Naval Ocean Systems Center
San Diego, CA 92152-5000

2

Technical Document 2171
September 1991

Development of a Modular Robotic Architecture

R. T. Laird
R. P. Smurlo
S. R. Timmer

Approved for public release; distribution is unlimited.

NAVAL OCEAN SYSTEMS CENTER
San Diego, California 92152-5000

J. D. FONTANA, CAPT, USN
Commander

R. T. SHEARER, Acting
Technical Director

ADMINISTRATIVE INFORMATION

The work in this report was performed during FY 91 by the Advanced Technology Branch (Code 535) of the Naval Ocean Systems Center as a project of Independent Exploratory Development (IED) program. Sponsorship was provided the Office of Chief of Naval Research, Arlington, VA.

Released by
S. W. Martin, Head
Advanced Technology
Development Branch

Under authority of
D. W. Murphy, Head
Advanced Systems Division

PK

CONTENTS

APPENDICES

FIGURES

1.0 INTRODUCTION

1.1 OBJECTIVE

The objective of this project is to develop the hardware and software components for constructing and controlling reconfigurable, modular robots (MODBOTs). While numerous robotic control architectures currently exist, few offer the degree of flexibility and modularity that is required to support rapid integration and prototyping of evolving (processor and sensor) technology into demonstrable systems. The proposed architecture emphasizes standard electrical and mechanical hardware interfaces between distributed processing modules, and standard software libraries that provide communication services and process control across a wide range of processors. The product is a standardized set of tools for building robotic systems that can be easily reconfigured as project requirements and technology change.

The first-year effort is concentrating on specification and design of the architecture, while the second- and (potential) third-year efforts will pursue implementation and demonstration of the MODBOT concept as applied to an actual application, such as physical indoor security.

1.2 SCOPE

This document describes the high-level architecture requirements and introduces the concepts related to MODBOT systems. The preliminary design and an example application of the architecture are detailed. The material is divided into the following sections:

Section 1.0 is an overview of the modular architecture.

Section 2.0 discusses the reasons why a new robotic architecture is needed, and gives examples of various MODBOT applications (both immediate and future). The section includes an overview of previous work, and briefly summarizes related systems that were investigated while developing the modular architecture.

Section 3.0 presents the requirements of a MODBOT architecture, and outlines what it must support in terms of capabilities, from both the developer's and the user's point of view.

Section 4.0 describes the MODBOT system architecture in terms of the major hardware and software subsystems. Details on the application of the architecture to a mobile security robot are included.

Section 5.0 describes MODBOT operation in terms of automatic or built-in capabilities such as self-diagnostics, self-configuration, and self-preservation. The communication and the coordination of multiple robots are outlined.

Section 6.0 presents a brief system development plan. Independent development of the MODBOT sensor, the actuator, and the processor modules is also discussed.

Section 7.0 defines the various acronyms and abbreviations used throughout the text.

Section 8.0 lists the reference material.

1.3 OVERVIEW

The Modular Robot Architecture (MRA) describes both the hardware and software components that are used to create a MODBOT. The MODBOT itself is a generic entity that must be customized by the developer or intelligent user for a particular application. The MRA facilitates customization of the MODBOT for specific tasks by providing sensor, actuator, and processing modules that can be configured in the manner demanded by the application. The Mobile Security Robot (MOSER) is an example of a MODBOT that will be developed using the modular architecture.

Conceptually, the MODBOT is similar to the IBM PC with its expansion slots; adding a module to a MODBOT is like adding a peripheral card to a PC. One simply plugs a card into an available slot, installs the supplied software drivers, and immediately incorporates the new capabilities of the card into the system. Adding smarter, better, and faster modules and capabilities to a MODBOT will be equally simple. The ability of the MODBOT to accept modules of increasing complexity provides the MRA with its evolutionary growth potential, and has been a primary motivating factor for the development of the architecture.

Simply stated, a MODBOT is a collection of independent modules of varying intelligence and sophistication connected together by a generalized, distributed network. The MRA does not require a particular physical module configuration nor does it require that all modules be located physically together. The generic MODBOT is illustrated in figure 1.

For systems involving direct human supervision, a MODBOT is divided into two physically separate computing systems: the Control Station (CS) and the Remote Platform (RP). The CS is a single module that is remote from the rest of the MODBOT. The RP consists of several modules and is connected to the CS by a telemetry link that acts as a network bridge. Two possible implementations to this approach are given in figures 1b and 1c. Only in systems that are strictly autonomous would the CS be located with the RP (figure 1b).

(Since most of the systems developed using the MRA will involve remote human control at various levels, the division of the MODBOT into separate CS and RP systems will be assumed throughout the remainder of this document.)

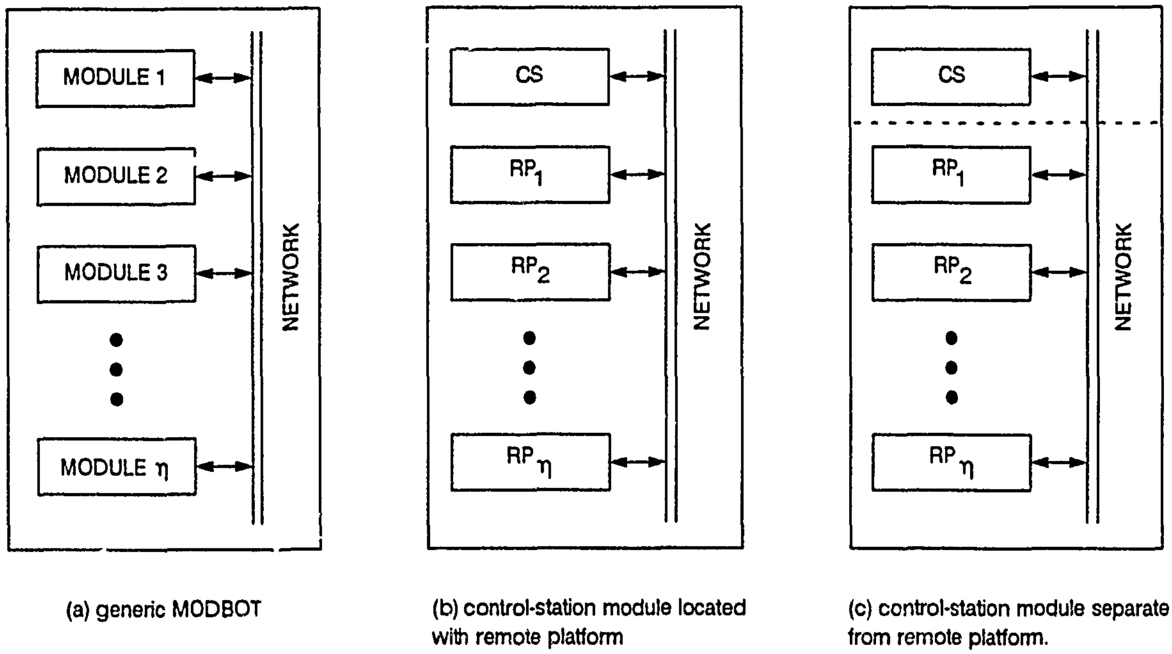


Figure 1. Robot module configurations.

The MRA is the framework around which robotic applications can be developed. The MRA supplies a set of standard hardware and software components that the user then assembles to build a modular system that can be easily upgraded as requirements and technology change. Standardized components provide a common interface for integrating the various parts of a system such as sensors, processors, and information. From a developmental viewpoint, the MRA is the "glue" that holds the pieces together.

A variety of applications can be addressed with the modular architecture, from indoor security to outdoor surveillance, but the architecture is especially useful for developmental or prototype applications. (Each hardware implementation of the architecture addresses a different set of robotic applications.) The ability to reconfigure and change modules lets developers quickly test new technology with a minimal amount of integration overhead (and expense). Figure 2 shows the relationship between the components of a typical modular system in an indoor application.

Modular robots will be particularly valuable in the laboratory environment where requirements continually change. Ideas can be implemented and tested quickly in a modular fashion and, when satisfactorily debugged, transferred to the deliverable system.

Operation of a MODBOT depends primarily upon the application. The MRA provides a software "kernel" around which application-specific control methodologies and algorithms can be implemented. Only rudimentary process control is provided by the modular architecture. More sophisticated coordination must be supplied by the developer as required.

The MRA is a generic tool that must be customized from both a hardware and software standpoint before a particular application can be addressed. The flexibility of the MRA, made possible by standard interfaces and a variety of hardware and software "hooks," allows developers to configure a robot to specific needs. Standardized interfaces and a modular hardware design allow for independent development of sensor, actuator, and processor subsystems that can be tested and debugged offline. Final system integration is performed much more efficiently since the functionality of the components being added has already been verified.

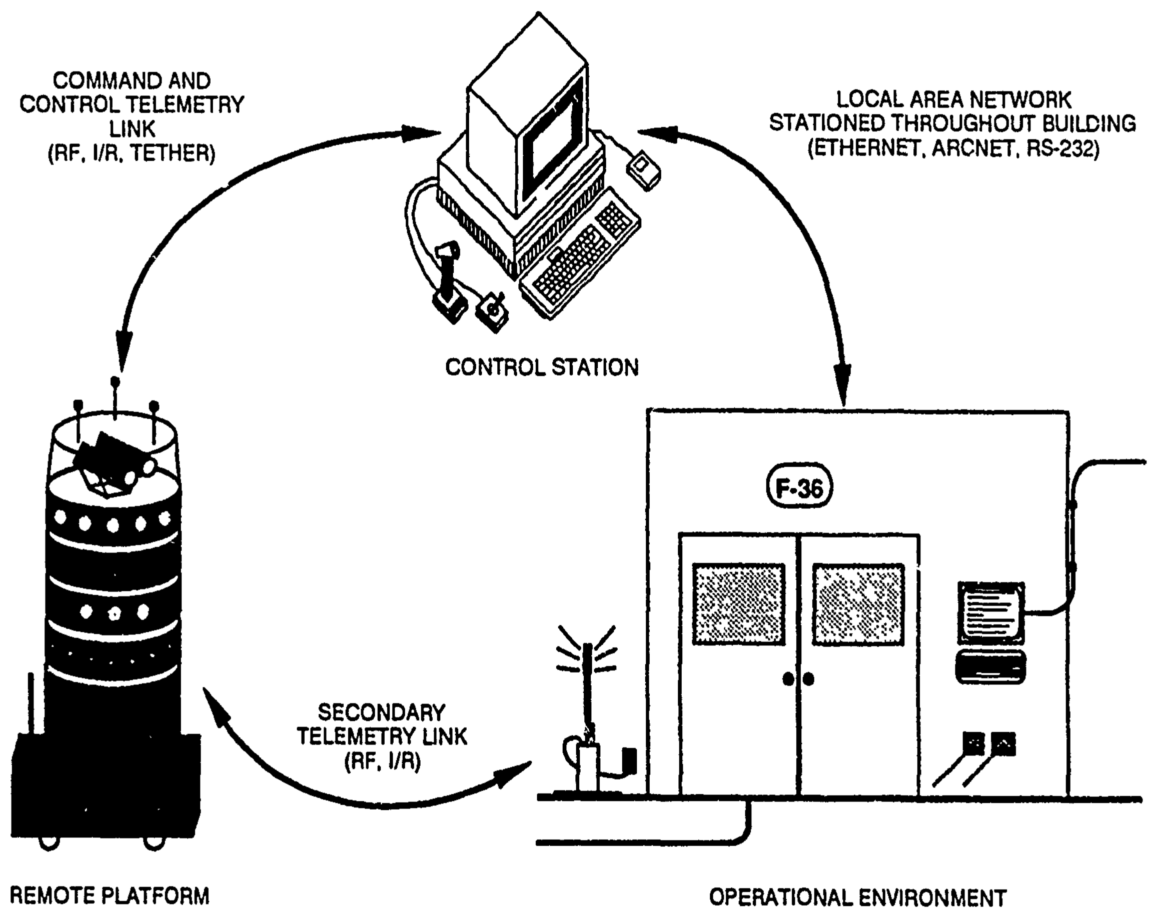


Figure 2. The relationship between components of a modular-robotic system.

2.0 BACKGROUND

2.1 THE NEED FOR A MODULAR ARCHITECTURE

Most current and projected nonindustrial Navy applications of robotics involve mobile systems. The development of a suitable processing and control architecture is a task that historically has been independently undertaken for individual robotic projects, each addressing different operational needs. Results often do not succeed due to insufficient funding, awareness of the issues and alternatives, or a tendency to become outdated. Furthermore, the majority of efforts have produced application-specific control systems that are difficult to adapt to more than the problem at hand. The development of a flexible, powerful, and widely available "core" high-level processing system with evolutionary growth potential for use on mobile robots (ground, air, surface, and underwater) will greatly alleviate these problems. An atmosphere of standardization and compatibility can be fostered among systems throughout the fleet.

A standardized, modular control system will also reduce the costs associated with development of customized architectures. Only the configuration of the pieces would have to be done each time, not the redesign of the entire system. In addition, a standardized architecture will promote software/hardware reusability in that identical modules and components will be used in several places, reducing both development time and cost.

The MRA is a generic control system with a standard set of hardware and software tools that can be used to design modular robots with a high degree of flexibility and extensibility. Several architectures currently exist that can be used to construct the control mechanisms for complex (robotic) systems. The Realtime Control System (RCS) developed by the National Bureau of Standards (NBS, also known as National Institute of Standards and Technology [NIST]), is an example (Barbera, Fitzgerald, & Albus, 1982). The MRA, however, is designed specifically to support the development of modular robots and modular control systems (in the general case). The MRA emphasizes standard hardware (electrical/mechanical) and software interfaces to promote development of capabilities by multiple activities (e.g., Navy, Army, Air Force, Marine Corps), the products of which can then be easily integrated to form a cooperative solution to a common problem.

2.2 APPLICATIONS OF THE MRA

The MRA can be used on a variety of applications ranging from a simple embedded device control to sophisticated autonomous robot control. Very little in the *specification* of the architecture restricts its use to a given class of control applications. Typically, however, a particular *implementation* will restrict the architecture to a specific set of problems. The implementation described in this document is aimed at the control of mobile robots.

2.2.1 Modular Developmental Testbed

The primary purpose of the MRA is to support the development of robot modules and control algorithms that will be used to build MODBOTs. Module development and integration is facilitated by the use of standard interfaces and procedures. Developers are required to conform to the standards, but are allowed a great deal of flexibility in the types of modules that can be developed. The actual control methodology (i.e., how the robot's action is controlled) is also "modular" in a sense and can be modified by the developer to obtain any desired behavior.

The module concept allows for independent development of new sensor, actuator, and software control components. These components are typically developed as modules for MODBOT applications, but the modules may serve as a simple means for testing new components that are not necessarily destined for MODBOTs (or even robotic applications).

Once the hardware modules and control algorithms have been developed, specific applications can be addressed. (Useful systems must solve real-world problems, and the architectures upon which they are based must be proven capable of performing. The developmental testbed is necessary but, alone, is not sufficient to solve problems.)

2.2.2 Physical Security

The first instance of a MODBOT to be developed under the MRA will be the Mobile Security Robot (MOSER). MOSER addresses the need for physical security within the confines of a structure such as an office building or a warehouse. The security robot will be an autonomous, modular, mobile system responsible for detecting intruders and responding to the assessed threat. It will duplicate several of the sensor and processing components found on ROBART II (Everett et al., 1990) with several improvements in the area of distributed processing, such as system networking, modularity (module design), and dynamic system configuration.

This application was chosen as the first implementation of the MRA for a number of reasons: (1) the familiarity of the security task to the development team, (2) the current availability of an existing, successful security robotic testbed (i.e., ROBART II), and (3) the desire to develop an inhouse mobile (security) robot capability.

MOSER is intended to be a fully autonomous system capable of operating within its environment without human supervision. However, because intelligent autonomous control is not readily achieved, MOSER's capabilities will be extended incrementally as levels of control (from teleoperated to autonomous control) are added to the robot's control scheme.

MOSER consists of the following hardware systems (items 4 to 7 are not specifically discussed here):

1) Control Station (figure 2).
2) Remote Platform (figure 3).
3) Telemetry Link (initially a tethered cable).
4) Environmental processor.
5) Fixed environmental sensors and actuators.
6) Navigational aids (i.e., beacons, freeway markers)
7) Local Area Network stationed throughout environment.

The software portions of the MRA will be used to tie the hardware systems together through the standard interfaces mentioned above. This document simply introduces MOSER and physical security as an application of the MRA; detailed design information for MOSER would be given in hardware and software design specifications.

2.2.3 Other Applications

Development of the MRA gives the Navy a rapid-robotic-prototyping capability, and makes immediately available the tools to construct modular (robotic) solutions to other problems. Almost any application involving distributed processing on an intermediate scale (e.g., less than 255 processors) can be implemented in a modular fashion. Three possible uses are given below:

Waterside Security

Physical Security could be performed by an outdoor version of MOSER. Problems that arise when moving from indoor to outdoor systems include autonomous navigation (waterside environments being a bit more harsh and unpredictable than an office building or even an enclosed warehouse). A MODBOT could be constructed to patrol a "secure" pier in an autonomous mode (with the aid of fixed beacons, perhaps), and could alert a remote operator of intruders or of the absence of important cargo. The advantages of robotic security guards are numerous (Everett, 1988).

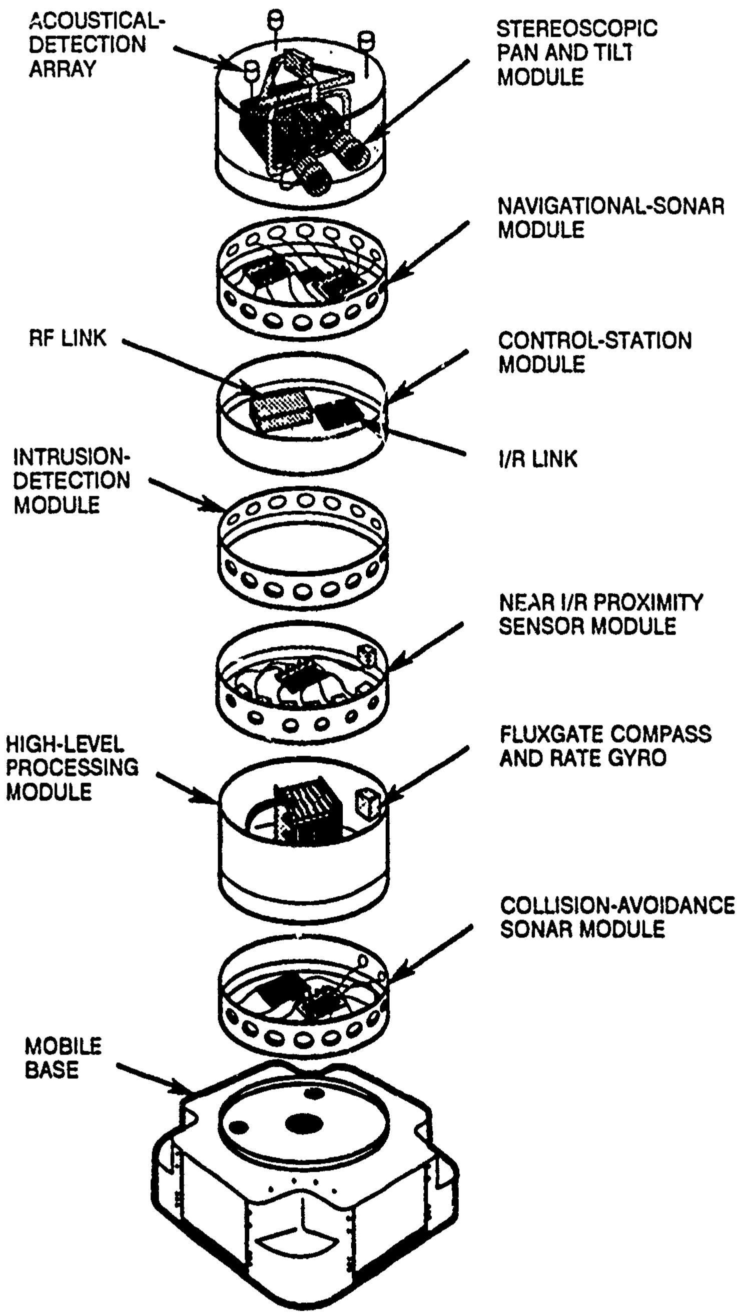


Figure 3. MOSER remote-platform (RP) module configuration.

Underwater Exploration

Either a tethered or an autonomous MODBOT could be used for ocean exploration and surveillance. A MODBOT could be easily adapted to a submergible platform. Special underwater sensors could detect and classify underwater objects. Acoustical (sonar)-sensor modules could also be developed for monitoring and locating underwater activity to be investigated autonomously or under the supervision of a remote operator. The navigational problem is even more difficult when another dimension is added.

Intelligent Underwater/Surface Sensor

Stationary MODBOTs placed at critical locations near harbors, sea access lanes, or other points of interest could be used as intelligent sensor platforms. Several highly sophisticated sensor modules employed on a single MODBOT would detect the presence (or absence) of specific objects (environmental conditions). The MODBOT could then be programmed to respond by simply recording the event or perhaps respond in a more active manner. In this application, the MODBOT is nothing more than a data-fusion machine with some heuristic applied to generate the desired response.

2.3 RELATED WORK

Extensive research in mobile-platform development has taken place at Stanford, Carnegie-Mellon, MIT, DARPA, Martin-Marietta, NBS, and NOSC (to name a few). The sections below summarize the architectures relevant to this effort that have been researched. Illustrations of the architectures are included for easy comparison between several different approaches to the control of intelligent machines (figures 4 through 9).

2.3.1 Generic Robotic Processing Architecture (GRPA)

The predecessor to the MRA (temporally, if not logically), the Generic Robotic Processing Architecture (GRPA), was used on the Unmanned Ground Vehicle/ Teleoperated Vehicle (UGV/TOV, formerly GATERS) program (Hughes et al., 1990). As implemented on the GATERS project, GRPA was basically a mechanism for mapping operator input on the control station to vehicle actuators on the remote platform, successfully demonstrating a relatively sophisticated degree of teleoperated control. This version of GRPA *implemented* only a portion of the architectural concepts as initially outlined (Aviles, Laird, & Myers, 1988), but did prove that the basic system design was capable of realtime device control (incentive enough to pursue further development of the modular architecture concept).

The GRPA processing architecture (figure 4) is based upon the Virtual Systems Interface (VSI) data structure. Objects (e.g., sensors, actuators, state variables) are divided into four categories: remote-system sensors, remote-system actuators, control-station sensors, and control-station actuators. The control portion of GRPA is responsible for mapping local controls onto remote actuators and for mapping remote sensors onto local displays. Virtual device drivers—hardware-specific I/O routines—are used to couple the physical world to objects in the VSI. A remoting system is used to transmit and receive encoded packets between the control station and the remote vehicle.

Several key features of the MRA are common to the original goals of GRPA (e.g., standard software interfaces, a core high-level control system, and an extendible modular design). The MRA is partially a renovation of some of the GRPA ideas as begun under the Distributed Robotic Control Architecture IED project in FY 88. Whereas GRPA emphasized "compilation" of robot descriptions into actual implementations, the MRA emphasizes distribution of function into modules that can be dynamically configured. GRPA was concerned primarily with the standard software interfaces; the MRA attempts to standardize both the software and hardware interfaces between distributed components. Additionally, system networking concepts introduced in GRPA will be expanded upon and implemented under the MRA.

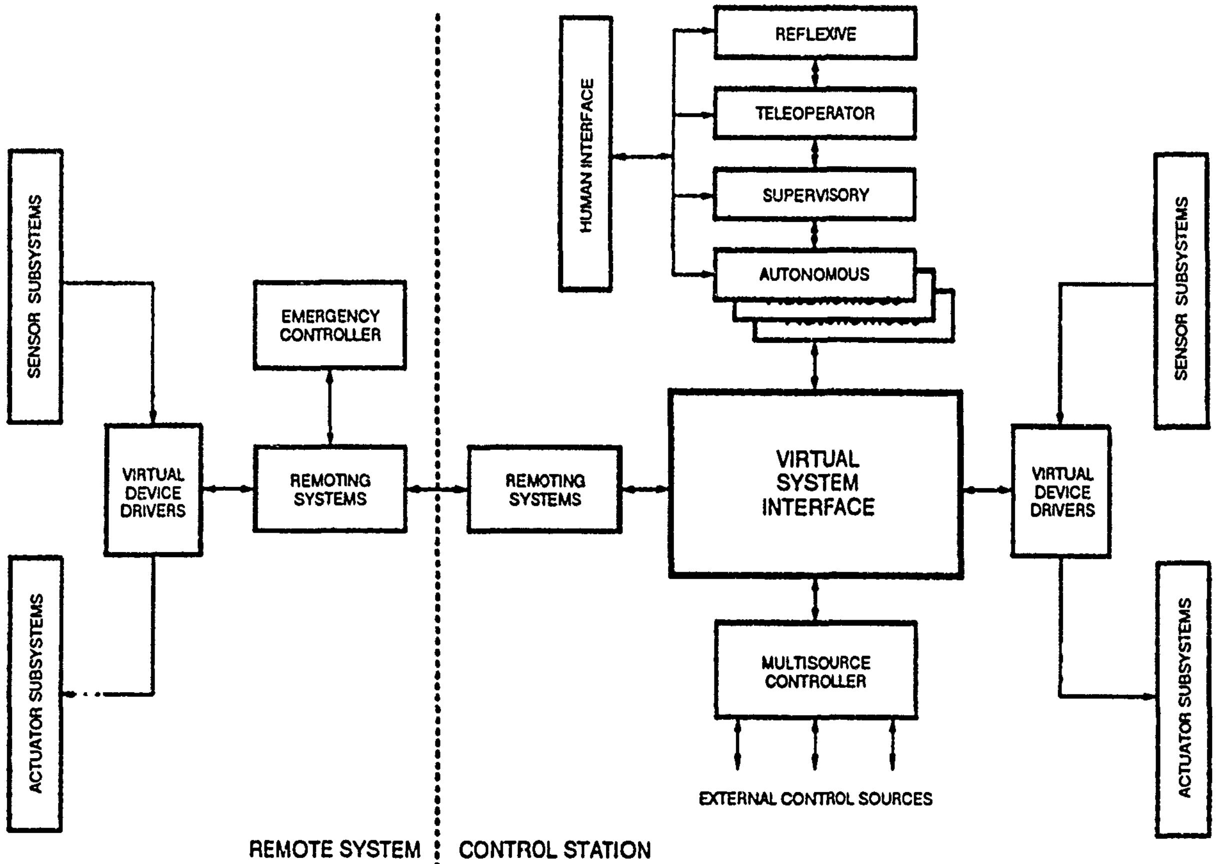


Figure 4. Overall GRPA architecture for all levels (Aviles, Laird, & Myers, 1988.)

2.3.2 ROBART II

ROBART II, an autonomous security robot being used at NOSC, is a testbed for the development of hardware/software solutions to problems facing mobile (sentry) robots. Over the last three years, ROBART II's sentry capabilities have been enhanced to make it one of the most advanced autonomous security robots used by the Navy (Everett et al., 1990).

The computer architecture used on ROBART II (figure 5) is designed as a distributed hierarchy of ten onboard microprocessors acting as dedicated controllers with a remote microcomputer used as the high-level system Planner. The Planner performs the functions of path planning, obstacle avoidance, position estimation, map making, sonar-range plotting, and security assessment. The onboard computers are dedicated to such functions as head positioning, sonar ranging, platform mobility, and speech synthesis; one of these processors acts as the system Scheduler and coordinates the activity of the others. Communication between the onboard computers is controlled by the Scheduler and is accomplished by means of an 8-bit parallel bus and a multiplexed RS-232 serial port. Communication between the remote Planner and the robot is via a 1200-baud radio link.

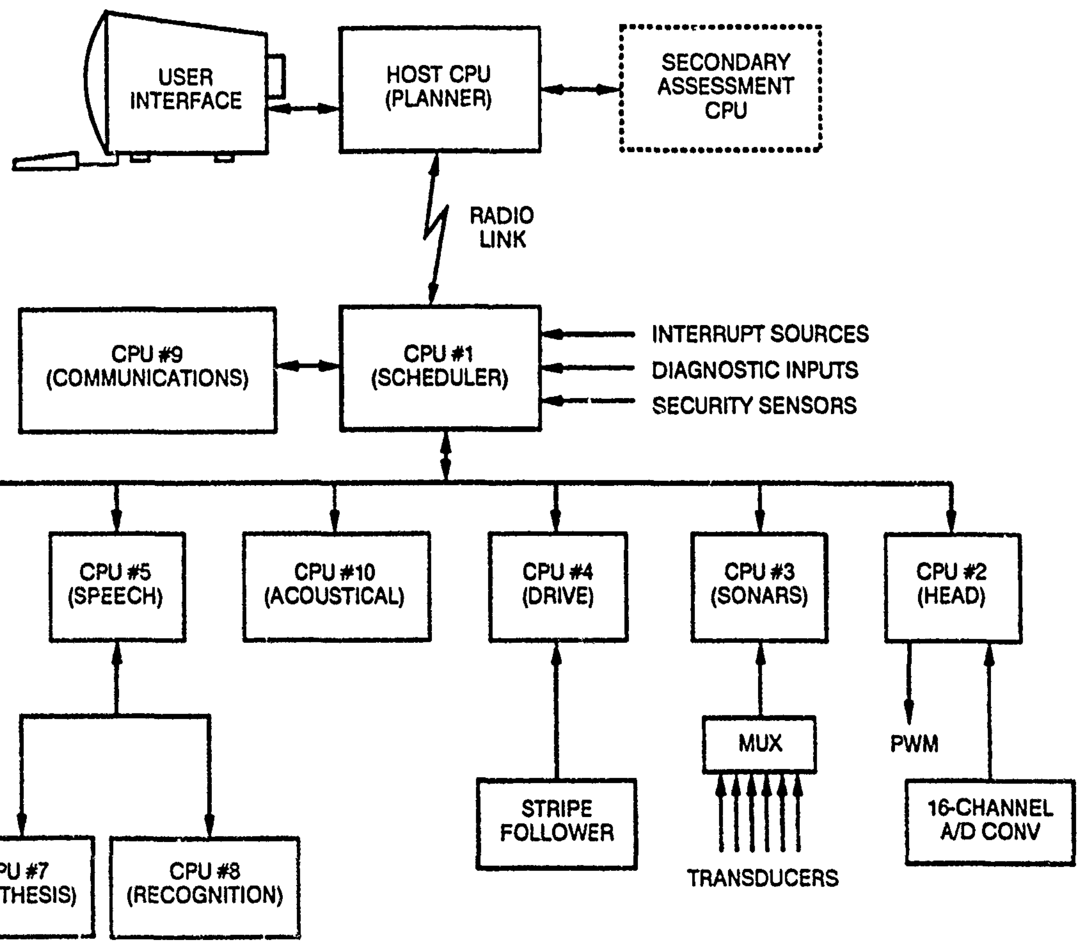


Figure 5. Distributed computer architecture for ROBART II (Everett et al., 1990.)

Because ROBART II is not owned by NOSC, a replacement robot is needed to enable research and development to continue. The MOSER will provide NOSC with a security robot having capabilities on the order of ROBART II as well as the potential to exceed those capabilities.

The MRA will borrow from the experience gained in the design and implementation of ROBART II and offer new solutions to problems that face all evolving, distributed computer systems (e.g., management of growth and effective communication between processing components). Much of the technology used on ROBART II will be transferred to MOSER, particularly the autonomous navigation and security-assessment concepts. This technology transfer will allow the MRA physical-security application to be developed much faster than would otherwise be possible (section 2.2.2).

MOSER's development is *partially* the result of the desire to create a more powerful ROBART II with the ability to easily add new capabilities; integration of additional sensor and processing components is becoming difficult for the developers of ROBART II because the robot's enclosure is nearly full. The modular approach will allow the technology originally intended for ROBART II to be implemented on MOSER (e.g., line following to aid in vehicle navigation).

2.3.3 Other Architectures

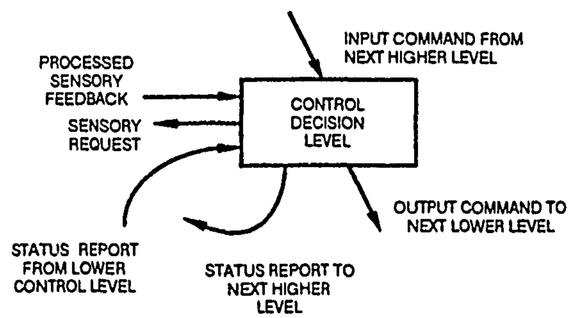
There are perhaps hundreds of control architectures that have been developed for use on intelligent robotic systems. However, the literature search revealed four approaches that stand out in terms of both relevance to development of the MRA and in terms of the unique architectural concepts they describe. These efforts influenced the generation of the high-level requirements for the MRA. The paragraphs below *briefly* summarize each of these architectures and provide insight into some of the design decisions made in developing the MRA.

Realtime Control System (RCS)

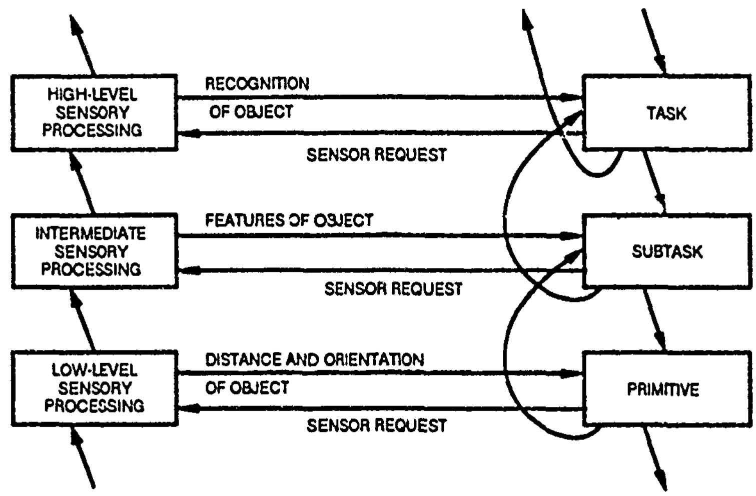
The RCS, developed by the National Bureau of Standards (NBS) as an environment for the design and implementation of distributed, task-based control applications (Barbera et al., 1984), has been successfully implemented in such efforts as the NBS Automated Manufacturing Research Facility (AMRF), a multirobot manufacturing shop (McGain, 1985).

The RCS does not include specifications for the actual control algorithms, but does include methods for organizing and interfacing those algorithms. The MRA is similar in that it specifies only module-level communication and data abstraction; it does not specify the algorithms used to control processes within a module. Both the RCS and the MRA provide developers with a set of tools to implement a variety of control systems in a structured, standardized manner.

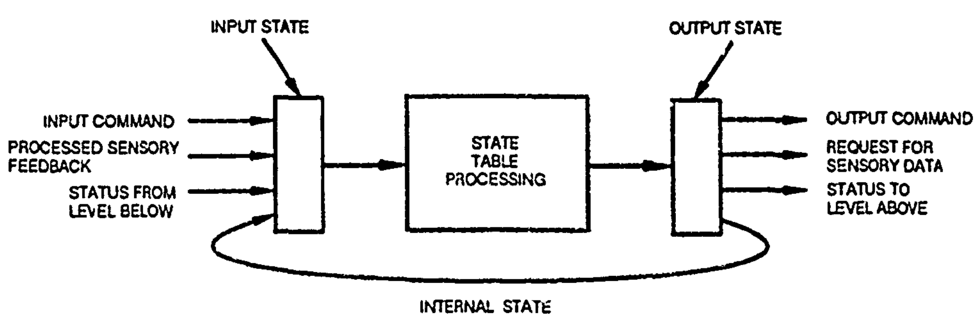
The RCS is based on the concept of generic control levels that are organized in a hierarchy based upon a task decomposition of the problem (figures 6a and 6b). Each of the control levels has a standard input and output data-set interface that facilitates modularity. Processing at each control level consists of sampling input data and generating output data based upon user-defined state table information (figure 6c). Sensory feedback is provided to each control level by sensory-processing modules available at that level (figure 6b). Realtime response is obtained by distribution of control levels onto multiple processors communicating through common memory.



(a) generic control level



(b) hierarchical structure and sensory-control interaction



(c) state-table inputs and outputs.

Figure 6. RCS architecture components (Barbera, Albus, & Fitzgerald, 1982).

Like the RCS, the MRA defines a set of standard interfaces that allow for modularity and the ability to easily introduce new capabilities. The MRA also uses distribution of function onto multiple processors to obtain realtime response where required. Unlike RCS, however, the MRA does not directly specify a particular control architecture (e.g., hierarchical decomposition); it only specifies how the modules that compose the architecture (whatever it may be) communicate, and what common functionality each module must provide.

NASA/NBS Standard Reference Model (NASREM)

NASREM is a hierarchical control-system architectural model designed by NBS to support development of the control-system architecture for the Flight Telerobot Servicer (Albus, McGain, & Lumina, 1984). NASREM, primarily intended for telerobotic systems, has been extended to include control of autonomous systems as well. As depicted in figure 7, control is divided into three conceptual processes at six different levels. Actions are performed by decomposing higher level commands into tasks that eventually produce real-world results (e.g., manipulator movement). The tasks perform different types of mathematical transformations at each level. An operator interface allows the user to control the system at any one of the six levels. NASREM is an implementation of RCS just as MOSER is an implementation of the MRA. NASREM was based upon the system architecture that evolved from RCS. However, each offers slightly different concepts that the MRA intends to support (or seeks not to exclude).

The goals of the MRA include features offered by and contained in NASREM: flexible and well-integrated system interface, control at various levels of interaction, world modeling at various levels of abstraction, and high-level command execution. Like NASREM, the MRA is a "standard model" that must be customized to meet the requirements of a particular application.

Subsumption

The subsumptive architecture developed by Brooks (1986) is based on task-achieving behaviors organized vertically as horizontal slices (figure 8a). This is different from the traditional horizontal decomposition of tasks (e.g., perception, modeling, planning) into vertical slices. Layers of behaviors can be subsumed by higher levels to produce more intelligent behavior (figure 8b). Layers are composed of modules that are finite state machines with associated data variables (figure 8c).

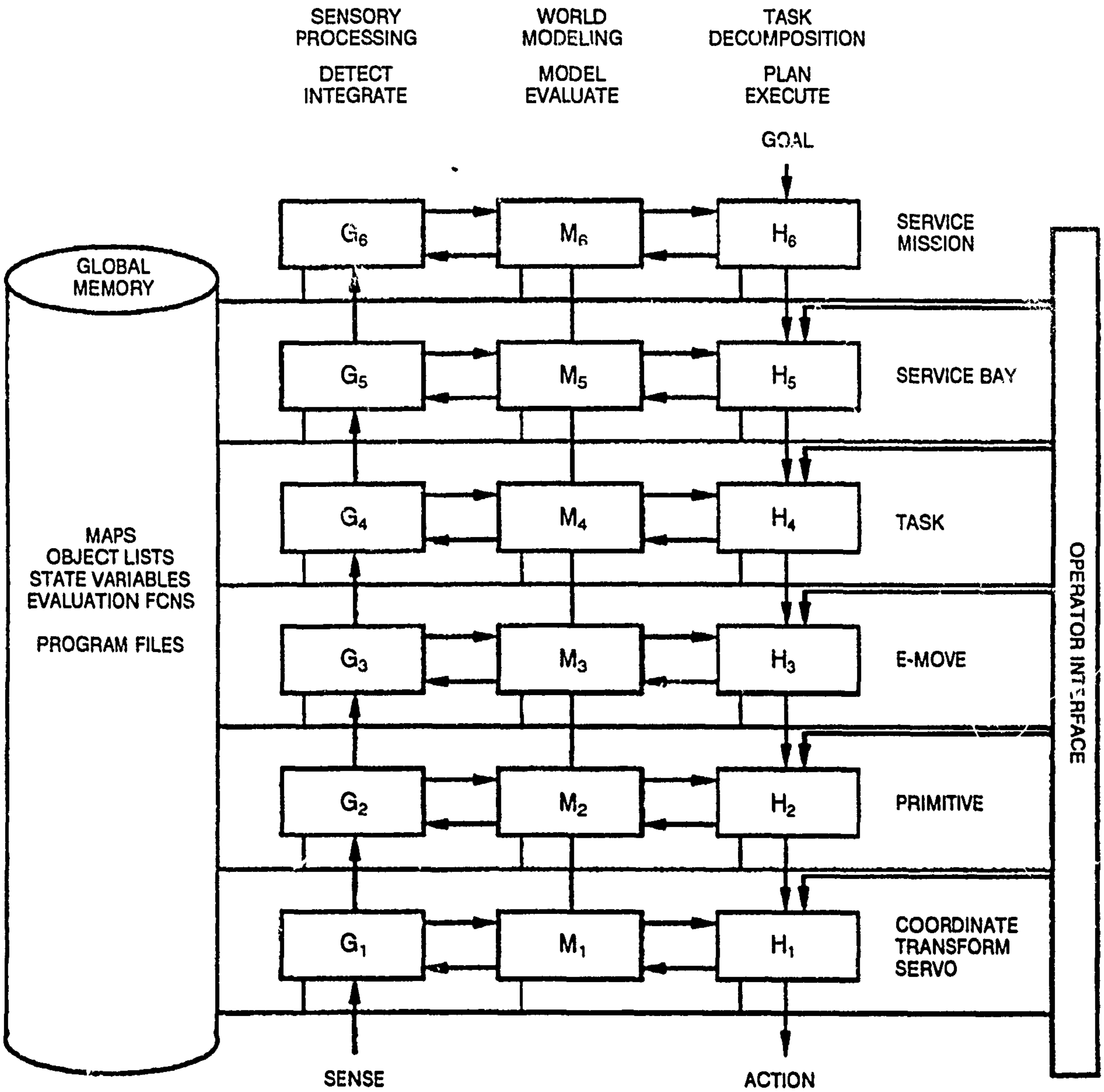
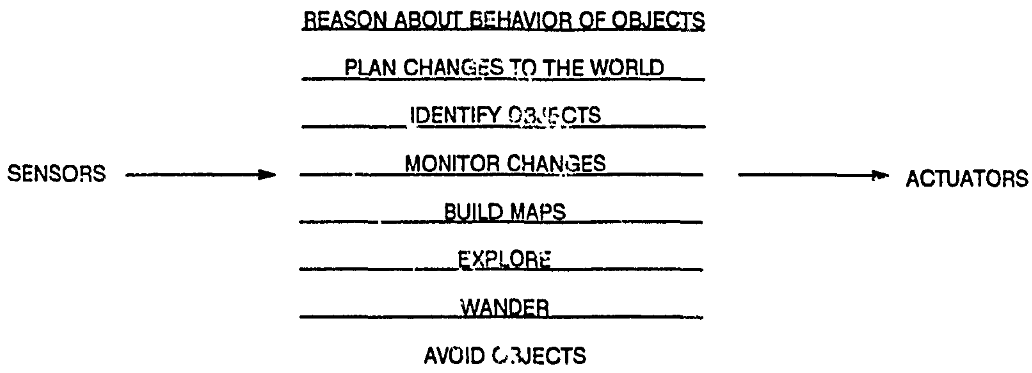
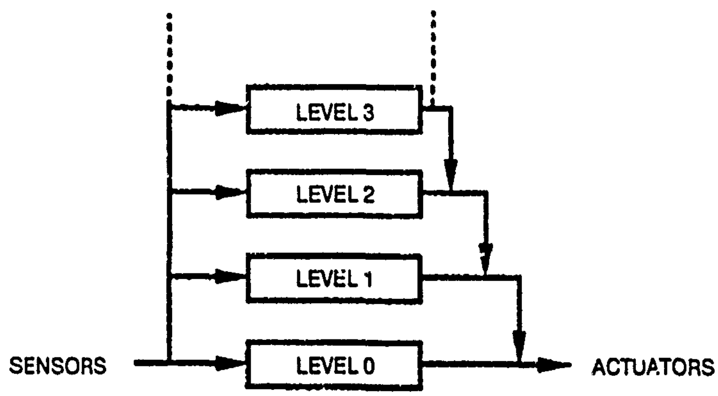


Figure 7. NASREM (RCS) hierarchical control system architecture (Albus, McGain, & Lumina, 1984.)

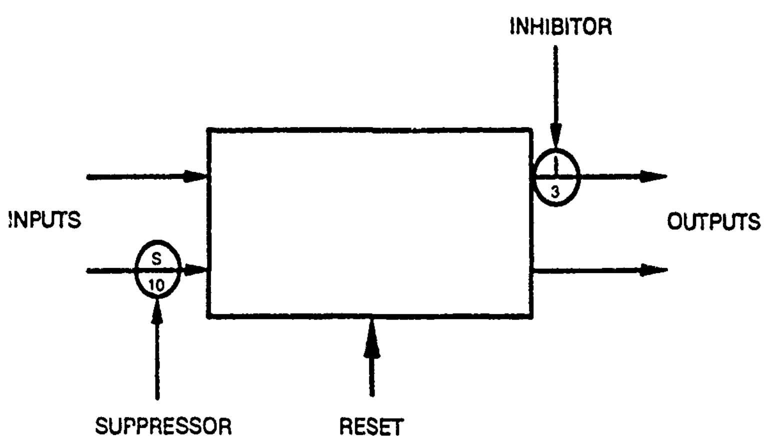
The subsumptive approach is significant in that it provides almost immediate functionality at lower levels of behavior, and allows for incremental growth toward an autonomous, useful purpose. The MRA can support subsumption directly by virtue of its modularity and flexibility in supporting behavior-based control algorithms at the module level. The control system requirements of multiple goals, multiple sensors, robustness, and extensibility identified by Brooks (1986) as being inherent to the subsumptive control architecture, also apply to the MRA.



(a) vertical decomposition of control system into horizontal slices



(b) layered control levels subsumed by higher levels



(c) subsumption module with input and output lines.

Figure 8. Subsumption architecture (Brooks, 1986).

Nested-Hierarchical Controller (NHC)

Figure 9 shows the architecture of the NHC that its developer (Meystel, 1988), states as being an integral part of all autonomous mobile robots (AMR). The NHC offers concepts that are applicable to the development of the MRA and to MODBOT implementations such as MOSER. Of particular interest is the decomposition of motion into planning, navigating, piloting, and control. This can easily be implemented under MRA by using distinct modules to carry out each of the levels of motion. Each module could also implement the corresponding level of perception and knowledge representation used by the motion process. Actuators and sensors can also be modeled and implemented under the MRA in a manner similar to that of the NHC.

The NHC architecture is divided into three functional areas: perception (how the robot sees its environment); knowledge (how the robot models the perceived environment); and planning (how the robot accomplishes goals and reacts to environmental situations). Each of the three areas is broken down into levels of data abstraction that are useful to the corresponding level of control. Direct sensory feedback is available to the lower-level control subsystems for realtime and reflexive response.

Although architectures do exist for building modular robotic systems, the modular approach proposed does not force a particular control scheme upon the developers. Instead, it offers rudimentary control mechanisms that can be replaced by those supplied by system developers. In addition, very few systems offer the degree of hardware modularity and flexibility that is achieved through *simple*, standardized interfaces supported by the modular architecture.

Each of the architectures above can be implemented using the components of the MRA. The modular architecture itself offers only a simple default controller. More sophisticated robot control can be obtained by implementing other methodologies in a modular fashion by using selected (standardized hardware/software) components; in this case, the modular architecture is nothing more than a development and integration tool. The modular architecture is not intended to replace those above, but to facilitate their implementation as modular systems.

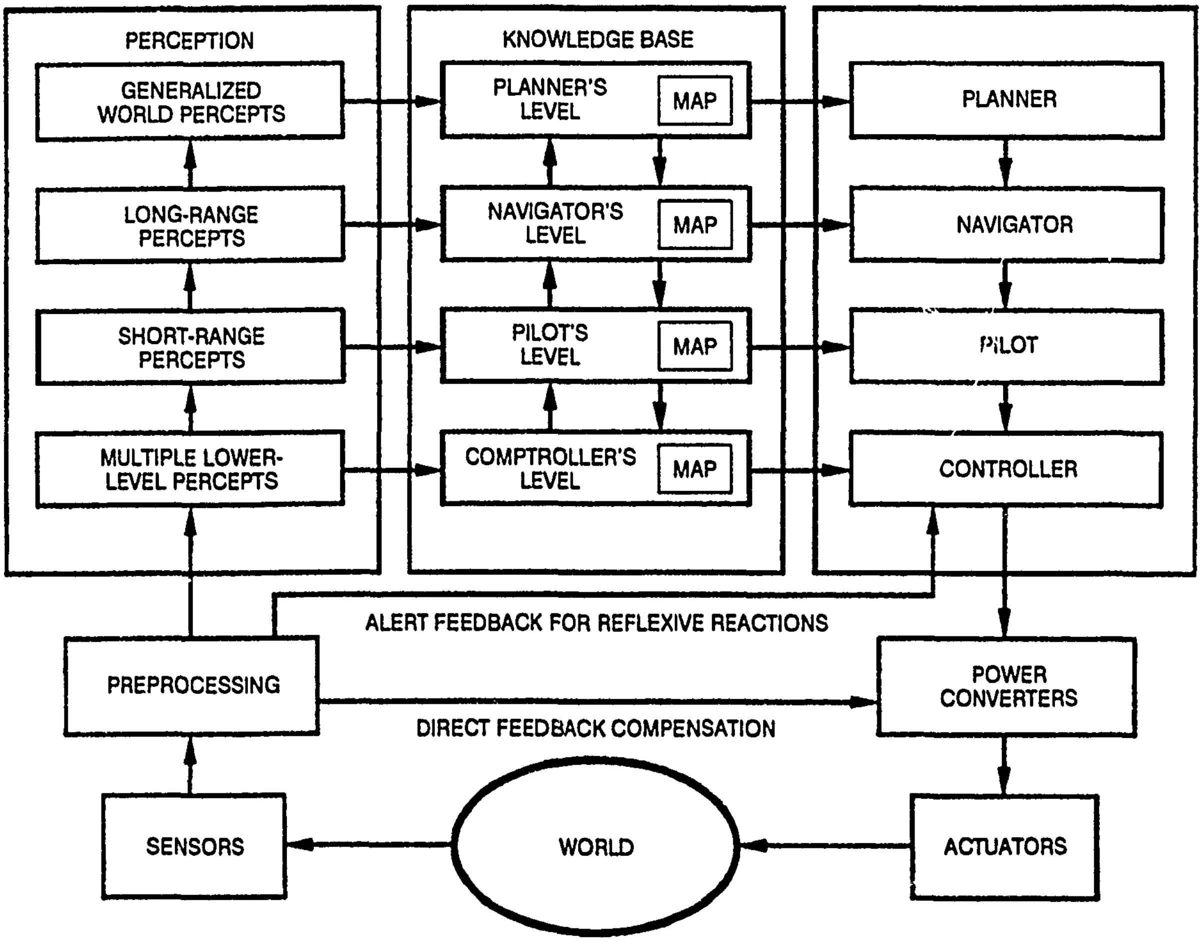


Figure 9. The Nested-Hierarchical Controller (NHC) architecture (Meystel, 1988).

3.0 A FRAMEWORK FOR DEVELOPING MODULAR SYSTEMS

3.1 SPECIFICATION REQUIREMENTS

Requirements, even for systems that are *not application specific,* drive the design and should be specified in advance. These requirements specify the characteristics of an architecture that can be used to create modular systems with a great degree of flexibility in terms of hardware configuration and software operation. If the architecture possesses these characteristics, then it should be capable of supporting construction of modular robots.

The need for a new architecture was discussed in section 2.1. This section outlines the high-level software and hardware requirements. In the sections following, the MRA is referred to as "the architecture".

3.2 MRA HARDWARE REQUIREMENTS

The general hardware requirement is for an architecture that will support production and assembly of sensor, actuator, and processor modules that can be connected together using a generalized power, communications, and auxiliary control/feedback "bus." The MRA provides hardware components and guidelines that system developers use to build modular robots. Specific hardware requirements are listed below.

3.2.1 Hardware Design

Emphasis is placed upon a simple, flexible, and maintainable hardware design. Complex designs lead to complex problems that require complicated tools to solve. These components will have simple, standardized interfaces that facilitate system integration.

3.2.2 Support for Add-On Modules

Incremental development of capabilities will be supported by add-on sensor, actuator, and processor modules that can be configured in any manner required. Modules will typically be constructed from materials that allow easy physical connection, and are structurally adequate for the operating environment. The modules may be separate and connected only by wires that form the generalized module bus. The architecture will support, at a minimum, 16 modules in a single system.

3.2.3 Standard Platform Interface

The architecture will provide a standard platform interface that decouples the rest of the modular robot from the vehicle platform or base. The interface is a mechanical and electrical connection between the platform and one of the robot modules (for mobile systems this will be the "mobility" module). Standardizing this connection allows a modular robot to be moved between platforms without modifications. The platform interface is also responsible for regulating power supplied by the base to standard voltages that are made available to the rest of the robot. In terms of power, the platform must be capable of supplying 24V DC (raw), which will then be regulated to standard voltages of +12V DC and +24V DC (clean).

3.2.4 Standard Module Interface

Modules are connected to *each other* through a common communications and power bus. The bus provides a standard interface that specifies the required mechanical and electrical connections for modules. The interface should allow for growth in the bus (auxiliary control and feedback lines, for example). The bus, a conceptual device, is implementation dependent. The bus may be a set of wires, a connectorized backplane, or a combination of both. The communications portion of the bus will support rates of at least 1.8 megabits-per-second (Mbps) with the ability to detect and correct for simultaneous (interfering) bus-access requests. The power portion of the bus will supply a minimum of 10A at +12V DC and 15A at +24V DC. Standard voltages of +5V DC (or +8V DC), +12V DC, and +24V DC will be distributed to each module by regulating devices attached to the power bus.

3.2.5 Standardized Low-Cost/Low-Power Components

Building modular robots should be cheap in terms of dollars and power. Low-cost (less than $200), low-power complementary metal-oxide semiconductors (CMOS) devices (e.g., integrated circuits, regulators, microprocessors, etc.) will be used in the design of the standard hardware components. Off-the-shelf components should be used where possible. Battery-operated systems, in particular, must minimize power consumption wherever possible to maintain operation for extended periods. Standardization produces reusable, easily maintainable components that can typically be manufactured at a lower cost.

3.2.6 Unlimited Expansion

Above all else, the hardware architecture must be expandable. That is, provisions (hardware "scars") must be in place that will allow additional capabilities to be added at a later date. Application modules must be developed that can be assembled into a system solution with little or no integration overhead, that is, the time spent on integration is minimized. Expansion of cap .ities is limited only by available technology and, as technology advances, so toc ıld the abilities of a modular robot incorporating that new technology as an add-(module.

3.3 MRA SOFTWARE REQUIREMENTS

The general software requirement is for a computer architecture to support distributed processing among numerous functional modules that can effectively communicate with one another in a coordinated manner to accomplish a task in real time. The MRA provides software functions via standard libraries targeted at various processors that application programmers use to build these modules. Specific software requirements are listed below.

3.3.1 Distributed-System Software Services

Modular robots are distributed systems. The architecture must provide for distributed processing by means of software functions that, at a minimum, support interprocess communication and coordination. These functions will allow two or more modules to communicate with one another in an efficient manner, and will allow module processes to be coordinated such as tasks are coordinated in a multitasking environment. Response time, measured as the time from transmission of a typical length command packet to receipt of an acknowledgment of that packet, will be less than 100 μs.

3.3.2 Standard Module Application Controller

Each robot module is controlled by an application program. For modules that do not require a specialized controller (e.g., simple sensor or actuator systems), a standard (default) application controller will be provided. The controller will be responsible for managing the subsystems of the architecture to perform the function of the module. Operation of the controller is fixed and cannot be changed. Modules requiring more sophisticated control will replace the default controller with a specialized (user-supplied) application controller.

3.3.3 Standard Communications Protocol

Modules communicate with one another by using a protocol designed for high-level control of robot functions. A standard protocol will be used for interprocess communication, and for communication between external devices such as the control station. The protocol will support efficient transmission of commands and status information between modules. An existing protocol should be used if possible to promote interoperability between different systems. Also, adherence to the International Standards Organization (ISO) Open Systems Interconnection (OSI) model should be maintained with respect to a layered communications protocol (ISO, 1980).

3.3.4 Standard Device Interface

Devices are hardware components that interact with the real world to either measure (observe) it or effect (manipulate) it. Sensors and actuators, as well as the parallel and serial ports of a computer are devices. Modules of a particular system are often constructed using similar components with similar interfaces. The architecture will provide a common interface to these and other devices. The standard device interface acts to decouple the details of the hardware implementation from higher-level software. Device "drivers" will be provided to support control of microcomputer/microcontroller devices such as serial ports, parallel ports, and timers. Drivers will be provided for a variety of target systems (e.g., 8031 microcontroller, STD-bus V20/8088/80286 microprocessors, etc.). A separate mechanism will also be provided for standard access to sensors and actuators, as well as to logical and virtual devices.

3.3.5 Flexible Control Methodology

The modular architecture does not specifically provide a formal method of control other than that offered by the standard module application controller. Programmers can develop their own control methodology to be implemented by using the subsystems of the architecture. The architecture must be flexible enough to support adaptation of new and existing control methodologies to modular robots.

3.3.6 Dynamic System Configuration (reconfiguration)

To support changing application requirements, modular robots must be able to be reconfigured at will with little or no software changes necessary. The architecture will provide a means for automatically recognizing the existence or absence of a module and to adapt system operation accordingly. The developer will be free to add or remove modules as desired without the need to change software module "addresses" or "identifiers". Configuration of the system with respect to software module interfaces will occur dynamically at initialization time.

3.3.7 Remote Function Operation

Each robot module can perform an associated set of functions. This applies to most systems developed under the modular architecture. Remote function execution is provided by the architecture so that one module can command another to perform some operation and return the results upon completion. The architecture is responsible for sending the appropriate command and for coordinating returned results with the original function call. Remote functions are used by application programmers in a manner similar to normal program function calls.

4.0 SYSTEM ARCHITECTURE

4.1 HARDWARE COMPONENTS

The MRA hardware architecture defines a MODBOT as a set of modules linked by a generalized distributed bus. The architecture does not require that the modules be arranged in any particular physical configuration, nor does the architecture specify the types of modules that a system must contain. Mobile systems, especially those involving human supervision, typically consist of a control station (CS), a remote platform (RP), and a telemetry link that connects the two. Note that the architecture does not force this decomposition upon an implementation, it is merely a convenient way of describing mobile systems involving human control.

4.1.1 Control Station (CS)

The CS is viewed conceptually as a MODBOT module whose central processing unit is located remotely from its associated module on the RP. The telemetry link connects the CS processor to the remote portion of the CS. Figure 10 shows a typical CS and its associated peripherals. The hardware architecture specifies only that the CS hardware be capable of supporting the MRA standard software systems.

4.1.1.1 Human Interface

The CS is the primary operator interface to the MODBOT. The CS acts as a control and display environment that allows the operator/developer to manipulate and observe any part of the robot and its world that is modeled by the system. The user can control operation of the robot at any one of several levels, e.g., teleoperated, reflexive, or supervisory depending upon the application software. Sensory and other state information is transmitted from the robot to the CS for display and analysis.

4.1.1.2 Displays

The CS uses one or more displays to present command, control, and sensory information to the operator. Realtime video transmitted from the RP can be displayed on separate monitors or integrated into a single-monitor system. Some applications may require more sophisticated devices such as headmounted displays that provide stereoscopic vision. The selection of displays is completely dependent upon the requirements of the application.

4.1.1.3 Controls

Possible controls include a speech-recognition device called the Voice Navigator by Articulate Systems, Inc. The Voice Navigator connects to the SCSI port of the Macintosh and employs a menu-driven voice training system to learn voice commands of a specific operator. The audio speaker available on the Macintosh allows the CS to verify the command to tell the operator when the robot is done performing a task. The joystick selected for this application is the Advanced Gravis MouseStick. The Mouse-Stick features three programmable pushbuttons to allow for mouse-like menu control of specific functions, and is used to teleoperate (drive) the robot.

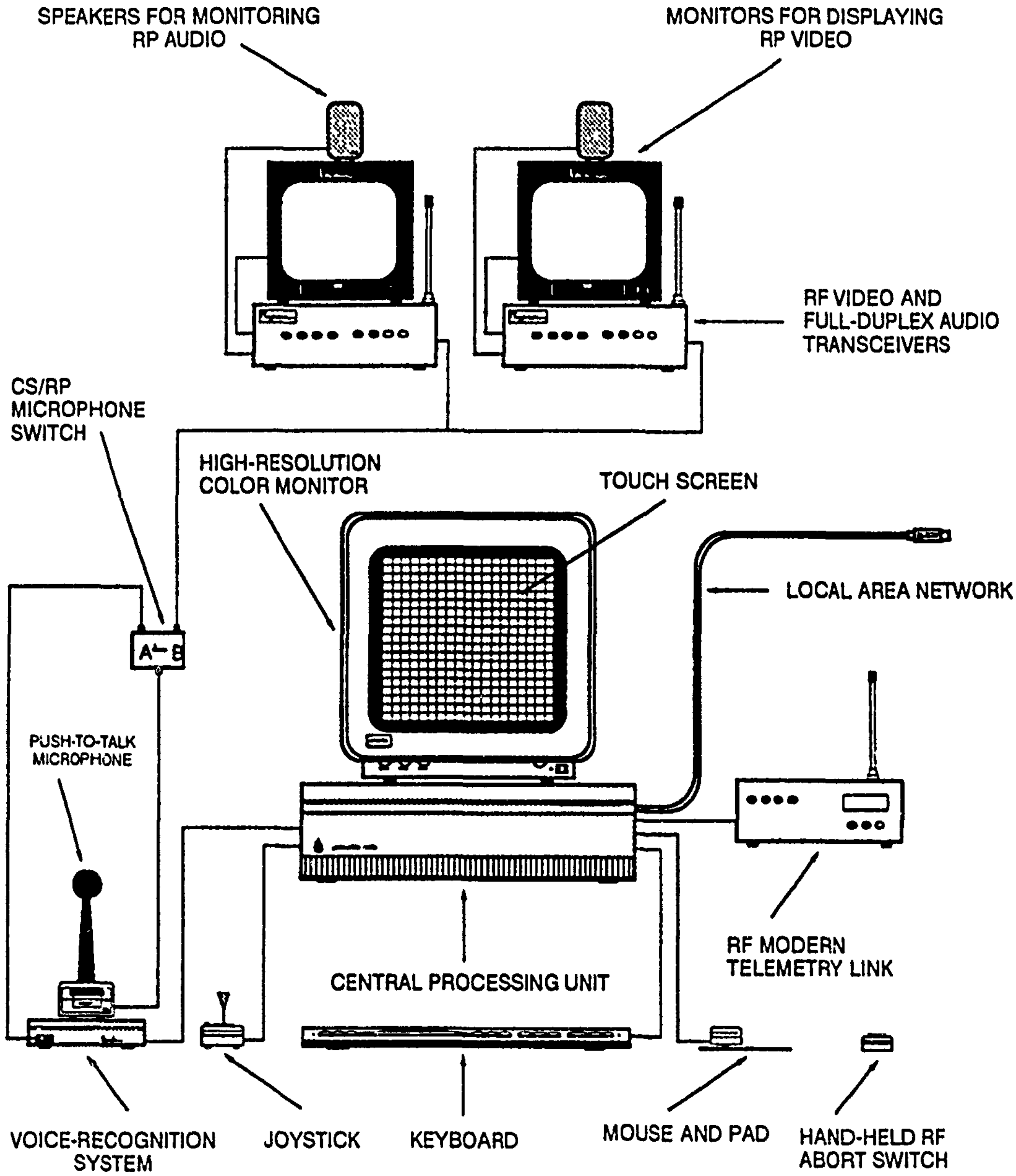


Figure 10. Control-station (CS) configuration showing external device connections.

4.1.2 Remote Platform (RP)

The RP, the remote portion of the MODBOT, represents what is normally referred to as the "robot". The RP is a series of modules that are connected together in a daisy-chain fashion by a distributed robot module bus. For mobile systems, the RP includes a propulsion system or base suitable for the application environment (i.e., land, air, sea, or other). The MRA specifies standard hardware interfaces between each RP module, and also indicates how the modules are attached to the base, electrically and mechanically. The architecture places no restrictions upon module arrangement.

4.1.2.1 Base

The base is comprised of the propulsion system and its accompanying power source. An outdoor robot would need a much more rugged platform and a fuel-driven system while its indoor counterpart would be most likely electrically driven. The only assumption the MRA places upon the base is that it must provide power to the rest of the system. For nonmobile robots, the base may consist simply of a mechanical module mount or frame and an electrical connection to a provided power source (e.g., a 24V DC transformer).

The base is viewed conceptually as an actuator with an associated mobility module that interfaces the base to the rest of the RP. The mobility module contains the standard MRA hardware components and is usually attached directly to the base.

For the indoor security robot, a modified version of the TRC Labmate is being used. The wiring of the original platform was unacceptable and was totally redone. In addition, the original motor controllers were replaced with much more reliable and robust units. The Labmate, a stand-alone, 24V DC battery-operated base, has an RS-232 interface to an onboard processor that controls basic platform functions such as point-to-point transit and continuous-path control. High-level commands, such as "go forward 5 meters" or "turn 45 degrees," can be issued from a host processor via the RS-232 interface. The Labmate also has a joystick interface that is used to manually control the base.

4.1.2.2 Robot Module Bus (MODBUS)

Power, communications, and auxiliary control are distributed to each of the MODBOT modules via the robot module bus or MODBUS. The bus provides a standard interface to each module and simplifies the connections between modules. As shown in figure 11, the MODBUS consists of three sets of signal lines that form the standard module interface. The lines are broken apart at each module where the different sets of signals are run to the appropriate devices; the communications lines are connected to the intelligent-communications node, the power lines are attached to the power distribution node, and the auxiliary lines are attached to components as required by the particular module to which they run (the auxiliary control/feedback lines have yet to be identified as to number and function, but will allow for a certain amount of growth in the MODBUS).

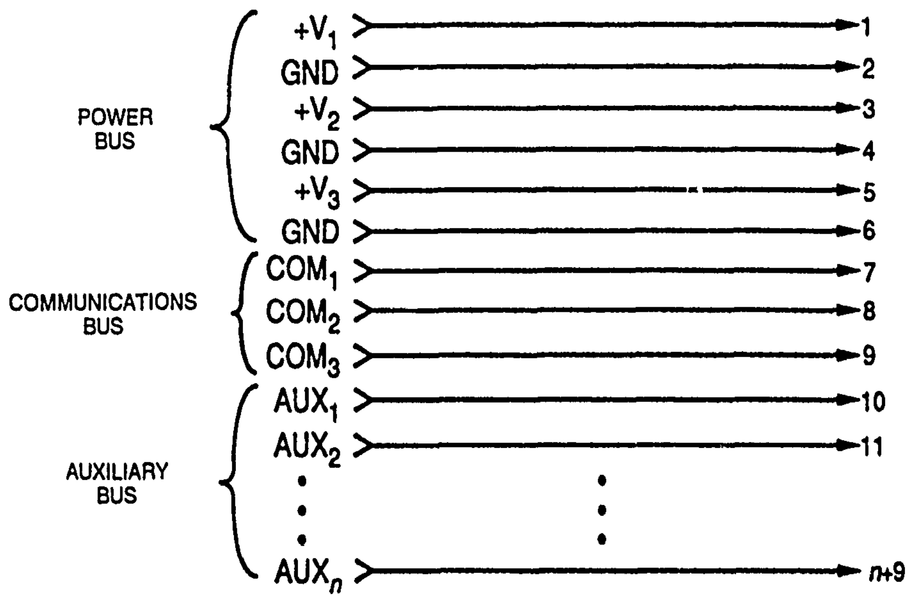


Figure 11. Generalized robot module "bus" (MODBUS).

The power lines carry common DC voltages (+24V, +12V, etc.) provided by the robot base and the power-conditioning unit. The communications lines support the intermodule local area network, while the auxiliary bus lines carry module-specific signals other than power and communications, such as video, audio, etc.

Modules are connected through the MODBUS, which is implemented on MOSER as a cable harness that is daisy-chained from one module to the next. This allows modules to be rearranged quickly and easily without any of the problems normally associated with relocating components of an embedded system (such as cable-handling nightmares).

4.1.2.3 Intelligent-Communications Node (ICN)

Each robot module contains an Intelligent-Communications Node (ICN) that provides a standard interface to the MODBOT local area network. The ICN manages medium-speed (>1 MBPS) data exchange between modules on the network by providing error detection/correction, collision detection, and general message passing services. Figure 12 is a block diagram of the ICN. The portion of the diagram contained in dashed lines may or may not be required. Single-chip microcontrollers are available that offer an integrated communications capability without the need for a separate network interface controller.

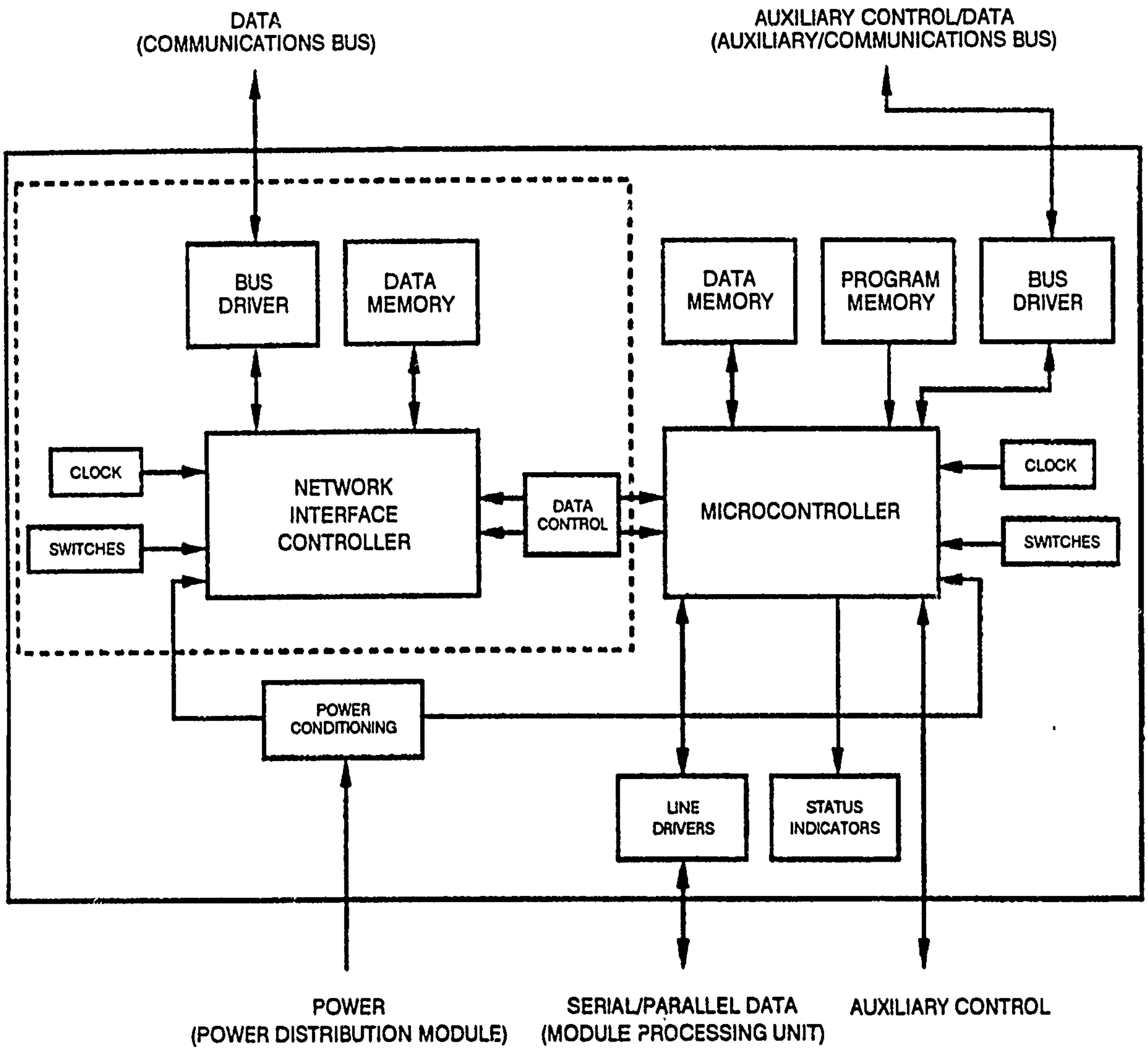


Figure 12. ICN block diagram.

The communications path is daisy-chained such that any module can talk to any other module—there is no central communications controller. Point-to-point as well as broadcast communications are supported. The ICN interfaces to the communications bus on one side and to the module processing unit on the other. The interface to the processing unit is configurable as either a standard RS-232 connection or as an 8-bit, high-speed, parallel connection.

The ICN on the MOSER is designed using CMOS components to minimize power consumption. An Intel 80C152 Universal Communications Controller is used as the ICN processor (Intel Corporation, 1989). The 80C152, an 8-bit microcontroller with an internal global-communications channel that implements the MODBOT local area network, can be configured to use one of four communications protocols: (1) Carrier-Sense Multiaccess with Collision Detection (CSMA/CD); (2) a subset of High-level

Data-Link-Control (HDLC); (3) Synchronous Data-Link Control (SDLC); and (4) a user-definable protocol. Baud rates of up to 1.8 MBPS are possible. A local-communications channel is also available for serial (RS-232) communications to the module's primary processor.

4.1.2.4 Power-Distribution Node (PDN)

Each robot module also contains a Power-Distribution Node (PDN) that provides local power conditioning, regulation, and short-circuit protection of power inputs from the MODBUS. Standard voltages such as +5V/+8V DC, +12V DC, and +24V DC are supplied by the PDN to the module. Distribution of power to individual components can be locally controlled via TTL-level inputs to "switches" on the PDN (figure 13). This allows specific subsystems to be turned off while not in use to conserve power. High-current thermal fuses are used in place of circuit breakers so that power that has

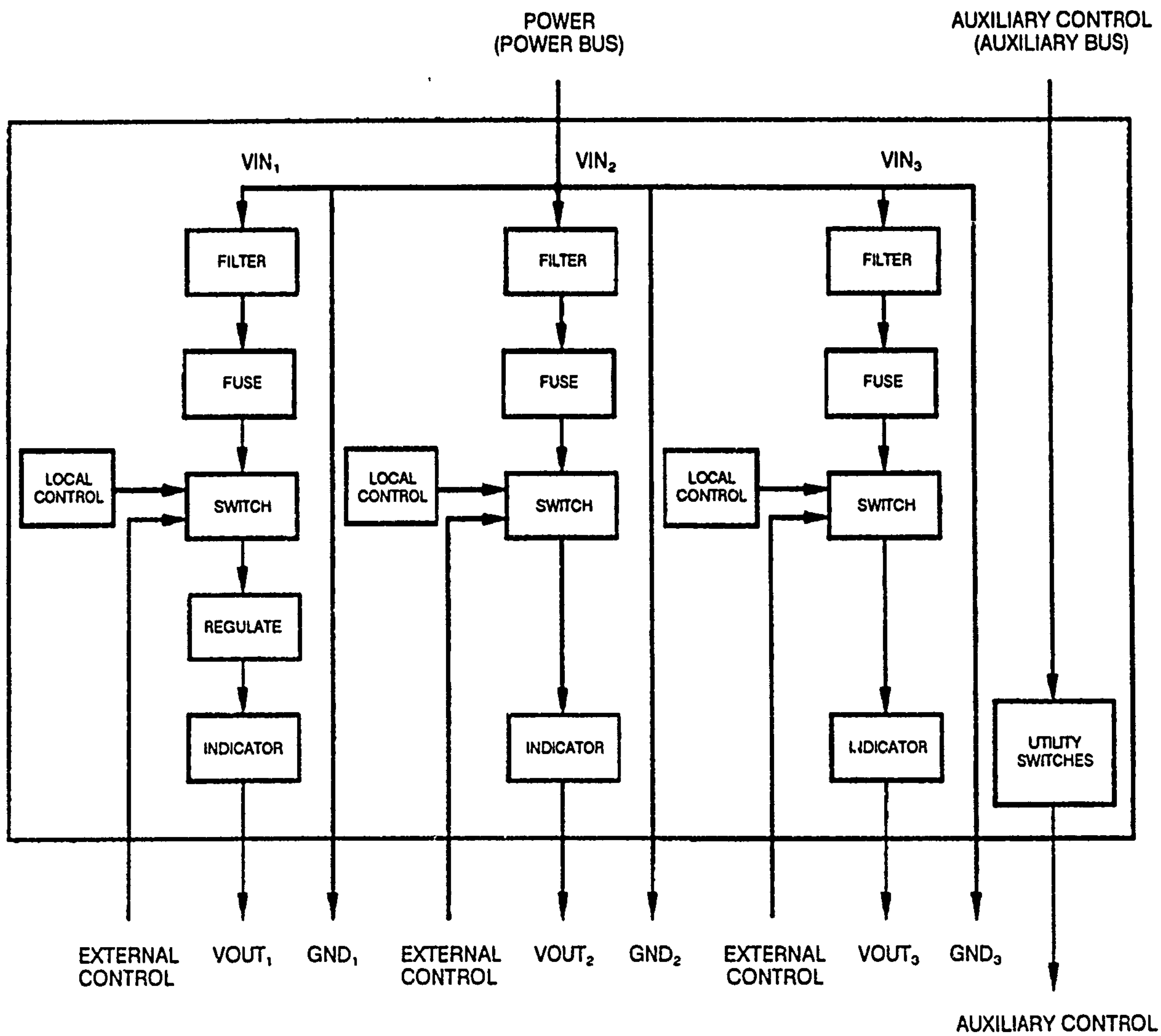


Figure 13. PDN block diagram.

been temporarily removed from a circuit will be automatically reapplied after a certain amount of time has passed. Thus, human intervention is not required to replace a blown fuse or reset a mechanical breaker, which is not always possible with a remote system.

Component selection (i.e., fuse size, regulator type, etc.) determines the current-carrying capabilities of each PDN and can be modified so that a particular module will be supplied with an appropriate amount of power. The MRA specifies limits on power consumption for each module so that a power budget for the entire system is achieved.

4.1.2.5 Platform-Power-Conditioning Unit (PPCU)

The Platform-Power-Conditioning Unit (PPCU) is responsible for converting the power supplied by the robot base to the standard voltages available on the MODBUS. The PPCU resides in the robot base and must be interfaced to the existing platform power system. It first protects and conditions the power supplied by the base with circuit breakers and spike suppressors, and then converts the power to standard voltages delivered over the MODBUS to the PDN. The supplied voltages are isolated from each other to avoid grounding problems. The only requirement that the PPCU places upon the platform is that it be capable of providing +24V DC at sufficient current. A block diagram for the PPCU is given in figure 14.

The PPCU also provides a standard interface between the robot platform and the power-distribution portion of the MODBUS. The PPCU makes the MODBUS base-independent, and decouples the power system of the base from the rest of the MODBOT.

4.1.2.6 Sensor, Actuator, and Processing Modules

A module is conceptually an independent unit with associated sensors, actuators, and/or processors. Each module must contain an ICN, a PDN, a central processing unit, also known as the Module Processing Unit (MPU) and, optionally, may contain sensors, actuators, additional processors or whatever else is required to support the function of the module (figure 15). The ICN is the module's interface to the robot local area network, while the PDN is the interface to the power system of the robot. The MPU implements the software systems of the MRA along with application-specific software provided by the developer or intelligent user. The only restrictions the MRA places upon the processing unit is that it be capable of implementing the MRA software subsystems, and capable of communicating with the ICN. Any processor meeting these requirements can be used, from simple single chip microcontrollers to sophisticated 68000-based microcomputers.

The ICN is connected to the MPU through either a standard RS-232 or an 8-bit parallel interface. Command and control data are transferred between the MODBUS local area network and the MPU via the ICN. The ICN receives power directly from the

PDN, and the ICN can control module power distribution through connections to local switches on the PDN.

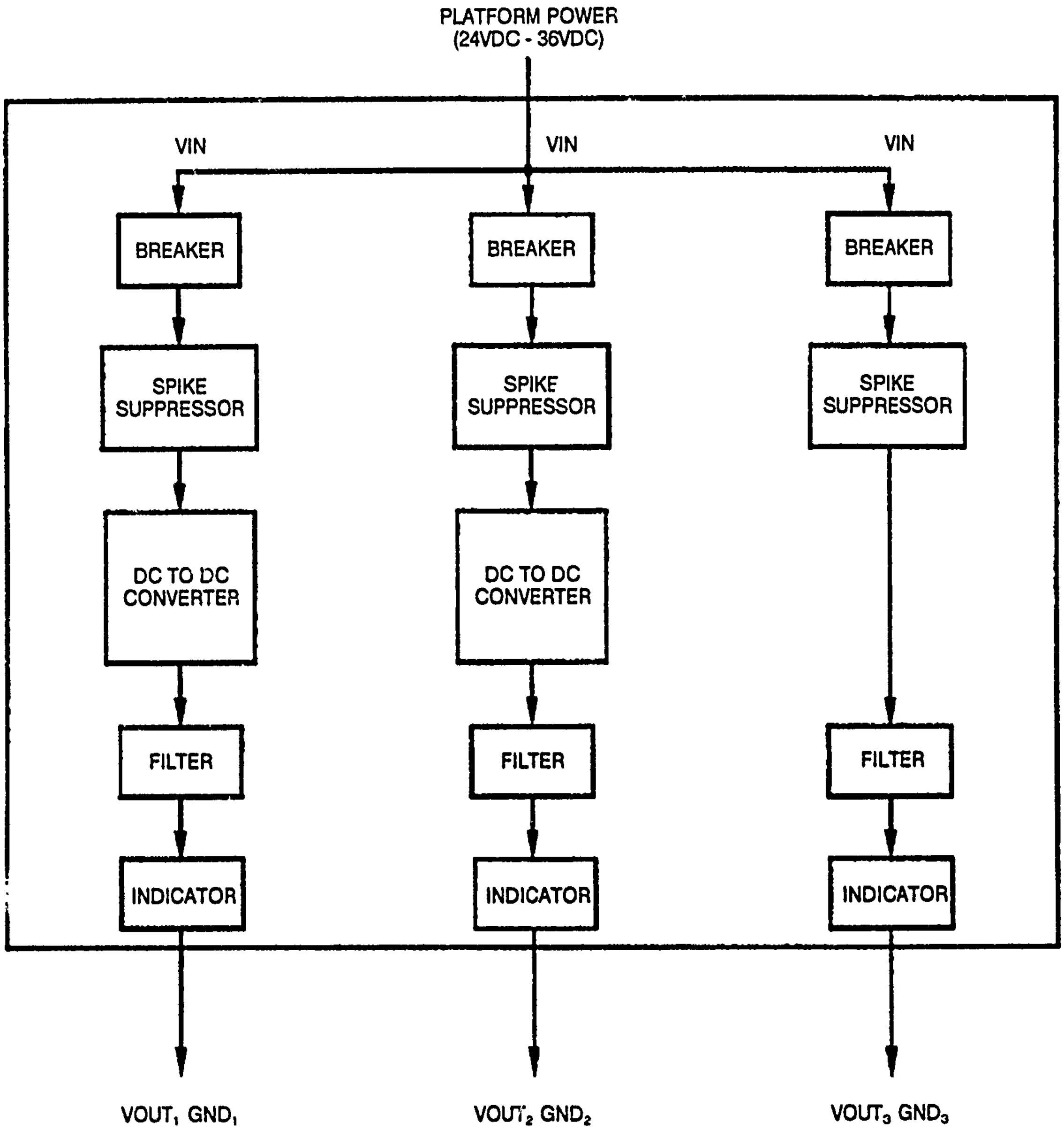


Figure 14. PPCU block diagram.

The MPU also receives power directly from the PDN, and can control module power distribution through connections to local switches on the PDN. The PDN is connected directly to the power portion of the MODBUS.

Modules interface to the real world through sensors and actuators connected to the module processor. A human interface—usually in the form of a serial connection—should be included on each module for diagnostic and test purposes. The environment interface shown in figure 15 refers to a possible connection to a secondary communications system such as an I/R link or high-speed local area network stationed throughout the environment.

Construction and assembly of modules should be standardized so as to maximize component interchangeability and ease of connectivity. The MRA does not specify what modules are to be made of, it only specifies certain components that each module must include (the ICN and PDN for example). The physical implementation of a module will vary between applications; the module in figure 15 is a conceptual entity that can be implemented in a number of ways depending upon the user's needs.

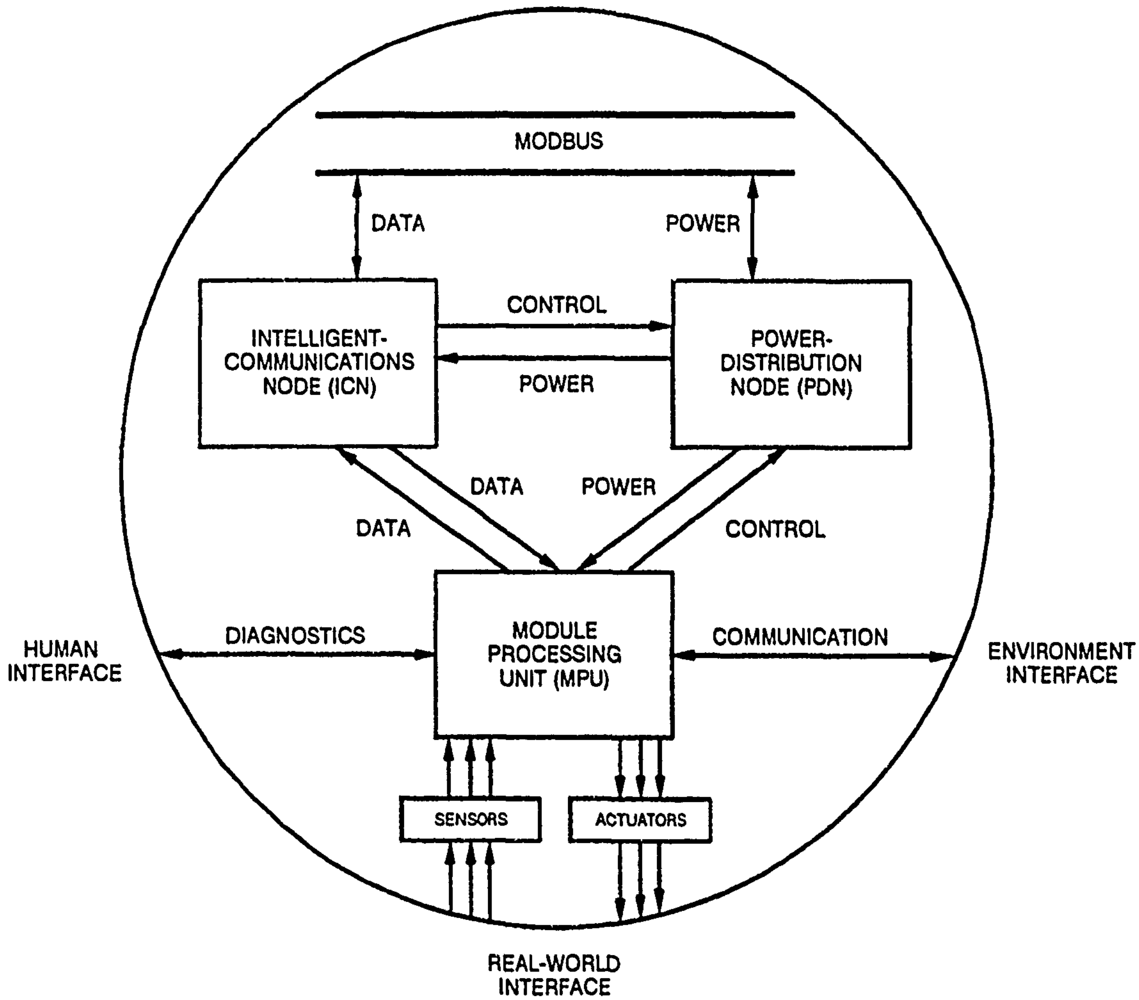


Figure 15. Generic robot module. The communications link to the external environment (environment interface) and the sensors and actuators (real-world interface) are optional.

Modules should have a single function or purpose. For example, a video transmitter should not be combined with an ultrasonic range sensor on a single module, rather the two should be implemented as separate units. This is equivalent to having distinct software modules that support separate functions of a program, and is consistent with modular-system-engineering practices.

Modules required for the mobile-security-robot application include actuator, sensor, and processing modules sufficient for intelligent navigation and for detection of "intruders." This includes a mobility module that interfaces to the robot platform, ultrasonic collision avoidance and navigation modules, an intrusion detection module, a near I/R proximity module used as a redundant-collision-avoidance sensor, an audio/visual module used during teleoperation, a CS module that houses the telemetry link to the CS processing unit, and a high-level processing module that is responsible for path planning, world modeling, and overall system coordination.

4.1.3 Telemetry Link

The telemetry link is the external connection between the CS and the RP, and is application dependent. Typical command and control links (as opposed to video or audio links) use radio frequency (RF), infrared (I/R), or some form of tether such as a fiber-optic cable. A combination of devices can be used in situations where the capabilities of a single device is not sufficient for the given environment (an intelligent communications controller could then switch between systems as fields of coverage or signal strength change). The link allows the CS processor sufficient bandwidth to effectively transfer information to the robot (and vice versa).

The telemetry system, part of the CS module, is an external connection of the CS module-processing unit to the CS remote-platform ICN. Telemetry units are located both with the CS processor and within the CS remote-platform module, and allow information to be transferred indirectly between the CS processor and the RP local area network just as if the CS processor were located onboard the RP (figure 16).

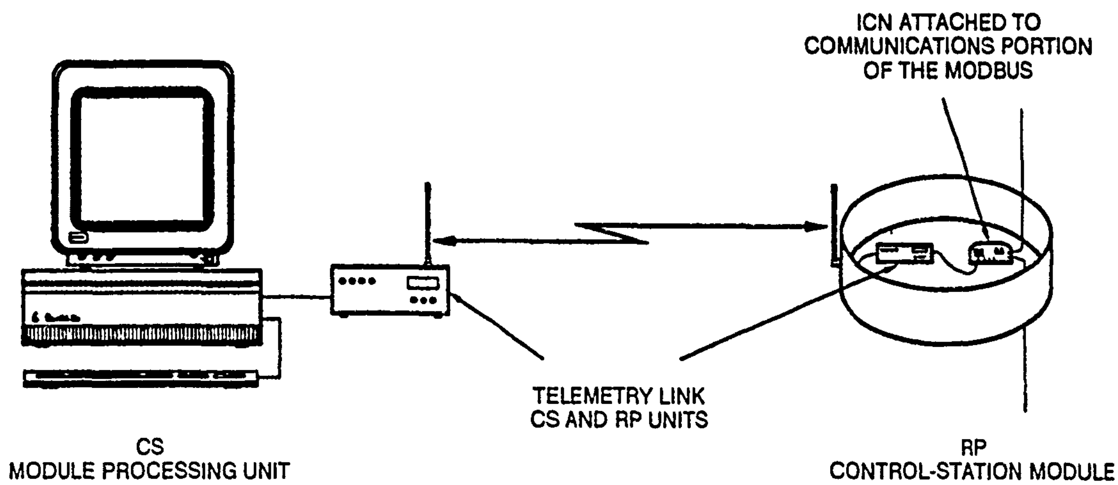


Figure 16. Telemetry link connecting the control-station (CS) processing unit and the remote-platform (RP) module.

MOSER uses a hard-wired tether that transmits command and control information between the CS processor and the remote CS module as an RS-232 signal operating at 9600 baud. In addition, the tether carries dual-channel audio and dual-channel video. Future plans call for replacement of the tether by an RF spread spectrum local area network controller for the command and control link and two RF video transmitters with dual-channel audio.

4.1.4 Communication Networks

Multipoint control—having one station control two or more RPs—is accomplished through the use of local-area wireless network (LAWN) modems. Data exchanged between the CS and the RP over the telemetry link contains addressing information that identifies the source and destination of transmissions. Each wireless modem has an associated address identifier. The modem controller examines incoming data and accepts them only if the address matches its own. By changing the destination address, a single CS can communicate with and control multiple MODBOTs. Figure 17 illustrates the concept.

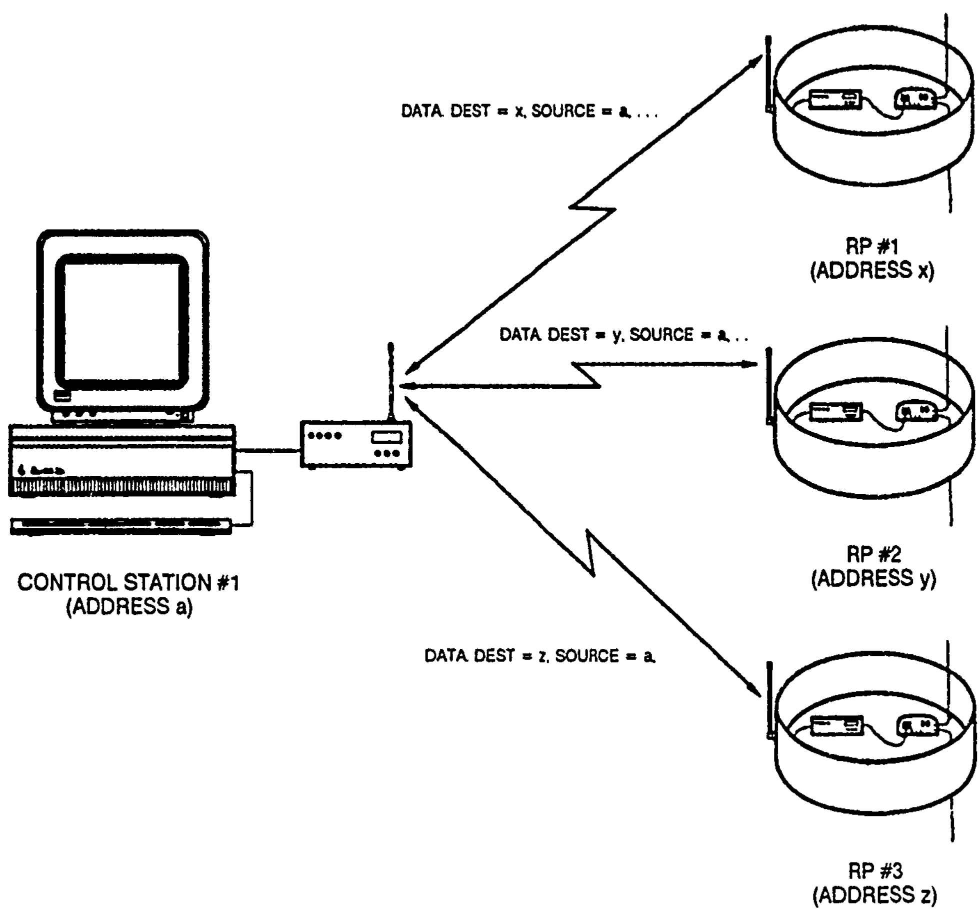


Figure 17. A single control-station (CS) controlling multiple remote platforms (RP) (note the addressing).

Multiple CSs can also be used to control a single RP. Each CS can be assigned a different subsystem to manage or monitor. Coordination between stations can be accomplished by communication over a separate local area network (figure 10).

4.2 SOFTWARE COMPONENTS

The MRA software architecture provides a framework around which distributed applications can be developed and implemented in a modular manner. The software systems are organized as layers in an *N,N-k* architecture where one software subsystem may have views (access) to multiple subsystems below it in the hierarchy (Lorin, 1988). Figure 18 is the modular-architecture-system image. The four primary software interfaces are given vertically on the right side of the diagram (e.g., Message-Manager Interface, LAN/MPU Interface, etc.) while the software subsystems that provide the interfaces are shown as layers within the hierarchy (e.g., METHOD LEVEL, COMMUNICATIONS LEVEL, etc.). MODBOT software application developers are able to use any subsystems as *desired*. However, applications must provide specific functionality that is established by the MRA and is otherwise obtained through the use of these subsystems. A standard software "library" is provided with functions available for inter-module communications, simple task control, and management of local and system module function execution. Software developers need only supply a minimal set of device-specific software "drivers" to implement the entire MRA software architecture (i.e., the software subsystems are portable between hardware implementations with only few modifications necessary).

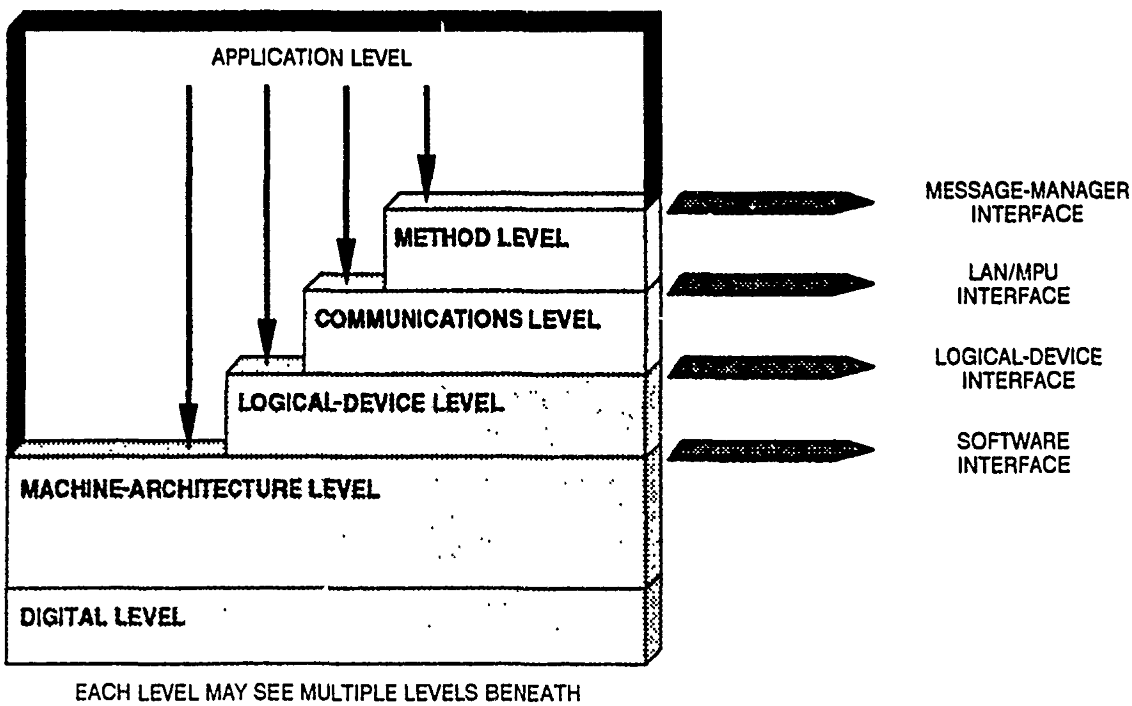


Figure 18. MRA system image (*N,N-k* architecture).

Each robot module can be thought of as an object that responds to a variety of commands represented by the object's methods and whose internal state is maintained by instance variables. Robot modules are of the class *Module* and possess certain capabilities common to all objects of that class. Modules also *inherit* the capabilities of their superclass (*Object*). Through classification and other properties of object-oriented programming, robot modules are given (by definition) common functionality.

The software subsystems of the MRA directly support object-oriented design and implementation of distributed, highly modular control systems for use on mobile (and nonmobile) robots. An object-oriented approach promotes both reusable software components and modularity at several levels. Additional capabilities can easily be added to a module simply by adding new methods to the object's class.

Below is a functional decomposition of the MRA software systems as implemented on the ICN and the MPU. (The ICN software is a subset of the MPU software with the exception of the Global Communications Subsystem, which is unique to the ICN.) Figure 19 is a block diagram of the MRA software subsystems as implemented on the MPU (note that the Global Communications Subsystem and the Logical-Device Interface are not shown).

4.2.1 ICN Software System

The ICN software system is responsible for managing medium-speed communication between the MODBOT local area network (LAN) and the MPU on board each module. The ICN provides a standard interface between the LAN and the MPU, and is the cornerstone of the MRA in that it provides for distributed MODBOT module communication and control.

The ICN software supports a 1.8-MBPS (maximum), CSMA/CD (peer-to-peer) communications network configured as a bus with deterministic access to a maximum of 255 modules (slots). There is no central or master-communications controller. Each node has equal access to the network and is responsible for managing its own resources. The Intel 80C152 global serial channel (GSC) is used as the node controller.

The ICN software system is composed of the following five functional units:

1) Global-Communications Device Handler.
2) Global-Communications Interface.
3) Local-Communications Device Handler.
4) Local-Communications Interface.
5) Intelligent-Communications Controller.

The Global-Communications Subsystem and the Intelligent-Communications Controller are unique to the ICN software system. The Local-Communications Subsystem is common to both the ICN and MPU processing systems.

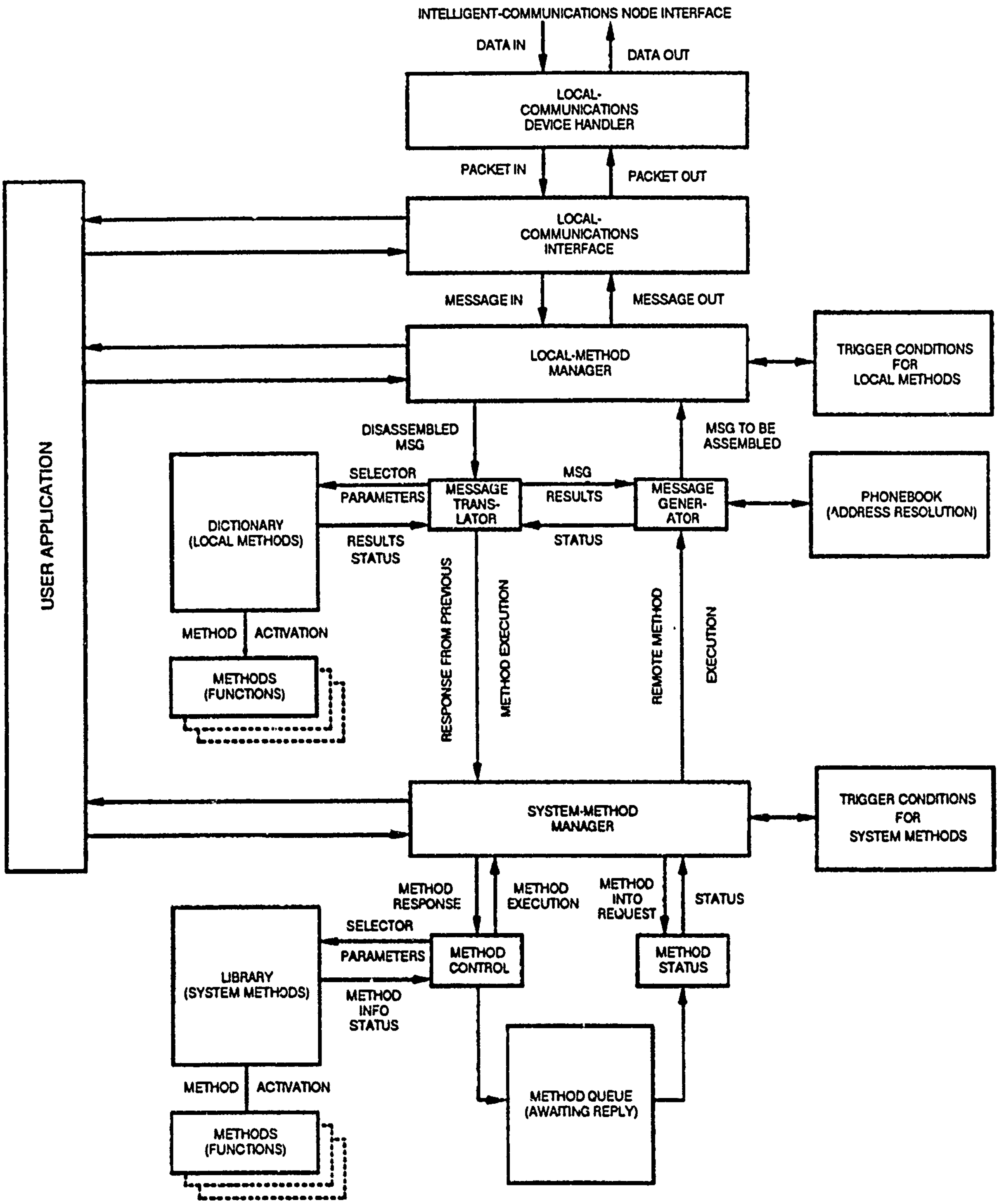


Figure 19. MRA-software block diagram. (Global-Communications Subsystem and Logical-Device Interface not shown.)

4.2.1.1 Global-Communications Subsystem

The Global-Communications Subsystem implements a standard set of functions on the target LAN hardware. These functions are used by higher-level software to access the LAN. The subsystem is broken down into the Global-Communications Device Handler (responsible for low-level control of the LAN hardware) and the Global-Communications Interface (which implements the standard network access functions available to other software subsystems).

The Global-Communications Device Handler interacts directly with the ICN hardware to provide external communications between the LAN and the ICN. The functions at this level are device-specific, and must be modified for the target (LAN) hardware. The functions provided by the GSC and the device handler implement the Physical Link Layer and the Data Link Layer of the ISO open-systems communication model (International Standards Organization, 1980).

The device handler logically decouples the standard functions of the Global-Communications Interface from the hardware implementation. To use a device other than the GSC as the MODBOT LAN, only the Global-Communications Device Handler functions must be rewritten.

The Global-Communications Interface is an abstraction of the lower-level functions and represents the user interface to the LAN. Functions at this level are completely hardware independent, and provide services for initializing and transmitting messages to/from the LAN.

4.2.1.2 Local-Communications Subsystem

The Local-Communications Subsystem implements a standard set of functions on the target-processor communications hardware. These functions are duplicated on both the ICN and MPU, and are used by higher-level software to communicate between the two systems. The subsystem is broken down into the Local-Communications Device Handler (responsible for low-level control of the serial or parallel communications hardware) and the Local-Communications Interface (which implements the standard communications port access functions available to other software subsystems).

The Local-Communications Device Handler is responsible for low-level data exchange between the MPU and the ICN. Functions at this level deal directly with the target hardware and are device-specific, so their implementation will change as the hardware changes. The device-handler functions are distributed between both the MPU and ICN, providing the communications link between the two systems.

The device handler logically decouples the higher-level functions provided by the Local-Communications Interface from the specific hardware implementation. Changing serial-communications controllers, for example, requires only that certain portions of the physical interface be modified to maintain consistency and compatibility at higher levels.

The Local-Communications Interface is an abstraction of the lower-level functions and represents the user interface to the interprocessor (local) communications port.

Functions at this level are completely hardware independent, and provide services for initializing and transmitting packets to/from the serial or parallel port.

4.2.1.3 Intelligent-Communications Controller

The "main" program of the ICN is the Intelligent-Communications Controller whose MPU counterpart is the User Application. The ICN controller is very simple: it initializes the Local- and Global-Communications Subsystems, and then coordinates transmission of information (data packets that represent module messages) between the ICN and the MPU. Messages are received from the network and passed along to the MPU. Messages are also received from the MPU and then sent on to the network for distribution as appropriate.

Currently, the communications controller is a simple message buffer between the MPU and the MODBOT LAN. Future plans include adding the message recognition and response capabilities of the Message Manager to the ICN for more sophisticated control. This would make the ICN software nearly identical to that of the MPU.

4.2.2 MPU Software System

The MPU software system is responsible for execution of both local and system methods (functions) as directed by the application module. The MPU provides the User Application with standard interfaces to the message-passing, function-execution, and logical-device-control facilities of the MRA.

The MPU software system is divided into two distinct components: the MRA MPU standard software services, and the MPU application program, which is the main routine supplied by the developer. A default controller is supplied by the MRA (for simple applications) that replaces the main program.

The MPU software system is composed of the following six functional units:

1) Local-Communications Device Handler.
2) Local-Communications Interface.
3) Local-Method Manager.
4) System-Method Manager.
5) Logical-Device Interface.
6) Application Controller.

The Message Manager, Logical-Device Subsystem, and the Application Controller are unique to the MPU. The Local-Communications Subsystem is common to both the MPU and ICN, and was described previously in section 4.2.1.2.

4.2.2.1 Message Manager

The Message Manager is responsible for translating and executing incoming messages, and for generating messages in response to external and internal requests. The

Message Manager is also responsible for external message address resolution, which relies on the ability to automatically determine the status of a module on the MODBOT network.

Functions that describe the behavior of a module are called *local methods* and are referred to as "internal." The functions available to all modules are called *system methods* and are referred to as "external." A dictionary contains compiled versions of the local methods that are executed upon receipt of the appropriate message. A *library* holds references to all of the system methods that an MPU needs for its application.

The Local-Method Manager coordinates receipt of incoming messages and their interpretation. Messages that request action or information are activated as local methods contained in the dictionary, while messages that are responses from previous external requests are passed on to the System-Method Manager.

The System-Method Manager coordinates activation of and response to system methods. A *method queue* is maintained by the System-Method Manager for external commands or data requests that require a response. Upon receipt of the required information, the method queue is searched according to message address and sequence number, and the external reference is resolved with the response being passed back to the calling function. The queue allows the application program to activate several methods sequentially and then coordinates receipt of responses, allowing the main program to continue execution until it is ready to process the incoming data. Services are available to the application program for examining the status of queued-method activations.

Initialization of the Message Manager includes resolving system-method address references. A *phone book* is maintained that contains the names of all externally referenced modules. Upon startup, each module whose name appears in the phone book is searched for on the MODBOT network. If found, the module's address is entered and subsequent references to that module can be resolved. If the module can't be located, then system methods referencing that module will fail.

A *trigger-condition table*, for both local and system methods, allows for execution of functions according to qualifiers placed on local (instance) variables. Functions can also be activated by an expiring timer with a given period (typically specified in milliseconds). Conditions for system methods are preprogrammed by the applications developer, while local-method conditions are set by external commands.

4.2.2.2 Logical-Device Subsystem

The logical-device subsystem consists of a logical-device interface and an associated *blackboard* data structure. The blackboard is used as a global module data storage and retrieval mechanism, and provides a convenient and consistent means of maintaining local variables (Aviles, Laird, & Myers, 1988).

The blackboard is based upon *logical devices* that have an abstracted real-world implementation. Logical sensors and actuators, for example, are used to represent devices whose state is maintained in the blackboard. Functions attached to the logical devices update their physical counterpart as information is requested from or entered into the blackboard. Devices that have no hard implementation are called *virtual*, and can be used to simulate an actual entity.

The logical-device interface provides software services for adding items to the blackboard, and for updating and retrieving the various data fields of those items. Activation of the functions attached to the items is automatic depending upon the device interface function used.

4.2.2.3 Application Controller

For applications that require no special processing (e.g., simple sensor or actuator modules), a default application controller is provided by the MRA MPU. The default controller takes the place of the User Application main program, and is responsible for initializing and coordinating the other subsystems of the MPU.

For specialized modules such as high-level path planning or distributed task controllers, the user must supply the main application program (i.e., the User Application). In this case, the application program is responsible for initializing the MPU subsystems and for managing the MPU software resources as required (see section 2.2.3 for a description of other applications).

4.2.3 Intermodule Message Format

The MODBOT network is based upon the ISO open-systems communication model, and implements the physical, the data-link, the presentation, and the application layers. The message format used at the presentation and application levels is based upon the Robotic-Vehicle Message Format (RVMF) developed by TACOM (Brendle, 1990). The primary differences between the RVMF and that used under the MRA is in the placement and bit requirements for the unit destination and source address fields as well as the message length and sequence number. These fields were modified to optimize message acknowledgment and function execution (only three bytes are required to ACK a message and only five bytes needed to execute a module function). The RVMF block address and unit ID correspond to the MODBOT address and module unit ID respectively.

The modified RVMF message format (RVMF*) is (nearly) maintained from layer to layer, that is, very few overhead bytes are added as the message is passed between the data link and application layers. This greatly increases data throughput and simplifies the MRA communications software interfaces. Figure 20 is a preliminary definition of the modified RVMF communications protocol. The figure does not break the protocol down into the multiple OSI layers. A message checksum or cyclic-redundancy check (CRC, not shown) would be added by the data-link layer before transmission.

DEST MODBOT ID	DEST UNIT ID	SOURCE MODBOT ID	SOURCE UNIT ID	SEQUENCE NUMBER	TRANSACTION DISPOSITION	TRANSACTION CATEGORY	FUNCTION ID	PARAMETER LENGTH	PARAMETER
DESTINATION ADDRESS		SOURCE ADDRESS		MESSAGE COORDINATION		FUNCTION SELECTION		FUNCTION PARAMETERS/RESULTS	

Figure 20. MODBOT communications protocol (multiple layers).

4.2.4 High-Level Module Definition

The MRA provides facilities for describing modules at a high level of abstraction. The module description is then translated into compilable code that can be included in the application. A syntax similar to the C programming language is used to define a module's methods as well as internal instance variables. Classes as well as objects can be defined. Figure 21 is an example of an object definition for an I/R proximity (I/RP) module.

```
with OBJECT;
with MODULE;
module IRP
{
        var OBJECT:
                char        Class[]                  = "MODULE";
                char        Superclass[]             = "OBJECT";

        var MODULE:
                byte        Address                  = 0;
                char        Name[]                   = "IRP";
                bit         Subsystem_Power          = OFF;
                int         Operational_Status       = IDLE;

        var IRP:
                int         IRP_Max_Update_Rate      = 10;      /* Hz        */
                int         IRP_Number_Sensors       = 11;
                int         IRP_Sensor_Range         = 30;      /* Inches    */

        method QUERY_OPERATIONAL_LIMITS:
                int         IRP_Max_Update_Rate();
                int         IRP_Number_Sensors();
                int         IRP_Sensor_Range();

        method STATUS_REQUEST, PERIODIC_STATUS_REQUEST:
                bit         IRP_Sensor_Report(int S : in);
                bit*        IRP_n_Sensor_Report(int S1, int S2 : in);
}
```

Figure 21. Example of an MRA definition for an I/R proximity module.

5.0 SYSTEM OPERATION

5.1 GENERAL PHILOSOPHY

Modular-robot operation depends upon the application and is normally the responsibility of the system developer. However, if the standard module application controllers are used, then operation of the robot is based upon the combined behavior of each module as implemented by the individual module functions. That is, the default controllers simply respond to incoming commands by executing the functions provided by the application programmer (section 4.2). If the standard controllers are not used, then operation is determined by the control methodology implemented by the developer. In this case, the developer is free to operate the robot however desired, and simply uses the hardware and software components of the MRA to implement the design.

5.2 AUTOMATIC CAPABILITIES

Independent of the approach above, all modular robots will have certain inherent capabilities that will automatically execute at system initialization. This includes built-in tests (BITs) for diagnostic purposes, and module identification and address-resolution functions for self-configuration. These capabilities are implemented at the lower levels of the hardware and software architecture and cannot normally be overridden or defeated.

5.2.1 Self-Diagnostics

Upon power up, the standard software systems (for example, the global- and local-communications systems) perform a variety of diagnostic tests to ensure the integrity of the hardware and software components. Errors are reported to the higher level subsystems for action. In case of a severe error, degraded performance is preferable over total loss of capability and will be attempted if possible. Certain failures will prevent a module from operating altogether, such as a low-level-communications-driver failure. Diagnostic indicators will be present on all standard hardware components such as the ICN, PDN, and PPCU.

5.2.2 Self-Configuration

Each robot module has an associated network address that is set by hardware and is read by initialization software. References to modules must be correlated with their addresses as in resolution of external function references in the compilation of computer programs. Address resolution takes place automatically upon power up by high-level software subsystems of the MRA. Unresolved references occur when a module cannot be located on the network, in which case an error is flagged. The process is dynamic and allows the robot to configure itself each time power is applied (or the system is reset).

5.3 SYSTEM COMMUNICATION

There are four levels of communication associated with modular robot control: (1) communication between the ICN and the robot LAN (on the MODBUS), (2) communication between the ICN and the MPU, (3) communication between the CS and the RP, and (4) communication between multiple CSs. These levels are shown in figure 22. (This topology does not apply to robots that do not have an associated CS.) When more than one CS or modular robot is employed at the same time, configurations can be conceptualized that take advantage of the multiple communication pathways. (The communications protocol addresses MODBOTs as well as individual modules, that is, both the MODBOT and the module are identified in the address.)

MODBOT Teams (LAN communication)

A MODBOT team consists of one or more CSs controlling two or more modular robots. The CSs are connected using a local area network (LAN) located throughout the workspace (figures 22 and 23). The application programmer is responsible for coordinating control between the multiple stations (not a trivial problem). MODBOT teams address problems such as physical security of very large spaces such as a warehouse.

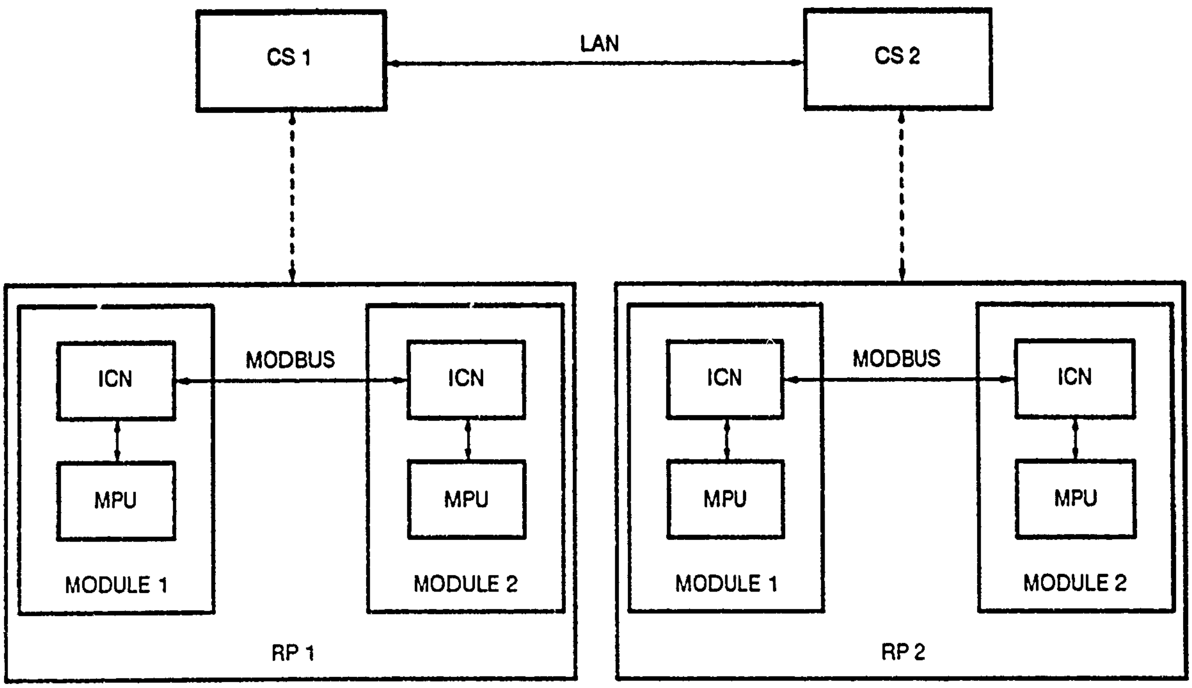


Figure 22. Communication paths between modular robot components.

MODBOT Divisions (WAN communication)

A MODBOT division consists of multiple MODBOT teams (figure 23). A wide area network (WAN) connects teams located at remote sites. Coordination at this level is very complex and may require separate CS dedicated for this purpose. MODBOT divisions can be used for applications such as inventory monitoring and control at several (possibly distant) sites.

5.4 OPERATION AS A SECURITY ROBOT

MOSER, the mobile-security modular robot, is responsible for autonomous navigation of an enclosed area such as an office building, and for the detection of intruders within that space. MOSER is modally operated, that is, one of several control modes is entered by the operator at the CS, and the robot behaves according to rules governing the selected mode.

Four logical levels of control are implemented on MOSER: teleoperated, reflexive, supervisory, and autonomous. Teleoperated control allows the user to directly navigate the MODBOT throughout its surroundings while monitoring sensory feedback on the CS displays. Teleoperation gives the operator full control of all MODBOT actions, including running the MODBOT into a wall as an extreme example. Reflexive control is a variation of teleoperation in which the MODBOT is now responsible for maintaining primitive reflexes that are conditioned by the operator. This prevents the MODBOT from running into walls. Supervisory control is a form of semiautonomous behavior. At this level, the operator is able to issue simple commands to the MODBOT and is able to instruct the MODBOT to traverse a path. Under supervisory control, the operator can intercede at any point and override MODBOT automatic functions. Under autonomous control, the operator need only specify goals to be reached or simple tasks to be executed. Autonomous control is attained with MOSER by its ability to operate as a self-sustaining mobile security robot that operates with a predetermined set of goals and responsibilities (e.g., identify and alert the operator to the presence of intruders).

A typical scenario would involve an operator manually piloting the robot to a starting point such as a door into a main hall, and then instructing the robot to patrol a path around the perimeter of the building and to sound an alarm if an intruder is detected within the area covered by specific sensors.

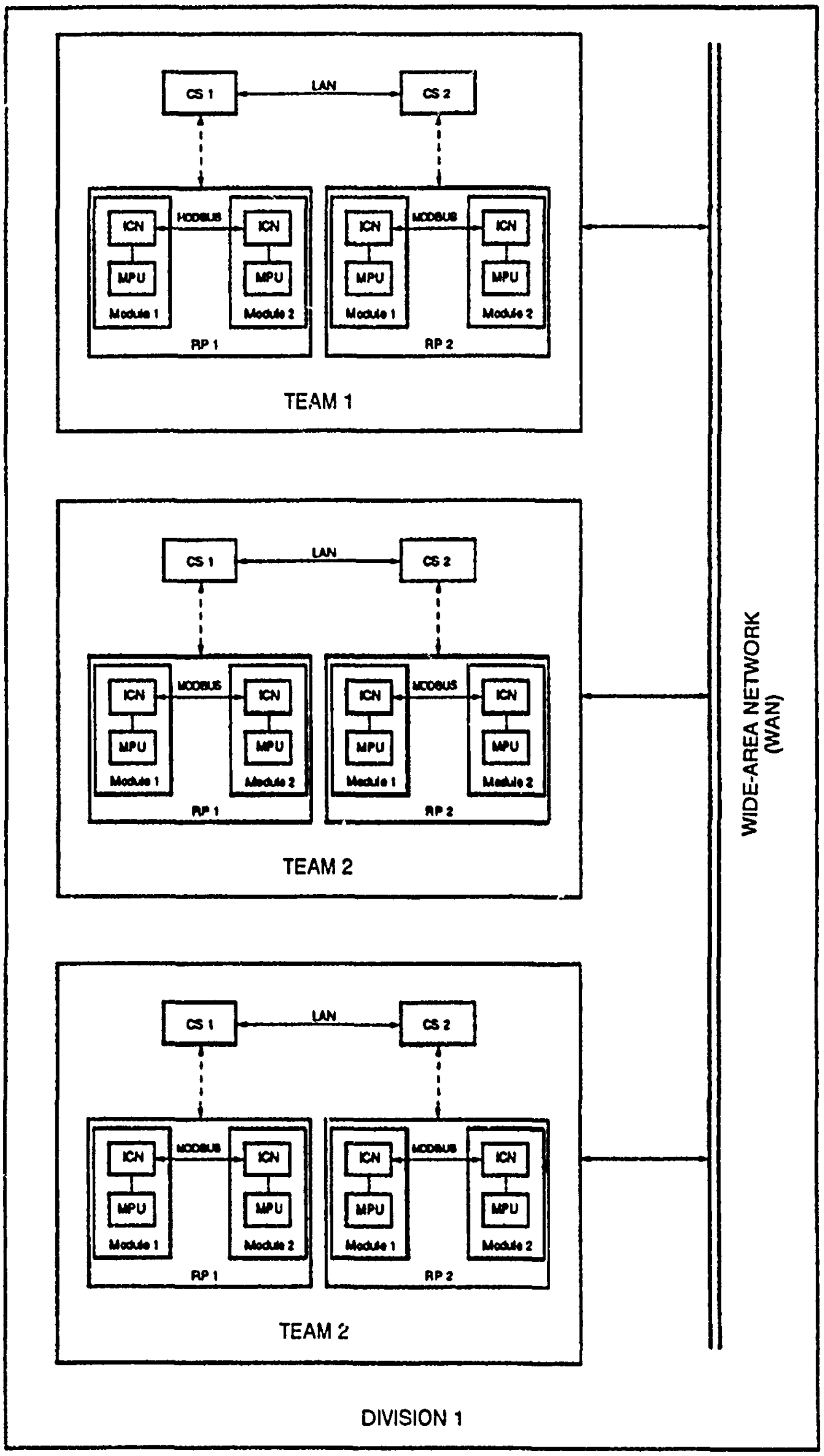


Figure 23. MODBOT teams (one or more MODBOTs) and divisions (one or more teams).

6.0 SYSTEM DEVELOPMENT

6.1 DOCUMENTATION

Development of the MRA follows standard software engineering practices as best as possible given a finite amount of time, money, and patience. The general plan is to research, design, review, and implement a modular architecture that meets an ever-changing array of requirements. Because the MRA is intended to be a standardized architecture—a common interface and control system that is shared among several agencies—emphasis is placed upon using "standard" (widely available or easily attainable) equipment, tools, and procedures so as to facilitate implementation of the "standard."

System documentation is a major portion of this effort and describes all aspects of the modular architecture and its development, from conception to implementation. This document is a high-level conceptual introduction to the MRA; it describes the preliminary architecture design and outlines an example application (i.e., a mobile security robot).

6.2 DEVELOPMENT EQUIPMENT AND STANDARDS

In developing the hardware and software systems of the MRA, considerable thought was given to the selection of development and target system equipment—making use of existing equipment was of major importance. For example, the decision to use a particular microcontroller as a (standard) module processing unit was initially based upon the availability of an existing cross compiler. In addition, the development process (especially implementation) can be simplified by trying to standardize on certain components and processes that are repeated throughout the system. Using a standardized computer programming language, for example, increases software portability and reusability.

The sections below list the software and hardware development tools and equipment that are being used in this project. The list is provided as background information and as a convenience for future MODBOT developers.

6.2.1 Software Development (Computers, Languages, etc.)

Below is a list of the major software tools and equipment used in developing the MRA. Only items that were used as "standard" equipment are listed.

Development systems

IBM PC-AT:

Software development.
Project documentation.
MS-DOS 3.2 and greater.

Macintosh IIx:

Software development.
Project documentation.
MultiFinder 6.03.
GRAVIS MouseStick GMPU joystick.
Farallon MacRecorder audio digitizer.
Articulate Systems Voice Navigator voice recognition system.

Programming language

Microsoft C compiler V5.1 (C programming language):
Path Planning module software development (IBM).
Franklin C-51 compiler (C programming language):
MPU software development (IBM).
ICN software development (IBM).
Application module software development (IBM).
Symantic Think C compiler (C programming language):
CS software development (Macintosh).

Software development tools

Documentation:
Wordstar 4.0 (IBM).
Microsoft Word 3.02 (Macintosh).
Cricket Draw 1.1.1 (Macintosh).
MacDraw II (Macintosh).

Other

Laplink(Mac) 2.0 (file transfer and data conversion).
Apple File Exchange .

6.2.2 Hardware Development (Computers, Platforms, etc.)

Below is a list of the major hardware components used in developing the MRA (and MOSER). Only components that were used as "standard" equipment are listed.

Target computer systems

CS:

Macintosh IIx.

MPU:

80C31 microcontroller.
Winsystems STD bus SBC8-8 (V20 processor).

ICN:

80C152 microcontroller.

High-level processing module:

Winsystems STD bus STD-AT (80286 processor).

Platform

MOSER:

TRC Labmate (modified extensively inhouse).

Module development

Robot module "ring" material:

1/8"–1/4" thick, 18" round plexiglass.
1/2" thick, 18" diameter PVC pipe.

6.3 INDEPENDENT DEVELOPMENT OF MODULES

An important aspect of the MRA is the ability to develop robot modules independent of normal system development and integration. The intention is that cooperating agencies independently develop MODBOT capabilities that can be easily integrated into an existing system or that can be coupled to form pieces of a new system. Modules are developed according to MRA standard interface requirements with the developer supplying the necessary hardware and software device-specific drivers. The concept is similar to third-party development of expansion or peripheral boards for the personal computer with the potential for product diversification as seen in this market applicable to the development of MODBOT modules.

6.3.1 Types of Modules That Should Be Developed

There are three major categories of MODBOT modules to be developed: (1) actuator modules, (2) sensor modules, and (3) processor modules. Development of specific modules is, of course, application dependent. Below is a brief listing of potential modules appropriate for indoor security applications.

Actuators

Actuator modules provide interaction with the physical environment:

1) Pan/tilt module for directional sensors such as cameras.
2) Manipulator module for grasping and activating controls.
3) Deterrence module for halting intruders.

Sensors

Sensor modules provide information for world modeling:

1) Environmental module for temperature, humidity, etc.
2) Intrusion-detection module having several different sensors.
3) Ultrasonic-ranging module for mapping the environment.
4) I/R-collision-avoidance modules for navigation.
5) Ultrasonic-collision-avoidance module also for navigation.
6) Video-motion-detection or vision-sensor module.

Processors

Processing modules provide system coordination and planning:

1) Vision-processing modules for navigation/object recognition.
2) Neural-network processing module for data reduction/analysis.
3) IBM AT processing module for path planning.
4) CP-31 microcontroller for use as a platform-interface module.
5) Control-station module for use as a human interface.

6.3.2 Adherence to Standard Interface Specifications

In developing MODBOT modules, adherence to the interface requirements specifications as given in the IRS is mandatory. Successful system integration is wholly dependent upon strict conformance to the MRA standards especially when multiple activities are involved. When possible, fabrication and distribution of standard architecture components should be done by the coordinating agency to ensure compatibility.

7.0 DEFINITIONS, ACRONYMS, AND ABBREVIATIONS

AMR	Autonomous Mobile Robot
AMRF	Automated Manufacturing Research Facility
ARDEC	Armament Research, Development, and Engineering Center
BIT	Built-in Test
CMOS	Complementary Metal-Oxide Semiconductor
CS	Control Station
CRC	Cyclic-Redundancy Check
CSMA/CD	Carrier-Sense Multiaccess with Collision Detection
GATERS	Ground-Air Telerobotic Systems
GRPA	Generic Robotic Processing Architecture
GSC	Global Serial Channel (of the Intel 80C152)
HDLC	High-level Data-Link Control
HDS	Hardware-Design Specification
ICN	Intelligent-Communication Node
IED	Independent Exploratory Development
I/R	Infrared
IRS	Interface-Requirements Specification
ISO	International Standards Organization
LAN	Local Area Network
LAWN	Local-Area Wireless Network
MB	Megabytes
Mbps	Megabits-per-second
MODBOT	Modular Robot
MODBUS	Robot Module Bus
MOSER	Mobile Security Robot (first application of the MRA)
MPU	Module Processing Unit
MRA	Modular Robotic Architecture
NASREM	NASA/NBS Standard Reference Model
NBS	National Bureau of Standards (a.k.a. NIST)
NHC	Nested-Hierarchical Controller
NIST	National Institute of Standards and Technology
NOSC	Naval Ocean Systems Center
OSI	Open-Systems Interconnection
PDN	Power-Distribution Node
PPCU	Platform-Power-Conditioning Unit

RCS	Realtime Control System
RP	Remote Platform
RVMF	Robotic-Vehicle Message Format
SBIR	Small-Business Innovative Research
SDLC	Synchronous Data-Link Control
SDS	Software-Design Specification
SPEC	System Specification
UGV/TOV	Unmanned Ground Vehicle/Teleoperated Vehicle
VSI	Virtual Systems Interface
WAN	Wide Area Network

8.0 REFERENCES

Albus, J. S., H. G. McGain, and R. Lumina. 4 December 1984. NASA/NBS *Standard Reference Model for Telerobot Control System Architecture (NASREM).*

Aviles, W. A., R. T., Laird, and M. E. Myers. November 1988. "Towards a Modular Robotic Architecture," *Proceedings SPIE Mobile Robots III*, pp. 271–278, Cambridge, MA.

Barbera, A. J., M. L. Fitzgerald, and J. S. Albus. April 1982. "Concepts for a Real-Time Sensory-Interactive Control System Architecture," *Proceedings of the Fourteenth Southeastern Symposium on System Theory*, pp. 121–126.

Barbera, A. J., M. L. Fitzgerald, J. S. Albus and L. S. Haynes. June 1984. "RCS: The NBS Real-Time Control System," presented at the *Robots 8 Conference and Exposition*, Detroit, MI.

Brendle, B. E., Jr. July 1990. "Robotic Vehicle Communications Interoperability Protocols," *Proceedings AUVS-90*, pp. 267–279, Dayton, OH.

Brooks, R. A. April 1986. "A Robust Layered Control System for a Mobile Robot," *IEEE Journal of Robotics and Automation*, RA-2, pp. 14–23, San Francisco, CA.

Everett, H. R., March 1988. "Security and Sentry Robots," *International Encyclopedia of Robotics Applications and Automation*, John Wiley & Sons, Inc., NY.

Everett, H. R., G. A. Gilbreath, T. T. Tran, and J. M. Nieusma. June 1990. *Modeling the Environment of a Mobile Security Robot*, NOSC Technical Document 1835, Naval Ocean Systems Center, San Diego, CA.

Hughes, T. W., H. R., Everett, A. Y. Umeda, S. W. Martin, W. A. Aviles, A. H. Koyomatsu, M. R. Solorzano, R. T. Laird, S. P. McArthur, and E. H. Spain. November 1990. "Issues in Mobile Robotics: Unmanned Ground Vehicle Program Teleoperated Vehicle," *Proceedings SPIE Mobile Robots V*, Boston, MA.

Intel Corporation. January 1989. "83C152 Hardware Description and Data Sheets," *Intel 8-Bit Embedded Controller Handbook*, No. 270645-002, pp. 9-1–9-87.

International Standards Organization (ISO). April 1980. *Reference Model for Open Systems Interconnection Architecture*, ISO/TC97/SC16 N309.

Lorin, H. 1988. *Aspects of Distributed Computer Systems, Second Edition*, John Wiley & Sons, Inc., NY.

McGain, H. G. March 1985. "A Hierarchically Controlled, Sensory Interactive Robot in the Automated Manufacturing Research Facility," *IEEE International Conference on Robotics and Automation*, pp. 931–939, St. Louis, MO.

Meystel, A. March 1988. "Mobile Robots, Autonomous," *International Encyclopedia of Robotics Applications and Automation*, John Wiley & Sons, Inc., NY.

APPENDIX A

SOFTWARE SYSTEM IMPLEMENTATION

Appendix A. Software System Implementation

The following section is a listing of the standard software subsystems of the MRA. The information provided herein is sufficient to implement application modules that can be configured to form a modular robot.

Configuration information in the form of directory and file specifications, as well as program build information (i.e., makefiles), are included in the listings.

The source for a single application module (Collision Avoidance Infrared - CAIR) is also included. This is an example of an object-oriented approach to the development of function-specific robot modules.

```
LD-List Directories, Advanced Edition 4.50, (C) Copr 1987-88, Peter Norton

  C:\MRA
  C:\MRA\APP
  C:\MRA\APP\BIN
  C:\MRA\APP\BIN\80152
  C:\MRA\APP\BIN\8031
  C:\MRA\APP\BIN\MSDOS
  C:\MRA\APP\BIN\MSDOS\CS
  C:\MRA\APP\BIN\MSDOS\WATCH
  C:\MRA\APP\BIN\SBC8
  C:\MRA\APP\SRC
  C:\MRA\APP\SRC\AV
  C:\MRA\APP\SRC\AV\TEST
  C:\MRA\APP\SRC\CAIR
  C:\MRA\APP\SRC\CAIR\TEST
  C:\MRA\APP\SRC\CAUS
  C:\MRA\APP\SRC\CAUS\TEST
  C:\MRA\APP\SRC\CS
  C:\MRA\APP\SRC\HLP
  C:\MRA\APP\SRC\HLP\TEST
  C:\MRA\APP\SRC\MOB
  C:\MRA\APP\SRC\MOB\TEST
  C:\MRA\APP\SRC\NAV
  C:\MRA\APP\SRC\NAV\TEST
  C:\MRA\APP\SRC\WATCH
  C:\MRA\COM
  C:\MRA\COM\BIN
  C:\MRA\COM\BIN\80152
  C:\MRA\COM\BIN\8031
  C:\MRA\COM\BIN\MSDOS
  C:\MRA\COM\BIN\SBC8
  C:\MRA\COM\SRC
  C:\MRA\COM\SRC\DEV
  C:\MRA\COM\SRC\DEV\TEST
  C:\MRA\COM\SRC\HDR
  C:\MRA\COM\SRC\LCS
  C:\MRA\COM\SRC\LCS\TEST
  C:\MRA\COM\SRC\MMS
  C:\MRA\COM\SRC\MMS\TEST
  C:\MRA\ICN
  C:\MRA\ICN\BIN
  C:\MRA\ICN\BIN\80152
  C:\MRA\ICN\BIN\80152\MON
  C:\MRA\ICN\SRC
  C:\MRA\ICN\SRC\AC
  C:\MRA\ICN\SRC\AC\TEST
  C:\MRA\ICN\SRC\GCS
  C:\MRA\ICN\SRC\GCS\TEST
  C:\MRA\LIB
  C:\MRA\MPU
  C:\MRA\MPU\BIN
  C:\MRA\MPU\BIN\80152
  C:\MRA\MPU\BIN\8031
  C:\MRA\MPU\BIN\MSDOS
  C:\MRA\MPU\BIN\SBC8
  C:\MRA\MPU\SRC
  C:\MRA\MPU\SRC\AC
  C:\MRA\MPU\SRC\AC\TEST
  C:\MRA\MPU\SRC\LDS
  C:\MRA\MPU\SRC\LDS\TEST

59 directories
```

```
FI-File Info, Advanced Edition 4.50, (C) Copr 1987-88, Peter Norton

 Directory of C:\MRA

.                      <DIR>      7-06-90    8:43a
..                     <DIR>      7-06-90    8:43a
APP                    <DIR>      8-07-90    2:24p
COM                    <DIR>      7-06-90    8:43a
ICN                    <DIR>      7-06-90    8:43a
LIB                    <DIR>      5-30-91    8:05a
MPU                    <DIR>      7-06-90    8:44a
makefile              11,243      5-30-91    1:00p
mradir                 1,413      5-30-91   12:56p
mrafiles                   0      5-30-91    1:01p

 Directory of C:\MRA\APP

.                      <DIR>      8-07-90    2:24p
..                     <DIR>      8-07-90    2:24p
BIN                    <DIR>      8-07-90    2:24p
SRC                    <DIR>      8-07-90    2:24p

 Directory of C:\MRA\APP\BIN

.                      <DIR>      8-07-90    2:24p
..                     <DIR>      8-07-90    2:24p
80152                  <DIR>      3-25-91    3:34p
8031                   <DIR>      3-25-91    2:19p
MSDOS                  <DIR>      3-25-91    2:19p
SBC8                   <DIR>      3-25-91    2:19p

 Directory of C:\MRA\APP\BIN\80152

.                      <DIR>      3-25-91    3:34p
..                     <DIR>      3-25-91    3:34p

 Directory of C:\MRA\APP\BIN\8031

.                      <DIR>      3-25-91    2:19p
..                     <DIR>      3-25-91    2:19p
cair                  37,465      5-28-91   10:20a
caus                  39,263      5-28-91   10:21a
mob                   64,302      5-29-91   11:21a
cair      hex         52,624      5-28-91   10:21a
caus      hex         54,964      5-28-91   10:21a
mob       hex         87,366      5-29-91   11:21a
caird     obj          4,597      4-18-91    8:05a
cairl     obj          4,026      5-21-91    8:04a
causd     obj          6,347      5-08-91   10:11a
causl     obj          4,660      5-21-91    8:05a
fgm       obj            405      5-20-91    4:08p
irs       obj          6,934      5-24-91    4:25p
mobd      obj         17,755      5-27-91    1:26p
mobl      obj          8,264      5-27-91    1:26p
trc       obj         19,313      5-29-91   11:20a

 Directory of C:\MRA\APP\BIN\MSDOS

.                      <DIR>      3-25-91    2:19p
..                     <DIR>      3-25-91    2:19p
CS                     <DIR>      4-05-91    2:50p
WATCH                  <DIR>      4-05-91    8:22a
cairl     obj          3,461      5-21-91    8:04a
causl     obj          3,895      5-21-91    8:05a
mobl      obj          6,757      5-27-91    1:27p

 Directory of C:\MRA\APP\BIN\MSDOS\CS

.                      <DIR>      4-05-91    2:50p
..                     <DIR>      4-05-91    2:50p
cs        exe         28,061      5-29-91    7:43a
cs        map             31      5-08-91    7:40a
csd       obj          1,287      5-27-91    3:34p
```



```
main      obj          8,079      5-29-91    7:43a
mra       obj          1,020      4-16-91   10:49a

 Directory of C:\MRA\APP\BIN\MSDOS\WATCH

.                      <DIR>      4-05-91    8:22a
..                     <DIR>      4-05-91    8:22a
icnwatch  exe        122,537      5-21-91    4:08p
icnwatch  log          1,680      4-16-91   10:54a
main      obj         17,254      4-16-91   10:32a
mra       obj            809      4-16-91   10:32a

 Directory of C:\MRA\APP\BIN\SBC8

.                      <DIR>      3-25-91    2:19p
..                     <DIR>      3-25-91    2:19p

 Directory of C:\MRA\APP\SRC

.                      <DIR>      8-07-90    2:24p
..                     <DIR>      8-07-90    2:24p
AV                     <DIR>      8-07-90    2:26p
CAIR                   <DIR>      8-07-90    2:26p
CAUS                   <DIR>      8-07-90    2:26p
CS                     <DIR>      3-30-91    1:11p
HLP                    <DIR>      8-07-90    2:26p
MOB                    <DIR>      8-07-90    2:26p
NAV                    <DIR>      8-07-90    2:26p
WATCH                  <DIR>      4-05-91    8:23a

 Directory of C:\MRA\APP\SRC\AV

.                      <DIR>      8-07-90    2:26p
..                     <DIR>      8-07-90    2:26p
TEST                   <DIR>      3-25-91    3:35p

 Directory of C:\MRA\APP\SRC\AV\TEST

.                      <DIR>      3-25-91    3:35p
..                     <DIR>      3-25-91    3:35p

 Directory of C:\MRA\APP\SRC\CAIR

.                      <DIR>      8-07-90    2:26p
..                     <DIR>      8-07-90    2:26p
TEST                   <DIR>      3-06-91    3:41p
count                     16      3-27-91    1:03p
makefile               5,620      5-30-91   12:43p
name                      16      3-27-91   12:43p
period1                   28      4-08-91   11:54a
period15                  28      4-05-91   10:33a
print                     34      5-30-91   10:53a
range                     16      3-27-91   12:47p
rate                      16      3-27-91    1:03p
report1                   20      3-27-91    3:19p
reportn                   24      3-27-91    3:19p
resetp                    28      3-30-91    6:40p
resetw                    24      4-05-91   10:39a
wait1                     22      4-05-91   10:41a
wait5                     22      4-08-91   11:56a
caird     31          55,948      4-18-91    8:05a
cairl     31          31,673      5-21-91    8:04a
cairl     at          11,315      5-21-91    8:04a
caird     c           20,426      4-18-91    7:45a
cairl     c            5,198      4-05-91    7:54a
caird     h            4,340      4-05-91    7:54a
cairl     h            2,953      5-16-91    4:50p
cair      m51        116,909      5-28-91   10:20a
cair      res            233      4-05-91    9:07a

 Directory of C:\MRA\APP\SRC\CAIR\TEST

.                      <DIR>      3-06-91    3:41p
```

```
..                <DIR>     3-06-91   3:41p

 Directory of C:\MRA\APP\SRC\CAUS

.                 <DIR>     8-07-90   2:26p
..                <DIR>     8-07-90   2:26p
TEST              <DIR>     3-25-91   3:35p
count                16     4-18-91  10:23a
makefile          5,673     5-30-91  12:43p
name                 16     4-18-91  10:23a
period1              28     4-18-91  11:53a
period15             28     4-18-91  11:54a
period5              28     5-06-91  11:53a
print                34     5-30-91  10:53a
range                16     4-18-91  10:23a
rate                 16     4-18-91  10:26a
report1              20     4-18-91  10:39a
reportn              24     4-18-91  10:25a
resetp               28     4-18-91  11:56a
resetw               24     5-06-91   4:26p
setmin               24     5-08-91  10:30a
waitmin              22     5-06-91   4:28p
causd     31     72,652     5-08-91  10:11a
causl     31     36,079     5-21-91   8:05a
causl     at     12,820     5-21-91   8:05a
causd     c      26,784     5-08-91  10:07a
causl     c       5,605     4-18-91   4:22p
snr12     c       8,568     6-13-90  12:40p
sonar     c       9,609     4-10-91   7:46a
causd     h       4,525     4-18-91   3:58p
causl     h       3,205     5-16-91   4:50p
caus      m51   123,693     5-28-91  10:21a
caus      res       233     4-18-91   8:02a

 Directory of C:\MRA\APP\SRC\CAUS\TEST

.                 <DIR>     3-25-91   3:35p
..                <DIR>     3-25-91   3:35p

 Directory of C:\MRA\APP\SRC\CS

.                 <DIR>     3-30-91   1:11p
..                <DIR>     3-30-91   1:11p
makefile          6,539     5-30-91  12:45p
print                34     5-30-91  10:53a
csd       at      6,836     5-27-91   3:34p
main      at     24,094     5-29-91   7:43a
mra       at     11,442     4-16-91  10:49a
csd       c       3,789     5-27-91   3:34p
main      c      11,167     5-29-91   7:42a
mra       c       7,063     4-05-91   9:59a
csd       h       3,007     4-05-91   7:58a
mra       h       3,644     4-05-91   9:58a
cs        map    16,877     5-29-91   7:43a
cs        res       309     5-27-91   3:33p

 Directory of C:\MRA\APP\SRC\HLP

.                 <DIR>     8-07-90   2:26p
..                <DIR>     8-07-90   2:26p
TEST              <DIR>     3-25-91   3:35p

 Directory of C:\MRA\APP\SRC\HLP\TEST

.                 <DIR>     3-25-91   3:35p
..                <DIR>     3-25-91   3:35p

 Directory of C:\MRA\APP\SRC\MOB

.                 <DIR>     8-07-90   2:26p
..                <DIR>     8-07-90   2:26p
TEST              <DIR>     8-09-90   4:52p
fwdjog               24     5-27-91  12:26p
```



```
fwdslow              24     5-27-91  12:17p
fwdstop              24     5-27-91  12:35p
halt                 18     5-27-91  12:15p
lftslow              24     5-27-91  12:52p
makefile          7,022     5-30-91  12:47p
name                 16     5-20-91   5:00p
print                34     5-30-91  10:53a
rate                 16     5-20-91   5:00p
reset                16     5-27-91   1:02p
revslow              24     5-27-91  12:53p
rltslow              24     5-27-91  12:52p
telemode             20     5-27-91  12:08p
fgm       31      8,482     5-20-91   4:08p
irs       31    109,106     5-24-91   4:25p
mobd      31    194,959     5-27-91   1:26p
mobl      31     65,153     5-27-91   1:26p
trc       31    238,803     5-29-91  11:20a
mobl      at     17,966     5-27-91   1:27p
fgm       c       5,262     5-20-91   4:08p
irs       c      40,778     5-20-91   4:24p
mobd      c      57,028     5-27-91   1:24p
mobl      c       8,596     5-27-91  11:27a
trc       c      90,951     5-29-91  11:18a
fgm       h       1,981     5-20-91   3:51p
irs       h       2,787     5-20-91  11:06a
mobd      h       6,177     5-27-91   1:24p
mobl      h       5,590     5-27-91   1:24p
trc       h       4,962     5-24-91   4:23p
mob       m51   209,381     5-29-91  11:21a
mob       res       322     5-16-91   4:38p

 Directory of C:\MRA\APP\SRC\MOB\TEST

.                 <DIR>     8-09-90   4:52p
..                <DIR>     8-09-90   4:52p
trct             14,073     8-11-90   1:22p
c         bat        27     7-11-90   8:27a
l         bat        13     8-09-90   5:48p
m         bat        31     7-16-90   4:34p
o         bat        11     7-10-90   4:18p
trcold    c      71,926     4-05-91   9:19a
trct      c       1,457     4-05-91   8:18a
trcold    h       4,189     4-05-91   8:16a
trct      hex    13,519     8-11-90   1:22p
trct      lst    10,414     8-11-90   1:22p
trct      m51    55,294     8-11-90   1:22p
trct      obj     1,509     8-11-90   1:22p
trct      res       145     8-09-90   5:48p

 Directory of C:\MRA\APP\SRC\NAV

.                 <DIR>     8-07-90   2:26p
..                <DIR>     8-07-90   2:26p
TEST              <DIR>     3-25-91   3:35p

 Directory of C:\MRA\APP\SRC\NAV\TEST

.                 <DIR>     3-25-91   3:35p
..                <DIR>     3-25-91   3:35p

 Directory of C:\MRA\APP\SRC\WATCH

.                 <DIR>     4-05-91   8:23a
..                <DIR>     4-05-91   8:23a
makefile          5,270     5-30-91  12:48p
print                34     5-30-91  10:53a
main      at     69,316     4-16-91  10:32a
mra       at     11,310     4-16-91  10:32a
main      c      36,559     4-09-91   2:35a
mra       c       7,063     4-05-91   8:48a
mra       h       3,644     4-05-91   8:46a
icnwatch  map    35,491     5-21-91   4:07p
icnwatch  res       177     4-05-91   9:11a
```

```
 Directory of C:\MRA\COM

.                <DIR>      7-06-90    8:43a
..               <DIR>      7-06-90    8:43a
BIN              <DIR>      7-06-90    8:45a
SRC              <DIR>      7-06-90    8:45a

 Directory of C:\MRA\COM\BIN

.                <DIR>      7-06-90    8:45a
..               <DIR>      7-06-90    8:45a
80152            <DIR>      3-20-91   11:58a
8031             <DIR>      3-20-91   11:56a
MSDOS            <DIR>      3-20-91   11:56a
SBC8             <DIR>      3-20-91   12:18p

 Directory of C:\MRA\COM\BIN\80152

.                <DIR>      3-20-91   11:58a
..               <DIR>      3-20-91   11:58a
lcd      obj     15,630     5-21-91    3:59p
lci      obj      1,755     3-28-91    3:53p
lmm      obj      5,753     4-08-91   10:29a
lmmdct   obj      1,299     4-16-91   10:13a
mm       obj     28,905     5-08-91   11:03a
pb       obj      3,855     5-28-91    8:25a
rtc      obj      2,077     4-01-91    9:47a
sio      obj      4,829     5-29-91    8:03a
smm      obj     12,199     4-08-91    2:08p
smmlib   obj      1,986     4-16-91   10:13a

 Directory of C:\MRA\COM\BIN\8031

.                <DIR>      3-20-91   11:56a
..               <DIR>      3-20-91   11:56a
lcd      obj     14,989     5-21-91    4:00p
lci      obj      1,755     3-28-91    3:54p
lmm      obj      5,753     4-08-91   10:36a
lmmdct   obj      1,299     4-16-91   10:15a
mm       obj     28,905     5-08-91   11:05a
pb       obj      3,855     5-28-91    8:25a
rtc      obj      2,077     4-01-91    9:47a
sio      obj      6,821     5-29-91    8:04a
smm      obj     12,199     4-08-91    2:09p
smmlib   obj      1,986     4-16-91   10:16a

 Directory of C:\MRA\COM\BIN\MSDOS

.                <DIR>      3-20-91   11:56a
..               <DIR>      3-20-91   11:56a
lcd      obj     15,143     5-21-91    4:01p
lci      obj      1,740     3-28-91    3:56p
lmm      obj      4,119     4-08-91   10:39a
lmmdct   obj      1,641     4-16-91   10:17a
mm       obj     17,349     5-08-91   11:06a
pb       obj      3,387     5-28-91    8:25a
rtc      obj      1,919     4-01-91    9:48a
sio      obj      3,611     5-29-91    8:05a
smm      obj      9,243     4-08-91    2:10p
smmlib   obj      2,194     4-16-91   10:18a

 Directory of C:\MRA\COM\BIN\SBC8

.                <DIR>      3-20-91   12:18p
..               <DIR>      3-20-91   12:18p
sio      obj      3,811     5-29-91    8:05a

 Directory of C:\MRA\COM\SRC

.                <DIR>      7-06-90    8:45a
..               <DIR>      7-06-90    8:45a
DEV              <DIR>      8-08-90    9:19a
```



```
HDR              <DIR>      7-08-90    4:50p
LCS              <DIR>     10-10-90   10:04a
MMS              <DIR>      7-06-90   12:28p

 Directory of C:\MRA\COM\SRC\DEV

.                <DIR>      8-08-90    9:19a
..               <DIR>      8-08-90    9:19a
TEST             <DIR>      8-08-90    3:18p
lib                  30     5-29-91    8:06a
makefile          5,434     5-30-91   12:35p
print                34     5-30-91   10:53a
rtc      152     31,021     4-01-91    9:47a
sio      152     97,153     5-29-91    8:03a
rtc      31      31,017     4-01-91    9:47a
sio      31     121,656     5-29-91    8:04a
rtc      at      20,997     4-01-91    9:48a
sio      at      54,177     5-29-91    8:05a
rtc      c       13,326     4-01-91    9:47a
sio      c       37,704     5-29-91    8:03a
rtc      h        1,764     3-28-91    9:11a
sio      h        5,455     5-15-91    8:19a
sio      sbc     54,375     5-29-91    8:05a

 Directory of C:\MRA\COM\SRC\DEV\TEST

.                <DIR>      8-08-90    3:18p
..               <DIR>      8-08-90    3:18p
siortc           10,098     5-29-91    1:27p
siotst            3,561     5-15-91    1:37p
c        bat         34     3-28-91    9:52a
l        bat         87     3-28-91    9:54a
o        bat         11     7-10-90    4:18p
siortc   c        1,510     5-29-91    1:26p
siotst   c          408     5-15-91    1:37p
siortc   exe     10,635     3-28-91    9:45a
siotst   exe      3,273     3-22-91   12:08p
siortc   hex     14,691     5-29-91    1:27p
siotst   hex      3,770     5-15-91    1:37p
siortc   lst     25,476     5-29-91    1:26p
siotst   lst      5,554     5-15-91    1:37p
siortc   m51     31,395     5-29-91    1:27p
siotst   m51     14,948     5-15-91    1:37p
siortc   obj      3,013     5-29-91    1:26p
siotst   obj        785     5-15-91    1:37p

 Directory of C:\MRA\COM\SRC\HDR

.                <DIR>      7-08-90    4:50p
..               <DIR>      7-08-90    4:50p
debug    h        1,431     3-22-91    1:47p
sysdefs  h        1,606     3-28-91    3:42p

 Directory of C:\MRA\COM\SRC\LCS

.                <DIR>     10-10-90   10:04a
..               <DIR>     10-10-90   10:04a
TEST             <DIR>     10-10-90   10:04a
lib                  30     5-21-91    4:02p
makefile          5,412     5-30-91   12:35p
print                34     5-30-91   10:53a
lcd      152    166,560     5-21-91    3:59p
lci      152     25,914     3-28-91    3:53p
lcd      31     159,933     5-21-91    4:00p
lci      31      25,910     3-28-91    3:54p
lcd      at      39,263     5-21-91    4:01p
lci      at      17,709     3-28-91    3:56p
bmf      c       26,091     3-22-91    3:56p
lcc      c       61,544     5-21-91    3:57p
lcd      c       23,403     3-28-91   12:14p
lci      c       10,848     3-22-91    3:56p
bmf      h        3,063    12-05-90   10:48a
lcc      h        5,722     3-28-91    4:24p
```

```
lcd       h        3,745    2-01-91    3:33p
lcl       h        3,209    3-26-91    4:00p

 Directory of C:\MRA\COM\SRC\LCS\TEST

.                  <DIR>    10-10-90  10:04a
..                 <DIR>    10-10-90  10:04a
packet            12,946     8-28-90   8:44a
c         bat         38     8-28-90   2:24p
l         bat         20     8-21-90   2:37p
o         bat         11     7-10-90   4:18p
packet    c          490     1-31-91   9:16a
packet    hex     15,507     8-28-90   8:44a
packet    lst      6,482     8-28-90   8:43a
packet    m51     45,605     8-28-90   8:44a
packet    obj      1,249     8-28-90   8:43a
packet    res        186     8-23-90   4:46p

 Directory of C:\MRA\COM\SRC\MMS

.                  <DIR>     7-06-90  12:28p
..                 <DIR>     7-06-90  12:28p
TEST               <DIR>    10-26-90   9:53a
llb                   30     5-28-91   8:28a
makefile           7,295     5-30-91  12:36p
print                 34     5-30-91  10:53a
lmmdct    152     11,332     4-16-91  10:12a
mm        152    280,122     5-08-91  11:03a
pb        152     49,173     5-28-91   8:25a
smmlib    152     17,995     4-16-91  10:13a
lmmdct    31      11,328     4-16-91  10:15a
mm        31     280,118     5-08-91  11:05a
pb        31      49,169     5-28-91   8:25a
smmlib    31      17,991     4-16-91  10:16a
lmmdct    at       4,948     4-16-91  10:17a
mm        at      65,321     5-08-91  11:06a
pb        at      20,712     5-28-91   8:25a
smmlib    at       7,678     4-16-91  10:18a
lmm       c       19,952     4-16-91  10:07a
lmmdct    c        2,584     3-28-91  12:12p
mm        c       37,764     5-08-91  11:00a
pb        c       11,843     5-28-91   8:23a
smm       c       39,448     4-16-91  10:08a
smmlib    c        3,292     4-08-91   9:06a
lmm       h        4,567     3-28-91  12:37p
mm        h        5,851     4-16-91   9:56a
pb        h        2,341     4-05-91   9:27a
smm       h        4,218     4-16-91   9:58a

 Directory of C:\MRA\COM\SRC\MMS\TEST

.                  <DIR>    10-26-90   9:53a
..                 <DIR>    10-26-90   9:53a
list               9,907     2-15-91  10:52a
test              42,383     4-01-91   4:12p
c         bat         23     1-10-91   8:23a
l         bat         14     2-14-91   9:02a
o         bat         11     7-10-90   4:18p
list      c        5,522     2-15-91  10:54a
test      c        1,915     5-08-91  10:52a
test      exe     22,551     5-08-91  11:11a
test      hex     53,424     1-24-91   4:31p
list      lst     91,206     2-15-91  10:52a
test      lst     36,310     4-01-91   4:07p
list      m51     26,704     2-15-91  10:52a
test      m51    116,262     4-01-91   4:12p
list      obj      9,926     2-15-91  10:52a
test      obj      1,666     5-08-91  11:11a
list      res         50     2-14-91   9:04a
test      res        116     4-01-91   4:11p

 Directory of C:\MRA\ICN
```



```
.                  <DIR>     7-06-90   8:43a
..                 <DIR>     7-06-90   8:43a
BIN                <DIR>     7-06-90   8:47a
SRC                <DIR>     7-06-90   8:43a

 Directory of C:\MRA\ICN\BIN

.                  <DIR>     7-06-90   8:47a
..                 <DIR>     7-06-90   8:47a
80152              <DIR>     3-25-91  11:33a

 Directory of C:\MRA\ICN\BIN\80152

.                  <DIR>     3-25-91  11:33a
..                 <DIR>     3-25-91  11:33a
MON                <DIR>     3-25-91  12:28p
iac               22,038     5-21-91   4:03p
iac       hex     27,405     5-21-91   4:04p
gcd       obj     13,781     3-29-91   1:16p
gci       obj      1,772     3-28-91   4:13p
main      obj        220     4-16-91  10:26a
mra       obj      1,331     4-16-91  10:26a

 Directory of C:\MRA\ICN\BIN\80152\MON

.                  <DIR>     3-25-91  12:28p
..                 <DIR>     3-25-91  12:28p
icnmon            22,039     5-21-91   4:04p
icnmon    hex     27,407     5-21-91   4:04p
gcd       obj     13,782     3-29-91   1:17p

 Directory of C:\MRA\ICN\SRC

.                  <DIR>     7-06-90   8:43a
..                 <DIR>     7-06-90   8:43a
AC                 <DIR>    10-10-90  10:01a
GCS                <DIR>     7-06-90   8:43a

 Directory of C:\MRA\ICN\SRC\AC

.                  <DIR>    10-10-90  10:01a
..                 <DIR>    10-10-90  10:01a
TEST               <DIR>    11-01-90   8:13a
makefile           6,332     5-30-91  12:50p
print                 34     5-30-91  10:53a
main      152      5,614     4-16-91  10:26a
mra       152     20,971     4-16-91  10:26a
main      c        1,876     3-17-91  10:18a
mra       c        7,879     4-05-91   8:49a
mra       h        3,644     4-05-91   8:59a
iac       m51     73,852     5-21-91   4:04p
icnmon    m51     73,927     5-21-91   4:04p
iac       res        241     3-25-91   1:08p
icnmon    res        285     3-25-91   1:14p

 Directory of C:\MRA\ICN\SRC\AC\TEST

.                  <DIR>    11-01-90   8:13a
..                 <DIR>    11-01-90   8:13a
pktio             11,114     2-05-91  10:39a
pktiol            11,047     2-05-91  10:40a
c         bat         37    11-01-90   8:25a
l         bat         19    11-01-90   8:27a
ll        bat         20     2-01-91   2:37p
o         bat         11     7-10-90   4:18p
pktio     c          519     3-28-91   8:09a
pktiol    c          359     2-05-91  10:39a
pktio     hex     15,252     2-05-91  10:39a
pktiol    hex     15,195     2-05-91  10:40a
pktio     lst      5,141     2-05-91  10:39a
pktiol    lst      3,925     2-05-91  10:39a
pktio     m51     34,644     2-05-91  10:39a
pktiol    m51     34,279     2-05-91  10:40a
```

```
pktio    obj        859   2-05-91  10:39a
pktiol   obj        736   2-05-91  10:39a
pktio    res        170  12-19-90   4:20p
pktiol   res        171   2-01-91   2:37p

 Directory of C:\MRA\ICN\SRC\GCS

.                 <DIR>     7-06-90   8:43a
..                <DIR>     7-06-90   8:43a
TEST              <DIR>     8-14-90   8:49a
lib                   30    5-30-91  10:44a
makefile           4,856    5-30-91  12:52p
print                 34    5-30-91  10:53a
gcd      152     148,333    3-29-91   1:16p
gci      152      26,897    3-28-91   4:13p
bmf      c        18,110    3-27-91   8:01a
gcd      c        26,732    3-28-91  10:19a
gci      c        11,005    2-01-91   3:38p
gsc      c        53,696    3-29-91   1:14p
bmf      h         2,993    3-27-91   7:56a
gcd      h         3,842    1-30-91   4:51p
gci      h         3,209    3-26-91   4:06p
gsc      h         7,566    3-14-91   2:55p
gcd      mon     148,345    3-29-91   1:17p

 Directory of C:\MRA\ICN\SRC\GCS\TEST

.                 <DIR>     8-14-90   8:49a
..                <DIR>     8-14-90   8:49a
c        bat          27    7-11-90   8:27a
l        bat          90    7-10-90   4:15p
o        bat          11    7-10-90   4:18p

 Directory of C:\MRA\LIB

.                 <DIR>     5-30-91   8:05a
..                <DIR>     5-30-91   8:05a
mra_1521 lib      78,072    5-30-91  10:44a
mra_311  lib      63,581    5-30-91   9:10a
mra_mss  lib      49,747    5-30-91  10:42a
mra_sbc8 lib       5,141    5-30-91  10:42a

 Directory of C:\MRA\MPU

.                 <DIR>     7-06-90   8:44a
..                <DIR>     7-06-90   8:44a
BIN               <DIR>     7-06-90   8:46a
SRC               <DIR>     7-06-90   8:46a

 Directory of C:\MRA\MPU\BIN

.                 <DIR>     7-06-90   8:46a
..                <DIR>     7-06-90   8:46a
80152             <DIR>     3-25-91   3:35p
8031              <DIR>     3-21-91   8:35a
MSDOS             <DIR>     3-21-91   8:35a
SBC8              <DIR>     3-25-91   1:39p

 Directory of C:\MRA\MPU\BIN\80152

.                 <DIR>     3-25-91   3:35p
..                <DIR>     3-25-91   3:35p
ldi      obj         296    5-30-91   9:09a

 Directory of C:\MRA\MPU\BIN\8031

.                 <DIR>     3-21-91   8:35a
..                <DIR>     3-21-91   8:35a
ldi      obj         296    5-30-91   9:09a
main     obj         367    4-16-91  10:28a
mra      obj         622    4-16-91  10:28a

 Directory of C:\MRA\MPU\BIN\MSDOS
```



```
.                 <DIR>     3-21-91   8:35a
..                <DIR>     3-21-91   8:35a
ldi      obj         530    5-30-91   9:09a
main     obj       1,124    4-16-91  10:28a
mra      obj       1,072    4-16-91  10:29a

 Directory of C:\MRA\MPU\BIN\SBC8

.                 <DIR>     3-25-91   1:39p
..                <DIR>     3-25-91   1:39p
ldi      obj         530    5-30-91   9:09a

 Directory of C:\MRA\MPU\SRC

.                 <DIR>     7-06-90   8:46a
..                <DIR>     7-06-90   8:46a
AC                <DIR>    10-10-90  10:01a
LDS               <DIR>     7-06-90  12:29p

 Directory of C:\MRA\MPU\SRC\AC

.                 <DIR>    10-10-90  10:01a
..                <DIR>    10-10-90  10:01a
TEST              <DIR>    11-01-90   8:14a
makefile           5,205    5-30-91  12:54p
print                 34    5-30-91  10:53a
main     31        6,988    4-16-91  10:28a
mra      31       14,930    4-16-91  10:28a
main     at        4,139    4-16-91  10:28a
mra      at       11,599    4-16-91  10:29a
main     c         2,301    4-05-91   8:29a
mra      c         7,063    4-05-91   8:50a
mra      h         3,644    4-05-91   8:57a

 Directory of C:\MRA\MPU\SRC\AC\TEST

.                 <DIR>    11-01-90   8:14a
..                <DIR>    11-01-90   8:14a

 Directory of C:\MRA\MPU\SRC\LDS

.                 <DIR>     7-06-90  12:29p
..                <DIR>     7-06-90  12:29p
TEST              <DIR>    10-10-90   2:17p
lib                   30    5-30-91   9:10a
makefile           4,722    5-30-91  12:55p
print                 34    5-30-91  10:53a
ldi      152       6,512    5-30-91   9:09a
ldi      31        6,508    5-30-91   9:09a
ldi      at        6,153    5-30-91   9:09a
ldi      c         3,819    5-30-91   9:08a
ldi      h         2,090    3-25-91  11:47a
ldi      sbc       6,153    5-30-91   9:09a

 Directory of C:\MRA\MPU\SRC\LDS\TEST

.                 <DIR>    10-10-90   2:17p
..                <DIR>    10-10-90   2:17p

538 files found   21,860,352 bytes free
```

```
#***************************************************************************#
#                              MAKEFILE                                     #
#***************************************************************************#
#
# CPCI:          IED90-MRA-MAKEFILE-TXT-R1C0
#
# Description:   Makefile for the Modular Robotic Architecture (MRA).
#                Updates any/all of the programming modules that reside
#                under the individual systems and subsystems.
#
#                Targets are available for the following systems/subsystems:
#
#                mra   - MRA subsystems (make all)
#
#                dev   - COM Standard Hardware Device Driver Subsystem
#                lcs   - COM Local Communications Subsystem
#                mms   - COM Method Manager Subsystem
#
#                gcs   - ICN Global Communications Subsystem
#                iac   - ICN Applications Controller
#
#                lds   - MPU Logical Device Subsystem
#                mac   - MPU Applications Controller
#
#                av    - MPU Audio/Visual Module
#                cair  - MPU Collision Avoidance IR Module
#                caus  - MPU Collision Avoidance Ultrasonic Module
#                cs    - MPU Control Station Module
#                hlp   - MPU High-Level Processing Module
#                mob   - MPU Mobility Module
#                nav   - MPU Navigation Module
#                watch - MPU ICNwatch Module
#
#                Compile-time literal definitions and their meanings follow:
#
#                I8031      - MPU using Intel 8031 running at 11.0592 MHz
#                IBMAT      - MPU using IBM-AT compatible runnning at 12 MHz
#                SBC8       - MPU using STD LPM-SBC8 (V20) running at 8 MHz
#                I80152     - ICN using Intel 80152 running at 14.7456 MHz
#                ICNMON     - ICN (80152) network monitor program
#                DEBUG      - Conditional compilation of debug code
#
# Notes:         1) The dependency and production rules are included here.
#                2) Linkage parameters for RAM-based systems using the CP-31:
#                   CODE SEGMENT  = 08000h
#                   XDATA SEGMENT = 0C000h
#                   STARTUP CODE  = \c51\cram.obj
#                3) Linkage parameters for ROM-based systems using the CP-31:
#                   CODE SEGMENT  = 00000h
#                   XDATA SEGMENT = 08000h
#                   STARTUP CODE  = \c51\crom.obj
#                4) Linkage parameters for ROM-based systems using the ICN:
#                   CODE SEGMENT  = 00000h
#                   XDATA SEGMENT = 00000h
#                   STARTUP CODE  = \c51\crom.obj
#                5) Source files for most subsystems are the same between
#                   compilers/target systems. Conditional compilation is used
#                   to generate the binary files for each system. The binary
#                   files are stored according to target system in the
#                   appropriate (\bin) directory. This allows the source file
#                   name to be maintained while separating the binaries.
#
#                   Target System      Processor        Binary Directory
#                   -------------------------------------------------------
#                   CP-31              8031             bin\8031
#                   CP-31/535          8031/80535       bin\8031
#                   ICN                80C152           bin\80152
#                   IBM-PC/AT          8088/80286       bin\MSDOS
#                   LPM-SBC8           8088/V20         bin\SBC8
#
#                   a) The binary files for different applications targeted
#                      for the same system are stored in subdirectories in
#                      the binary directory. For example, the Global Device
```



```
#                      Driver (GCD) subsystem binary file for the ICN monitor
#                      application is stored as icn\bin\80152\mon\gcd.obj.
#                   b) File suffixes are used to distinguish between various
#                      target system files with the same (root) source file
#                      name.
#
#                   Target System      File Suffix      Example
#                   -------------------------------------------------------
#                   CP-31              .31              rtc.31   (listing)
#                   CP-31/535          .31              trc.31
#                   ICN                .152             lcd.152
#                   IBM-PC/AT          .at              main.at
#                   LPM-SBC8           .sbc             sio.sbc
#
# Edit History: 07/07/90 - Written by Robin T. Laird.
#               05/27/91 - Last modified by Robin T. Laird.
#
#***************************************************************************#



################################### RULES ###################################

.SUFFIXES : .hex .exe .obj .c .a51

.IGNORE   :

# Control settings for Franklin 8031 development

CC          =c51
AS          =a51
LINK        =l51
OTOH        =ohs51
CFLAGS      =cd la db sb
ASFLAGS     =
LFLAGS      =
OFLAGS      =
STARTUP     =\c51\crom.obj
CODESEG     =00000h
XDATASEG    =00000h

# Control settings for Microsoft MS-DOS development

MSC         =cl
MSAS        =masm
MSLINK      =link
MSCFLAGS    =/AS /c /Oi /Zi /Od
MSASFLAGS   =
MSLNKFLGS   =/co
LOADLIBES   =

.c.obj :
	$(CC) $< $(CFLAGS)

.a51.obj :
	$(AS) $< $(ASFLAGS)

.obj.exe :
	$(LINK) $(STARTUP),$< TO $@ code ($(CODESEG)) xdata ($(XDATASEG)) ixref

.exe.hex :
	$(OTOH) $< $(OFLAGS)



################################ DEFINITIONS ################################

# Project, system, and application level definitions

PROJ            = \src\mra
APPSYS          = app
COMSYS          = com
ICNSYS          = icn
```

```
MPUSYS          = mpu

MRALIB          = \$(PROJ)\lib                              # Library binaries

APPSRC          = \$(PROJ)\$(APPSYS)\src                    # Source files
COMSRC          = \$(PROJ)\$(COMSYS)\src
ICNSRC          = \$(PROJ)\$(ICNSYS)\src
MPUSRC          = \$(PROJ)\$(MPUSYS)\src

APPBIN31        = \$(PROJ)\$(APPSYS)\bin\8031               # 8031 binaries
COMBIN31        = \$(PROJ)\$(COMSYS)\bin\8031
ICNBIN31        = \$(PROJ)\$(ICNSYS)\bin\8031
MPUBIN31        = \$(PROJ)\$(MPUSYS)\bin\8031

APPBIN152       = \$(PROJ)\$(APPSYS)\bin\80152              # 80152 binaries
COMBIN152       = \$(PROJ)\$(COMSYS)\bin\80152
ICNBIN152       = \$(PROJ)\$(ICNSYS)\bin\80152
MPUBIN152       = \$(PROJ)\$(MPUSYS)\bin\80152
ICNBIN152_MON   = \$(PROJ)\$(ICNSYS)\bin\80152\mon

APPBINMS        = \$(PROJ)\$(APPSYS)\bin\msdos              # MS-DOS binaries
COMBINMS        = \$(PROJ)\$(COMSYS)\bin\msdos
ICNBINMS        = \$(PROJ)\$(ICNSYS)\bin\msdos
MPUBINMS        = \$(PROJ)\$(MPUSYS)\bin\msdos
MPUBINMS_WATCH  = \$(PROJ)\$(MPUSYS)\bin\msdos\watch

APPBINSBC8      = \$(PROJ)\$(APPSYS)\bin\sbc8               # LPM-SBC8 binaries
COMBINSBC8      = \$(PROJ)\$(COMSYS)\bin\sbc8
ICNBINSBC8      = \$(PROJ)\$(ICNSYS)\bin\sbc8
MPUBINSBC8      = \$(PROJ)\$(MPUSYS)\bin\sbc8

# Common subsystem level source directories

DEVSRC          = $(COMSRC)\dev
HDRSRC          = $(COMSRC)\hdr
LCSSRC          = $(COMSRC)\lcs
MMSSRC          = $(COMSRC)\mms

# ICN subsystem level source directories

GCSSRC          = $(ICNSRC)\gcs
IACSRC          = $(ICNSRC)\ac

# MPU subsystem level source directories

LDSSRC          = $(MPUSRC)\lds
MACSRC          = $(MPUSRC)\ac

# Application subsystem level source directories

AVSRC           = $(APPSRC)\av
CAIRSRC         = $(APPSRC)\cair
CAUSSRC         = $(APPSRC)\caus
CSSRC           = $(APPSRC)\cs
HLPSRC          = $(APPSRC)\hlp
MOBSRC          = $(APPSRC)\mob
NAVSRC          = $(APPSRC)\nav
WATCHSRC        = $(APPSRC)\watch



##################################### TARGETS ####################################

# MRA system target (the whole thing)

mra     : dev lcs mms iac gcs mac lds av cair caus cs hlp mob nav watch
          cd \mra


# Common system targets

dev     :
          cd $(DEVSRC)
```



```
          make
          make lib

lcs     :
          cd $(LCSSRC)
          make
          make lib

mms     :
          cd $(MMSSRC)
          make
          make lib


# ICN system targets

iac     :
          cd $(IACSRC)
          make

gcs     :
          cd $(GCSSRC)
          make
          make lib


# MPU system targets

mac     :
          cd $(MACSRC)
          make

lds     :
          cd $(LDSSRC)
          make
          make lib


# Application system targets

av      :
          cd $(AVSRC)
          make

cair    :
          cd $(CAIRSRC)
          make

caus    :
          cd $(CAUSSRC)
          make

cs      :
          cd $(CSSRC)
          make

hlp     :
          cd $(HLPSRC)
          make

mob     :
          cd $(MOBSRC)
          make

nav     :
          cd $(NAVSRC)
          make

watch   :
          cd $(WATCHSRC)
          make
```

```
#*************************************************************************#
#                              MAKEFILE                                   #
#*************************************************************************#
#                                                                         #
# CPCI:          IED90-MRA-APP-CAIR-MAKEFILE-TXT-ROCO                     #
#                                                                         #
# Description:   Makefile for the Mobile Security Robot application.      #
#                Makes the Collision Avoidance IR module subsystems.      #
#                                                                         #
#                Targets are available for the following systems/subsystems: #
#                                                                         #
#                cair     - MPU Collision Avoidance IR Module             #
#                cair.hex - MPU Collision Avoidance IR Program (8031)     #
#                print    - Print Collision Avoidance IR source files     #
#                                                                         #
# Notes:         1) The dependency and production rules are included here. #
#                2) See also \mra\makefile.                               #
#                                                                         #
# Edit History: 03/25/91 - Written by Robin T. Laird.                     #
#                                                                         #
#*************************************************************************#



#################################### RULES ####################################

.SUFFIXES : .hex .exe .obj .c .a51

# Control settings for Franklin 8031 development

CC          =c51
AS          =a51
LINK        =l51
OTOH        =ohs51
CFLAGS      =cd la db sb
ASFLAGS     =
LFLAGS      =
OFLAGS      =
STARTUP     =\c51\crom.obj
CODESEG     =00000h
XDATASEG    =00000h

# Control settings for Microsoft MS-DOS development

MSC         =cl
MSAS        =masm
MSLINK      =link
MSCFLAGS    =/AL /c /Oi /Zi /Od
MSASFLAGS   =
MSLNKFLGS   =/co
LOADLIBES   =

.c.obj :
        $(CC) $< $(CFLAGS)

.a51.obj :
        $(AS) $< $(ASFLAGS)

.obj.exe :
        $(LINK) $(STARTUP),$< TO $@ code ($(CODESEG)) xdata ($(XDATASEG)) ixref

.exe.hex :
        $(OTOH) $< $(OFLAGS)


################################# DEFINITIONS #################################

# Project, system, and application level definitions

PROJ            = mra
APPSYS          = app
COMSYS          = com
```



```
ICNSYS          = icn
MPUSYS          = mpu

APPSRC          = \$(PROJ)\$(APPSYS)\src
COMSRC          = \$(PROJ)\$(COMSYS)\src
ICNSRC          = \$(PROJ)\$(ICNSYS)\src
MPUSRC          = \$(PROJ)\$(MPUSYS)\src

APPBIN31        = \$(PROJ)\$(APPSYS)\bin\8031
APPBIN152       = \$(PROJ)\$(APPSYS)\bin\80152
APPBINMS        = \$(PROJ)\$(APPSYS)\bin\msdos
APPBINSBC8      = \$(PROJ)\$(APPSYS)\bin\sbc8

COMBIN31        = \$(PROJ)\$(COMSYS)\bin\8031

MPUBIN31        = \$(PROJ)\$(MPUSYS)\bin\8031

# Common subsystem level source directories

DEVSRC          = $(COMSRC)\dev
HDRSRC          = $(COMSRC)\hdr
LCSSRC          = $(COMSRC)\lcs
MMSSRC          = $(COMSRC)\mms

# ICN subsystem level source directories

GCSSRC          = $(ICNSRC)\gcs
IACSRC          = $(ICNSRC)\ac

# MPU subsystem level source directories

LDSSRC          = $(MPUSRC)\lds
MACSRC          = $(MPUSRC)\ac

# Application subsystem level source directories

CAIRSRC         = $(APPSRC)\cair

# Common subsystem global include and compilation units

SYSDEFS         = $(HDRSRC)\sysdefs.h

# Application subsystem compilation units

CAIR            = $(APPBIN31)\caird.obj        $(APPBIN31)\cairl.obj   \
                  $(APPBINMS)\cairl.obj        $(MPUBIN31)\main.obj    \
                  $(MPUBIN31)\mra.obj          $(COMBIN31)\rtc.obj     \
                  $(COMBIN31)\mm.obj           $(COMBIN31)\pb.obj      \
                  $(COMBIN31)\lmm.obj          $(COMBIN31)\smm.obj     \
                  $(COMBIN31)\lmmdct.obj       $(COMBIN31)\smmlib.obj  \
                  $(COMBIN31)\lci.obj          $(COMBIN31)\lcd.obj



################################### TARGETS ###################################

cair                    : $(APPBIN31)\cair.hex

$(APPBIN31)\cair.hex    : $(APPBIN31)\cair
                          $(OTOH) $< hex($@)

$(APPBIN31)\cair        : $(CAIR)
                          $(LINK) @$(CAIRSRC)\cair.res

print                   : $(CAIR)
                          %-a2ps -nf caird.h | post
                          %-a2ps -nf caird.c | post
                          %-a2ps -nf cairl.h | post
                          %-a2ps -nf cairl.c | post
                          touch print
```

```
############################ CAIR SYSTEM DEPENDENCIES ##########################

# Collision Avoidance IR dictionary system dependencies

$(APPBIN31)\caird.obj : $(SYSDEFS)                                      \
                        $(MMSSRC)\mm.h                                  \
                        $(MMSSRC)\lmm.h                                 \
                        $(CAIRSRC)\caird.h                              \
                        $(CAIRSRC)\caird.c
  $(CC) $(CAIRSRC)\$*.c $(CFLAGS) df(18031) pr($(CAIRSRC)\$*.31) oj($(APPBIN31)


# Collision Avoidance IR library system dependencies

CAIRL = $(SYSDEFS)              $(MMSSRC)\mm.h          $(MMSSRC)\smm.h \
        $(CAIRSRC)\cairl.h      $(CAIRSRC)\cairl.c

$(APPBIN31)\cairl.obj : $(CAIRL)
  $(CC) $(CAIRSRC)\$*.c $(CFLAGS) df(18031) pr($(CAIRSRC)\$*.31) oj($(APPBIN31)

$(APPBINMS)\cairl.obj : $(CAIRL)
  $(MSC) $(MSCFLAGS) /DIBMAT /Fs$(CAIRSRC)\$*.at /Fo$(APPBINMS)\$* $(CAIRSRC)\$
```

```
\c51\crom.obj,                  &
\mra\app\bin\8031\cairrd.obj,   &
\mra\mpu\bin\8031\main.obj,     &
\mra\mpu\bin\8031\mra.obj,      &
\mra\lib\mra_31l.lib            &
to \mra\app\bin\8031\cair       &
pr(\mra\app\src\cair\cair.m51) co(00000h) xd(08000h) ix
```

```
/**********************************************************************\
*                              CAIRD.H                                 *
************************************************************************
*                                                                      *
* CPCI:          IED90-MRA-APP-CAIR-CAIRD-H-ROCO                       *
*                                                                      *
* Description:   Collision Avoidance IR (CAIR) module dictionary.      *
*                Contains prototypes for module-specific functions.    *
*                Defines local method dictionary data structure.       *
*                Defines the number of available module functions.     *
*                                                                      *
*                Module CAIRD.H exports the following variables/functions: *
*                                                                      *
*                Module LMM.H:                                         *
*                int lmm_num_funcs;                                    *
*                int *lmm_dictionary[]();                              *
*                                                                      *
*                Module LMM.H:                                         *
*                char v_obj_class[];                                   *
*                char v_obj_superclass[]                               *
*                char v_unit_name[];                                   *
*                int  v_unit_reset;                                    *
*                                                                      *
*                CAIR (QUERY_OPERATING_LIMITS):                        *
*                cair_max_update_rate();                               *
*                cair_number_sensors();                                *
*                cair_sensor_range();                                  *
*                                                                      *
*                CAIR (STATUS_REQUEST, PERIODIC_STATUS_REQUEST):       *
*                cair_report_sensor();                                 *
*                cair_ir1_ir2_report_sensor();                         *
*                cair_wait_delta_sense();                              *
*                                                                      *
* Notes:         1) Dictionaries include methods (functions) for all   *
*                   component classes of an object extending back to   *
*                   the root object (OBJECT).                          *
*                2) This file is included by the Local Method Manager (LMM). *
*                3) Error codes must be consistent with those in CAIRL.H. *
*                                                                      *
* Edit History: 01/04/91 - Written by Robin T. Laird.                  *
*                                                                      *
\**********************************************************************/

#ifndef CAIRD_MODULE_CODE
#define CAIRD_MODULE_CODE       20000

/* Public Data Structures:                                             */

/* Number of functions in dictionary (including component functions). */
/* Local method manager (LMM) global variable is defined/initialized here. */

#define CAIR_NUM_FUNCS          10
int lmm_num_funcs               = CAIR_NUM_FUNCS;

/* Object class instance variables and values for this module.         */

char v_obj_class[]              = OBJ_MODULE_CLASS;
char v_obj_superclass[]         = OBJ_OBJECT_CLASS;

/* Unit class instance variables and values for this module.           */

char v_unit_name[]              = "CAIR";
int  v_unit_reset               = UNIT_NOT_INIT;

/* Prototype declarations for the dictionary functions.                */
/* All functions return an int value indicating success/failure.       */

int cair_max_update_rate(), cair_number_sensors();
int cair_sensor_range(), cair_report_sensor();
int cair_ir1_ir2_report_sensor();
int cair_wait_delta_sense();

/* Function error codes.
```



```
/* Error codes indicate source of function execution failure.          */

#define CAIR_BAD_SENSOR_NUMBER  1

/* Dictionary data structure (pointers to the various functions).      */
/* Object (obj_) and unit (unit_) class methods are defined in LMM.DCT. */
/* The function d_unit_reset() is called first upon system start-up.   */

int (*lmm_dictionary[CAIR_NUM_FUNCS]) () =
      { d_obj_class,          d_obj_superclass,       d_unit_name,
        d_unit_reset,         cair_max_update_rate,   cair_number_sensors,
        cair_sensor_range,    cair_report_sensor,     cair_ir1_ir2_report_sen
        cair_wait_delta_sense
      };

#endif
```

```c
/***********************************************************************\
*                               CAIRD.C                                 *
*************************************** ********************************
*                                                                       *
* CPCI:         IED90-MRA-APP-CAIR-CAIRD-C-ROCO                         *
*                                                                       *
* Description:  Collision Avoidance IR (CAIR) dictionary functions.     *
*               These functions are available to all modules in the system. *
*               Implements the (dictionary) functions for the collision *
*               avoidance IR module (cair_).                            *
*                                                                       *
*               Parameters are passed to/from the functions via the standard *
*               I/O buffers (mm_stdin and mm_stdout). All local methods must *
*               return an integer value indicating success or failure as in *
*               MM_COMMAND_EXECUTED or MM_COMMAND_EXECUTION_FAILURE. Also, *
*               for commands that fail, the reason for failure must be put *
*               in the output buffer as a byte value.                   *
*                                                                       *
*               Module CAIRD.C exports the following variables/functions: *
*                                                                       *
*               reset();                                                *
*               cair_max_update_rate();                                 *
*               cair_number_sensors();                                  *
*               cair_sensor_range();                                    *
*               cair_report_sensor();                                   *
*               cair_ir1_ir2_report_sensor();                           *
*               cair_wait_delta_sense();                                *
*                                                                       *
* Notes:        1) The calling sequence for each function is listed below. *
*                                                                       *
* Edit History: 01/10/91 - Written by Robin T. Laird.                   *
*                                                                       *
\***********************************************************************/

#include <absacc.h>                 /* Absolute addressing defs.        */
#include <sysdefs.h>                /* System constants and types.      */
#include <mm.h>                     /* Method manager.                  */
#include <lmm.h>                    /* Local method manager.            */
#include "caird.h"                  /* CAIR dictionary entries.         */

/* Global "instance variables".                                         */

static XDATA byte v_cair_max_update_rate= 10;   /* Hz.                  */
static XDATA byte v_cair_number_sensors = 11;   /* Discrete.            */
static XDATA int  v_cair_sensor_range   = 300;  /* Tenths of inches.    */

/* The IR sensors are connected to the 8255 of the 8031 (CP-31).        */
/* IR sensors 1-8 are in port A, sensor 1 at bit 0, sensor 8 at bit 7.  */
/* IR sensors 9-11 are in port C, sensor 9 at bit 0, sensor 11 at bit 2. */

/* Connector continuity is wired to the remaining bits of ports B and C. */
/* If a sensor is connected, port bit value is 1, otherwise 0.          */
/* Continuity for IRs 1-8 are in port B, cont 1 at bit 0, cont 8 at bit 7. */
/* Continuity for IRs 9-11 are in port C, cont 9 at bit 4, cont 11 at bit 6.*/

/* Addressing of the CP-31 puts 8255 ports A, B, and C in external data. */

#define PA_8255         XBYTE[0x0000]   /* 8255 port A.                 */
#define PB_8255         XBYTE[0x0001]   /* 8255 port B.                 */
#define PC_8255         XBYTE[0x0002]   /* 8255 port C.                 */
#define PORTCON_8255    XBYTE[0x0003]   /* 8255 control register.       */

/* 8255 control values (see Intel books).                               */

#define ALL_INPUT       0x9B            /* Ports A, B, and C as input.  */

/* Continuity check bit shift values and masks.                         */

#define PC_CONT_SHIFT   4               /* Shift to make a word.        */
#define PC_CONT_MASK    0x0070          /* Keep bits 4, 5, and 6.       */
#define PC_CONT_UNUSED  0x07FF          /* Masks out missing sensor bits. */

/* Continuity check mask global variable.                               */
```



```c
/* Used to mask out (force to 0=OFF) all disconnected sensors.          */

static XDATA word cont_mask;


/***********************************************************************\
*                               reset                                   *
*************************************************************************
*                                                                       *
* Function:     Resets (initializes) the CAIR.C module.                 *
*               All "housekeeping" required to initialize the module    *
*               is done here. This routine is called upon system power-up, *
*               before any of the other module functions are called.    *
*                                                                       *
* Input:        reset();                                                *
*                                                                       *
* Output:       Command execution status as an integer value (see MM.H). *
*                                                                       *
* Globals:      cont_mask : module CAIR.C                               *
*                                                                       *
* Edit History: 03/06/91 - Written by Robin T. Laird.                   *
*                                                                       *
\***********************************************************************/

int reset()
{
  /* Set up the CP-31 8255 for input on ports A, B, and C.              */

  PORTCON_8255 = ALL_INPUT;

  /* Check the IR sensor plug continuity, and set global sensor mask.   */
  /* Complement mask since 0 = installed, 1 = not-installed.            */

  cont_mask = (((word)PC_8255 & PC_CONT_MASK) << PC_CONT_SHIFT) | (word)PB_8255
  cont_mask = PC_CONT_UNUSED & ~cont_mask;
}


/***********************************************************************\
*                         cair_max_update_rate                          *
*************************************************************************
*                                                                       *
* Function:     Returns the maximum update rate for the IR sensor report *
*               functions (in Hz or updates per second). The value is   *
*               returned in the standard parameter output buffer.       *
*                                                                       *
* Input:        cair_max_update_rate();                                 *
*                                                                       *
* Output:       Max update rate as a byte value (mm_stdout).            *
*                                                                       *
* Globals:      mm_stdout : module MM.C                                 *
*               v_cair_max_update_rate : module CAIR.C                  *
*                                                                       *
* Edit History: 03/06/91 - Written by Robin T. Laird.                   *
*                                                                       *
\***********************************************************************/

int cair_max_update_rate()
{
  /* Put max update rate in the standard parameter output buffer.       */
  /* Return with command exec OK.                                       */

  mm_sprintfb(&mm_stdout, "%b", v_cair_max_update_rate);
  return(MM_COMMAND_EXECUTED);
}


/***********************************************************************\
*                         cair_number_sensors                           *
*************************************************************************
*                                                                       *
* Function:     Returns the number of IR sensors available.             *
*               The value is returned in the standard parameter output  *
```

```
 *                 buffer.                                                   *
 *                                                                           *
 * Input:          cair_number_sensors();                                    *
 *                                                                           *
 * Output:         Number of IR sensors as a byte value (mm_stdout).        *
 *                                                                           *
 * Globals:        mm_stdout : module MM.C                                   *
 *                 v_cair_number_sensors : module CAIR.C                     *
 *                                                                           *
 * Edit History: 03/06/91 - Written by Robin T. Laird.                       *
 *                                                                           *
\***********************      *********************************************/

int cair_number_sensors
{
  /* Put number of sensors in the standard parameter output buffer.         */
  /* Return with command exec OK.                                           */

  mm_sprintfb(&mm_stdout, "%b", v_cair_number_sensors);
  return(MM_COMMAND_EXECUTED);
}


/***************************************************************************\
*                           cair_sensor_range                               *
*****************************************************************************
 *                                                                           *
 * Function:       Returns the maximum range of the IR sensor (in tenths of  *
 *                 inches). The value is returned in the standard parameter  *
 *                 output buffer.                                            *
 *                                                                           *
 * Input:          cair_sensor_range();                                      *
 *                                                                           *
 * Output:         IR sensor range as an integer value (mm_stdout).          *
 *                                                                           *
 * Globals:        mm_stdout : module MM.C                                   *
 *                 v_cair_sensor_range : module CAIR.C                       *
 *                                                                           *
 * Edit History: 03/06/91 - Written by Robin T. Laird.                       *
 *                                                                           *
\***************************************************************************/

int cair_sensor_range()
{
  /* Put sensor range in the standard parameter output buffer.              */
  /* Return with command exec OK.                                           */

  mm_sprintfb(&mm_stdout, "%d", v_cair_sensor_range);
  return(MM_COMMAND_EXECUTED);
}


/***************************************************************************\
*                           cair_report_sensor                              *
*****************************************************************************
 *                                                                           *
 * Function:       Reports the state (0=OFF, 1=ON) of the indicated sensor.  *
 *                 The number of the sensor to report is passed in the       *
 *                 standard parameter input buffer as a bit value.           *
 *                 If the sensor number is invalid, an error code reporting so *
 *                 is returned via the standard parameter output buffer.     *
 *                                                                           *
 *                 The sensors are attached to ports A, B, and C of the 8255. *
 *                 IR sensors 1-8 are in port A, IR1 at bit 0, IR8 at bit 7. *
 *                 IR sensors 9-11 are in port C, IR9 at bit 0, IR11 at bit 2. *
 *                                                                           *
 * Input:          cair_report_sensor(                                       *
 *                   byte s; number of the sensor to report (mm_stdin).      *
 *   .             );                                                        *
 *                                                                           *
 * Output:         State of given sensor (0/1) as a bit value (mm_stdout).   *
 *                                                                           *
 * Globals:        mm_stdin  : module MM.C                                   *
```



```
 *                 mm_stdout : module MM.C                                   *
 *                 cont_mask : module CAIR.C                                 *
 *                 v_cair_number_sensors : module CAIR.C                     *
 *                                                                           *
 * Edit History: 05/24/90 - Original code written by Richard P. Smurlo.      *
 *               03/06/91 - Dictionary implementation by Robin T. Laird.     *
 *                                                                           *
\***************************************************************************/

int cair_report_sensor()
{
  byte s, v;

  /* Get the sensor number from the standard parameter input buffer.        */

  mm_sscanfb(&mm_stdin, "%b", &s);

  /* Make sure requested sensor is in range.                                */

  if (s > v_cair_number_sensors)
  {
    /* Report invalid sensor number, and command failed.                    */

    mm_sprintfb(&mm_stdout, "%b", CAIR_BAD_SENSOR_NUMBER);
    return(MM_COMMAND_EXECUTION_FAILURE);
  }
  else
  {
    /* Get the sensor value (0=OFF/not activated, 1=ON/activated).          */
    /* Continuity mask is used to force disconnected sensors to 0=OFF.      */

    v = (((((word)PC_8255 << 8) | (word)PA_8255) & cont_mask) >> s-1) & 1;

    /* Put sensor value in standard parameter output buffer.                */

    mm_sprintfb(&mm_stdout, "%y", v);
    return(MM_COMMAND_EXECUTED);
  }
}


/***************************************************************************\
*                         cair_ir1_ir2_report_sensor                        *
*****************************************************************************
 *                                                                           *
 * Function:       Returns the state (0=OFF, 1=ON) of the indicated sensors. *
 *                 The numbers of the sensors to report are passed in the    *
 *                 standard parameter input buffer as two byte values.       *
 *                 The states of sensors s1 through s2 are returned as a     *
 *                 sequence of bits (length = |s1-s2|+1). First sensor number *
 *                 should be less than the last sensor number.               *
 *                 If either sensor number is invalid, an error code reporting *
 *                 so is returned via the standard parameter output buffer.  *
 *                                                                           *
 *                 The sensors are attached to ports A, B, and C of the 8255. *
 *                 IR sensors 1-8 are in port A, IR1 at bit 0, IR8 at bit 7. *
 *                 IR sensors 9-11 are in port C, IR9 at bit 0, IR11 at bit 2. *
 *                                                                           *
 * Input:          cair_ir1_ir2_report_sensor(                               *
 *                   byte s1; first sensor to sample (s1 <= s2) (mm_stdin).  *
 *                   byte s2; last sensor to sample (mm_stdin).              *
 *                 );                                                        *
 *                                                                           *
 * Output:         States of given sensors (0/1) as a bit string (mm_stdout). *
 *                                                                           *
 * Globals:        mm_stdin  : module MM.C                                   *
 *                 mm_stdout : module MM.C                                   *
 *                 cont_mask : module CAIR.C                                 *
 *                 v_cair_number_sensors : module CAIR.C                     *
 *                                                                           *
 * Edit History: 05/24/90 - Original code written by Richard P. Smurlo.      *
 *               03/06/91 - Dictionary implementation by Robin T. Laird.     *
 *                                                                           *
```

```
\*****************************************************************************/

int cair_ir1 ir2_report_sensor()
{
  byte s1, s2, stemp, v;
  word vall, i;

  /* Get the sensor numbers from the standard parameter input buffer.     */

  mm_sscanfb(&mm_stdin, "%b%b", &s1, &s2);

  /* Make sure requested sensors are in range.                            */

  if (s1 > v_cair_number_sensors || s2 > v_cair_number_sensors)
  {
    /* Report invalid sensor range, and command failed.                   */

    mm_sprintfb(&mm_stdout, "%b", CAIR_BAD_SENSOR_NUMBER);
    return(MM_COMMAND_EXECUTION_FAILURE);
  }
  else
  {
    /* Make sure first sensor is less than last sensor.                   */

    if (s1 > s2)
    {
      stemp = s1;
      s1 = s2;
      s2 = stemp;
    }

    /* Get all sensor values (0=OFF/not activated, 1=ON/activated).       */
    /* Continuity mask is used to force disconnected sensors to 0=OFF.    */

    vall = (((word)PC_8255 << 8) | (word)PA_8255) & cont_mask;

    /* Loop through the sensors.                                          */

    for (i = s1; i <= s2; i++)
    {
      /* Extract particular bit value for this sensor.                    */

      v = (vall >> i-1) & 1;

      /* Put sensor value in standard parameter output buffer.            */

      mm_sprintfb(&mm_stdout, "%y", v);
    }
    return(MM_COMMAND_EXECUTED);
  }
}


/*****************************************************************************\
*                           cair_wait_delta_sense                             *
*******************************************************************************
*                                                                             *
* Function:     Returns the new state of the IR sensors (all of them)         *
*               if a change in states is seen between function calls.        *
*               All sensors are checked for a sense change in any             *
*               one sensor. The values of all sensors is returned upon        *
*               detecting a change (so the calling program must               *
*               determine which sensors actually changed states).             *
*                                                                             *
* Input:        cair_wait_delta_sense();                                      *
*                                                                             *
* Output:       States of all sensors as a word value (mm_stdout).            *
*                                                                             *
* Globals:      mm_stdout : module MM.C                                       *
*               cont_mask : module CAIR.C                                     *
*                                                                             *
* Edit History: 05/24/90 - Original code written by Richard P. Smurlo.        *
*               03/06/91 - Dictionary implementation by Robin T. Laird.       *
```



```
*                                                                             *
\*****************************************************************************/

int cair_wait_delta_sense()
{
  static word prev_state = 0xFF;
  word        curr_state;

  /* Get current state of sensors.                                        */

  curr_state = (((word)PC_8255 << 8) | (word)PA_8255) & cont_mask;

  /* If we've detected a change, return sensor values, response required. */
  /* Otherwise, indicate no response message required.                    */

  if (prev_state != curr_state)
  {
    prev_state = curr_state;
    mm_sprintfb(&mm_stdout, "%u", curr_state);
    return(MM_COMMAND_EXECUTED);
  }
  else
    return(MM_SUPPRESS_OUTPUT);
}
```

```
/************************************************************************\
*                               CAIRL.H                                  *
**************************************************************************
*                                                                        *
* CPCI:          IED90-MRA-APP-CAIR-CAIRL-H-ROCO                         *
*                                                                        *
* Description:   Collision Avoidance IR (CAIR) library function include file. *
*                Contains function prototypes and literals (#defines) for the *
*                collision avoidance IR module (cair_).                  *
*                                                                        *
*                Module CAIRL.H exports the following variables/functions: *
*                                                                        *
*                CAIR (QUERY_OPERATING_LIMITS):                          *
*                cair_max_update_rate();                                 *
*                cair_number_sensors();                                  *
*                cair_sensor_range();                                    *
*                                                                        *
*                CAIR (STATUS_REQUEST, PERIODIC_STATUS_REQUEST):         *
*                cair_report_sensor();                                   *
*                cair_ir1_ir2_report_sensor();                           *
*                cair_wait_delta_sense();                                *
*                                                                        *
* Notes:         1) The calling sequence for each function is listed below. *
*                2) Error codes must be consistent with those in CAIRD.H. *
*                                                                        *
* Edit History: 01/22/91 - Written by Robin T. Laird.                    *
*                                                                        *
\************************************************************************/

#ifndef CAIRL_MODULE_CODE
#define CAIRL_MODULE_CODE       20500

#include <smm.h>                        /* System method manager.        */

/* Public data structures:                                               */

#define CAIR_UNIT_NAME          "CAIR"

#define CAIR_FN_MAX_UPDATE_RATE (0+SMM_NUM_INHERITED_FNS)
#define CAIR_FN_NUMBER_SENSORS  (1+SMM_NUM_INHERITED_FNS)
#define CAIR_FN_SENSOR_RANGE    (2+SMM_NUM_INHERITED_FNS)
#define CAIR_FN_REPORT_SENSOR   (3+SMM_NUM_INHERITED_FNS)
#define CAIR_FN_IR1_IR2_REPORT  (4+SMM_NUM_INHERITED_FNS)
#define CAIR_FN_WAIT_DELTA      (5+SMM_NUM_INHERITED_FNS)

/* Function error codes.
/* Error codes indicate source of function execution failure.            */

#define CAIR_BAD_SENSOR_NUMBER  1

/* Public functions:                                                     */

int cair_max_update_rate(void);
int cair_number_sensors(void);
int cair_sensor_range(void);
int cair_report_sensor(byte s, word period);
int cair_ir1_ir2_report_sensor(byte s1, byte s2, word period);
int cair_wait_delta_sense(word period);

#endif
```

```c
/***********************************************************************\
*                               CAIRL.C                                 *
*************************************************************************
*                                                                       *
* CPCI:          IED90-MRA-APP-CAIR-CAIRL-C-ROCO                        *
*                                                                       *
* Description:   Collision Avoidance IR (CAIR) library functions.       *
*                These functions are available to all modules in the system. *
*                Implements the (library) functions for the collision   *
*                avoidance IR module (cair_).                           *
*                                                                       *
*                Parameters are passed to/from the functions via the standard *
*                I/O buffers (mm_stdin and mm_stdout). All local methods must *
*                return an integer value indicating success or failure as in *
*                MM_COMMAND_EXECUTED or MM_COMMAND_EXECUTION_FAILURE. Also, *
*                for commands tha fail, the reason for failure must be put *
*                in the output buffer as a byte value.                  *
*                                                                       *
*                Module CAIRL.C exports the following variables/functions: *
*                                                                       *
*                cair_max_update_rate();                                *
*                cair_number_sensors();                                 *
*                cair_sensor_range();                                   *
*                cair_report_sensor();                                  *
*                cair_ir1_ir2_report_sensor();                          *
*                cair_wait_delta_sense();                               *
*                                                                       *
* Notes:         1) The calling sequence for each function is listed below. *
*                                                                       *
* Edit History: 01/10/91 - Written by Robin T. Laird.                   *
*                                                                       *
\***********************************************************************/

#include <sysdefs.h>                 /* System constants and types.     */
#include <mm.h>                      /* Method manager.                 */
#include <smm.h>                     /* System method manager.          */
#include "cairl.h"                   /* Unit name and function IDs.     */


int cair_max_update_rate()
{
  int          event;
  mm_message m;

  m.trans_disposition = MM_INITIATING;
  m.trans_category    = MM_QUERY_OPERATING_LIMITS;
  m.function_id       = CAIR_FN_MAX_UPDATE_RATE;
  smm_generate_message(CAIR_UNIT_NAME, &m, &event);
  return(event);
}


int cair_number_sensors()
{
  int          event;
  mm_message m;

  m.trans_disposition = MM_INITIATING;
  m.trans_category    = MM_QUERY_OPERATING_LIMITS;
  m.function_id       = CAIR_FN_NUMBER_SENSORS;
  smm_generate_message(CAIR_UNIT_NAME, &m, &event);
  return(event);
}


int cair_sensor_range()
{
  int          event;
  mm_message m;

  m.trans_disposition = MM_INITIATING;
  m.trans_category    = MM_QUERY_OPERATING_LIMITS;
  m.function_id       = CAIR_FN_SENSOR_RANGE;
```



```c
  smm_generate_message(CAIR_UNIT_NAME, &m, &event);
  return(event);
}


int cair_report_sensor(s, period)
byte s;
word period;
{
  int          event;
  mm_message m;

  m.trans_disposition = MM_INITIATING;
  m.function_id       = CAIR_FN_REPORT_SENSOR;

  /* If a periodic status request, then period will be non-NULL.          */

  if (period == NULL)
  {
    m.trans_category  = MM_STATUS_REQUEST;
    mm_sprintfb(&mm_stdout, "%b", s);
  }
  else
  {
    m.trans_category  = MM_PERIODIC_STATUS_REQUEST;
    mm_sprintfb(&mm_stdout, "%u%b", period, s);
  }
  smm_generate_message(CAIR_UNIT_NAME, &m, &event);
  return(event);
}


int cair_ir1_ir2_report_sensor(s1, s2, period)
byte s1, s2;
word period;
{
  int          event;
  mm_message m;

  m.trans_disposition = MM_INITIATING;
  m.function_id       = CAIR_FN_IR1_IR2_REPORT;

  /* If a periodic status request, then period will be non-NULL.          */

  if (period == NULL)
  {
    m.trans_category  = MM_STATUS_REQUEST;
    mm_sprintfb(&mm_stdout, "%b%b", s1, s2);
  }
  else
  {
    m.trans_category  = MM_PERIODIC_STATUS_REQUEST;
    mm_sprintfb(&mm_stdout, "%u%b%b", period, s1, s2);
  }
  smm_generate_message(CAIR_UNIT_NAME, &m, &event);
  return(event);
}


int cair_wait_delta_sense(period)
word period;
{
  int          event;
  mm_message m;

  m.trans_disposition = MM_INITIATING;
  m.function_id       = CAIR_FN_WAIT_DELTA;

  /* If a periodic status request, then period will be non-NULL.          */

  if (period == NULL)
  {
    m.trans_category  = MM_STATUS_REQUEST;
```

```
    }
    else
    {
      m.trans_category  = MM_PERIODIC_STATUS_REQUEST;
      mm_sprintfb(&mm_stdout, "%u", period);
    }
    amm_generate_message(CAIR_UNIT_NAME, &m, &event);
    return(event);
  }
```

```
#***************************************************************************#
#                                 MAKEFILE                                  #
#***************************************************************************#
#                                                                           #
# CPCI:          IED90-MRA-COM-DEV-MAKEFILE-TXT-R0C0                        #
#                                                                           #
# Description:   Makefile for the Modular Robotic Architecture (MRA).       #
#                Makes the common device driver subsystem.                  #
#                                                                           #
#                Targets are available for the following systems/subsystems:#
#                                                                           #
#                dev   - COM Standard Hardware Device Driver Subsystem      #
#                lib   - Add modules to MRA library                         #
#                print - Print COM device driver files                      #
#                                                                           #
# Notes:         1) The dependency and production rules are included here.  #
#                2) See also \mra\makefile.                                 #
#                                                                           #
# Edit History: 03/22/91 - Written by Robin T. Laird.                       #
#                                                                           #
#***************************************************************************#



################################### RULES ###################################

.SUFFIXES : .hex .exe .obj .c .a51

# Control settings for Franklin 8031 development

CC              =c51
AS              =a51
LINK            =l51
OTOH            =ohs51
CFLAGS          =cd la db sb
ASFLAGS         =
LFLAGS          =
OFLAGS          =
STARTUP         =\c51\crom.obj
CODESEG         =00000h
XDATASEG        =00000h

# Control settings for Microsoft MS-DOS development

MSC             =cl
MSAS            =masm
MSLINK          =link
MSCFLAGS        =/AL /c /Oi /Zi /Od
MSASFLAGS       =
MSLNKFLGS       =/co
LOADLIBES       =

.c.obj :
        $(CC) $< $(CFLAGS)

.a51.obj :
        $(AS) $< $(ASFLAGS)

.obj.exe :
        $(LINK) $(STARTUP),$< TO $@ code ($(CODESEG)) xdata ($(XDATASEG)) ixref

.exe.hex :
        $(OTOH) $< $(OFLAGS)



################################ DEFINITIONS ################################

# Project, system, and application level definitions

PROJ                = mra
APPSYS              = app
COMSYS              = com
```



```

COMLIB              = \$(PROJ)\lib
COMSRC              = \$(PROJ)\$(COMSYS)\src
COMBIN31            = \$(PROJ)\$(COMSYS)\bin\8031
COMBIN152           = \$(PROJ)\$(COMSYS)\bin\80152
COMBINMS            = \$(PROJ)\$(COMSYS)\bin\msdos
COMBINSBC8          = \$(PROJ)\$(COMSYS)\bin\sbc8

# Common subsystem level source directories

DEVSRC              = $(COMSRC)\dev
HDRSRC              = $(COMSRC)\hdr

# Common subsystem global include and compilation units

SYSDEFS             = $(HDRSRC)\sysdefs.h

DEV                 = $(COMBIN152)\rtc.obj          $(COMBIN31)\rtc.obj     \
                      $(COMBINMS)\rtc.obj           $(COMBIN152)\sio.obj    \
                      $(COMBIN31)\sio.obj           $(COMBINMS)\sio.obj     \
                      $(COMBINSBC8)\sio.obj



################################## TARGETS ##################################

dev                 : $(DEV)

lib                 : $(DEV)
                      -lib51 delete $(COMLIB)\mra_1521.lib (rtc, sio)
                      -lib51 add $(COMBIN152)\rtc.obj to $(COMLIB)\mra_1521.lib
                      -lib51 add $(COMBIN152)\sio.obj to $(COMLIB)\mra_1521.lib
                      -lib51 delete $(COMLIB)\mra_311.lib (rtc, sio)
                      -lib51 add $(COMBIN31)\rtc.obj to $(COMLIB)\mra_311.lib
                      -lib51 add $(COMBIN31)\sio.obj to $(COMLIB)\mra_311.lib
                      -lib $(COMLIB)\mra_mss.lib -rtc.obj+$(COMBINMS)\rtc.obj;
                      -lib $(COMLIB)\mra_msl.lib -rtc.obj+$(COMBINMS)\rtc.obj;
                      -lib $(COMLIB)\mra_mss.lib -sio.obj+$(COMBINMS)\sio.obj;
                      -lib $(COMLIB)\mra_msl.lib -sio.obj+$(COMBINMS)\sio.obj;
                      -lib $(COMLIB)\mra_sbc8.lib -sio.obj+$(COMBINSBC8)\sio.obj;
                      touch lib

print:                $(DEV)
                      %-a2ps -nf rtc.h | post
                      %-a2ps -nf rtc.c | post
                      %-a2ps -nf sio.h | post
                      %-a2ps -nf sio.c | post
                      touch print



############################ COM DEV DEPENDENCIES ###########################

# COM Real Time Clock module dependencies for 80152, 8031, MS-DOS

RTC = $(SYSDEFS) $(DEVSRC)\rtc.h $(DEVSRC)\rtc.c

$(COMBIN152)\rtc.obj            : $(RTC)
  $(CC) $(DEVSRC)\$*.c $(CFLAGS) df(I80152) pr($(DEVSRC)\$*.152) oj($(COMBIN152

$(COMBIN31)\rtc.obj             : $(RTC)
  $(CC) $(DEVSRC)\$*.c $(CFLAGS) df(I8031)  pr($(DEVSRC)\$*.31)  oj($(COMBIN31)

$(COMBINMS)\rtc.obj             : $(RTC)
  $(MSC) $(MSCFLAGS) /DIBMAT /Fs$(DEVSRC)\$*.at /Fo$(COMBINMS)\$* $(DEVSRC)\$*.


# COM Serial I/O module dependencies 80152, 8031, MS-DOS

SIO = $(SYSDEFS) $(DEVSRC)\sio.h $(DEVSRC)\sio.c

$(COMBIN152)\sio.obj            : $(SIO)
  $(CC) $(DEVSRC)\$*.c $(CFLAGS) df(I80152) pr($(DEVSRC)\$*.152) oj($(COMBIN152
```

```
$(COMBIN31)\sio.obj              : $(SIO)
  $(CC) $(DEVSRC)\$*.c $(CFLAGS) df(I8031)  pr($(DEVSRC)\$*.31)  oj($(COMBIN31)

$(COMBINMS)\sio.obj              : $(SIO)
  $(MSC) $(MSCFLAGS) /DIBMAT /Fs$(DEVSRC)\$*.at /Fo$(COMBINMS)\$* $(DEVSRC)\$*.

$(COMBINSBC8)\sio.obj            : $(SIO)
  $(MSC) $(MSCFLAGS) /DSBC8 /Fs$(DEVSRC)\$*.sbc /Fo$(COMBINSBC8)\$* $(DEVSRC)\$
```

```
/************************************************************************\
*                               rtc.h                                   *
*************************************************************************
*                                                                       *
* CPCI:          IED90-MRA-COM-DEV-RTC-H-ROC1                           *
*                                                                       *
* Description:   Real-time clock device driver external declarations.   *
*                Contains the constant definitions and function prototypes *
*                for the RTC.C module. These functions provide standard *
*                access to the on-board real-time clock.                *
*                                                                       *
*                Module RTC exports the following functions:            *
*                                                                       *
*                rtc_init();                                            *
*                rtc_wait();                                            *
*                rtc_time();                                            *
*                                                                       *
* Notes:         1) The RTC functions are part of the MRA standard      *
*                   hardware device driver (DEV) subsystem.             *
*                                                                       *
* Edit History: 03/28/91 - Modified by Robin T. Laird.                  *
*                                                                       *
\***********************************************************************/

/* Public Data Structures:                                              */

#ifndef RTC_MODULE_CODE
#define RTC_MODULE_CODE          5000

#define ERR_RTC_NOT_INIT         1+RTC_MODULE_CODE

#define RTC_RANDOM_TIME          0L

/* External module global error variable.                               */

extern int rtc_error;

/* Public Functions:                                                    */

void rtc_init(void);
void rtc_wait(unsigned long duration);
unsigned long rtc_time(void);

#endif
```

```c
/****************************************************************************\
*                                 rtc.c                                      *
******************************************************************************
*                                                                            *
* CPCI:         IED90-MRA-COM-DEV-RTC-C-R0C1                                 *
*                                                                            *
* Description:  Real-time clock device driver functions.                     *
*               Implements a real-time clock with approximate millesecond    *
*               resolution. The functions below are currently implemented    *
*               Currently implemented for the Intel 8031 microcontroller.    *
*                                                                            *
*               Module RTC exports the following functions:                  *
*                                                                            *
*               rtc_init();                                                  *
*               rtc_wait();                                                  *
*               rtc_time();                                                  *
*                                                                            *
* Notes:        1) The RTC functions are part of the MRA standard            *
*                  hardware device driver (DEV) subsystem.                   *
*               2) See the Intel 8-Bit Embedded Controllers handbook         *
*                  for more information (No. 270645-002, pp. 7-7 - 7-11).    *
*                                                                            *
* Edit History: 05/02/90 - Modified by Robin T. Laird.                       *
*                                                                            *
\****************************************************************************/

#if defined(IBMAT)
#include <timeb.h>                        /* MS-DOS time struct definitions.  */
#else
#include <reg51.h>                        /* SFR definitions for 8031.        */
#endif
#include <stdlib.h>                       /* Random number generator fns.     */
#include <sysdefs.h>                      /* System constants and types.      */
#include <rtc.h>                          /* Real-time clock constants.       */

/* Public Variables:                                                          */

/* Global module error variable, rtc_error.                                   */
/* rtc_error contains code of last error occurence.                           */
/* Should be set to AOK after each successful function call.                  */
/* Variable can be examined by other software after each function call.       */

XDATA int rtc_error = ERR_RTC_NOT_INIT; /* Global module error variable.      */

/* Declare internal clock tick counter.                                       */
/* Rolls over after 4,294,967,295 counts.                                     */

#if defined(I80152) || defined(I8031)
static data unsigned long tickcnt = 0L;
#endif


/****************************************************************************\
*                               rtc_tickint                                  *
******************************************************************************
*                                                                            *
* Function:     Tick count interrupt routine. Increments the module          *
*               variable tickcnt every timer interrupt. Represents           *
*               elapsed time since the initialization routine was            *
*               last called. The rate at which the function is               *
*               called depends upon the TIMER/COUNTER mode and other         *
*               parameters specified in the initialization routine.          *
*               8031 family interrupt number 1 is TIMER/COUNTER-0.           *
*               Use register bank 2 for handling this interrupt.             *
*                                                                            *
* Input:        rtc_tickint();                                               *
*                                                                            *
* Output:       Nothing.                                                     *
*                                                                            *
* Globals:      tickcnt : module RTC.C                                       *
*                                                                            *
* Edit History: 06/19/90 - Written by Robin T. Laird.                        *
*                                                                            *
```



```c
\****************************************************************************/

#if !defined(IBMAT)
static void rtc_tickint() interrupt 1 using 2
{
  tickcnt++;
}
#endif


/****************************************************************************\
*                                rtc_init                                    *
******************************************************************************
*                                                                            *
* Function:     Routine to initialize the CP-31 (8031) TIMER 0 for           *
*               use as a real-time (pseudo-millisecond resoultion)           *
*               clock. TIMER 0 is configured to operate in mode 0            *
*               which is a 13-bit counter that interrupts the CPU            *
*               every OSCfrequency/(12*8192*1000) milliseconds.              *
*               This routine also clears the module tick count variable      *
*               tickcnt, and then starts the timer.                          *
*                                                                            *
* Input:        rtc_init();                                                  *
*                                                                            *
* Output:       Nothing.                                                     *
*                                                                            *
* Globals:      rtc_error : module RTC.C                                     *
*               tickcnt   : module RTC.C                                     *
*                                                                            *
* Edit History: 06/19/90 - Written by Robin T. Laird.                        *
*               03/28/91 - Modified by Robin T. Laird to init srand().       *
*                                                                            *
\****************************************************************************/

/* Define 8031 timer/counter control register bit functions (for TIMER 0).  */

#define GATE_CONTROL              0x08
#define NO_GATE_CONTROL           0x00

#define TIMER_FUNCTION            0x00
#define COUNTER_FUNCTION          0x04

#define MODE_0                    0x00
#define MODE_1                    0x01
#define MODE_2                    0x02
#define MODE_3                    0x03

void rtc_init()
{
  rtc_error = AOK;              /* Assume function successful...           */

  #if !defined(IBMAT)

  /* Clear TIMER 0 control bits.                                            */
  /* Set TIMER 0 to MODE 0: 13-bit timer (MCS-48 compatible).               */

  TMOD &= 0xF0;
  TMOD |= NO_GATE_CONTROL | TIMER_FUNCTION | MODE_0;

  /* Expicitly zero tick counter (one way of resetting clock).              */
  /* Enable TIMER/COUNTER-0 interrupts.                                     */
  /* Turn TIMER/COUNTER on (1=on, 0=off).                                   */

  tickcnt = 0L;

  ET0 = 1;                      /* TIMER 0 interrupt enable bit.           */
  TR0 = 1;                      /* TIMER 0 run control bit.                */
  EA  = 1;                      /* Enable processor interrupts.            */

  #endif

  /* Seed random number generator used for wait function below.             */
```

```
  srand((int)rtc_time());
}


/*************************************************************************\
*                              rtc_wait                                   *
***************************************************************************
*                                                                         *
* Function:     Waits for a specified period of time (in ms).             *
*               Function does not return until time expired.              *
*               If duration parameter is RTC_RANDOM_TIME then the function *
*               waits a random amount of time between 2000 and 7000 ms.   *
*                                                                         *
* Input:        rtc_wait(                                                 *
*                 unsigned long duration; time to wait or RTC_RANDOM_TIME. *
*               );                                                        *
*                                                                         *
* Output:       Nothing.                                                  *
*                                                                         *
* Globals:      rtc_error : module RTC.C                                  *
*                                                                         *
* Edit History: 03/28/91 - Written by Robin T. Laird.                     *
*                                                                         *
\*************************************************************************/

#define MIN_RANDOM_TIME         2000    /* Wait no less than 2 seconds.    */
#define MAX_RANDOM_TIME         7000    /* Wait no more than 7 seconds.    */

void rtc_wait(duration)
unsigned long duration;
{
  int t;
  unsigned long t0;

  rtc_error = AOK;              /* Assume function successful...           */

  t0 = rtc_time();
  if (duration == RTC_RANDOM_TIME)
  {
    /* Gen a random time >= MIN_RANDOM_TIME and <= MAX_RANDOM_TIME.        */
    /* Make sure we don't wait any longer than MAX_RANDOM_TIME trying.     */

    t = rand();
    while (t < MIN_RANDOM_TIME || t > MAX_RANDOM_TIME)
    {
      t = rand();
      if (rtc_time() > t0 + MAX_RANDOM_TIME) return;
    }

    /* Wait that amount of time.                                           */

    while (rtc_time() < t0 + t);
  }
  else
  {
    /* Wait amount of time specified by parameter duration.                */

    while (rtc_time() < t0 + duration);
  }
}


/*************************************************************************\
*                              rtc_time                                   *
***************************************************************************
*                                                                         *
* Function:     Routine to return time in milliseconds. The timer         *
*               period (as configured in the initialization routine)      *
*               is approximately 8.888...ms for an 11.0592 MHz CPU (or    *
*               6.666...ms for 14.7456 MHz CPU). Add whole portion of ms  *
*               value derived from tickcnt and fraction of timer period   *
*               derived from TIMER 0 to calculate actual time. This gives *
*               a time value that is accurate within +/- 1 ms.            *
```



```
*                                                                         *
* Input:        rtc_time();                                               *
*                                                                         *
* Output:       Returns absolute time value in milliseconds.              *
*                                                                         *
* Globals:      rtc_error : module RTC.C                                  *
*               tickcnt   : module RTC.C                                  *
*                                                                         *
* Edit History: 06/19/90 - Written by Robin T. Laird.                     *
*                                                                         *
\*************************************************************************/

/* Define divisor for calculation of time value (based on tickcnt).       */
/* E.g., OSCf/(12*timer roll-over*ms scale) = 11.0592e6/(12*8192*1000).   */
/* Play some games to avoid floating point math.                          */

#if defined(I80152)
#define MS_SCALE_FACTOR         20      /* Reduced fraction equivalent.   */
#define OSCF_SCALE_FACTOR       3       /* Reduced fraction equivalent.   */
#define MS_PER_COUNT            5       /* ms/count dividend.             */
#define COUNT_PER_MS            6144    /* count/ms divisor.              */
#else
#define MS_SCALE_FACTOR         80      /* Reduced fraction equivalent.   */
#define OSCF_SCALE_FACTOR       9       /* Reduced fraction equivalent.   */
#define MS_PER_COUNT            5       /* ms/count dividend.             */
#define COUNT_PER_MS            4608    /* count/ms divisor.              */
#endif

unsigned long rtc_time()
{
  #if defined(IBMAT)
  static struct timeb timebuf;
  #else
  static word count;
  #endif

  rtc_error = AOK;              /* Assume function successful...          */

  #if !defined(IBMAT)

  /* Calculate portion of timer period (in ms) based on current timer value.*/
  /* Add this value to time based on tickcnt to give actual time.          */

  count = ((((word)TH0<<5) | ((word)TL0&0x1F)) * MS_PER_COUNT) / COUNT_PER_MS;
  return((unsigned long)((tickcnt*MS_SCALE_FACTOR)/OSCF_SCALE_FACTOR) + count);

  #else

  /* Get time and store in timebuf structure (this includes ms portion).   */
  /* Return number of seconds * 1000 plus ms time for total ms time.       */

  ftime(&timebuf);
  return((unsigned long)timebuf.time*1000+timebuf.millitm);

  #endif
}
```

```
/*********************************************************************\
*                              sio.h                                  *
*.....................................................................*
*                                                                     *
* CPCI:          IED90-MRA-COM-DEV-SIO-H-ROC3                         *
*                                                                     *
* Description:   Serial I/O device driver public declarations.        *
*                Contains the constant definitions and function prototypes *
*                for the SIO.C module. These functions provide standard *
*                access to the on-board serial I/O port.              *
*                                                                     *
*                Module SIO exports the following variables/functions: *
*                                                                     *
*                int sio_error;                                       *
*                                                                     *
*                sio_init();                                          *
*                sio_putbyte();                                       *
*                sio_getbyte();                                       *
*                sio_byteavail();                                     *
*                sio_putstr();                                        *
*                sio_getstr();                                        *
*                sio_putnstr();                                       *
*                                                                     *
* Notes:         1) The SIO functions are part of the MRA standard    *
*                   hardware device driver (DEV) subsystem.           *
*                                                                     *
* Edit History: 05/15/91 - Modified by Robin T. Laird.                *
*                                                                     *
\*********************************************************************/

/* Public Data Structures:                                            */

#ifndef SIO_MODULE_CODE
#define SIO_MODULE_CODE           6000

#define SIO_ERR_NOT_INIT          1+SIO_MODULE_CODE
#define SIO_ERR_BAUD_RATE         2+SIO_MODULE_CODE
#define SIO_ERR_PARITY            3+SIO_MODULE_CODE
#define SIO_ERR_WORD_LENGTH       4+SIO_MODULE_CODE
#define SIO_ERR_STOP_BITS         5+SIO_MODULE_CODE
#define SIO_ERR_STRING_LENGTH     6+SIO_MODULE_CODE

/* I80152 signifies baud rate settings for 14.7456 MHz CPU.           */
/* I8031  signifies baud rate settings for 11.0592 MHz CPU.           */
/* IBMAT signifies baud rate settings for IBM-PC/AT using 8250.       */
/* SBC8 signifies baud rate settings for Winsystems LPM-SBC8-9.       */

#if defined(I80152)
#define MAX_BAUD_RATE             0xFF     /* 38.4 KBPS (SMOD=0)       */
#define BR19200                   0xFE
#define BR9600                    0xFC
#define BR4800                    0xF8
#define BR2400                    0xF0
#define BR1200                    0xE0
#define BR600                     0xC0
#define BR300                     0x80

#elif defined(I8031)
#define MAX_BAUD_RATE             0xFF     /* 57.6 KBPS (SMOD=1)       */
#define BR19200                   0xFD
#define BR9600                    0xFA
#define BR4800                    0xF4
#define BR2400                    0xE8
#define BR1200                    0xD0
#define BR600                     0xA0
#define BR300                     0x40

#elif defined(IBMAT)
#define BR38400                   3        /* Baud rate divisor settings. */
#define BR19200                   6
#define BR9600                    12
#define BR4800                    24
#define BR2400                    48
```



```
#define BR1200                    96
#define BR600                     192
#define BR300                     384

#elif defined(SBC8)
#define BR38400                   0x7E     /* Baud rate CO SELECT=0 DIVISOR=4. */
#define BR19200                   0x02
#define BR9600                    0x06
#define BR4800                    0x0E
#define BR2400                    0x16
#define BR1200                    0x22
#define BR600                     0x26
#define BR300                     0x2A
#endif

#if defined(I8031) || defined(I80152)
#define PEVEN                     0xC0     /* Parity settings (80535).        */
#define PODD                      0x80
#define PNONE                     0x00

#define WL5                       0xC0     /* Word length settings (80535).   */
#define WL6                       0x80
#define WL7                       0x40
#define WL8                       0x00

#define SB1                       0x00     /* Stop bit settings (80535).      */
#define SB2                       0x20

/* Local serial channel port identifier (8031 and 80152).             */
/* Auxiliary serial channel port identifier (80535 only).             */

#define LSC                       1
#define ASC                       2

#elif defined(IBMAT)
#define PEVEN                     0x18     /* Parity settings.                */
#define PODD                      0x08
#define PNONE                     0x00

#define WL5                       0x00     /* Word length settings.           */
#define WL6                       0x01
#define WL7                       0x02
#define WL8                       0x03

#define SB1                       0x00     /* Stop bit settings.              */
#define SB2                       0x04

/* COM port base addresses.                                           */

#define COM1                      0x3F0    /* 3F8-3FF.                        */
#define COM2                      0x2F0    /* 2F8-2FF.                        */
#define COM3                      0x3E0    /* 3E8-3EF.                        */
#define COM4                      0x2E0    /* 2E8-2EF.                        */

#elif defined(SBC8)
#define PEVEN                     0x00     /* Parity settings.                */
#define PODD                      0x02
#define PNONE                     0x0C

#define WL5                       0x00     /* Word length settings.           */
#define WL6                       0x10
#define WL7                       0x20
#define WL8                       0x30
                                           /* Stop bit settings.              */
#define SB1                       0x00
#define SB2                       0x01

/* COM port base addresses.                                           */

#define COM1                      0x90     /* LPM-SBC8 J1.                    */
#define COM2                      0x20     /* LPM-SIO4 J10.                   */
#define COM3                      0x24     /* LPM-SIO4 J8.                    */
#define COM4                      0x28     /* LPM-SIO4 J4.                    */
```

```
#define OM5                 0x2C    /* LPM-SIO4 J3.                 */
#endif

/* External module global error variable.                           */

extern int sio_error;

/* Public Functions:                                                */

void sio_init(int port, int baud_rate, int parity, int word_len, int stop_bit);
void sio_putbyte(int port, byte c);
byte sio_getbyte(int port);
int  sio_byteavail(int port);
void sio_putstr(int port, byte *s);
int  sio_getstr(int port, byte *s);
void sio_putnstr(int port, int numbytes, byte *s);

#endif
```

```
/************************************************************************\
*                              sio.c                                    *
*************************************************************************
*                                                                       *
* CPCI:          IED90-MRA-COM-DEV-SIO-C-ROC3                           *
*                                                                       *
* Description:   Serial I/O device driver functions.                    *
*                Implements a standard set of serial I/O functions for  *
*                transmitting/receiving data over the local serial port.*
*                The actual serial port used depends upon the implementation, *
*                but the functions and their prototypes must remain the same. *
*                Currently implemented for the Intel 8031, 80152, 80535, and *
*                IBM-AT (8250).                                         *
*                                                                       *
*                Module SIO exports the following variables/functions:  *
*                                                                       *
*                int sio_error;                                         *
*                                                                       *
*                sio_init();                                            *
*                sio_putbyte();                                         *
*                sio_getbyte();                                         *
*                sio_byteavail();                                       *
*                sio_putstr();                                          *
*                sio_getstr();                                          *
*                sio_putnstr();                                         *
*                                                                       *
* Notes:         1) The SIO functions are part of the MRA standard      *
*                   hardware device driver (DEV) subsystem.             *
*                2) See the Intel 8-Bit Embedded Controllers handbook   *
*                   for more information (No. 270645-002, pp. 7-11 - 7-21). *
*                3) The serial I/O functions for the auxiliary serial channel *
*                   (ASC) of the CP-31/535 are interrupt driven, whereas all *
*                   other functions are based on polling.               *
*                4) The CP-31/535 must be configured so that serial I/O *
*                   interrupts of the 8256 (level L4 and L5) are "connected" *
*                   to the INT0 line of the 80535.                      *
*                                                                       *
* Edit History: 05/21/91 - Modified by Robin T. Laird.                  *
*                                                                       *
\************************************************************************/

#if defined(I80152) || defined(I8031)
#include <reg515.h>                     /* SFR definitions for 8031, etc.  */
#include <absacc.h>                     /* Absolute memory access defs.    */
#endif
#include <string.h>                     /* Standard string functions.      */
#include <sysdefs.h>                    /* System constants and types.     */
#include <sio.h>                        /* Serial I/O constants.           */

/* Public Variables:                                                        */

/* Global module error variable, sio_error.                                 */
/* sio_error contains code of last error occurence.                         */
/* Should be set to AOK after each successful function call.                */
/* Variable can be examined by other software after each function call.     */

XDATA int sio_error = SIO_ERR_NOT_INIT; /* Global module error variable.    */

/* Communications port definitions for MPU (IBM-AT 8250).                   */

#if defined(IBMAT)

/* "8250" communications controller (CC) register address offsets.         */

#define CC_THR                  0x08    /* Transmit holding reg (write).   */
#define CC_RBR                  0x08    /* Receiver buffer reg (read).     */
#define CC_DLSB                 0x08    /* Divisor latch LSB.              */
#define CC_DMSB                 0x09    /* Divisor latch MSB.              */
#define CC_IER                  0x09    /* Interrupt enable register.      */
#define CC_IIR                  0x0A    /* Interrupt identification reg.   */
#define CC_LCR                  0x0B    /* Line control register.          */
#define CC_MCR                  0x0C    /* Modem control register.         */
#define CC_LSR                  0x0D    /* Line status register.           */
```



```
#define CC_MSR                  0x0E    /* Modem status register.          */

#define CC_DATA_READY           0x01    /* LSR data ready bit.             */
#define CC_RECEIVE_ERRS         0x1E    /* LSR error bits mask.            */
#define CC_TRANSMIT_HOLD_EMPTY  0x20    /* LSR xmtr holding register empty. */

#define CC_DLAB0                0x00    /* DLAB set low (0).               */
#define CC_DLAB1                0x80    /* DLAB set high (1).              */

#define CC_DISABLE_INT          0x00    /* Disable all interrupts.         */
#define CC_ENABLE_RCV           0x01    /* Enable receive interrupts.      */
#define CC_ENABLE_XMT           0x02    /* Enable transmit interrupts.     */
#define CC_ENABLE_STAT          0x04    /* Enable line status interrupts.  */
#define CC_MODEM_BUS_ENABLE     0x0B    /* DTR, RTS, and OUT2 active.      */

#elif defined(SBC8)

/* "8252" communications controller (CC) register address offsets.         */

#define CC_RBR                  0x00    /* Receiver buffer register.       */
#define CC_TBR                  0x00    /* Transmit buffer register.       */
#define CC_UCR                  0x01    /* UART control register.          */
#define CC_USR                  0x01    /* UART status register.           */
#define CC_MCR                  0x02    /* Modem control register.         */
#define CC_MSR                  0x03    /* Modem status register.          */
#define CC_BRSR                 0x03    /* Baud rate select register.      */

#define CC_MCR_RESET            0x0023  /* "Safe" MCR init value.          */
#define CC_RECV_ERR_MASK        0x000F  /* Receive error bit mask.         */
#define CC_TBRE_MASK            0x0040  /* Xfr-buff-ready bit mask.        */
#define CC_DR_MASK              0x0080  /* Data-ready bit mask.            */

/* Variables for saved USR, since with 8252 USR is reset after it's read.   */

static int old_usr = 0;
static int new_usr = 0;
static int byte_avail = FALSE;

/* "8256" MUART register definitions for CP-31/535 (80535).                 */

#else

/* 8256 MUART is addressed at absolute addresses F000H to FFFFH on 80535.   */

#define MUART_CMD_1     XBYTE[0xF000]   /* Command reg 1 (stop bit, parity).*/
#define MUART_CMD_2     XBYTE[0xF001]   /* Command reg 2 (baud rate).      */
#define MUART_CMD_3     XBYTE[0xF002]   /* Command reg 3 (RCV enable).     */

#define MUART_IER       XBYTE[0xF005]   /* Interrupt enable register.      */
#define MUART_IAR       XBYTE[0xF006]   /* Interrupt address register.     */
#define MUART_RBR       XBYTE[0xF007]   /* Data receive buffer.            */
#define MUART_TBR       XBYTE[0xF007]   /* Data transmit buffer.           */
#define MUART_MSR       XBYTE[0xF00F]   /* MUART status register.          */

#define MUART_RCV_ENABLE        0xC0    /* RCV enable, SET bit hi.         */
#define MUART_NESTED_INTS       0x90    /* Nested ints enabled, SET bit hi. */
#define MUART_EOI               0x88    /* End-of-interrupt, SET bit hi.   */

#define MUART_RECV_ERR_MASK     0x07    /* RCV errors in LSB of MSR.       */
#define MUART_TBE_MASK          0x20    /* Transmit buffer empty in MSR.   */
#define MUART_RBF_MASK          0x40    /* Receive buffer full in MSR.     */

#define MUART_RCV_INT_ENABLE    0x10    /* Level 4 interrupt enable.       */
#define MUART_XMT_INT_ENABLE    0x20    /* Level 5 interrupt enable.       */

#define MUART_RCV_INT_MASK      0x10    /* Level 4 interrupt identifier.   */
#define MUART_XMT_INT_MASK      0x14    /* Level 5 interrupt identifier.   */

/* Baud rate settings (which are different than those for 8031/80152).      */

#define _BR19200                0x03    /* Internal baud rate generator.   */
#define _BR9600                 0x04
#define _BR4800                 0x05
```

```
#define _BR2400                0x06
#define _BR1200                0x07
#define _BR600                 0x08
#define _BR300                 0x09

/* Type definition for I/O data queue.                                     */

#define MAX_BUFFER_SIZE        128

typedef struct { byte item[MAX_BUFFER_SIZE];
                 word front;
                 word rear;
                 byte empty;
                 byte full;
               } buffer;

/* Input (receive) and output (transmit) global data queues.               */

static XDATA buffer xmt_buffer;
static XDATA buffer rcv_buffer;

#endif

/***************************************************************************\
*                               buf_inc                                    *
****************************************************************************
*                                                                          *
* Function:     Macro that increments either the front or rear index       *
*               of a circular buffer of max size MAX_BUFFER_SIZE.          *
*               Should be used with care, i.e., not nested within a        *
*               compound statement like if ... then ... else.              *
*                                                                          *
* Input:        buf_inc(                                                   *
*                 word x; index to increment.                              *
*               );                                                         *
*                                                                          *
* Output:       Nothing.                                                   *
*                                                                          *
* Globals:      None.                                                      *
*                                                                          *
* Edit History: 12/04/90 - Written by Robin T. Laird.                      *
*                                                                          *
\***************************************************************************/

#define buf_inc(x)  (x = ((x == MAX_BUFFER_SIZE-1) ? 0 : x+1))


/***************************************************************************\
*                               sio_init                                   *
****************************************************************************
*                                                                          *
* Function:     Serial port initialization function. Sets up the           *
*               indicated serial port to operate according to the          *
*               specified parameters (e.g., baud rate, etc.).              *
*                                                                          *
*               For the 8031, the following settings are forced:           *
*                                                                          *
*               parity     : PNONE  (no parity)                            *
*               word_len   : WL8    (8 data bits)                          *
*               stop_bits  : SB1    (1 stop bit)                           *
*                                                                          *
* Input:        sio_init(                                                  *
*                 int port;      serial port number (identifier).          *
*                 int baud_rate; serial port baud rate.                    *
*                 int parity;    serial port parity.                       *
*                 int word_len;  serial port word length.                  *
*                 int stop_bit;  serial port stop bits.                    *
*               );                                                         *
*                                                                          *
* Output:       Nothing.                                                   *
*                                                                          *
* Globals:      sio_error : module SIO.C                                   *
```



```
*                       8031 regs : module REG515.H                        *
*                                                                          *
* Edit History: 10/10/86 - Written by Robin T. Laird.                      *
*                                                                          *
\***************************************************************************/

/* 8031 register values for initializing the local serial channel (LSC).   */
/* If using higher clock speed, SMOD is set to 0, otherwise set to 1.      */
/* SMOD is the MSB of the PCON Power Control register on the 8031.         */

#if defined(I80152) || defined(I8031)

#define SERIAL_PORT_MODE_1      0x50        /* SCON = 0bx1x1xxxx            */
#define SERIAL_PORT_MASK        0x5F        /* SCON = 0b0101xxxx            */
#define TIMER_1_MODE_2          0x20        /* TMOD = 0bxx1xxxxx            */
#define TIMER_1_MASK            0x2F        /* TMOD = 0b0010xxxx            */

#if defined(I80152)
#define DOUBLE_BAUD_RATE        0x00        /* PCON = 0b0xxxxxxx SMOD=0     */
#else
#define DOUBLE_BAUD_RATE        0x80        /* PCON = 0b1xxxxxxx SMOD=1     */
#endif

/* IBM-AT (8250) clear receive buffer attempt count.                       */

#elif defined(IBMAT)

#define MAX_RFIFO_READS         10          /* Clear this many bytes.       */

#endif

void sio_init(port, baud_rate, parity, word_len, stop_bit)
int port;
int baud_rate;
int parity;
int word_len;
int stop_bit;
{
  int i;

  sio_error = AOK;              /* Assume function successful...           */

  /* Check settings for validity.                                          */

  switch(baud_rate)
  {
   #if defined(IBMAT) || defined(SBC8)
   case BR38400:
   #elif defined(I80152) || defined(I8031)
   case MAX_BAUD_RATE:
   #endif
   case BR19200:
   case BR9600:
   case BR4800:
   case BR2400:
   case BR1200:
   case BR600:
   case BR300:
     break;
   default:
     sio_error = SIO_ERR_BAUD_RATE;
     return;
  }

  switch(parity)
  {
   case PEVEN:
   case PODD:
   case PNONE:
     break;
   default:
     sio_error = SIO_ERR_PARITY;
     return;
```

```
    }

    switch(word_len)
    {
      case WL5:
      case WL6:
      case WL7:
      case WL8:
        break;
      default:
        sio_error = SIO_ERR_WORD_LENGTH;
        return;
    }

    switch(stop_bit)
    {
      case SB1:
      case SB2:
        break;
      default:
        sio_error = SIO_ERR_STOP_BITS;
        return;
    }

    #if defined(I80152) || defined(I8031)

    /* Local and auxiliary serial channels are initialized differently.     */

    if (port == ASC)
    {
      /* Determine 8256 baud rate from parameter baud rate.                 */

      switch(baud_rate)
      {
        case MAX_BAUD_RATE:
        case BR19200:
          baud_rate = _BR19200;
          break;
        case BR9600:
          baud_rate = _BR9600;
          break;
        case BR4800:
          baud_rate = _BR4800;
          break;
        case BR2400:
          baud_rat.   _BR2400;
          break;
        case BR1200:
          baud_rate = _BR1200;
          break;
        case BR600:
          baud_rate = _BR600;
          break;
        case BR300:
          baud_rate = _BR300;
          break;
        default:
          baud_rate = _BR19200;
          break;
        break;
      }

      /* Init COMMAND REGISTER 1: stop bits, character length.              */
      /* Init COMMAND REGISTER 2: baud rate, parity.                        */
      /* Init COMMAND REGISTER 3: receiver enable.                          */
      /* Clear the receive buffer.                                          */

      MUART_CMD_1 = word_len | stop_bit;
      MUART_CMD_2 = parity   | baud_rate;
      MUART_CMD_3 = MUART_RCV_ENABLE;
      while ((MUART_MSR & MUART_RBF_MASK) == MUART_RBF_MASK) i = MUART_RBR;

      #if !defined(MUART_POLLED)
```



```

      /* Init I/O data queue structures.                                    */

      xmt_buffer.front = xmt_buffer.rear = 0;
      xmt_buffer.empty = TRUE;
      xmt_buffer.full  = FALSE;

      rcv_buffer.front = rcv_buffer.rear = 0;
      rcv_buffer.empty = TRUE;
      rcv_buffer.full  = FALSE;

      /* Indicate low-level triggered interrupt from 8256.                  */
      /* Enable external interrupt 0 (EX0) on CP-31/535 80535.              */

      IT0 = 0;
      EX0 = 1;

      /* Enable receive and transmit interrupts on the CP-31/535 8256.      */
      /* Assumes some other function enables processor interrupts.          */

      MUART_CMD_3 = MUART_NESTED_INTS;
      MUART_IER  |= MUART_RCV_INT_ENABLE;
      MUART_IER  |= MUART_XMT_INT_ENABLE;

      #endif
    }
    else
    {
      /* Set registers according to chosen options.                         */
      /* TIMER 1 is used in MODE 2 for variable baud rates.                 */
      /* TH1 sets TIMER 1 re-load value.                                    */
      /* SCON controls the serial port modes settings.                      */
      /* TMOD controls the timer mode settings.                             */
      /* TCON controls the timer itself (turns it on/off).                  */
      /* SMOD controls baud rate doubling depending upon CPU speed.         */

      PCON |= DOUBLE_BAUD_RATE;    /* Set Power Control Mode register.      */
      TH1   = baud_rate;           /* Set TIMER 1 re-load value for baud.   */
      TMOD |= TIMER_1_MODE_2;      /* Set Timer Mode Control register.      */
      TMOD &= TIMER_1_MASK;
      SCON |= SERIAL_PORT_MODE_1;  /* Set Serial Port Control register.     */
      SCON &= SERIAL_PORT_MASK;
      TR1   = 1;                   /* Set TIMER 1 run bit.                  */
    }

    #elif defined(IBMAT)

    /* Initialize 8250, see "Technical Reference Personal Computer AT".     */
    /* Set baud rate (divisor latches).                                     */
    /* DELAB must be 1 to write to baud rate divisor latches.               */
    /* Parameter baud_rate is ignored.                                      */

    outp(port+CC_LCR, CC_DLAB1);
    outp(port+CC_DMSB, (baud_rate >> 8) & 0xFF);
    outp(port+CC_DLSB, baud_rate & 0xFF);

    /* Set serial port operational parameters.                              */
    /* DELAB must be 0 to write/read LCR, IER, and data registers.          */

    outp(port+CC_LCR, CC_DLAB0 | parity | word_len | stop_bit);

    /* Disable all 8250 interrupts on this port.                            */

    outp(port+CC_IER, CC_DISABLE_INT);

    /* Clear line status and modem status registers.                        */

    inp(port+CC_LSR);
    inp(port+CC_MSR);

    /* Clear chars from receive register.                                   */
    /* Line status register (bit 0) indicates if data ready.                */

```

```
    for (i = 0; i < MAX_RFIFO_READS; i++)
    {
     if (inp(port+CC_LSR) & CC_DATA_READY)
       inp(port+CC_RBR);
     else
       break;
    }
    if (i > MAX_RFIFO_READS || inp(port+CC_LSR) & CC_DATA_READY)
    {
      sio_error = SIO_ERR_NOT_INIT;
      return;
    }

    #elif defined(SBC8)

    /* Set 8252 baud rate divisor (and CO SELECT).                          */
    /* Set 8252 UART control register.                                      */
    /* Re-set 8252 modem control register.                                  */

    outp(port+CC_BRSR, baud_rate);
    outp(port+CC_UCR,  parity | word_len | stop_bit);
    outp(port+CC_MCR,  CC_MCR_RESET);
    #endif
  }


/***************************************************************************\
*                              sio_putbyte                                   *
*****************************************************************************
*                                                                            *
* Function:     Output a byte to the specified serial port.                  *
*               Byte is output to the transmit buffer of the port.           *
*                                                                            *
* Input:        sio_putbyte(                                                 *
*                 int port; serial port number (identifier).                 *
*                 byte c;   character (value) to output.                     *
*               );                                                           *
*                                                                            *
* Output:       Nothing.                                                     *
*                                                                            *
* Globals:      sio_error : module SIO.C                                     *
*               8031 regs : module REG515.H                                  *
*                                                                            *
* Edit History: 10/10/86 - Written by Robin T. Laird.                        *
*                                                                            *
\***************************************************************************/

void sio_putbyte(port, c)
int  port;
byte c;
{
  /* Output value to serial port.                                          */
  /* Wait for character to be sent (transmit buffer empty).                */

  sio_error = AOK;            /* Assume function successful...              */

  #if defined(I80152) || defined(I8031)

  /* Local and auxiliary serial channels are managed differently.          */

  if (port == ASC)
  {
    #if defined(MUART_POLLED)

    while((MUART_MSR & MUART_TBE_MASK) != MUART_TBE_MASK);
    MUART_TBR = c;

    #else

    /* If buffer full, then wait until it's not.                           */

    while (xmt_buffer.full);

```



```
    /* Place byte in output queue.                                         */
    /* Adjust rear index to account for added byte.                        */
    /* See if buffer is full.                                              */
    /* If transmit buffer is empty, move byte to transmitter (generate INT).*/
    /* Buffer can't be empty since we just added a byte.                   */

    xmt_buffer.item[xmt_buffer.rear] = c;
    buf_inc(xmt_buffer.rear);
    if (xmt_buffer.front == xmt_buffer.rear) xmt_buffer.full = TRUE;
    if (xmt_buffer.empty)
    {
      xmt_buffer.empty = FALSE;
      MUART_TBR = c;
    }
    else
      xmt_buffer.empty = FALSE;

    #endif
  }
  else
  {
    SBUF = c;                       /* Put byte into serial buffer.         */
    while(TI != 1);                 /* Wait for transmit interrupt flag to set. */
    TI = 0;                         /* Clear interrupt flag for next xmit.  */
  }

  #elif defined(IBMAT)

  outo(port+CC_THR, c);
  while((inp(port+CC_LSR) & CC_TRANSMIT_HOLD_EMPTY) == 0);

  #elif defined(SBC8)

  /* Output value to serial port.                                          */
  /* Wait for character to be sent (transmit buffer empty).                */
  /* Need to save status of USR if byte avail.                             */

  outp(port+CC_TBR, c);
  do {
    new_usr = inp(port+CC_USR);
    if ((new_usr & CC_DR_MASK) == CC_DR_MASK) old_usr = new_usr;
  } while((inp(port+CC_USR) & CC_TBRE_MASK) != CC_TBRE_MASK);

  #endif
}


/***************************************************************************\
*                              sio_getbyte                                   *
*****************************************************************************
*                                                                            *
* Function:     Input character from specified serial port.                  *
*               Character is input from data register.                       *
*               Waits forever for input.                                     *
*                                                                            *
* Input:        sio_getbyte(                                                 *
*                 int port; serial port number (identifier).                 *
*               );                                                           *
*                                                                            *
* Output:       Returns character read from serial port.                     *
*                                                                            *
* Globals:      sio_error : module SIO.C                                     *
*               8031 regs : module REG515.H                                  *
*                                                                            *
* Edit History: 10/10/86 - Written by Robin T. Laird.                        *
*                                                                            *
\***************************************************************************/

byte sio_getbyte(port)
int port;
{
  byte c;

```

```
    /* Wait until there is a character available to input.              */
    /* Get character from port.                                         */
    /* Return input character (as a byte).                              */

    sio_error = AOK;                /* Assume function successful...    */

    #if defined(I80152) || defined(I8031)

    /* Local and auxiliary serial channels are managed differently.     */

    if (port == ASC)
    {
      #if defined(MUART_POLLED)

      while ((MUART_MSR & MUART_RBF_MASK) != MUART_RBF_MASK);
      return(MUART_RBR);

      #else

      /* If the buffer is empty, then wait for a char to be received.  */

      while(rcv_buffer.empty);

      /* Remove byte from buffer.                                       */
      /* Adjust front index to account for byte removed.                */
      /* Buffer can't be full since we just removed a byte.             */
      /* See if the buffer is empty.                                    */

      c = rcv_buffer.item[rcv_buffer.front];
      buf_inc(rcv_buffer.front);
      rcv_buffer.full = FALSE;
      if (rcv_buffer.front == rcv_buffer.rear) rcv_buffer.empty = TRUE;
      return(c);

      #endif
    }
    else
    {
      while (RI != 1);              /* Wait for receive interrupt flag to set.  */
      RI = 0;                       /* Clear interrupt flag for next recv.      */
      return(SBUF);                 /* Return byte in serial buffer.            */
    }

    #elif defined(IBMAT)

    while ((inp(port+CC_LSR) & CC_DATA_READY) == 0);
    return(inp(port+CC_RBR));

    #elif defined(SBC8)

    /* Indicate that byte no longer available.                          */
    /* Get current value of USR.                                        */
    /* If USR value is 0, then use saved USR value.                     */
    /* Otherwise (if USR != 0), wait until a byte is avail, and return it. */

    byte_avail = FALSE;
    new_usr = inp(port+CC_USR);
    if (new_usr == 0)
    {
      if ((old_usr & CC_DR_MASK) == CC_DR_MASK)
      {
        old_usr = 0;
        return(inp(port+CC_RBR));
      }
      else
      {
        while ((inp(port+CC_USR) & CC_DR_MASK) != CC_DR_MASK);
        return(inp(port+CC_RBR));
      }
    }
    else if ((new_usr & CC_DR_MASK) == CC_DR_MASK)
      return(inp(port+CC_RBR));

```



```
    while ((inp(port+CC_USR) & CC_DR_MASK) != CC_DR_MASK);
    return(inp(port+CC_RBR));

    #endif
}


/*************************************************************************\
*                              sio_byteavail                              *
***************************************************************************
*                                                                         *
* Function:     Determines whether or not a byte is available at the serial *
*               port. Does not remove the byte. Will return TRUE until the *
*               byte is removed and no new byte is received.              *
*                                                                         *
* Input:        sio_byteavail(                                            *
*                 int port; serial port number (identifier).              *
*               );                                                        *
*                                                                         *
* Output:       Returns TRUE if byte availabe, otherwise returns FALSE.   *
*                                                                         *
* Globals:      sio_error : module SIO.C                                  *
*               8031 regs : module REG515.H                               *
*                                                                         *
* Edit History: 10/10/86 - Written by Robin T. Laird.                     *
*                                                                         *
\*************************************************************************/

int sio_byteavail(port)
int port;
{
  sio_error = AOK;                /* Assume function successful...      */

  #if defined(I80152) || defined(I8031)

  /* Local and auxiliary serial channels are managed differently.       */

  if (port == ASC)
  {
    #if defined(MUART_POLLED)

    /* Check MUART status register receive buffer full flag.            */

    return(MUART_MSR & MUART_RBF_MASK ? TRUE : FALSE);

    #else

    /* Check receive buffer empty flag.                                 */

    return(!rcv_buffer.empty);

    #endif
  }
  else
  {
    /* Check RI for byte availability.                                  */

    return(RI == 1 ? TRUE : FALSE);
  }

  #elif defined(IBMAT)

  /* Check data ready for byte. Data ready reset by read of rcv buffer. */

  return((inp(port+CC_LSR) & CC_DATA_READY) == 1 ? TRUE : FALSE);

  #elif defined(SBC8)

  /* Since USR (input status reg) is reset on read we have to play games. */
  /* Check byte avail flag to see if set from previous call.            */
  /* If flag not set, get current value of USR.                         */
  /* Check old value of USR to see if byte was available.               */

```

```
    if (byte_avail)
      return(TRUE);
    else
    {
      new_usr = inp(port+CC_USR);
      if ((old_usr & CC_DR_MASK) == CC_DR_MASK)
      {
        byte_avail = TRUE;
      }
      else
      {
        old_usr = new_usr;
        byte_avail = (new_usr & CC_DR_MASK) ? TRUE : FALSE;
      }
      return(byte_avail);
    }

    #endif
}


/**********************************************************************\
*                              sio_putstr                              *
************************************************************************
*                                                                      *
* Function:     Outputs a NULL terminated string to the serial port.   *
*                                                                      *
* Input:        sio_putstr(                                            *
*                 byte *s;  pointer to string to output.               *
*                 int port; serial port number (identifier).           *
*               );                                                     *
*                                                                      *
* Output:       Nothing.                                               *
*                                                                      *
* Globals:      sio_error : module SIO.C                               *
*                                                                      *
* Edit History: 10/10/86 - Written by Robin T. Laird.                  *
*                                                                      *
\**********************************************************************/

void sio_putstr(port, s)
int  port;
byte *s;
{
  byte *stemp;
  word room_in_queue;

  /* Output string, char by char (as done above) until NULL char.        */

  sio_error = AOK;               /* Assume function successful...        */

  #if defined(I8031)

  if (port == ASC)
  {
    /* Wait until there's enough room for string to be inserted into queue. */

    do {
      if (xmt_buffer.front <= xmt_buffer.rear)
        room_in_queue = MAX_BUFFER_SIZE - xmt_buffer.rear + xmt_buffer.front;
      else
        room_in_queue = xmt_buffer.front - xmt_buffer.rear - 1;
    } while (room_in_queue < strlen((char*)s));

    /* Copy output string to transmit queue (save original string ptr).     */

    stemp = s;
    while(*s)
    {
      xmt_buffer.item[xmt_buffer.rear] = *s++;
      buf_inc(xmt_buffer.rear);
    }
```



```
    /* If transmit buffer was empty, move byte to transmitter (gen INT).     */
    /* Buffer can't be empty since we just added a byte.                     */

    if (xmt_buffer.empty)
    {
      xmt_buffer.empty = FALSE;
      MUART_TBR = *stemp;
    }
    else
      xmt_buffer.empty = FALSE;
    return;
  }

  #endif

  while(*s) sio_putbyte(port, *s++);
}


/**********************************************************************\
*                              sio_getstr                              *
************************************************************************
*                                                                      *
* Function:     Inputs a NULL terminated string from the serial port.  *
*                                                                      *
* Input:        sio_getstr(                                            *
*                 int port; serial port number (identifier).           *
*                 char *s;  pointer to string where bytes will be placed. *
*               );                                                     *
*                                                                      *
* Output:       Length of input string as an integer.                  *
*                                                                      *
* Globals:      sio_error : module SIO.C                               *
*                                                                      *
* Edit History: 10/10/86 - Written by Robin T. Laird.                  *
*                                                                      *
\**********************************************************************/

int sio_getstr(port, s)
int  port;
byte *s;
{
int i;

  /* Input chars (as done above) until we hit a NULL char.               */
  /* Stuff chars into s, using s as string pointer.                      */
  /* Keep track of string length in i, return i.                         */

  sio_error = AOK;               /* Assume function successful...        */

  i = 0;
  do {
    *s = sio_getbyte(port);
    i++;
  } while(*s++);
  return(i);
}


/**********************************************************************\
*                              sio_putnstr                             *
************************************************************************
*                                                                      *
* Function:     Outputs a (byte) string of numbytes in length.         *
*               Sets sio_error to SIO_ERR_STRING_LENGTH if the number of *
*               bytes to send is less than or equal to 0.              *
*                                                                      *
* Input:        sio_putnstr(                                           *
*                 int port;      serial port number (identifier).      *
*                 int numbytes;  number of bytes to output (numbytes > 0). *
*                 byte *s;       pointer to (byte) string to output.   *
*                                                                      *
* Output:       Nothing.                                               *
```

```
 *
 * Globals:      sio_error : module SIO.C                                  *
 *                                                                         *
 * Edit History: 06/19/90 - Written by Robin T. Laird.                     *
 *                                                                         *
\**************************************************************************/

void sio_putnstr(port, numbytes, s)
int  port;
int  numbytes;
byte *s;
{
  byte *stemp;
  word room_in_queue;

  /* Check the length of the string and indicate if invalid.              */

  if (numbytes <= 0)
  {
    sio_error = SIO_ERR_STRING_LENGTH;
  }
  else
  {
    sio_error = AOK;

    #if defined(I8031)

    if (port == ASC)
    {
      /* Wait until there's enough room for string to be inserted into que. */

      do {
        if (xmt_buffer.front <= xmt_buffer.rear)
          room_in_queue = MAX_BUFFER_SIZE - xmt_buffer.rear + xmt_buffer.front;
        else
          room_in_queue = xmt_buffer.front - xmt_buffer.rear - 1;
      } while (room_in_queue < numbytes);

      /* Copy output string to transmit queue (save original string ptr).   */

      stemp = s;
      while(numbytes--)
      {
        xmt_buffer.item[xmt_buffer.rear] = *s++;
        buf_inc(xmt_buffer.rear);
      }

      /* If transmit buffer was empty, move byte to transmitter (gen INT).  */
      /* Buffer can't be empty since we just added a byte.                  */

      if (xmt_buffer.empty)
      {
        xmt_buffer.empty = FALSE;
        MUART_TBR = *stemp;
      }
      else
        xmt_buffer.empty = FALSE;
      return;
    }

    #endif

    while (numbytes--) sio_putbyte(port, *s++);
  }
}

/**************************************************************************\
 *                              sio_8256_int                               *
 ***************************************************************************
 *                                                                         *
 * Function:     Handles transmit/receive serial interrupts from the 8256. *
 *               Assumes the use of the CP-31/535, and that the serial rcv *
```



```
 *               and xmt interrupts of the 8256 peripheral IC (level L4/L5) *
 *               are connected to the INT0 interrupt input of the 80535.   *
 *               (This requires jumpering the bottom two pins of W3 on the *
 *               CP-31/535 which connects 8256 INT to 80535 EXT INT0 input).*
 *                                                                         *
 *               Serial INPUT interrupts (for chars being received) move   *
 *               incoming data to a global queue where it can be removed by *
 *               the other serial I/O routines (e.g., sio_getbyte()).      *
 *                                                                         *
 *               Serial OUTPUT interrupts (for chars being transmitted) move *
 *               outgoing data from a global queue that is filled by the   *
 *               other serial I/O routines (e.g., sio_putbyte()).          *
 *                                                                         *
 *               Input (RCV) interrupts are distinguished from output (XMT) *
 *               interrupts by reading the 8256 interrupt address register *
 *               which holds a value equal to the interrupt level * 4.     *
 *                                                                         *
 * Input:        sio_8256_int();                                           *
 *                                                                         *
 * Output:       Nothing.                                                  *
 *                                                                         *
 * Globals:      sio_error   : module SIO.C                                *
 *               sio_in_que  : module SIO.C                                *
 *               sio_out_que : module SIO.C                                *
 *                                                                         *
 * Edit History: 05/21/91 - Written by Robin T. Laird.                     *
 *                                                                         *
\**************************************************************************/

#if defined(I8031)

static void sio_8256_int() interrupt 0 using 2
{
  static byte iar;

  /* Get interrupt address register.                                      */

  iar = MUART_IAR;

  /* Check for receive (input) interrupt.                                 */
  /* Reading IAR clears int and indicates INT ACK to int controller.      */

  if (iar == MUART_RCV_INT_MASK)
  {
    /* Make sure receive buffer is not full.                              */
    /* Move byte from input register to receive buffer.                   */
    /* Adjust buffer rear index.                                          */
    /* See if buffer is full.                                             */
    /* Mark buffer as not empty since we just received something.         */

    if (!rcv_buffer.full)
    {
      rcv_buffer.item[rcv_buffer.rear] = MUART_RBR;
      buf_inc(rcv_buffer.rear);
      if (rcv_buffer.front == rcv_buffer.rear) rcv_buffer.full = TRUE;
      rcv_buffer.empty = FALSE;
    }
  }

  /* Check for transmit (output) interrupt.                               */

  else if (iar == MUART_XMT_INT_MASK)
  {
    /* Mark buffer as not full since we just removed an item.             */
    /* Byte has been transmitted so adjust front index accordingly.       */
    /* Check to see if buffer is empty. Set empty flag if so.             */
    /* If not empty, move byte from front of buffer to transmit register. */

    if (!xmt_buffer.empty)
    {
      xmt_buffer.full = FALSE;
      buf_inc(xmt_buffer.front);
      if (xmt_buffer.front == xmt_buffer.rear)
```

```
        xmt_buffer.empty = TRUE;
      else
        MUART_TBR = xmt_buffer.item[xmt_buffer.front];
    }
  }

  /* Issue End-of-Interrupt (EOI).                                    */

  MUART_CMD_3 = MUART_EOI;
}

#endif
```

```c
/**************************************************************************\
*                               DEBUG.H                                    *
****************************************************************************
*                                                                          *
* CPCI:          IED90-MRA-COM-HDR-DEBUG-H-ROCO                            *
*                                                                          *
* Description:   Global debug definitions/declarations.                    *
*                Includes headers required to output debug messages to the *
*                local serial port (or standard output device) of the target *
*                system.                                                   *
*                                                                          *
* Notes:         1) All modules should include this file for debugging.    *
*                2) XDATA modifier not required for Microsoft C (MSC).     *
*                                                                          *
* Edit History: 03/22/91 - Written by Robin T. Laird.                      *
*                                                                          *
\**************************************************************************/

#if !defined(IBMAT)
#include <reg51.h>
#endif
#include <stdio.h>
#include <sio.h>

static XDATA char spf[80];            /* Buffer for sprintf() calls.      */
```

```
/**************************************************************************\
*                                SYSDEFS.H                                 *
****************************************************************************
*                                                                          *
* CPCI:          IED90-MRA-COM-HDR-SYSDEFS-H-R0C0                          *
*                                                                          *
* Description:   Global system definitions/declarations.                   *
*                Defines absolute and common values used by all modules.   *
*                                                                          *
* Notes:         1) All modules should include this file.                  *
*                2) XDATA modifier not required for Microsoft C (MSC).     *
*                                                                          *
* Edit History: 07/07/90 - Written by Robin T. Laird.                      *
*                                                                          *
\**************************************************************************/

#ifndef SYS_MODULE_CODE
#define SYS_MODULE_CODE          0

#define AOK                      1

#define TRUE                     1
#define FALSE                    0

#define YES                      1
#define NO                       0

#if !defined(NULL)
#define NULL                     0
#endif

#if !defined(NULLPTR)
#define NULLPTR                  ((char*)0)
#endif

#define SYS_MAX_PACKET_SIZE      256

#if defined(MSDOS)
#define XDATA
#else
#define XDATA                    xdata
#endif

typedef unsigned char byte;
typedef unsigned int word;

#endif
```

```
#*********************************************************************#
#                               MAKEFILE                              #
#*********************************************************************#
#                                                                     #
# CPCI:          IED90-MRA-COM-LCS-MAKEFILE-TXT-R0C0                  #
#                                                                     #
# Description:   Makefile for the Modular Robotic Architecture (MRA). #
#                Makes the common local communications subsystem.     #
#                                                                     #
#                Targets are available for the following systems/subsystems: #
#                                                                     #
#                lcs   - COM Local Communications Subsystem           #
#                lib   - Add modules to MRA library                   #
#                print - Print COM local communications files         #
#                                                                     #
# Notes:         1) The dependency and production rules are included here. #
#                2) See also \mra\makefile.                           #
#                                                                     #
# Edit History: [illegible]/91 - Written by Robin T. Laird.           #
#                                                                     #
#*********************************************************************#



##################################### RULES ##################################

.SUFFIXES : .hex .exe .obj .c .a51

# Control settings for Franklin 8031 development

CC          =c51
AS          =a51
LINK        =l51
OTOH        =oh51
CFLAGS      =cd la db sb
ASFLAGS     =
LFLAGS      =
OFLAGS      =
STARTUP     =\c51\crom.obj
CODESEG     =00000h
XDATASEG    =00000h

# Control settings for Microsoft MS-DOS development

MSC         =cl
MSAS        =masm
MSLINK      =link
MSCFLAGS    =/AL /c /Oi /Zi /Od
MSASFLAGS   =
MSLNKFLGS   =/co
LOADLIBES   =

.c.obj :
        $(CC) $< $(CFLAGS)

.a51.obj :
        $(AS) $< $(ASFLAGS)

.obj.exe :
        $(LINK) $(STARTUP),$< TO $@ code ($(CODESEG)) xdata ($(XDATASEG)) ixref

.exe.hex :
        $(OTOH) $< $(OFLAGS)



########################### DEFINITIONS ######################################

# Project, system, and application level definitions

PROJ          = mra
APPSYS        = app
COMSYS        = com
```



```
ICNSYS          = icn
MPUSYS          = mpu

COMLIB          = \$(PROJ)\lib
COMSRC          = \$(PROJ)\$(COMSYS)\src
COMBIN31        = \$(PROJ)\$(COMSYS)\bin\8031
COMBIN152       = \$(PROJ)\$(COMSYS)\bin\80152
COMBINMS        = \$(PROJ)\$(COMSYS)\bin\msdos
COMBINSBC8      = \$(PROJ)\$(COMSYS)\bin\sbc8

# Common subsystem level source directories

HDRSRC          = $(COMSRC)\hdr
LCSSRC          = $(COMSRC)\lcs

# Common subsystem global include and compilation units

SYSDEFS         = $(HDRSRC)\sysdefs.h

LCS             = $(COMBIN152)\lcd.obj        $(COMBIN152)\lci.obj   \
                  $(COMBIN31)\lcd.obj         $(COMBIN31)\lci.obj    \
                  $(COMBINMS)\lcd.obj         $(COMBINMS)\lci.obj



################################### TARGETS ##################################

lcs             : $(LCS)

lib             : $(LCS)
                  -lib51 delete $(COMLIB)\mra_1521.lib (lcd, lci)
                  -lib51 add $(COMBIN152)\lcd.obj to $(COMLIB)\mra_1521.lib
                  -lib51 add $(COMBIN152)\lci.obj to $(COMLIB)\mra_1521.lib
                  -lib51 delete $(COMLIB)\mra_311.lib (lcd, lci)
                  -lib51 add $(COMBIN31)\lcd.obj to $(COMLIB)\mra_311.lib
                  -lib51 add $(COMBIN31)\lci.obj to $(COMLIB)\mra_311.lib
                  -lib $(COMLIB)\mra_mss.lib -lcd.obj+$(COMBINMS)\lcd.obj;
                  -lib $(COMLIB)\mra_msl.lib -lcd.obj+$(COMBINMS)\lcd.obj;
                  -lib $(COMLIB)\mra_mss.lib -lci.obj+$(COMBINMS)\lci.obj;
                  -lib $(COMLIB)\mra_msl.lib -lci.obj+$(COMBINMS)\lci.obj;
                  touch lib

print:            $(LCS)
                  $-a2ps -nf lcd.h | post
                  $-a2ps -nf lcd.c | post
                  $-a2ps -nf lci.h | post
                  $-a2ps -nf lci.c | post
                  touch print



############################ COM LCS DEPENDENCIES ############################

# COM Local Communications Device module dependencies for 80152, 8031, MS-DOS

LCD = $(SYSDEFS)          $(LCSSRC)\bmf.h        $(LCSSRC)\bmf.c      \
      $(LCSSRC)\lcc.h     $(LCSSRC)\lcc.c        $(LCSSRC)\lcd.h      \
      $(LCSSRC)\lcd.c

$(COMBIN152)\lcd.obj             : $(LCD)
  $(CC) $(LCSSRC)\$*.c $(CFLAGS) df(I80152) pr($(LCSSRC)\$*.152) oj($(COMBIN152

$(COMBIN31)\lcd.obj              : $(LCD)
  $(CC) $(LCSSRC)\$*.c $(CFLAGS) df(I8031)  pr($(LCSSRC)\$*.31)  oj($(COMBIN31)

$(COMBINMS)\lcd.obj              : $(LCD)
  $(MSC) $(MSCFLAGS) /DIBMAT /Fs$(LCSSRC)\$*.at /Fo$(COMBINMS)\$* $(LCSSRC)\$*.


# COM Local Communications Interface module dependencies for 8031, MS-DOS

LCI = $(SYSDEFS) $(LCSSRC)\lcd.h $(LCSSRC)\lci.h $(LCSSRC)\lci.c
```

```
$(COMBIN152)\lci.obj              : $(LCI)
  $(CC) $(LCSSRC)\$*.c $(CFLAGS) df(180152) pr($(LCSSRC)\$*.152) oj($(COMBIN152

$(COMBIN31)\lci.obj               : $(LCI)
  $(CC) $(LCSSRC)\$*.c $(CFLAGS) df(18031) pr($(LCSSRC)\$*.31) oj($(COMBIN31)\$

$(COMBINMS)\lci.obj               : $(LCI)
  $(MSC) $(MSCFLAGS) /DIBMAT /Fa$(LCSSRC)\$*.at /Fo$(COMBINMS)\$* $(LCSSRC)\$*.
```

```c
/**************************************************************************\
*                                  BMF.H                                   *
****************************************************************************
*                                                                          *
* CPCI:          IED90-MRA-COM-LCS-BMF-H-ROCO                              *
*                                                                          *
* Description:   Frame Buffer Management variables and definitions.        *
*                Contains constant declarations and type definitions for the *
*                circular packet buffer management functions. Used by the  *
*                LCS communications device handler module.                 *
*                                                                          *
*                Module BMF exports the following types/functions:         *
*                                                                          *
*                typedef buffer;                                           *
*                                                                          *
*                buf_full();                                               *
*                buf_empty();                                              *
*                buf_clear();                                              *
*                buf_insert();                                             *
*                buf_remove();                                             *
*                                                                          *
* Notes:         1) This file is included by LCD.C.                        *
*                2) Module BMF requires type definitions from LCD.H.       *
*                                                                          *
* Edit History: 06/28/90 - Written by Robin T. Laird.                      *
*                                                                          *
\**************************************************************************/

/* Private Data Structures:                                                */

#ifndef BMF_MODULE_CODE
#define BMF_MODULE_CODE           3100

#define ERR_FULL_BUFFER           1+BMF_MODULE_CODE
#define ERR_EMPTY_BUFFER          2+BMF_MODULE_CODE

#define ERR_BOP_NOT_FOUND         3+BMF_MODULE_CODE
#define ERR_CRC_INVALID           4+BMF_MODULE_CODE
#define ERR_INVALID_LENGTH        5+BMF_MODULE_CODE

/* MAX_BUFFER_SIZE defines the number of bytes/buffer.                     */

#define MAX_BUFFER_SIZE           1024

/* Circular buffer (queue) to hold incoming/outgoing data packets.         */
/* Must be initialized using buf_clear().                                  */

typedef struct { byte item[MAX_BUFFER_SIZE];
                 word front;
                 word rear;
                 byte empty;
                 byte full;
               } buffer;

/* Private Functions:                                                      */

static int buf_full(buffer *b);
static int buf_empty(buffer *b);
static void buf_clear(buffer *b);
static void buf_insert(buffer *b, lcd_packet p);
static void buf_remove(buffer *b, lcd_packet p);

#endif
```

```
/***********************************************************************\
*                                BMF.C                                  *
*************************************************************************
*                                                                       *
* CPCI:          IED90-MRA-COM-LCS-BMF-C-ROC2                          *
*                                                                       *
* Description:   Frame Buffer Management functions.                     *
*                Contains functions for initializing and managing the   *
*                circular packet buffers for the LCS local communications *
*                device handler.                                        *
*                                                                       *
*                Module BMF exports the following functions:            *
*                                                                       *
*                buf_front()  macro;                                    *
*                buf_rear()   macro;                                    *
*                buf_inc()    macro;                                    *
*                buf_full();                                            *
*                buf_empty();                                           *
*                buf_clear();                                           *
*                buf_insert();                                          *
*                buf_remove();                                          *
*                                                                       *
* Notes:         1) This module is NOT a stand-alone compilation unit.  *
*                   It is included by the module LCD.C and is compiled there. *
*                   It is assumed that the file LCD.H is included before it. *
*                   Note that all of the functions herein are static.   *
*                                                                       *
* Edit History: 08/14/90 - Written by Richard P. Smurlo and Robin T. Laird. *
*                                                                       *
\***********************************************************************/

/* Private Variables:                                                   */

/* Declarations for the module circular receive and transmit buffers.   */
/* These are global to the LCC.C module and to the LCD.C module.        */

static XDATA buffer xmt_buffer;
static XDATA buffer rcv_buffer;

/* The packet field lengths below are given in number of bytes.         */

#define BOP_LENGTH              3              /* Beginning-of-packet len. */
#define ADDR_LENGTH             1              /* Source address len.    */
#define LENGTH_LENGTH           1              /* Packet length len.     */
#define CRC_LENGTH              2              /* Checksum (CRC) len.    */

/* Packet overhead is number of bytes added to packet before it is sent. */
/* This includes the beginning-of-packet (BOP) and the CRC.             */
/* The BMF software adds the BOP flag (3 bytes) and the CRC (2 bytes).  */

#define PACKET_OVERHEAD         (BOP_LENGTH+CRC_LENGTH)

/* The minimum number of bytes for a valid packet (as seen from this level) */
/* is the length of the packet overhead plus the length of the (source) */
/* address field and the length field.                                  */

#define MIN_BYTES_IN_PACKET     (PACKET_OVERHEAD+ADDR_LENGTH+LENGTH_LENGTH)

/* BOP is defined as a "unique" sequence of bits that precedes each packet. */

static byte BOP[] = {0xAA, 0xAA, 0xAA};        /* 1010 1010 1010         */

/***********************************************************************\
*                              buf_front                                *
*************************************************************************
*                                                                       *
* Function:      Macro that returns a pointer to the front (current)    *
*                element in the parameter buffer. Used to obtain the    *
*                address of the next packet to be received/transmitted. *
*                                                                       *
* Input:         buf_front(                                             *
*                  buffer b; buffer structure (NOT a pointer to it).    *
```



```
*                );                                                     *
*                                                                       *
* Output:        Pointer to front (current) packet in the buffer.       *
*                                                                       *
* Globals:       None.                                                  *
*                                                                       *
* Edit History: 07/08/90 - Written by Robin T. Laird.                   *
*                                                                       *
\***********************************************************************/

#define buf_front(b)  (&b.item[b.front])


/***********************************************************************\
*                              buf_rear                                 *
*************************************************************************
*                                                                       *
* Function:      Macro that returns a pointer to the rear (last)        *
*                element in the parameter buffer. Used to obtain the    *
*                address of the next packet to be received/transmitted. *
*                                                                       *
* Input:         buf_rear(                                              *
*                  buffer b; buffer structure (NOT a pointer to it);    *
*                );                                                     *
*                                                                       *
* Output:        Pointer to rear (last) packet in the buffer.           *
*                                                                       *
* Globals:       None.                                                  *
*                                                                       *
* Edit History: 07/10/90 - Written by Robin T. Laird.                   *
*                                                                       *
\***********************************************************************/

#define buf_rear(b)  (&b.item[b.rear])


/***********************************************************************\
*                              buf_inc                                  *
*************************************************************************
*                                                                       *
* Function:      Macro that increments either the front or rear index   *
*                of a circular buffer of max size MAX_BUFFER_SIZE.      *
*                Should be used with care, i.e., not nested within a    *
*                compound statement like if ... then ... else.          *
*                                                                       *
* Input:         buf_inc(                                               *
*                  word x; index to increment.                          *
*                );                                                     *
*                                                                       *
* Output:        Nothing.                                               *
*                                                                       *
* Globals:       None.                                                  *
*                                                                       *
* Edit History: 12/04/90 - Written by Robin T. Laird.                   *
*                                                                       *
\***********************************************************************/

#define buf_inc(x)  (x = ((x == MAX_BUFFER_SIZE-1) ? 0 : x+1))


/***********************************************************************\
*                              buf_full                                 *
*************************************************************************
*                                                                       *
* Function:      Function that returns the boolean of whether or not the *
*                parameter buffer is full (TRUE if so, FALSE if not).   *
*                                                                       *
* Input:         buf_full(                                              *
*                  buffer *b; pointer to buffer structure.              *
*                );                                                     *
*                                                                       *
* Output:        Integer, TRUE if buffer FULL, FALSE if buffer not FULL. *
*                                                                       *
```

```
 * Globals:      None.                                                  *
 *                                                                      *
 * Edit History: 07/08/90 - Written by Robin T. Laird.                  *
 *                                                                      *
\************************************************************************/

static int buf_full(b)
buffer *b;
{
  lcd_error = AOK;                           /* Assume function successful...    */
  return(b->full);
}


/************************************************************************\
 *                              buf_empty                               *
 ************************************************************************
 *                                                                      *
 * Function:     Function that returns the boolean of whether or not the *
 *               parameter buffer is empty (TRUE if so, FALSE if not).  *
 *               Operation depends upon the parameter buffer as follows: *
 *                                                                      *
 *               If buffer b == xmt_buffer then just returns empty flag. *
 *               If buffer b == rcv_buffer then returns whether packet avail. *
 *                                                                      *
 *               The receive buffer is considered empty if either 1) there *
 *               are not enough bytes to determine the length of the incoming *
 *               packet, or 2) the number of bytes received is less than the *
 *               number indicated by the packet length byte (which HAS been *
 *               received).                                             *
 *                                                                      *
 *               Note that this routine advances the front buffer pointer to *
 *               the beginning of a valid packet. It MUST be called before a *
 *               packet is actually removed from the buffer.            *
 *                                                                      *
 * Input:        buf_empty(                                             *
 *                 buffer *b; pointer to buffer structure.              *
 *               );                                                     *
 *                                                                      *
 * Output:       Integer, TRUE if buffer EMPTY, FALSE if buffer not EMPTY. *
 *                                                                      *
 * Globals:      None.                                                  *
 *                                                                      *
 * Edit History: 12/10/90 - Written by Richard P. Smurlo and Robin T. Laird. *
 *               03/08/91 - Fixed bug when first packet bad, Robin T. Laird. *
 *                                                                      *
\************************************************************************/

static int buf_empty(b)
buffer *b;
{
  int  bopix;                                /* BOP byte sequence index.         */
  int  bfull;                                /* Buffer full flag.                */
  word bytes_in_buff;                        /* Num bytes currently in buffer.   */
  word packet_length;                        /* Actual calculated packet length. */
  word bfront, brear;                        /* Static buffer front and rear.    */

  lcd_error = AOK;                           /* Assume function successful...    */

  /* Just return structure flag for transmit buffer.                             */
  /* Otherwise, determine if there are enough bytes for an entire packet.        */

  if (b == &xmt_buffer)
    return(xmt_buffer.empty);
  else
  {
    /* Quick check to see if buffer REALLY empty.                                */

    if ((b->front == b->rear) && !b->full) return(TRUE);

    /* Set local buffer front, rear, full variables.                             */
    /* Interrupts are happening, we don't want our vars changing as we go.       */

```



```
    bfront = b->front;
    brear  = b->rear;
    bfull  = b->full;

    /* Calculate the number of bytes currently in the buffer.                    */
    /* Num bytes in buffer must be > minimum bytes for a valid packet.           */

    if (bfull)
      bytes_in_buff = MAX_BUFFER_SIZE;
    else if (bfront < brear)
      bytes_in_buff = brear - bfront;
    else
      bytes_in_buff = brear + MAX_BUFFER_SIZE - bfront;

    /* If there aren't enough bytes, report AOK, return buffer empty.            */

    if (bytes_in_buff < MIN_BYTES_IN_PACKET)
    {
      return(TRUE);
    }
    else
    {
      /* We have enough bytes to VERIFY if we have a valid packet.               */
      /* Find the BOP sequence in the buffer (may have a bad packet).            */
      /* If we hit buffer end, report BOP not found, return buffer empty.        */
      /* Also, buffer is actually empty so make front=rear and full=FALSE.       */

      bopix = 0;
      while (bopix < BOP_LENGTH)
      {
        if (b->item[bfront] == BOP[bopix])
          bopix++;
        else
           bopix = 0;
        buf_inc(bfront);
        if ((bfront == brear) && (bopix < BOP_LENGTH))
        {
          lcd_error = ERR_BOP_NOT_FOUND;
          b->front = bfront;
          b->full  = FALSE;
          return(TRUE);
        }
      }

      /* Must have a valid BOP and bfront points to byte after BOP.              */
      /* Move actual buffer front (b->front) to beginning of BOP.                */

      b->front = bfront;
      for (bopix = 0; bopix < BOP_LENGTH; bopix++)
      {
        if (b->front == 0)
          b->front = MAX_BUFFER_SIZE - 1;
        else
          b->front--;
      }

      /* Now recalculate number of bytes in buffer to see if packet valid.       */
      /* Must add BOP_LENGTH since bfront points to byte past BOP.               */

      if (bfront < brear)
        bytes_in_buff = brear - bfront + BOP_LENGTH;
      else
        bytes_in_buff = brear + MAX_BUFFER_SIZE - bfront + BOP_LENGTH;

      /* If there aren't enough bytes, report AOK, return buffer empty.          */

      if (bytes_in_buff < MIN_BYTES_IN_PACKET)
      {
        return(TRUE);
      }
      else
      {
        /* If we reach here...                                                   */
```

```
        /*   b->front points to beginning of BOP.                            */
        /*   bfront points to first byte in packet (past BOP).               */
        /*   bytes_in_buff holds num bytes currently in buffer.              */
        /*   There are enough bytes in buffer to VERIFY valid packet.        */

        /* Retrieve length byte of packet.                                   */
        /* Length byte is at LCD_LEN_POS bytes past first byte in packet.    */
        /* Routine FAILS if packet length byte is invalid (sometimes).       */
        /* Result depends on if bad packet length > num bytes in buffer.     */

        if (bfront < MAX_BUFFER_SIZE - LCD_LEN_POS)
          packet_length = b->item[bfront + LCD_LEN_POS];
        else
          packet_length = b->item[bfront + LCD_LEN_POS - MAX_BUFFER_SIZE];

        /* Packet INVALID if num bytes in buff < packet_length + overhead.   */
        /* Return buffer empty if so, otherwise return buffer NOT empty.     */

        if (bytes_in_buff < packet_length + PACKET_OVERHEAD)
          return(TRUE);
        else
          return(FALSE);
      }
    }
  }
}


/***************************************************************************\
*                                buf_clear                                  *
*****************************************************************************
*                                                                           *
* Function:     Initializes the parameter buffer and clears its contents.  *
*               The front and rear pointers are reset and the boolean      *
*               state flags (i.e., empty, full) are set accordingly.       *
*                                                                           *
* Input:        buf_clear(                                                  *
*                 buffer *b; pointer to buffer structure to clear.         *
*               );                                                          *
*                                                                           *
* Output:       Nothing.                                                    *
*                                                                           *
* Globals:      lcd_error : module LCD.C                                    *
*                                                                           *
* Edit History: 07/09/90 - Written by Robin T. Laird.                       *
*                                                                           *
\***************************************************************************/

static void buf_clear(b)
buffer *b;
{
  word i;

  lcd_error = AOK;                        /* Assume function successful...  */

  /* Process each of the packets in the buffer.                             */
  /* Set all bytes in packet to 0.                                          */

  for (i = 0; i < MAX_BUFFER_SIZE; i++)
  {
    b->item[i] = 0x00;
  }

  /* Set the front and rear indexes equal to indicate empty buffer.         */
  /* Set the empty and full flags as appropriate.                           */

  b->front = b->rear = 0;
  b->empty = TRUE;
  b->full  = FALSE;
}


/***************************************************************************\
```



```
*                                buf_insert                                 *
*****************************************************************************
*                                                                           *
* Function:     Inserts a packet into the parameter buffer if room.        *
*               An error is returned if the buffer is already full.        *
*               Inserts beginning-of-packet header and CRC trailer.        *
*                                                                           *
* Input:        buf_insert(                                                 *
*                 buffer *b;    pointer to buffer structure to clear.      *
*                 lcd_packet p; packet to be inserted.                     *
*               );                                                          *
*                                                                           *
* Output:       Nothing.                                                    *
*                                                                           *
* Globals:      lcd_error : module LCD.C                                    *
*                                                                           *
* Edit History: 07/09/90 - Written by Robin T. Laird.                       *
*                                                                           *
\***************************************************************************/

static void buf_insert(b, p)
buffer *b;
lcd_packet p;
{
  word i, crc, length, size;

  lcd_error = AOK;                        /* Assume function successful...  */

  /* Make sure buffer isn't already full.                                   */

  if (b->full)
  {
    lcd_error = ERR_FULL_BUFFER;
    return;
  }

  /* Calculate size and remaining space in buffer to see if packet will fit.*/
  /* If there's not enough room, report buffer full and return.             */
  /* Length of packet is stored at LCD_LEN_POS in packet data.              */

  if (b->front <= b->rear)
    size = b->rear - b->front;
  else
    size = b->rear + MAX_BUFFER_SIZE - b->front + 1;

  length = p[LCD_LEN_POS];
  if (length + PACKET_OVERHEAD > MAX_BUFFER_SIZE - size)
  {
    lcd_error = ERR_FULL_BUFFER;
    return;
  }

  /* Insert beginning of packet (BOP) header into buffer.                   */

  for (i = 0; i < BOP_LENGTH; i++)
  {
    b->item[b->rear] = BOP[i];
    buf_inc(b->rear);
  }

  /* Copy element into buffer and set appropriate structure fields.         */
  /* Adjust rear index (MAX_BUFFER_SIZE-1 is last element in buffer).       */

  for (i = 0; i < length; i++)
  {
    b->item[b->rear] = p[i];
    buf_inc(b->rear);
  }

  /* Calculate CRC checksum and insert into buffer.                         */

  crc = lcc_crc16(p, length);
```

```
  b->item[b->rear] = crc >> 8;
  buf_inc(b->rear);
  b->item[b->rear] = crc & 0x00FF;
  buf_inc(b->rear);

  /* Check and see if we've filled the buffer (set full flag if so).      */
  /* Set empty flag to FALSE since we just inserted.                      */

  if ((b->front == b->rear) && !b->empty) b->full = TRUE;
  b->empty = FALSE;
}


/***********************************************************************\
*                              buf_remove                               *
*************************************************************************
*                                                                       *
* Function:     Removes a packet from the parameter buffer if available.*
*               An error is returned if the buffer is already empty.    *
*               The beginning-of-packet header and CRC trailer are removed
*               before the data packet is returned.                     *
*                                                                       *
* Input:        buf_remove(                                             *
*                 buffer *b;     pointer to buffer structure to clear.  *
*                 lcd_packet p; packet to be removed.                   *
*               );                                                      *
*                                                                       *
* Output:       Nothing.                                                *
*                                                                       *
* Globals:      lcd_error : module LCD.C                                *
*                                                                       *
* Edit History: 07/09/90 - Written by Robin T. Laird.                   *
*               03/08/91 - Compatible with new buf_empty(), Robin T. Laird.
*                                                                       *
\***********************************************************************/

static void buf_remove(b, p)
buffer *b;
lcd_packet p;
{
  word i, length;
  word crc_calc, crc_rcvd, crc_hi, crc_lo;

  lcd_error = AOK;                        /* Assume function successful...  */

  /* Make sure buffer isn't already empty.                                */
  /* After buf_empty() call, b->front will point to beginning of BOP.     */

  if (buf_empty(b))
  {
    lcd_error = ERR_EMPTY_BUFFER;
    return;
  }

  /* Valid packet in buffer (buffer not empty), proceed to remove.        */
  /* Discard and skip past BOP.                                           */

  for (i = 0; i < BOP_LENGTH; i++) buf_inc(b->front);

  /* Calculate the length of the packet.                                  */

  if (b->front < MAX_BUFFER_SIZE - LCD_LEN_POS)
    length = b->item[b->front + LCD_LEN_POS];
  else
    length = b->item[b->front + LCD_LEN_POS - MAX_BUFFER_SIZE];

  /* Copy length number of bytes.                                         */

  for (i = 0; i < length; i++)
  {
     p[i] = b->item[b->front];
     buf_inc(b->front);
  }
```



```

  /* Strip CRC off the returned packet.                                   */

  crc_hi = b->item[b->front];
  buf_inc(b->front);

  crc_lo = b->item[b->front];
  buf_inc(b->front);

  /* Calculate CRC based on data received.                                */
  /* Compare calculated CRC against CRC received.                         */
  /* If they don't match, report error as invalid CRC.                    */

  crc_rcvd = (crc_hi << 8) | crc_lo;
  crc_calc = lcc_crc16(p, length);

  if (crc_calc != crc_rcvd)
  {
    lcd_error = ERR_CRC_INVALID;
    return;
  }

  /* Buffer can't be full since we just removed a packet.                 */

  b->full = FALSE;

  /* Check and see if we've depleted the buffer (set empty flag if so).   */

  if (b->front == b->rear) b->empty = TRUE;
}
```

```c
/*********************************************************************\
 *                               LCC.H                                *
 **********************************************************************
 *                                                                    *
 * CPCI:          IED90-MRA-COM-LCS-LCC-H-ROCO                        *
 *                                                                    *
 * Description:   Local Communications Channel (LCC) variables and functions. *
 *                Contains constant function parameter and hardware register *
 *                declarations for initializatin and control of the Intel *
 *                8031 LCC. Implements the low-level data link layer portion *
 *                of the MRA LCS communications device handler functions. *
 *                                                                    *
 *                Module LCC exports the following functions:         *
 *                                                                    *
 *                lcc_mode();                                         *
 *                lcc_baud();                                         *
 *                lcc_crc16();                                        *
 *                lcc_sys_init();                                     *
 *                lcc_set_rcv_dst();                                  *
 *                lcc_set_xmt_src();                                  *
 *                lcc_start_rcv();                                    *
 *                lcc_start_xmt();                                    *
 *                lcc_stop_rcv();                                     *
 *                lcc_stop_xmt();                                     *
 *                lcc_enable_interrupts();                            *
 *                lcc_sio_int()  interrupt;                           *
 *                lcc_sio1_int() interrupt;                           *
 *                lcc_sio2_int() interrupt;                           *
 *                lcc_sio3_int() interrupt;                           *
 *                lcc_sio4_int() interrupt;                           *
 *                lcc_pio_int()  interrupt;                           *
 *                                                                    *
 * Notes:         1) See the Intel 8-Bit Embedded Controller Handbook for *
 *                   more information (No. 270645-002, pp. 7-11 - 7-21). *
 *                                                                    *
 * Edit History: 08/14/90 - Written by Robin T. Laird and Richard P. Smurlo. *
 *                                                                    *
\*********************************************************************/

/* Private Data Structures:                                           */

#ifndef LCC_MODULE_CODE
#define LCC_MODULE_CODE          3200

#define ERR_IO_MODE              1+LCC_MODULE_CODE
#define ERR_BAUD_RATE            2+LCC_MODULE_CODE
#define ERR_RFIFO_NOT_CLEAR      3+LCD_MODULE_CODE

/* I/O mode settings (specifies serial or parallel channel operation). */

#define LCC_SIO_MODE             0
#define LCC_PIO_MODE             1

/* Serial I/O Constants:                                              */

/* I80152 signifies baud rate settings for 14.7456 MHz CPU.           */
/* I8031  signifies baud rate settings for 11.0592 MHz CPU.           */
/* IBMAT signifies settings for the IBM-AT (8250).                    */

#if defined(I80152)
#define MAX_BAUD_RATE            0xFF    /* 38.4 KBPS (SMOD=0).       */
#define BR19200                  0xFE
#define BR9600                   0xFC
#define BR4800                   0xF8
#define BR2400                   0xF0
#define BR1200                   0xE0
#define BR600                    0xC0
#define BR300                    0x80

#elif defined(I8031)
#define MAX_BAUD_RATE            0xFF    /* 57.6 KBPS (SMOD=1).       */
#define BR19200                  0xFD
#define BR9600                   0xFA
```



```c
#define BR4800                   0xF4
#define BR2400                   0xE8
#define BR1200                   0xD0
#define BR600                    0xA0
#define BR300                    0x40

#elif defined(IBMAT)
#define BR38400                  3       /* Baud rate divisor settings.   */
#define BR19200                  6
#define BR9600                   12
#define BR4800                   24
#define BR2400                   48
#define BR1200                   96
#define BR600                    192
#define BR300                    384

#define PEVEN                    0x18    /* Parity settings.              */
#define PODD                     0x08
#define PNONE                    0x00

#define WL5                      0x00    /* Word length settings.         */
#define WL6                      0x01
#define WL7                      0x02
#define WL8                      0x03

#define SB1                      0x00    /* Stop bit settings.            */
#define SB2                      0x04

/* COM port base addresses.                                           */

#define COM1                     0x3F0   /* 3F8-3FF.                  */
#define COM2                     0x2F0   /* 2F8-2FF.                  */
#define COM3                     0x3E0   /* 3E8-3EF.                  */
#define COM4                     0x2E0   /* 2E8-2EF.                  */

/* Global port and line control variables for customization.          */

extern unsigned int  lcc_com_port;
extern unsigned int  lcc_baud_rate;
extern unsigned char lcc_tty_control;
#endif

/* Parallel I/O Constants:                                            */

/* Data direction (PDIR) definitions: 0 = input, 1 = output.          */

#define PIO_INPUT                0
#define PIO_OUTPUT               1

/* Private Functions:                                                 */

static int  lcc_mode(void);
static int  lcc_baud(void);
static void lcc_set_rcv_dst(lcd_packet p, word length);
static void lcc_set_xmt_src(lcd_packet p, word length);
static word lcc_crc16(lcd_packet p, word length);
static void lcc_sys_init(int mode, int baud_rate);
static void lcc_start_rcv(void);
static void lcc_start_xmt(void);
static void lcc_stop_rcv(void);
static void lcc_stop_xmt(void);
static void lcc_enable_interrupts();
#if defined(IBMAT)
static void interrupt far lcc_sio1_int();
static void interrupt far lcc_sio2_int();
static void interrupt far lcc_sio3_int();
static void interrupt far lcc_sio4_int();
#endif

#endif
```

```c
/*************************************************************************\
*                              LCC.C                                       *
***************************************************************************
*                                                                          *
* CPCI:          IED90-MRA-COM-LCS-LCC-C-ROCO                              *
*                                                                          *
* Description:   Local Communications Channel (LCC) functions.             *
*                Implements the MRA low-level data link layer functions    *
*                for the LCS local communications device handler.          *
*                Currently supports the 8031 Local Serial Channel (LSC), the *
*                the 80C152 Local Parallel Channel (LPC - I/O ports P4/P5), *
*                and the serial communications ports of the IBM-AT (8250). *
*                                                                          *
*                Module LCC exports the following functions:               *
*                                                                          *
*                xptr_lo_offset() macro;                                   *
*                xptr_hi_offset() macro;                                   *
*                lcc_mode();                                               *
*                lcc_baud();                                               *
*                lcc_crc16();                                              *
*                lcc_sys_init();                                           *
*                lcc_set_rcv_dst();                                        *
*                lcc_set_xmt_src();                                        *
*                lcc_start_rcv();                                          *
*                lcc_start_xmt();                                          *
*                lcc_stop_rcv();                                           *
*                lcc_stop_xmt();                                           *
*                lcc_enable_interrupts();                                  *
*                lcc_sio_int()  interrupt;                                 *
*                lcc_sio1_int() interrupt;                                 *
*                lcc_sio2_int() interrupt;                                 *
*                lcc_sio3_int() interrupt;                                 *
*                lcc_sio4_int() interrupt;                                 *
*                lcc_pio_int()  interrupt;                                 *
*                                                                          *
* Notes:         1) This module is NOT a stand-alone compilation unit.     *
*                   It is included by the module LCD.C and is compiled there. *
*                   It is assumed that the file LCD.H is included before it. *
*                   Note that all of the functions herein are static.      *
*                                                                          *
* Edit History: 07/10/90 - Written by Richard P. Smurlo and Robin T. Laird. *
*                                                                          *
\*************************************************************************/

#if defined(IBMAT)
#include <dos.h>                        /* DOS IBM-AT service definitions.  */
#else
#include <reg152.h>                     /* 8031 and 80152 definitions.      */
#endif

/* Private Variables:                                                       */

/* Variable that indicates operational mode (serial or parallel).           */
/* Set by the lcc_sys_init() routine (default is serial mode).              */

static XDATA int lcc_io_mode = LCC_SIO_MODE;

/* Communications port definitions for MPU (IBM-AT 8250).                   */
/* COM port selection and settings are done in MAIN.C and passed here.      */

#if defined(IBMAT)

/* "8250" communications controller (CC) register address offsets.          */

#define CC_THR              0x08    /* Transmit holding reg (write).        */
#define CC_RBR              0x08    /* Receiver buffer reg (read).          */
#define CC_DLSB             0x08    /* Divisor latch LSB.                   */
#define CC_DMSB             0x09    /* Divisor latch MSB.                   */
#define CC_IER              0x09    /* Interrupt enable register.           */
#define CC_IIR              0x0A    /* Interrupt identification reg.        */
#define CC_LCR              0x0B    /* Line control register                */
#define CC_MCR              0x0C    /* Modem control register.              */
#define CC_LSR              0x0D    /* Line status register.                */
```



```c
#define CC_MSR                  0x0E    /* Modem status register.           */

#define CC_DATA_READY           0x01    /* LSR data ready bit.              */
#define CC_RECEIVE_ERRS         0x1E    /* LSR error bits mask.             */
#define CC_TRANSMIT_HOLD_EMPTY  0x20    /* LSR xmtr holding register empty. */

#define CC_DLAB0                0x00    /* DLAB set low (0).                */
#define CC_DLAB1                0x80    /* DLAB set high (1).               */

#define CC_DISABLE_INT          0x00    /* Disable all interrupts.          */
#define CC_ENABLE_RCV           0x01    /* Enable receive interrupts.       */
#define CC_ENABLE_XMT           0x02    /* Enable transmit interrupts.      */
#define CC_ENABLE_STAT          0x04    /* Enable line status interrupts.   */
#define CC_MODEM_BUS_ENABLE     0x0B    /* DTR, RTS, and OUT2 active.       */

/* "8250" interrupt source numbers (in order of priority from high to low). */

#define LINE_STATUS_ERR_INT     0x06
#define RCV_DATA_AVAIL_INT      0x04
#define XMT_EMPTY_INT           0x02
#define MODEM_INT               0x00

/* 8259 interrupt controller (IC) register (absolute) addresses.            */

#define IC_OCW2                 0x20    /* Operation control word 2.        */
#define IC_OCW1                 0x21    /* Operation control word 1.        */

#define IC_ENABLE_IRQ3          0xF7    /* Enable ints on IRQ3.             */
#define IC_ENABLE_IRQ4          0xEF    /* Enable ints on IRQ4.             */
#define IC_EOI                  0x20    /* Non-specific end-of-interrupt.   */

/* Globals that can be changed by user as desired BEFORE call to lci_init().*/

unsigned int  lcc_com_port    = COM1;
unsigned int  lcc_baud_rate   = BR19200;
unsigned char lcc_tty_control = (PNONE | WL8 | SB1);
unsigned char lcc_ic_ocw1;
void (interrupt far *lcc_irq3_vect)();
void (interrupt far *lcc_irq4_vect)();

#else

/* Switch select bits for the ICN and MPU (80152 and 8031).                 */
/* Define I/O mode (parallel/serial select) special function bit variable.  */

sbit PSSEL = P1^6;

/* Define baud rate select special function bit variables.                  */

sbit BRS0  = P1^5;
sbit BRS1  = P1^4;
sbit BRS2  = P1^3;

/* Define special function register bit/byte variables for LPC.             */

#define PIO  P4                     /* Parallel I/O (xmt/rcv) data register. */
sbit PDIR  = P5^3;                  /* Parallel data direction (0=IN, 1=OUT). */
sbit RTSO  = P5^4;                  /* Request to send output  (active low). */
sbit CTSI  = P5^5;                  /* Clear to send input     (active low). */
sbit RTSI  = P5^6;                  /* Request to send input   (active low). */
sbit CTSO  = P5^7;                  /* Clear to send output    (active low). */

#endif


/*************************************************************************\
*                            xptr_lo_offset                                *
***************************************************************************
*                                                                          *
* Function:      Obtains the low-offset portion of an external data (XDATA) *
*                pointer. Specific to the Franklin 8031 C compiler (V2.4). *
*                                                                          *
* Input:         xptr_lo_offset(                                           *
```

```
 *                 xdata void *xp; XDATA pointer.                       *
 *               );                                                      *
 *                                                                       *
 * Output:       Returns the low-offset portion of the external data pointer. *
 *                                                                       *
 * Globals:      None.                                                   *
 *                                                                       *
 * Edit History: 07/07/90 - Written by Robin T. Laird.                   *
 *                                                                       *
\***********************************************************************/

#define xptr_lo_offset(xp)  (((unsigned int)xp)&0x00FF)


/***********************************************************************\
 *                          xptr_hi_offset                              *
 ************************************************************************
 *                                                                       *
 * Function:     Obtains the hi-offset portion of an external data (XDATA) *
 *               pointer. Specific to the Franklin 8031 C compiler (V2.4). *
 *                                                                       *
 * Input:        xptr_hi_offset(                                         *
 *                 xdata void *xp; XDATA pointer.                        *
 *               );                                                      *
 *                                                                       *
 * Output:       Returns the hi-offset portion of the external data pointer. *
 *                                                                       *
 * Globals:      None.                                                   *
 *                                                                       *
 * Edit History: 07/07/90 - Written by Robin T. Laird.                   *
 *                                                                       *
\***********************************************************************/

#define xptr_hi_offset(xp)  ((((unsigned int)xp)>>8)&0x00FF)


/***********************************************************************\
 *                            lcc_mode                                  *
 ************************************************************************
 *                                                                       *
 * Function:     Determines the I/O mode for external communications.    *
 *                                                                       *
 *               For the 80C152:                                         *
 *                                                                       *
 *               The mode is read from a switch connected to P1 bit 6 of *
 *               the 80C152 parallel port. Low (0) selects serial mode,  *
 *               while high (1) selects parallel mode. The mode bit      *
 *               corresponds to P/S SEL on the ICN schematic.            *
 *                                                                       *
 *               For the 8031 (or any other processor):                  *
 *                                                                       *
 *               The mode is always returned as serial mode selected (0). *
 *                                                                       *
 * Input:        lcc_mode();                                             *
 *                                                                       *
 * Output:       Returns the local communications mode, 0 for serial mode, *
 *               1 for parallel mode.                                    *
 *                                                                       *
 * Globals:      lcd_error : module LCD.C                                *
 *               8031 regs : module REG152.H                             *
 *                                                                       *
 * Edit History: 08/14/90 - Written by Robin T. Laird.                   *
 *                                                                       *
\***********************************************************************/

static int lcc_mode()
{
  lcd_error = AOK;                          /* Assume function successful...   */

  #if defined(I80152)
  return ((int)PSSEL);
  #else
  return (LCC_SIO_MODE);
```



```
  #endif
}


/***********************************************************************\
 *                            lcc_baud                                  *
 ************************************************************************
 *                                                                       *
 * Function:     Determines the baud rate for external serial communications. *
 *                                                                       *
 *               For the 80C152:                                         *
 *                                                                       *
 *               The baud rate is read from a switch connected to P1 of  *
 *               the 80C152 parallel port. Bits 3, 4, and 5 select one of *
 *               eight available baud rates from 300 baud to 38400 baud. *
 *               Bit 3 corresponds to BSR2 on the ICN schematic, 4 is BSR1, *
 *               and bit 5 to BSR0.                                      *
 *                                                                       *
 *               Bits 3, 4, and 5 are decoded as follows (see also LCC.H): *
 *                                                                       *
 *               Bit  3  4  5   Function Return Value   Baud Rate        *
 *               ------------------------------------------------        *
 *                    0  0  0        0 (decimal)        300              *
 *                    0  0  1        1 (decimal)        600              *
 *                    0  1  0        2 (decimal)        1200             *
 *                    0  1  1        3 (decimal)        2400             *
 *                    1  0  0        4 (decimal)        4800             *
 *                    1  0  1        5 (decimal)        9600             *
 *                    1  1  0        6 (decimal)        19200            *
 *                    1  1  1        7 (decimal)        38400 (57600)    *
 *                                                                       *
 *               If a 14 MHz CPU is being used (I80152), then the max    *
 *               baud rate is 38400, otherwise it is assumed that an     *
 *               11 MHz CPU (or similar flavor) is being used and the max *
 *               baud rate is 57600. So, the baud rate selected by code 111 *
 *               is either 38400 or 57600 depending upon the target CPU. *
 *                                                                       *
 *               For the IBM-AT (8250):                                  *
 *                                                                       *
 *               The baud rate is taken from global variable in MAIN.C.  *
 *                                                                       *
 *               For all other processors:                               *
 *                                                                       *
 *               The baud rate defaults to 19200.                        *
 *                                                                       *
 * Input:        lcc_baud();                                             *
 *                                                                       *
 * Output:       Returns the baud rate code for external serial          *
 *               communications as given by the table above.             *
 *                                                                       *
 * Globals:      lcd_error : module LCD.C                                *
 *               8031 regs : module REG152.H                             *
 *               lcc_baud_rate : module MAIN.C                           *
 *                                                                       *
 * Edit History: 08/14/90 - Written by Robin T. Laird.                   *
 *                3/14/91 - Added code for IBM-At, Robin T. Laird.       *
 *                                                                       *
\***********************************************************************/

static int lcc_baud()
{
  int baud_rate;

  lcd_error = AOK;                          /* Assume function successful...   */

  /* Decode baud rate from bits 3, 4, and 5 of parallel port P1.              */

  #if defined(I80152)
  switch((byte)BRS2<<2 | (byte)BRS1<<1 | (byte)BRS0)
  {
    case 0:
      baud_rate = BR300;
      break;
```

```
    case 1:
      baud_rate = BR600;
      break;
    case 2:
      baud_rate = BR1200;
      break;
    case 3:
      baud_rate = BR2400;
      break;
    case 4:
      baud_rate = BR4800;
      break;
    case 5:
      baud_rate = BR9600;
      break;
    case 6:
      baud_rate = BR19200;
      break;
    case 7:
      baud_rate = MAX_BAUD_RATE;          /* Either 38.4K or 57.6K.          */
      break;
    default:
      lcd_error = ERR_BAUD_RATE;
      baud_rate = 0;
  }
  #elif defined(IBMAT)
  baud_rate = lcc_baud_rate;
  #else
  baud_rate = BR19200;
  #endif

  return (baud_rate);
}


/***************************************************************************\
*                                lcc_crc16                                  *
*****************************************************************************
*                                                                           *
* Function:      Generates a simple 16-bit checksum used as the CRC for     *
*                detecting communication errors. CRC is calculated as the   *
*                one's complement of the sum of the values of all the bytes *
*                in the packet (from p[0] through p[length-1]).             *
*                                                                           *
* Input:         lcc_crc16(                                                 *
*                  lcd_packet p; packet for which checksum is to calculated.*
*                  word length;  length of packet (data portion only).      *
*                                                                           *
* Output:        Unsigned integer result of checksum used as CRC.           *
*                                                                           *
* Globals:       lcd_error : module LCD.C                                   *
*                                                                           *
* Edit History: 08/14/90 - Written by Robin T. Laird.                       *
*                                                                           *
\***************************************************************************/

static word lcc_crc16(p, length)
lcd_packet p;
word length;
{
  word i, sum;

  lcd_error = AOK;                     /* Assume function successful...   */

  /* Add up the data bytes.                                                 */
  /* Complement and return as checksum CRC.                                 */

  for (i = sum = 0; i < length; i++) sum += p[i];
  return (~sum );
}


/***************************************************************************\
```



```
*                              lcc_sys_init                                  *
******************************************************************************
*                                                                            *
* Function:      Initializes the Local Serial/Parallel Channel.              *
*                Sets the global operational mode variable, lcc_io_mode.     *
*                If SERIAL operation is selected the following is done:      *
*                                                                            *
*                TIMER 1 is used to generate the parameter baud rate.        *
*                PCON, TMOD, and SMOD are initialized appropriately.         *
*                TIMER 1 is enabled to start baud rate generation.           *
*                LCC receiver is enabled.                                    *
*                                                                            *
*                If PARALLEL operation is selected the following is done:    *
*                                                                            *
*                Baud rate parameter is ignored.                             *
*                Port direction register is set to input.                    *
*                Control registers are cleared.                              *
*                Data port is cleared.                                       *
*                                                                            *
* Input:         lcc_sys_init(                                               *
*                  byte mode;      I/O operational mode (serial/parallel).   *
*                  byte baud_rate; baud rate for serial operation.           *
*                );                                                          *
*                                                                            *
* Output:        Nothing.                                                    *
*                                                                            *
* Globals:       lcd_error   : module LCD.C                                  *
*                lcc_io_mode : module LCC.C                                  *
*                lcc_com_id  : module LCC.C                                  *
*                lcc_com_set : module LCC.C                                  *
*                8031 regs   : module REG152.H                               *
*                lcc_com_port     : module MAIN.C                            *
*                lcc_baud_rate    : module MAIN.C                            *
*                lcc_tty_control  : module MAIN.C                            *
*                lcc_irq3_vect    : module MAIN.C                            *
*                lcc_irq4_vect    : module MAIN.C                            *
*                lcc_ic_ocw1      : module MAIN.C                            *
*                                                                            *
* Edit History: 08/14/90 - Written by Robin T. Laird.                        *
*               03/14/91 - Added code for IBM-At, Robin T. Laird.            *
*                                                                            *
\****************************************************************************/

#if defined(IBMAT)

/* IBM-At 8250 and 8259 local definitions.                                  */
/* Interrupt numbers assigned to IRQs.                                      */

#define IRQ3                         0x0B    /* Interrupt request 3.        */
#define IRQ4                         0x0C    /* Interrupt request 4.        */

/* IRQs assigned to COM ports (default).                                    */
/* Note that COM1/COM3 share interrupts, and COM2/COM4 share interrupts.    */

#define COM1_INT_NUMBER              IRQ4
#define COM2_INT_NUMBER              IRQ3
#define COM3_INT_NUMBER              IRQ4
#define COM4_INT_NUMBER              IRQ3

#define MAX_RFIFO_READS              10             /* Clear this many bytes.   */

#else

/* 8031 register values for initializing the local serial channel (LSC).    */
/* If using higher clock speed, SMOD is set to 0, otherwise set to 1.       */
/* SMOD is the MSB of the PCON Power Control register on the 8031.          */

#define SERIAL_PORT_MODE_1           0x50           /* SCON = 0bx1x1xxxx     */
#define SERIAL_PORT_MASK             0x5F           /* SCON = 0b0101xxxx     */
#define TIMER_1_MODE_2               0x20           /* TMOD = 0bxx1xxxxx     */
#define TIMER_1_MASK                 0x2F           /* TMOD = 0b0010xxxx     */

#if defined(I80152)
```

```
#define DOUBLE_BAUD_RATE          0x00            /* PCON = 0b0xxxxxxx SMOD=0 */
#else
#define DOUBLE_BAUD_RATE          0x80            /* PCON = 0b1xxxxxxx SMOD=1 */
#endif

#endif

static void lcc_sys_init(mode, baud_rate)
int mode;
int baud_rate;
{
  byte i, temp;

  lcd_error = AOK;                         /* Assume function successful...    */

  /* Initialize the I/O channel depending upon the mode...                     */

  /* For serial mode operation:                                                */
  /*   For 8031 or 80152:
  /*      TIMER 1 is used in MODE 2 for variable baud rates.                   */
  /*      TH1 sets TIMER 1 re-load value.                                      */
  /*      SCON controls the serial port modes settings.                        */
  /*      TMOD controls the timer mode settings.                               */
  /*      TCON controls the timer itself (turns it on/off).                    */
  /*      SMOD controls baud rate doubling depending upon CPU speed.           */
  /*      Clear serial buffer by reading data.                                 */
  /*   For 8250 (IBM-AT) (NOSC serial port library functions are used):       */
  /*      Configure 8250 with baud rate and serial params defined elsewhere.   */
  /*      Disable serial interrupts for now.                                   */
  /*      Clear all registers (MSR, LSR, and data buffer).                     */
  /*      Save old interrupt vectors (so someone else can restore).            */
  /*      Save old 8259 interrupt control word (so someone else can restore).*/
  /*      Enable processor bus interrupts via the MODEM control register.      */

  /* For parallel operation:                                                   */
  /*   Set port direction to input.                                            */
  /*   Clear data control lines.                                               */
  /*   Clear data port.                                                        */

  switch(mode)
  {
    case LCC_SIO_MODE:
      lcc_io_mode = LCC_SIO_MODE;

      #if defined(IBMAT)

      /* Disable interrupts while we're changing vectors, ports, etc.          */

      _disable();

      /* Initialize 8250, see "Technical Reference Personal Computer AT".      */
      /* Set baud rate (divisor latches).                                      */
      /* DELAB must be 1 to write to baud rate divisor latches.                */
      /* Parameter baud_rate is ignored.                                       */

      outp(lcc_com_port+CC_LCR, CC_DLAB1);
      outp(lcc_com_port+CC_DMSB, (lcc_baud_rate >> 8) & 0xFF);
      outp(lcc_com_port+CC_DLSB, lcc_baud_rate & 0xFF);

      /* Set serial port operational parameters.                               */
      /* DELAB must be 0 to write/read LCR, IER, and data registers.           */

      outp(lcc_com_port+CC_LCR, CC_DLAB0 | lcc_tty_control);

      /* Disable all 8250 interrupts for now.                                  */

      outp(lcc_com_port+CC_IER, CC_DISABLE_INT);

      /* Clear line status and modem status registers.                         */

      inp(lcc_com_port+CC_LSR);
      inp(lcc_com_port+CC_MSR);

```



```
      /* Clear chars from receive register.                                    */
      /* Line status register (bit 0) indicates if data ready.                 */

      for (i = 0; i < MAX_RFIFO_READS; i++)
      {
       if (inp(lcc_com_port+CC_LSR) & CC_DATA_READY)
        inp(lcc_com_port+CC_RBR);
       else
         break;
      }
      if (i > MAX_RFIFO_READS || inp(lcc_com_port+CC_LSR) & CC_DATA_READY)
      {
        lcd_error = ERR_RFIFO_NOT_CLEAR;
        return;
      }

      /* Save old interrupt vectors so we can restore them later.              */
      /* Set new vectors to our interrupt routines.                            */

      switch(lcc_com_port)
      {
        case COM1:
          lcc_irq4_vect = _dos_getvect(COM1_INT_NUMBER);
           _dos_setvect(COM1_INT_NUMBER, lcc_sio1_int);
          break;

        case COM3:
          lcc_irq4_vect = _dos_getvect(COM3_INT_NUMBER);
           _dos_setvect(COM3_INT_NUMBER, lcc_sio3_int);
          break;

        case COM2:
          lcc_irq3_vect = _dos_getvect(COM2_INT_NUMBER);
           _dos_setvect(COM2_INT_NUMBER, lcc_sio2_int);
          break;

        case COM4:
          lcc_irq3_vect = _dos_getvect(COM4_INT_NUMBER);
           _dos_setvect(COM4_INT_NUMBER, lcc_sio4_int);
          break;

        default:
          break;
      }

      /* Initialize the 8259 interrupt controller.                             */
      /* Save old control interrupt mask.                                      */

      lcc_ic_ocw1 = inp(IC_OCW1);

      /* Set MODEM control reg to enable bus interrupts.                       */

      outp(lcc_com_port+CC_MCR, CC_MODEM_BUS_ENABLE);

      #else

      /* Disable processor interrupts while we're changing things.             */

      EA = 0;

      PCON |= DOUBLE_BAUD_RATE;          /* Set Power Control Mode register. */
      TH1   = (byte)baud_rate;           /* Set TIMER 1 auto-reload value.   */
      TMOD |= TIMER_1_MODE_2;            /* Set Timer Mode Control register. */
      TMOD &= TIMER_1_MASK;
      SCON |= SERIAL_PORT_MODE_1;        /* Set Serial Port Control register.*/
      SCON &= SERIAL_PORT_MASK;
      TR1   = 1;                         /* Set TIMER 1 run bit.             */
      temp  = SBUF;                      /* Clear serial receive buffer.     */

      #endif
      break;

    case LCC_PIO_MODE:
```

```
        lcc_io_mode = LCC_PIO_MODE;

        #if defined(IBMAT)
        lcd_error = ERR_IO_MODE;
        #else

        /* Disable processor interrupts while we're changing things.          */

        EA = 0;

        PDIR   = PIO_INPUT;                  /* Parallel I/O direction.       */
        RTSO_  = 0;                          /* Data control lines...         */
        CTSI_  = 0;
        RTSI_  = 0;
        CTSO_  = 0;
        PIO    = 0;                          /* Parallel I/O data port.       */

        #endif
        break;

      default:
        lcd_error = ERR_IO_MODE;
        return;
    }
}


/*************************************************************************\
*                            lcc_set_rcv_dst                              *
***************************************************************************
*                                                                         *
* Function:      Sets the communications channel receiver destination address *
*                and packet size. The destination for the reception is the *
*                parameter packet address. Special care must be taken so that *
*                data in the destination is not over written (or processed) *
*                before the entire packet is received.                    *
*                                                                         *
* Input:         lcc_set_rcv_dst(                                         *
*                  lcd_packet p; pointer to receive packet (dest).        *
*                  word length;  size of receive packet (in bytes).       *
*                );                                                       *
*                                                                         *
* Output:        Nothing.                                                 *
*                                                                         *
* Globals:       lcd_error : module LCD.C                                 *
*                                                                         *
* Edit History: 03/09/91 - Written by Robin T. Laird.                     *
*                                                                         *
\*************************************************************************/

static void lcc_set_rcv_dst(p, length)
lcd_packet p;
word length;
{
  lcd_error = AOK;                         /* Assume function successful...   */
}


/*************************************************************************\
*                            lcc_set_xmt_src                              *
***************************************************************************
*                                                                         *
* Function:      Sets the communications channel transmission source address *
*                and packet size. The source for the transmission is the  *
*                parameter packet address. Special care must be taken so  *
*                that data in the source is not over written (or processed) *
*                before the entire packet is transmitted.                 *
*                                                                         *
* Input:         lcc_set_xmt_src(                                         *
*                  lcd_packet p; pointer to transmit packet.              *
*                  word length;  size of receive packet (in bytes).       *
*                );                                                       *
*                                                                         *
```



```
* Output:        Nothing.                                                 *
*                                                                         *
* Globals:       lcd_error : module LCD.C                                 *
*                                                                         *
* Edit History: 03/09/91 - Written by Robin T. Laird.                     *
*                                                                         *
\*************************************************************************/

static void lcc_set_xmt_src(p, length)
lcd_packet p;
word length;
{
  lcd_error = AOK;                         /* Assume function successful...   */
}


/*************************************************************************\
*                            lcc_start_rcv                                *
***************************************************************************
*                                                                         *
* Function:      Initiates reception of a data packet.                    *
*                Enables the LCC/LPC interrupt service routines.          *
*                                                                         *
* Input:         lcc_start_rcv();                                         *
*                                                                         *
* Output:        ~thing.                                                  *
*                                                                         *
* Globals:       lcd_error    : module LCD.C                              *
*                lcc_io_mode  : module LCC.C                              *
*                8031 regs    : module REG152.H                           *
*                lcc_com_port : module MAIN.C                             *
*                                                                         *
* Edit History: 07/10/90 - Written by Richard P. Smurlo and Robin T. Laird. *
*               03/14/91 - Added code for IBM-At, Robin T. Laird.         *
*                                                                         *
\*************************************************************************/

static void lcc_start_rcv()
{
  lcd_error = AOK;                         /* Assume function successful...   */

  /* Enable processor interrupts according to operational mode.             */

  #if defined(IBMAT)

  /* Enable interrupts on COM port IRQ (0 a bit to enable IRQ int).          */
  /* Assumes that COM1/COM3 on IRQ4 and COM2/COM4 on IRQ3).                  */

  switch(lcc_com_port)
  {
    case COM1:
    case COM3:
      outp(IC_OCW1, inp(IC_OCW1) & IC_ENABLE_IRQ4);
      break;

    case COM2:
    case COM4:
      outp(IC_OCW1, inp(IC_OCW1) & IC_ENABLE_IRQ3);
      break;

    default:
      break;
  }

  /* Enable receive and line status interrupts on the 8250.                  */

  outp(lcc_com_port+CC_IER, inp(lcc_com_port+CC_IER) | CC_ENABLE_RCV
                                                     | CC_ENABLE_STAT);

  #else

  switch(lcc_io_mode)
  {
```

```
    case LCC_SIO_MODE:
      ES = 1;                             /* Enable serial interrupts.        */
      break;

    case LCC_PIO_MODE:
      EX0 = 1;                            /* Enable external interrupt 0.     */
      EX1 = 1;                            /* Enable external interrupt 1.     */
      break;

    default:
      break;
  }

  #endif
}


/******************************************************************************\
*                                lcc_start_xmt                                 *
********************************************************************************
*                                                                              *
* Function:     Initiates transmission of data packets.                        *
*               Enables the LCC/LPC interrupt service routines.                *
*                                                                              *
* Input:        lcc_start_xmt();                                               *
*                                                                              *
* Output:       Nothing.                                                       *
*                                                                              *
* Globals:      lcd_error    : module LCD.C                                    *
*               lcc_io_mode  : module LCC.C                                    *
*               8031 regs    : module REG152.H                                 *
*               lcc_com_port : module MAIN.C                                   *
*                                                                              *
* Edit History: 07/10/90 - Written by Richard P. Smurlo and Robin T. Laird.    *
*               03/14/91 - Added code for IBM-At, Robin T. Laird.              *
*                                                                              *
\******************************************************************************/

static void lcc_start_xmt()
{
  lcd_error = AOK;                        /* Assume function successful...   */

  /* Enable processor interrupts according to operational mode.              */

  /* If serial mode is used, have to cause interrupt to happen.              */
  /* This is done by setting the transmit interrupt flag (TI).               */

  #if defined(IBMAT)

  /* Enable interrupts on COM port IRQ (0 a bit to enable IRQ int).          */
  /* Assumes that COM1/COM3 on IRQ4 and COM2/COM4 on IRQ3).                  */

  switch(lcc_com_port)
  {
    case COM1:
    case COM3:
      outp(IC_OCW1, inp(IC_OCW1) & IC_ENABLE_IRQ4);
      break;

    case COM2:
    case COM4:
      outp(IC_OCW1, inp(IC_OCW1) & IC_ENABLE_IRQ3);
      break;

    default:
      break;
  }

  /* Enable transmit interrupts on the 8250.                                 */

  outp(lcc_com_port+CC_IER, inp(lcc_com_port+CC_IER) | CC_ENABLE_XMT);

  /* Call interrupt routine to start transmissions.                          */
```



```
  /* Interrupt won't happen until interrupts are actually enabled.           */

  switch(lcc_com_port)
  {
    case COM1:
      lcc_sio1_int();
      break;

    case COM2:
      lcc_sio2_int();
      break;

    case COM3:
      lcc_sio3_int();
      break;

    case COM4:
      lcc_sio4_int();
      break;

    default:
      break;
  }

  #else

  switch(lcc_io_mode)
  {
    case LCC_SIO_MODE:
      ES = 1;                             /* Enable serial interrupts.        */
      TI = 1;                             /* Cause interrupt to happen.       */
      break;

    case LCC_PIO_MODE:
      EX0 = 1;                            /* Enable external interrupt 0.     */
      EX1 = 1;                            /* Enable external interrupt 1.     */
      break;

    default:
      break;
  }

  #endif
}


/******************************************************************************\
*                                 lcc_stop_rcv                                 *
********************************************************************************
*                                                                              *
* Function:     Disables the LCC/LPC receiver by turning off the enable bit.   *
*               Any incoming packet is dropped.                                *
*                                                                              *
* Input:        lcc_stop_rcv();                                                *
*                                                                              *
* Output:       Nothing.                                                       *
*                                                                              *
* Globals:      lcd_error    : module LCD.C                                    *
*               lcc_io_mode  : module LCC.C                                    *
*               8031 regs    : module REG152.H                                 *
*               lcc_com_port : module MAIN.C                                   *
*               lcc_ic_ocw1  : module MAIN.C                                   *
*                                                                              *
* Edit History: 07/10/90 - Written by Richard P. Smurlo.                       *
*               03/14/91 - Added code for IBM-At, Robin T. Laird.              *
*                                                                              *
\******************************************************************************/

static void lcc_stop_rcv()
{
  lcd_error = AOK;                        /* Assume function successful...   */

  /* Disable processor interrupts according to operational mode.             */
```

```
    #if defined(IBMAT)

    /* Disable receive and line status interrupts on the 8250.          */

    outp(lcc_com_port+CC_IER, inp(lcc_com_port+CC_IER) & ~CC_ENABLE_RCV
                                                       & ~CC_ENABLE_STAT);

    /* Disable IRQ3 and IRQ4 interrupts on 8259.                        */
    /* Restore old 8259 interrupt control bit mask.                     */

    outp(IC_OCW1, lcc_ic_ocw1);

    #else

    switch(lcc_io_mode)
    {
      case LCC_SIO_MODE:
        ES = 0;                      /* Disable serial interrupts.      */
        break;

      case LCC_PIO_MODE:
        EX0 = 0;                     /* Disable external interrupt 0.   */
        EX1 = 0;                     /* Disable external interrupt 1.   */
        break;

      default:
        break;
    }

    #endif
  }

  /*********************************************************************\
  *                          lcc_stop_xmt                               *
  ***********************************************************************
  *                                                                     *
  * Function:     Disables the LCC/LPC transmitter by turning off enable bit. *
  *               Any outgoing packet is terminated.                    *
  *                                                                     *
  * Input:        lcc_stop_xmt();                                       *
  *                                                                     *
  * Output:       Nothing.                                              *
  *                                                                     *
  * Globals:      lcd_error    : module LCD.C                           *
  *               lcc_io_mode  : module LCC.C                           *
  *               8031 regs    : module REG152.H                        *
  *               lcc_com_port : module MAIN.C                          *
  *               lcc_ic_ocw1  : module MAIN.C                          *
  *                                                                     *
  * Edit History: 07/10/90 - Written by Richard P. Smurlo.              *
  *               03/14/91 - Added code for IBM-At, Robin T. Laird.     *
  *                                                                     *
  \*********************************************************************/

  static void lcc_stop_xmt()
  {
    lcd_error = AOK;                 /* Assume function successful...   */

    /* Disable processor interrupts according to operational mode.     */

    #if defined(IBMAT)

    /* Disable transmit interrupts on the 8250.                         */

    outp(lcc_com_port+CC_IER, inp(lcc_com_port+CC_IER) & ~CC_ENABLE_XMT);

    /* Disable IRQ3 and IRQ4 interrupts on 8259.                        */
    /* Restore old 8259 interrupt control bit mask.                     */

    outp(IC_OCW1, lcc_ic_ocw1);
```



```
    #else

    switch(lcc_io_mode)
    {
      case LCC_SIO_MODE:
        ES = 0;                      /* Disable serial interrupts.      */
        break;

      case LCC_PIO_MODE:
        EX0 = 0;                     /* Disable external interrupt 0.   */
        EX1 = 0;                     /* Disable external interrupt 1.   */
        break;

      default:
        break;
    }

    #endif
  }

  /*********************************************************************\
  *                       lcc_enable_interrupts                         *
  ***********************************************************************
  *                                                                     *
  * Function:     Enables processor interrupts for the 8031/80152.      *
  *                                                                     *
  * Input:        lcc_enable_interrupts();                              *
  *                                                                     *
  * Output:       Nothing.                                              *
  *                                                                     *
  * Globals:      lcd_error    : module LCD.C                           *
  *               8031 regs    : module REG152.H                        *
  *               lcc_com_port : module MAIN.C                          *
  *                                                                     *
  * Edit History: 03/09/91 - Written by Richard P. Smurlo.              *
  *               03/14/91 - Added code for IBM-At, Robin T. Laird.     *
  *                                                                     *
  \*********************************************************************/

  static void lcc_enable_interrupts()
  {
    lcd_error = AOK;                 /* Assume function successful...   */

    #if defined(IBMAT)
    _enable();                       /* Enable all interrupts (STI).    */
    #else
    EA = 1;                          /* Enable all interrupts.          */
    #endif
  }


  /*********************************************************************\
  *                           lcc_sio_int                               *
  ***********************************************************************
  *                                                                     *
  * Function:     Processes a valid serial interrupt (ESV) from the LSC. *
  *               This routine is called upon completion of a serial interrupt.*
  *               This happens after a char has been transmitted or after a *
  *               char has been received. The routine determines which int *
  *               has occured and proceeds accordingly.                 *
  *                                                                     *
  *               This interrupt routine services the global rcv_buffer. *
  *               As bytes are received, they are placed in the next position *
  *               in the receive buffer, and the rear buffer index is updated. *
  *               If the buffer is full, then bytes are dropped. The full flag *
  *               is also updated.                                      *
  *                                                                     *
  *               This interrupt routine also services the global xmt_buffer. *
  *               As bytes are transmitted, they are removed from the next *
  *               position in the transmit buffer, and the front buffer index *
  *               is updated. If the buffer is empty, then transmission is *
  *               terminated. The full empty is also updated.           *
```

```
 *                                                                      *
 * Input:       lcc_sio_int() interrupt;       I80152 and I8031         *
 *              lcc_sio1_int() interrupt;      IBMAT COM1               *
 *              lcc_sio2_int() interrupt;      IBMAT COM2               *
 *              lcc_sio3_int() interrupt;      IBMAT COM3               *
 *              lcc_sio4_int() interrupt;      IBMAT COM4               *
 *                                                                      *
 * Output:      Nothing.                                                *
 *                                                                      *
 * Globals:     rcv_buffer : module BMF.C                               *
 *              xmt_buffer : module BMF.C                               *
 *               lcd_state : module LCD.C                               *
 *              8031_regs  : module REG152.H                            *
 *                                                                      *
 * Edit History: 07/10/90 - Written by Robin T. Laird.                  *
 *               03/14/91 - Added code for IBM-At, Robin T. Laird.      *
 *                                                                      *
\***********************************************************************/

#if defined(IBMAT)

unsigned int ctr = 0;
unsigned int errctr = 0;

static void interrupt far lcc_sio1_int()
{
  static byte b;

  /* Interrupt occured on COM1.                                          */
  /* When LSB of interrupt ID register set then no more interrupts pending. */
  /* Loop until this condition is met. This allows multiple int servicing.  */

  do {
    /* Increment interrupt service counter.                              */

    ++ctr;

    /* Interrupt ID register holds number of interrupt source(s).        */
    /* Transmit interrupt is serviced last to avoid interrupting ourself. */

    switch(inp(COM1+CC_IIR))
    {
      case MODEM_INT:
      {
        /* Should never have one of these.                               */
        /* Just clear line status and modem status registers.            */

        /* Reading the MSR register clears int in IIR.                   */

        inp(COM1+CC_LSR);
        inp(COM1+CC_MSR);
      }
      break;

      case LINE_STATUS_ERR_INT:
      {
        /* Get (and clear) line status.                                  */
        /* Check status to see if there is a byte waiting to be received. */
        /* If there is, get and throw away bad byte.                     */
        /* Otherwise, all done.                                          */

        /* Reading the line status register clears int in IIR.           */

        if (inp(COM1+CC_LSR) & CC_DATA_READY)
        {
          inp(COM1+CC_RBR);
          ++errctr;
        }
      }
      break;

      case RCV_DATA_AVAIL_INT:
      {
```



```
        /* Get the byte from the receive buffer (put it in b).           */
        /* Check for receive errors in LSR (if any err bit set then error). */
        /* If bad byte or buffer full throw byte away.                   */
        /* Otherwise...                                                  */
        /*   Put byte at end of buffer.                                  */
        /*   Increment rear pointer.                                     */
        /*   Check if buffer full, and set flag appropriately.           */

        /* Reading the receive buffer register clears int in IIR.        */

        b = inp(COM1+CC_RBR);
        if ((inp(COM1+CC_LSR) & CC_RECEIVE_ERRS) || rcv_buffer.full)
        {
        }
        else
        {
          rcv_buffer.item[rcv_buffer.rear] = b;
          buf_inc(rcv_buffer.rear);
          if (rcv_buffer.front == rcv_buffer.rear) rcv_buffer.full = TRUE;
        }
      }
      break;

      case XMT_EMPTY_INT:
      default:
      {
        /* Get (and clear) line status.                                  */
        /* Make sure transmitter holding register is empty.              */
        /* If it's not, we got problems, so just return.                 */
        /* If the transmit buffer is empty, just return also.            */
        /* Otherwise...                                                  */
        /*   Output byte at front of buffer.                             */
        /*   Increment front pointer.                                    */
        /*   Buffer can't be full since we just transmitted a byte.      */

        /* Reading IIR or writing to transmit buff reg clears int in IIR. */

        if ((inp(COM1+CC_LSR) & CC_TRANSMIT_HOLD_EMPTY) && !xmt_buffer.empty)
        {
          xmt_buffer.full = FALSE;
          outp(COM1+CC_THR, xmt_buffer.item[xmt_buffer.front]);
          buf_inc(xmt_buffer.front);
          if (xmt_buffer.front == xmt_buffer.rear) xmt_buffer.empty = TRUE;
        }
        else
        {
        }
      }
    }
  } while (inp(COM1+CC_IIR) != 1);

  /* Acknowledge interrupt on 8259 by sending end-of-interrupt command.  */

  outp(IC_OCW2, IC_EOI);
}

static void interrupt far lcc_sio2_int()
{
  static byte b;

  /* Interrupt occured on COM2.                                          */

  do {
    ++ctr;
    switch(inp(COM2+CC_IIR))
    {
      case MODEM_INT:
      {
        inp(COM2+CC_LSR);
        inp(COM2+CC_MSR);
      }
      break;
```

```
      case LINE_STATUS_ERR_INT:
      {
        if (inp(COM2+CC_LSR) & CC_DATA_READY)
        {
          inp(COM2+CC_RBR);
        ++errctr;
        }
      }
      break;

      case RCV_DATA_AVAIL_INT:
      {
        b = inp(COM2+CC_RBR);
        if ((inp(COM2+CC_LSR) & CC_RECEIVE_ERRS) || rcv_buffer.full)
        {
        }
        else
        {
          rcv_buffer.item[rcv_buffer.rear] = b;
          buf_inc(rcv_buffer.rear);
          if (rcv_buffer.front == rcv_buffer.rear) rcv_buffer.full = TRUE;
        }
      }
      break;

      case XMT_EMPTY_INT:
      default:
      {
        if ((inp(COM2+CC_LSR) & CC_TRANSMIT_HOLD_EMPTY) && !xmt_buffer.empty)
        {
          xmt_buffer.full = FALSE;
          outp(COM2+CC_THR, xmt_buffer.item[xmt_buffer.front]);
          buf_inc(xmt_buffer.front);
          if (xmt_buffer.front == xmt_buffer.rear) xmt_buffer.empty = TRUE;
        }
        else
        {
        }
      }
    }
  } while (inp(COM2+CC_IIR) != 1);
  outp(IC_OCW2, IC_EOI);
}

static void interrupt far lcc_sio3_int()
{
  static byte b;

  /* Interrupt occured on COM3.                                              */

  do {
    ++ctr;
    switch(inp(COM3+CC_IIR))
    {
      case MODEM_INT:
      {
        inp(COM3+CC_LSR);
        inp(COM3+CC_MSR);
      }
      break;

      case LINE_STATUS_ERR_INT:
      {
        if (inp(COM3+CC_LSR) & CC_DATA_READY)
        {
          inp(COM3+CC_RBR);
        ++errctr;
        }
      }
      break;

      case RCV_DATA_AVAIL_INT:
      {
```



```
        b = inp(COM3+CC_RBR);
        if ((inp(COM3+CC_LSR) & CC_RECEIVE_ERRS) || rcv_buffer.full)
        {
        }
        else
        {
          rcv_buffer.item[rcv_buffer.rear] = b;
          buf_inc(rcv_buffer.rear);
          if (rcv_buffer.front == rcv_buffer.rear) rcv_buffer.full = TRUE;
        }
      }
      break;

      case XMT_EMPTY_INT:
      default:
      {
        if ((inp(COM3+CC_LSR) & CC_TRANSMIT_HOLD_EMPTY) && !xmt_buffer.empty)
        {
          xmt_buffer.full = FALSE;
          outp(COM3+CC_THR, xmt_buffer.item[xmt_buffer.front]);
          buf_inc(xmt_buffer.front);
          if (xmt_buffer.front == xmt_buffer.rear) xmt_buffer.empty = TRUE;
        }
        else
        {
        }
      }
    }
  } while (inp(COM3+CC_IIR) != 1);
  outp(IC_OCW2, IC_EOI);
}

static void interrupt far lcc_sio4_int()
{
  static byte b;

  /* Interrupt occured on COM4.                                              */

  do {
    ++ctr;
    switch(inp(COM4+CC_IIR))
    {
      case MODEM_INT:
      {
        inp(COM4+CC_LSR);
        inp(COM4+CC_MSR);
      }
      break;

      case LINE_STATUS_ERR_INT:
      {
        if (inp(COM4+CC_LSR) & CC_DATA_READY)
        {
          inp(COM4+CC_RBR);
        ++errctr;
        }
      }
      break;

      case RCV_DATA_AVAIL_INT:
      {
        b = inp(COM4+CC_RBR);
        if ((inp(COM4+CC_LSR) & CC_RECEIVE_ERRS) || rcv_buffer.full)
        {
        }
        else
        {
          rcv_buffer.item[rcv_buffer.rear] = b;
          buf_inc(rcv_buffer.rear);
          if (rcv_buffer.front == rcv_buffer.rear) rcv_buffer.full = TRUE;
        }
      }
      break;
```

```
        case XMT_EMPTY_INT:
        default:
          {
            if ((inp(COM4+CC_LSR) & CC_TRANSMIT_HOLD_EMPTY) && !xmt_buffer.empty)
            {
              xmt_buffer.full = FALSE;
              outp(COM4+CC_THR, xmt_buffer.item[xmt_buffer.front]);
              buf_inc(xmt_buffer.front);
              if (xmt_buffer.front == xmt_buffer.rear) xmt_buffer.empty = TRUE;
            }
            else
            {
            }
          }
      }
    } while (inp(COM4+CC_IIR) != 1);
    outp(IC_OCW2, IC_EOI);
}

#else

static void lcc_sio_int() interrupt 4 using 2
{
  /* Determine which interrupt occured (receive or transmit).                  */

  if (RI)
  {
    /* For a valid reception:                                                 */
    /*    Clear receive interrupt flag.                                       */
    /*    Make sure buffer is not full (if it is, reset RI and drop chars).   */
    /*    Move byte from input register to receive buffer.                    */
    /*    Adjust buffer rear index.                                           */
    /*    See if the buffer is full.                                          */

    RI = 0;
    if (!rcv_buffer.full)
    {
      rcv_buffer.item[rcv_buffer.rear] = SBUF;
      buf_inc(rcv_buffer.rear);
      if (rcv_buffer.front == rcv_buffer.rear) rcv_buffer.full = TRUE;
    }
  }

  if (TI)
  {
    /* For valid transmit interrupt:                                          */
    /*    Clear transmit interrupt flag.                                      */
    /*    Mark buffer as not full since we just removed an item.              */
    /*    Make sure buffer is not empty (if so just clear int flag).          */
    /*    Move byte from front of buffer to transmit register.                */
    /*    Adjust buffer front index.                                          */
    /*    See if the buffer is empty.                                         */

    TI = 0;
    if (!xmt_buffer.empty)
    {
      xmt_buffer.full = FALSE;
      SBUF = xmt_buffer.item[xmt_buffer.front];
      buf_inc(xmt_buffer.front);
      if (xmt_buffer.front == xmt_buffer.rear) xmt_buffer.empty = TRUE;
    }
  }
}

#endif


/***************************************************************************\
*                               lcc_pio_int                                 *
*****************************************************************************
*                                                                           *
* Function:     Processes valid parallel I/O interrupts.                    *
```



```
*               Both parallel input and output requests are processed via   *
*               this interrupt handler.                                     *
*                                                                           *
* Input:        lcc_pio_int() interrupt;                                    *
*                                                                           *
* Output:       Nothing.                                                    *
*                                                                           *
* Globals:      None.                                                       *
*                                                                           *
* Edit History: 07/10/90 - Written by Richard P. Smurlo.                    *
*                                                                           *
\***************************************************************************/

#if defined(I80152)

static void lcc_pio_int() interrupt 0 using 2
{
}

#endif
```

```c
/*********************************************************************\
*                              LCD.H                                 *
***********************************************************************
*                                                                    *
* CPCI:              MRA-COM-LCS-LCD-H-ROCO                          *
*                                                                    *
* Description:  LCS communications device handler variables and functions. *
*               Contains constant function parameter declarations as well  *
*               as function return values (for success and failure of all  *
*               operations). Contains the function prototypes for the LCD.C *
*               module.                                              *
*                                                                    *
*               Module LCD exports the following types/variables/functions: *
*                                                                    *
*               typedef lcd_packet;                                  *
*               typedef lcd_state;                                   *
*                                                                    *
*               int lcd_error;                                       *
*                                                                    *
*               lcd_init();                                          *
*               lcd_reset();                                         *
*               lcd_enable();                                        *
*               lcd_disable();                                       *
*               lcd_receive_packet();                                *
*               lcd_transmit_packet();                               *
*               lcd_status();                                        *
*                                                                    *
* Notes:        1) See SDS pp. 5-6 through 5-x for more information. *
*                                                                    *
* Edit History: 08/14/90 - Written by Robin T. Laird.                *
*                                                                    *
\*********************************************************************/

/* Public Data Structures:                                           */

#ifndef LCD_MODULE_CODE
#define LCD_MODULE_CODE          3000

#define LCD_ERR_NOT_INIT         1+LCD_MODULE_CODE
#define LCD_ERR_PACKET_LENGTH    2+LCD_MODULE_CODE
#define LCD_ERR_NUM_ATTEMPTS     3+LCD_MODULE_CODE

#define LCD_ERR_RECEIVE_PACKET   4+LCD_MODULE_CODE
#define LCD_ERR_TRANSMIT_PACKET  5+LCD_MODULE_CODE

#define LCD_FAIL_RECEIVER        6+LCD_MODULE_CODE
#define LCD_FAIL_TRANSMITTER     7+LCD_MODULE_CODE

#define LCD_WAIT_FOREVER         65535
#define LCD_DONT_WAIT            0

#define LCD_MAX_PACKET_LENGTH    SYS_MAX_PACKET_SIZE
#define LCD_MAX_ATTEMPTS         60000

#define LCD_DEST_ADDR_POS        0
#define LCD_LEN_POS              1
#define LCD_SRC_ADDR_POS         2

/* Type for receive/transmit data packet, simply a 256-element array.      */

typedef byte lcd_packet[LCD_MAX_PACKET_LENGTH];

/* Structure type for LCD module status (holds rcv/xmt error counts).      */

typedef struct { word r_valid_cnt;
                 word r_err_cnt;
                 word r_crce_err_cnt;
                 word r_bop_err_cnt;
                 word x_valid_cnt;
                 word x_err_cnt;
               } lcd_state;

/* External module global error variable.                            */
```



```c
extern int lcd_error;

/* Public Functions:                                                 */

void lcd_init(void);
void lcd_reset(void);
void lcd_enable(void);
void lcd_disable(void);
void lcd_receive_packet(lcd_packet p, word retry);
void lcd_transmit_packet(lcd_packet p, word retry);
void lcd_status(lcd_state *s);

#endif
```

```
/*********************************************************************\
*                              LCD.C                                  *
***********************************************************************
*                                                                     *
* CPCI:          IED90-MRA-COM-LCS-LCD-C-R0C0                         *
*                                                                     *
* Description:   LCS communications device handler functions.         *
*                Implements the MRA standard Local Communications     *
*                Device (LCD) Handler module. This module contains the *
*                standard device handler functions and must include the *
*                low-level data link layer functions for the actual hardware *
*                implementation (currently implemented for the 8031 LSC and *
*                the 80C152 LSC/LPC - Local Serial/Parallel Channel). *
*                                                                     *
*                Message format at this level (ISO OSI data link layer) is: *
*                                                                     *
*                 | byte 0 | byte 1 | byte 2 | byte 3 | byte 4 | byte n| *
*                 |-----------------------------------------------------| *
*                 |  DEST  | LENGTH | SOURCE |  xxxx  |  xxxx  |  .... | *
*                                                                     *
*                Module LCD exports the following variables/functions: *
*                                                                     *
*                int lcd_error;                                       *
*                                                                     *
*                lcd_init();                                          *
*                lcd_reset();                                         *
*                lcd_enable();                                        *
*                lcd_disable();                                       *
*                lcd_receive_packet();                                *
*                lcd_transmit_packet();                               *
*                lcd_status();                                        *
*                                                                     *
* Notes:         1) The files BMF.H and BMF.C contain the support functions *
*                   for managing the receive and transmit circular buffers. *
*                2) The files LCC.H and LCC.C contain the required support *
*                   functions for the Local Communications Channel hardware. *
*                                                                     *
* Edit History: 08/14/90 - Written by Richard P. Smurlo and Robin T. Laird. *
*                                                                     *
\*********************************************************************/

#include <sysdefs.h>                    /* System constants and types.       */
#include "lcd.h"                        /* LCD public literals/functions.    */
#include "lcc.h"                        /* Local Communications Channel.     */
#include "bmf.h"                        /* Buffer management functions.      */

/* Public Variables:                                                         */

/* Global module error variable, lcd_error.                                  */
/* lcd_error contains code of last error occurence.                          */
/* Should be set to AOK after each successful function call.                 */
/* Variable can be examined by other software after each function call.      */

XDATA int lcd_error = LCD_ERR_NOT_INIT; /* Global module error variable.     */

/* Global module state variable, _lcd_state.                                 */
/* Holds LCD module error counts for packet reception/transmission.          */

static XDATA lcd_state _lcd_state;      /* Global module state variable.     */

/* Buffer management functions should be included here.                      */

#include "bmf.c"                        /* Buffer management functions.      */

/* Low-level data link layer support functions should be included here.      */

#include "lcc.c"                        /* Local communications channel.     */


/*********************************************************************\
*                             lcd_init                                *
***********************************************************************
*                                                                     *
```



```
* Function:      Initializes the Local Communications Device Handler. *
*                Clears the transmit and receive buffers, obtains the local *
*                communications channel I/O mode and baud rate, initializes *
*                the Local Serial/Parallel Channel, and starts operation *
*                of the LCC. The LCC reception/transmission error and valid *
*                packet counters of the module state variable are cleared. *
*                                                                     *
* Input:         lcd_init();                                          *
*                                                                     *
* Output:        Nothing.                                             *
*                                                                     *
* Globals:       lcd_error  : module LCD.C                            *
*                _lcd_state : module LCD.C                            *
*                rcv_buffer : module BMF.C                            *
*                xmt_buffer : module BMF.C                            *
*                                                                     *
* Edit History: 07/08/90 - Written by Robin T. Laird.                 *
*                                                                     *
\*********************************************************************/

void lcd_init()
{
  lcd_error = AOK;                      /* Assume function successful...     */

  /* Reset all error counting registers in module state variable.            */
  /* Valid reception/transmission counters are also cleared.                 */

  _lcd_state.r_valid_cnt     = 0;
  _lcd_state.r_err_cnt       = 0;
  _lcd_state.r_crce_err_cnt  = 0;
  _lcd_state.r_bop_err_cnt   = 0;
  _lcd_state.x_valid_cnt     = 0;
  _lcd_state.x_err_cnt       = 0;

  /* Initialize the transmit and receive buffers.                            */

  buf_clear(&xmt_buffer);
  buf_clear(&rcv_buffer);

  /* Get the I/O mode and baud rate (if applicable) for this system.         */
  /* Pass to LCC initialization function (which sets up I/O channel).        */
  /* No errors are currently fatal (they are only warnings).                 */

  lcc_sys_init(lcc_mode(), lcc_baud());

  /* Start the LCC receiver (begin receiving data packets).                  */
  /* Start the LCC transmitter.                                              */
  /* Errors are non-fatal and would cause only degraded performance.         */

  lcc_start_rcv();
  lcc_start_xmt();

  /* Enable interrupts...                                                    */

  lcc_enable_interrupts();
}


/*********************************************************************\
*                             lcd_reset                               *
***********************************************************************
*                                                                     *
* Function:      Performs a soft reset of the LCD systems.            *
*                The receive and transmit buffers are cleared and the *
*                receive and transmit destination and source addresses *
*                for the LCC I/O channels are reset to the beginning of *
*                the buffers. The LCC reception/transmission error and *
*                valid packet counters of the module state structure are *
*                cleared.                                             *
*                                                                     *
* Input:         lcd_reset();                                         *
*                                                                     *
* Output:        Nothing.                                             *
```

```
 *                                                                        *
 * Globals:      lcd_error   : module LCD.C                               *
 *               _lcd_state  : module LCD.C                               *
 *               rcv_buffer  : module BMF.C                               *
 *               xmt_buffer  : module BMF.C                               *
 *                                                                        *
 * Edit History: 07/09/90 - Written by Robin T. Laird.                    *
 *                                                                        *
\**************************************************************************/

void lcd_reset()
{
  lcd_error = AOK;                      /* Assume function successful...   */

  /* Reset all error counting registers in module state variable.         */
  /* Valid reception/transmission counters are also cleared.              */

  _lcd_state.r_valid_cnt     = 0;
  _lcd_state.r_err_cnt       = 0;
  _lcd_state.r_crce_err_cnt  = 0;
  _lcd_state.r_bop_err_cnt   = 0;
  _lcd_state.x_valid_cnt     = 0;
  _lcd_state.x_err_cnt       = 0;

  /* Re-initialize the transmit and receive buffers.                      */

  buf_clear(&xmt_buffer);
  buf_clear(&rcv_buffer);
}


/**************************************************************************\
 *                              lcd_enable                                *
 **************************************************************************
 *                                                                        *
 * Function:     Enables reception and transmission of data packets.      *
 *               The LCC receiver and transmitter are re-enabled.         *
 *               The function assumes that the LCC has been disabled      *
 *               for some reason. It is normally not necessary to enable  *
 *               the LCC after it has been initialized (by lcd_init()).   *
 *                                                                        *
 * Input:        lcd_enable();                                            *
 *                                                                        *
 * Output:       Nothing.                                                 *
 *                                                                        *
 * Globals:      lcd_error : module LCD.C                                 *
 *                                                                        *
 * Edit History: 07/09/90 - Written by Robin T. Laird.                    *
 *                                                                        *
\**************************************************************************/

void lcd_enable()
{
  lcd_error = AOK;                      /* Assume function successful...   */

  /* Start both the receiver and transmitter.                             */

  lcc_start_rcv();
  lcc_start_xmt();
}


/**************************************************************************\
 *                              lcd_disable                               *
 **************************************************************************
 *                                                                        *
 * Function:     Disables reception and transmission of data packets.     *
 *               The LCC receiver and transmitter are disabled.           *
 *               Any incoming data will be lost (dropped). The LCC must   *
 *               be re-enabled using lcd_enable() before packets may be   *
 *               received or transmitted again.                           *
 *                                                                        *
 * Input:        lcd_disable();                                           *
```



```
 *                                                                        *
 * Output:       Nothing.                                                 *
 *                                                                        *
 * Globals:      lcd_error : module LCD.C                                 *
 *                                                                        *
 * Edit History: 07/09/90 - Written by Robin T. Laird.                    *
 *                                                                        *
\**************************************************************************/

void lcd_disable()
{
  lcd_error = AOK;                      /* Assume function successful...   */

  /* Stop both the receiver and transmitter.                              */

  lcc_stop_rcv();
  lcc_stop_xmt();
}


/**************************************************************************\
 *                          lcd_receive_packet                            *
 **************************************************************************
 *                                                                        *
 * Function:     Removes a packet from the receive buffer if possible.    *
 *               If a packet is available, it is removed from the receive *
 *               buffer, and the packet length is updated. If a packet is not *
 *               available from the receive buffer, the function will perform *
 *               as follows depending upon the retry value:               *
 *                                                                        *
 *               LCD_DONT_WAIT    : remove packet if available, return if not.*
 *               LCD_WAIT_FOREVER : wait forever for packet to be received. *
 *               retry            : try retry times to receive packet.    *
 *                                                                        *
 * Input:        lcd_receive_packet(                                      *
 *                 lcd_packet p; packet to be received (destination).     *
 *                 word retry;   number of times to try receiving packet. *
 *               );                                                       *
 *                                                                        *
 * Output:       Nothing.                                                 *
 *                                                                        *
 * Globals:      lcd_error  : module LCD.C                                *
 *               rcv_buffer : module BMF.C                                *
 *                                                                        *
 * Edit History: 07/08/90 - Written by Robin T. Laird.                    *
 *                                                                        *
\**************************************************************************/

void lcd_receive_packet(p, retry)
lcd_packet p;
word retry;
{
  lcd_error = AOK;                      /* Assume function successful...   */

  /* If we don't want to wait (LCD_DONT_WAIT):                            */
  /*   Try and remove packet from buffer.                                 */
  /*   If one not available then lcd_error = ERR_BUF_EMPTY.               */
  /*   Else, we've successfully removed a received packet.                */

  /* If we want to wait forever (LCD_WAIT_FOREVER):                       */
  /*   Wait until there is a packet in the buffer, then remove it.        */
  /*   This function WILL wait forever, i.e., try indefinitely.           */

  /* If we want to wait for a period of time:                             */
  /*   Set up and zero auxilliary timer.                                  */
  /*   Loop, checking to see if a packet is available and timer not expired.*/
  /*   If the timer expires, indicate buffer empty and return.            */
  /*   Else, get incoming packet and return.                              */

  if (retry == LCD_DONT_WAIT)
  {
    buf_remove(&rcv_buffer, p);
  }
```

```
  else if (retry == LCD_WAIT_FOREVER)
  {
    while (buf_empty(&rcv_buffer));
    buf_remove(&rcv_buffer, p);
  }
  else if (retry <= LCD_MAX_ATTEMPTS)
  {
    while (buf_empty(&rcv_buffer))
      if (retry-- == 0)
      {
        lcd_error = LCD_ERR_RECEIVE_PACKET;
        return;
      }
    buf_remove(&rcv_buffer, p);
  }
  else
  {
    lcd_error = LCD_ERR_NUM_ATTEMPTS;
  }

  /* If an error occured while removing packet from buffer, log type.      */
  /* If no error ocurred, log successful reception.                        */

  if (lcd_error == ERR_BOP_NOT_FOUND)
  {
    _lcd_state.r_bop_err_cnt++;
    _lcd_state.r_err_cnt++;
  }
  else if (lcd_error == ERR_CRC_INVALID)
  {
    _lcd_state.r_crce_err_cnt++;
    _lcd_state.r_err_cnt++;
  }
  else if (lcd_error == AOK)
    _lcd_state.r_valid_cnt++;
}


/***********************************************************************\
*                         lcd_transmit_packet                           *
*************************************************************************
*                                                                       *
* Function:     Adds the parameter packet to the transmit buffer if possible.*
*               If the transmit buffer is full an error is generated.   *
*               Otherwise, if the buffer is empty, the packet is transmitted *
*               immediately, and the packet information is inserted into the *
*               transmit buffer. Depending upon the retry value, the    *
*               function will perform as follows:                       *
*                                                                       *
*               LCD_DONT_WAIT    : return immediately, pkt added to buffer. *
*               LCD_WAIT_FOREVER : wait forever for packet to be transmitted.*
*               retry            : try retry times to transmit packet.  *
*                                                                       *
*               Currently, only the LCD_WAIT_FOREVER option is supported. *
*               This is equivalent to a one packet deep transmit buffer. *
*                                                                       *
* Input:        lcd_transmit_packet(                                    *
*                 lcd_packet p; packet to be transmitted.               *
*                 word retry;   number of times to try transmitting packet. *
*               );                                                      *
*                                                                       *
* Output:       Nothing.                                                *
*                                                                       *
* Globals:      lcd_error  : module LCD.C                               *
*               xmt_buffer : module BMF.C                               *
*               _lcd_state : module LCD.C                               *
*                                                                       *
* Edit History: 07/08/90 - Written by Robin T. Laird.                   *
*                                                                       *
\***********************************************************************/

void lcd_transmit_packet(p, retry)
lcd_packet p;
```



```
word retry;
{
  lcd_error = AOK;                        /* Assume function successful...    */

  /* Insert packet into transmit buffer and send it.                          */
  /* If the transmit buffer is empty then the transmitter is disbaled:        */
  /*   Reset the transmitter source address and byte count.                   */
  /*   Re-start the transmitter (it was turned off when buf empty).           */
  /* Else, just insert packet into buffer for transmission (sometime later).*/
  /* If the insert failed, abort transmission and return an error.            */

  if (buf_empty(&xmt_buffer))
  {
    buf_insert(&xmt_buffer, p);
    if (lcd_error != AOK)
    {
      lcd_error = LCD_ERR_TRANSMIT_PACKET;
      return;
    }
    else lcc_start_xmt();
  }
  else
  {
    buf_insert(&xmt_buffer, p);
    if (lcd_error != AOK)
    {
      lcd_error = LCD_ERR_TRANSMIT_PACKET;
      return;
    }
  }

  /* At a later date, it will be possible to add a packet to the transmit   */
  /* buffer and then return to the calling function immediately, the packet */
  /* would be transmitted when it moved to the front of the buffer.         */
  /* A retry of zero would indicate that the packet is to be transmitted */
  /* in the above manner (i.e., don't wait for packet to be transmitted).   */

  /* For now, always wait for completion.                                   */

  if (retry == LCD_DONT_WAIT)
  {
    lcd_error = LCD_ERR_TRANSMIT_PACKET;
  }
  else if (retry == LCD_WAIT_FOREVER)
  {
    while (!buf_empty(&xmt_buffer));
  }
  else if (retry <= LCD_MAX_ATTEMPTS)
  {
    while (!buf_empty(&xmt_buffer))
      if (retry-- == 0)
      {
        lcd_error = LCD_ERR_TRANSMIT_PACKET;
        return;
      }
  }
  else
  {
    lcd_error = LCD_ERR_NUM_ATTEMPTS;
  }

  /* If an error occured while inserting packet into buffer, log type.      */
  /* If no error ocurred, log successful transmission.                      */

  if (lcd_error != AOK)
    _lcd_state.x_err_cnt++;
  else
    _lcd_state.x_valid_cnt++;
}


/***********************************************************************\
*                              lcd_status                               *
```

```
 ***************************************************************************
 *                                                                         *
 * Function:     Returns the operational status of the LCD subsystem.     *
 *               This is accomplished by returning the contents of the    *
 *               module state structure that records various operational  *
 *               parameters such as the number of valid packets received, etc.*
 *                                                                         *
 * Input:        lcd_status(                                               *
 *                 lcd_state *s  pointer to module state structure.        *
 *               );                                                        *
 *                                                                         *
 * Output:       Nothing.                                                  *
 *                                                                         *
 * Globals:      lcd_error : module LCD.C                                  *
 *                                                                         *
 * Edit History: 07/09/90 - Written by Richard P. Smurlo.                  *
 *                                                                         *
\***************************************************************************/

void lcd_status(s)
lcd_state *s;
{
  lcd_error = AOK;                        /* Assume function successful...    */

  *s = _lcd_state;
}
```

```
/**********************************************************************\
*                              LCI.H                                   *
************************************************************************
*                                                                      *
* CPCI:         IED90-MRA-COM-LCS-LCI-H-ROCO                           *
*                                                                      *
* Description:  LCS communications interface variables and functions.  *
*               Contains constant function parameter declarations as well *
*               as function return values (for success and failure of all *
*               operations). Contains the function prototypes for the LCI.C *
*               module.                                                *
*                                                                      *
*               Module LCI exports the following types/variables/functions: *
*                                                                      *
*               typedef lci_message;                                   *
*                                                                      *
*               int lci_error;                                         *
*                                                                      *
*               lci_init();                                            *
*               lci_receive_message();                                 *
*               lci_send_message();                                    *
*                                                                      *
* Notes:        1) See SDS pp. 5-6 through 5-x for more information.   *
*                                                                      *
* Edit History: 08/14/90 - Written by Robin T. Laird.                  *
*                                                                      *
\**********************************************************************/

/* Public Data Structures:                                             */

#ifndef LCI_MODULE_CODE
#define LCI_MODULE_CODE           4000

#define LCI_ERR_NOT_INIT          1+LCI_MODULE_CODE
#define LCI_ERR_RECEIVE_MESSAGE 2+LCI_MODULE_CODE
#define LCI_ERR_SEND_MESSAGE      3+LCI_MODULE_CODE

/* Maximum and minimum retry values for send/receive of messages.      */
/* Values must correspond with related definitions in module LCD.H.    */

#define LCI_WAIT_FOREVER          65535
#define LCI_DONT_WAIT             0
#define LCI_MAX_ATTEMPTS          60000

/* Maximum message length SHOULD be two less than maximum frame length. */

#define LCI_MAX_MESSAGE_LENGTH  SYS_MAX_PACKET_SIZE

/* MRA inter-module message type defined as a sequence of bytes.       */

typedef byte lci_message[LCI_MAX_MESSAGE_LENGTH];

/* External module global error variable.                              */

extern int lci_error;

/* Public Functions:                                                   */

void lci_init();
void lci_receive_message(lci_message m, word retry);
void lci_send_message(lci_message m, word retry);

#endif
```

```
/**************************************************************************\
*                                  LCI.C                                   *
****************************************************************************
*                                                                          *
* CPCI:          IED90-MRA-COM-LCS-LCI-C-R0C0                              *
*                                                                          *
* Description:   LCS communications interface functions.                   *
*                Implements the MRA standard Local Communications          *
*                Interface (LCI) module. This module contains the standard *
*                communications interface functions that provide higher-level *
*                software access to the Local Communications Channel of the *
*                host processing system (ICN or MPU).                      *
*                                                                          *
*                Message format at this level (ISO OSI network layer) is:  *
*                                                                          *
*                 | byte 0 | byte 1 | byte 2 | byte 3 | byte 4 | byte n|   *
*                 |----------------------------------------------------|   *
*                 |  DEST  | LENGTH | SOURCE |  xxxx  |  xxxx  |  .... |   *
*                                                                          *
*                Module LCI exports the following variables/functions:    *
*                                                                          *
*                int lci_error;                                            *
*                                                                          *
*                lci_init();                                               *
*                lci_receive_message();                                    *
*                lci_send_message();                                       *
*                                                                          *
* Notes:         1) The LCI functions are implementation independent.      *
*                2) Module LCI represents the MRA Communications Level.    *
*                                                                          *
* Edit History: 08/14/90 - Written by Robin T. Laird.                      *
*                                                                          *
\**************************************************************************/

#include <sysdefs.h>                    /* System constants and types.     */
#include "lcd.h"                        /* LCD public literals/functions.  */
#include "lci.h"                        /* LCI public literals/functions.  */

/* Public Variables:                                                       */

/* Global module error variable, lci_error.                                */
/* lci_error contains code of last error occurence.                        */
/* Should be set to AOK after each successful function call.               */
/* Variable can be examined by other software after each function call.    */

XDATA int lci_error = LCI_ERR_NOT_INIT; /* Global module error variable.   */


/**************************************************************************\
*                                 lci_init                                 *
****************************************************************************
*                                                                          *
* Function:      Initializes the Local Communications Interface.           *
*                The Local Communications Devie Handler (LCD) subsystem    *
*                along with all module variables are initialized. Any errors *
*                are examined for severity and an attempt is made to recover *
*                from non-fatal conditions. If initialization is unsuccessful *
*                then the error LCI_ERR_NOT_INIT is returned in lci_error. *
*                                                                          *
* Input:         lci_init();                                               *
*                                                                          *
* Output:        Nothing.                                                  *
*                                                                          *
* Globals:       lci_error : module LCI.C                                  *
*                lcd_error : module LCD.C                                  *
*                                                                          *
* Edit History: 07/28/90 - Written by Robin T. Laird.                      *
*                                                                          *
\**************************************************************************/

void lci_init()
{
  lci_error = AOK;                      /* Assume function successful...   */
```



```

  /* Initialize the LCD subsystem (sets up LSC/LPC hardware and software). */

  lcd_init();
  if (lcd_error != AOK)
  {
    switch(lcd_error)
    {
      case LCD_FAIL_RECEIVER:
      case LCD_FAIL_TRANSMITTER:
        lci_error = LCI_ERR_NOT_INIT;
        break;
      default:
        lci_error = lcd_error;
    }
  }
}


/**************************************************************************\
*                          lci_receive_message                            *
****************************************************************************
*                                                                          *
* Function:      Receives the latest (oldest) message from the LSC/LPC.   *
*                If a message is not immediately available, the function   *
*                waits for a specific period of time for an incoming message, *
*                and then returns regardless. If a message is never received, *
*                then the global variable lci_error is set to the literal  *
*                LCI_ERR_RECEIVE_MESSAGE. If a message is available then it *
*                is returned immediately.                                  *
*                                                                          *
*                The number of receive errors is tracked so that if it     *
*                exceeds a maximum value over time, the LCS device handler *
*                is reset to try and remedy the problem.                   *
*                                                                          *
* Input:         lci_receive_message(                                      *
*                  lci_message m; received message.                        *
*                  word retry;    number of times to try receiving message. *
*                );                                                        *
*                                                                          *
* Output:        Nothing.                                                  *
*                                                                          *
* Globals:       lci_error : module LCI.C                                  *
*                lcd_error : module LCD.C                                  *
*                                                                          *
* Edit History: 08/11/90 - Written by Robin T. Laird.                      *
*                                                                          *
\**************************************************************************/

#define MAX_RECV_ERRORS         1000    /* On > 1000 errors, reset LCD.    */

void lci_receive_message(m, retry)
lci_message m;
word retry;
{
  lcd_state status;                     /* Holds LCD status.               */

  lci_error = AOK;                      /* Assume function successful...   */

  /* Request packet. Iterate retry number of times.                        */

  lcd_receive_packet(m, retry);

  /* If a packet is not available, check integrity of receiver.            */
  /* If number of receive errors is excessive then reset the LCD subsystem. */
  /* If a packet is available then return it.                              */

  if (lcd_error != AOK)
  {
    lcd_status(&status);
    if (status.r_err_cnt > MAX_RECV_ERRORS) lcd_reset();
    lci_error = LCI_ERR_RECEIVE_MESSAGE;
  }
```

```
}


/***********************************************************************\
*                           lci_send_message                           *
*************************************************************************
*                                                                       *
* Function:     Sends the parameter message to the LSC/LPC.            *
*               If the message cannot be sent immediately, the function *
*               waits a specific period of time for the transmission and*
*               then returns regardless. If the message is never sent, then *
*               the global variable lci_error is set to LCI_ERR_SEND_MESSAGE.*
*                                                                       *
*               The number of send errors is tracked so that if it     *
*               exceeds a maximum value over time, the LCS device handler *
*               is reset to try and remedy the problem.                 *
*                                                                       *
* Input:        lci_send_message(                                       *
*                 lci_message m; message to be sent.                    *
*                 word retry;    number of times to try sending message. *
*               );                                                      *
*                                                                       *
* Output:       Nothing.                                                *
*                                                                       *
* Globals:      lci_error : module LCI.C                                *
*               lcd_error : module LCD.C                                *
*                                                                       *
* Edit History: 08/11/90 - Written by Robin T. Laird.                  *
*                                                                       *
\***********************************************************************/

#define MAX_SEND_ERRORS        1000    /* > 1000 errors reset LCD.      */

void lci_send_message(m, retry)
lci_message m;
word retry;
{
  lcd_state status;                    /* Holds LCD status.             */

  lci_error = ACK;                     /* Assume function successful... */

  /* Send packet. Iterate retry number of times.                        */

  lcd_transmit_packet(m, retry);

  /* If packet could not be transmitted, check integrity of transmitter. */
  /* If number of transmit errors is excessive then reset the LCD subsystem.*/

  if (lcd_error != AOK)
  {
    lcd_status(&status);
    if (status.x_err_cnt > MAX_SEND_ERRORS) lcd_reset();
    lci_error = LCI_ERR_SEND_MESSAGE;
  }
}
```

```
#*****************************************************************#
#                           MAKEFILE                              #
#*****************************************************************#
#                                                                 #
# CPCI:          IED90-MRA-COM-MMS-MAKEFILE-TXT-ROCO              #
#                                                                 #
# Description:   Makefile for the Modular Robotic Architecture (MRA).  #
#                Makes the common method manager subsystem.       #
#                                                                 #
#                Targets are available for the following systems/subsystems: #
#                                                                 #
#                mms   - COM Method Manager Subsystem             #
#                lib   - Add modules to MRA library               #
#                print - Print COM method manager files           #
#                                                                 #
# Notes:         1) The dependency and production rules are included here. #
#                2) See also \mra\makefile.                       #
#                                                                 #
# Edit History: 03/22/91 - Written by Robin T. Laird.             #
#                                                                 #
#*****************************************************************#


################################### RULES ###################################

.SUFFIXES : .hex .exe .obj .c .a51

# Control settings for Franklin 8031 development

CC          =c51
AS          =a51
LINK        =l51
OTOH        =ohs51
CFLAGS      =cd la db sb
ASFLAGS     =
LFLAGS      =
OFLAGS      =
STARTUP     =\c51\crom.obj
CODESEG     =00000h
XDATASEG    =00000h

# Control settings for Microsoft MS-DOS development

MSC         =cl
MSAS        =masm
MSLINK      =link
MSCFLAGS    =/AL /c /Oi /Zi /od
MSASFLAGS   =
MSLNKFLGS   =/co
LOADLIBES   =

.c.obj :
        $(CC) $< $(CFLAGS)

.a51.obj :
        $(AS) $< $(ASFLAGS)

.obj.exe :
        $(LINK) $(STARTUP),$< TO $@ code ($(CODESEG)) xdata ($(XDATASEG)) ixref

.exe.hex :
        $(OTOH) $< $(OFLAGS)


################################ DEFINITIONS ################################

# Project, system, and application level definitions

PROJ            = mra
APPSYS          = app
COMSYS          = com
```



```
ICNSYS          = icn
MPUSYS          = mpu

COMLIB          = \$(PROJ)\lib
COMSRC          = \$(PROJ)\$(COMSYS)\src
COMBIN31        = \$(PROJ)\$(COMSYS)\bin\8031
COMBIN152       = \$(PROJ)\$(COMSYS)\bin\80152
COMBINMS        = \$(PROJ)\$(COMSYS)\bin\msdos
COMBINSBC8      = \$(PROJ)\$(COMSYS)\bin\sbc8

# Common subsystem level source directories

HDRSRC          = $(COMSRC)\hdr
LCSSRC          = $(COMSRC)\lcs
MMSSRC          = $(COMSRC)\mms

# Common subsystem global include a · compilation units

SYSDEFS         = $(HDRSRC)\sysdefs.h

MMS             = $(COMBIN152)\mm.obj           $(COMBIN152)\pb.obj     \
                  $(COMBIN152)\lmmdct.obj       $(COMBIN152)\smmlib.obj \
                  $(COMBIN31)\mm.obj            $(COMBIN31)\pb.obj      \
                  $(COMBIN31)\lmmdct.obj        $(COMBIN31)\smmlib.obj  \
                  $(COMBINMS)\mm.obj            $(COMBINMS)\pb.obj      \
                  $(COMBINMS)\lmmdct.obj        $(COMBINMS)\smmlib.obj



################################## TARGETS ##################################

mms             : $(MMS)

lib             : $(MMS)
                  -lib51 delete $(COMLIB)\mra_1521.lib (mm, pb, lmmdct, smmlib)
                  -lib51 add $(COMBIN152)\mm.obj to $(COMLIB)\mra_1521.lib
                  -lib51 add $(COMBIN152)\pb.obj to $(COMLIB)\mra_1521.lib
                  -lib51 add $(COMBIN152)\lmmdct.obj to $(COMLIB)\mra_1521.lib
                  -lib51 add $(COMBIN152)\smmlib.obj to $(COMLIB)\mra_1521.lib
                  -lib51 delete $(COMLIB)\mra_311.lib (mm, pb, lmmdct, smmlib)
                  -lib51 add $(COMBIN31)\mm.obj to $(COMLIB)\mra_311.lib
                  -lib51 add $(COMBIN31)\pb.obj to $(COMLIB)\mra_311.lib
                  -lib51 add $(COMBIN31)\lmmdct.obj to $(COMLIB)\mra_311.lib
                  -lib51 add $(COMBIN31)\smmlib.obj to $(COMLIB)\mra_311.lib
                  -lib $(COMLIB)\mra_mss.lib -mm.obj+$(COMBINMS)\mm.obj;
                  -lib $(COMLIB)\mra_msl.lib -mm.obj+$(COMBINMS)\mm.obj;
                  -lib $(COMLIB)\mra_mss.lib -pb.obj+$(COMBINMS)\pb.obj;
                  -lib $(COMLIB)\mra_msl.lib -pb.obj+$(COMBINMS)\pb.obj;
                  -lib $(COMLIB)\mra_mss.lib -lmmdct.obj+$(COMBINMS)\lmmdct.obj
                  -lib $(COMLIB)\mra_msl.lib -lmmdct.obj+$(COMBINMS)\lmmdct.obj
                  -lib $(COMLIB)\mra_mss.lib -smmlib.obj+$(COMBINMS)\smmlib.obj
                  -lib $(COMLIB)\mra_msl.lib -smmlib.obj+$(COMBINMS)\smmlib.obj
                  touch lib

print           : $(MMS)
                  %-a2ps -nf lmm.h | post
                  %-a2ps -nf lmm.c | post
                  %-a2ps -nf lmmdct.c | post
                  %-a2ps -nf mm.h | post
                  %-a2ps -nf mm.c | post
                  %-a2ps -nf pb.h | post
                  %-a2ps -nf pb.c | post
                  %-a2ps -nf smm.h | post
                  %-a2ps -nf smm.c | post
                  %-a2ps -nf smmlib.c | post
                  touch print



############################# COM MMS DEPENDENCIES ##########################

# COM Local Method Manager dictionary module dependencies
```

```
LMMDCT = $(SYSDEFS) $(MMSSRC)\mm.h $(MMSSRC)\lmm.h $(MMSSRC)\lmmdct.c

$(COMBIN152)\lmmdct.obj          : $(LMMDCT)
   $(CC) $(MMSSRC)\$*.c $(CFLAGS) df(I80152) pr($(MMSSRC)\$*.152) oj($(COMBIN152

$(COMBIN31)\lmmdct.obj           : $(LMMDCT)
   $(CC) $(MMSSRC)\$*.c $(CFLAGS) df(I8031) pr($(MMSSRC)\$*.31) oj($(COMBIN31)\$

$(COMBINMS)\lmmdct.obj           : $(LMMDCT)
   $(MSC) $(MSCFLAGS) /DIBMAT /Fs$(MMSSRC)\$*.at /Fo$(COMBINMS)\$* $(MMSSRC)\$*.


# COM System Method Manager library module dependencies

SMMLIB = $(SYSDEFS) $(MMSSRC)\mm.h $(MMSSRC)\smm.h $(MMSSRC)\smmlib.c

$(COMBIN152)\smmlib.obj          : $(SMMLIB)
   $(CC) $(MMSSRC)\$*.c $(CFLAGS) df(I80152) pr($(MMSSRC)\$*.152) oj($(COMBIN152

$(COMBIN31)\smmlib.obj           : $(SMMLIB)
   $(CC) $(MMSSRC)\$*.c $(CFLAGS) df(I8031) pr($(MMSSRC)\$*.31) oj($(COMBIN31)\$

$(COMBINMS)\smmlib.obj           : $(SMMLIB)
   $(MSC) $(MSCFLAGS) /DIBMAT /Fs$(MMSSRC)\$*.at /Fo$(COMBINMS)\$* $(MMSSRC)\$*.


# COM Phone Book module dependencies

PB = $(SYSDEFS) $(MMSSRC)\mm.h $(MMSSRC)\smm.h $(MMSSRC)\pb.h $(MMSSRC)\pb.c

$(COMBIN152)\pb.obj              : $(PB)
   $(CC) $(MMSSRC)\$*.c $(CFLAGS) df(I80152) pr($(MMSSRC)\$*.152) oj($(COMBIN152

$(COMBIN31)\pb.obj               : $(PB)
   $(CC) $(MMSSRC)\$*.c $(CFLAGS) df(I8031) pr($(MMSSRC)\$*.31) oj($(COMBIN31)\$

$(COMBINMS)\pb.obj               : $(PB)
   $(MSC) $(MSCFLAGS) /DIBMAT /Fs$(MMSSRC)\$*.at /Fo$(COMBINMS)\$* $(MMSSRC)\$*.


# COM Method Manager module dependencies

MM = $(SYSDEFS)           $(LCSSRC)\lci.h         $(MMSSRC)\lmm.h        \
     $(MMSSRC)\smm.h      $(MMSSRC)\mm.h          $(MMSSRC)\lmm.c        \
     $(MMSSRC)\smm.c      $(MMSSRC)\mm.c

$(COMBIN152)\mm.obj              : $(MM)
   $(CC) $(MMSSRC)\$*.c $(CFLAGS) df(I80152) pr($(MMSSRC)\$*.152) oj($(COMBIN152

$(COMBIN31)\mm.obj               : $(MM)
   $(CC) $(MMSSRC)\$*.c $(CFLAGS) df(I8031) pr($(MMSSRC)\$*.31) oj($(COMBIN31)\$

$(COMBINMS)\mm.obj               : $(MM)
   $(MSC) $(MSCFLAGS) /DIBMAT /Fs$(MMSSRC)\$*.at /Fo$(COMBINMS)\$* $(MMSSRC)\$*.
```

```
/***************************************************************************\
*                              LMM.H                                        *
*****************************************************************************
*                                                                           *
* CPCI:          IED90-MRA-COM-MMS-LMM-H-ROCO                               *
*                                                                           *
* Description:   Local Method Manager (LMM) variables and functions.        *
*                Contains constant function parameter declarations (#defines) *
*                as well as function return values (for success and failure *
*                of all operations). Contains the function prototypes for the *
*                LMM.C module.                                              *
*                                                                           *
*                Defines the dictionary functions for the inherited classes. *
*                Each .DCT file defines the dictionary for that MODBOT unit. *
*                The functions in the dictionary represent a unit's methods *
*                that are available to other units in the MODBOT system.    *
*                Only one module-specific dictionary is valid at a time.    *
*                                                                           *
*                Module LMM exports the following variables/functions:      *
*                                                                           *
*                int lmm_error;                                             *
*                                                                           *
*                lmm_init();                                                *
*                lmm_process();                                             *
*                lmm_fire();                                                *
*                lmm_generate_message();                                    *
*                lmm_translate_message();                                   *
*                                                                           *
*                int lmm_num_funcs;                                         *
*                int (*lmm_dictionary[]) ();                                *
*                                                                           *
*                v_obj_class[];                                             *
*                v_obj_superclass[];                                        *
*                                                                           *
*                v_unit_name[];                                             *
*                v_unit_reset;                                              *
*                                                                           *
* Notes:         1) See SDS pp. 5-6 through 5-x for more information.       *
*                                                                           *
* Edit History:  10/01/90 - Written by Robin T. Laird.                      *
*                                                                           *
\***************************************************************************/

#ifndef LMM_MODULE_CODE
#define LMM_MODULE_CODE                 8000

/* Public Data Structures:                                                  */

#define LMM_ERR_NOT_INIT                1+LMM_MODULE_CODE
#define LMM_ERR_MSG_TRANSLATION         2+LMM_MODULE_CODE
#define LMM_ERR_MSG_GENERATION          3+LMM_MODULE_CODE

/* External module global error variable.                                   */

extern int lmm_error;

/* Public Functions:                                                        */

void lmm_init();
void lmm_process(mm_message *m_in, mm_message *m_out, byte *status);
void lmm_fire(void);
void lmm_generate_message(mm_message *m_in, mm_message *m_out, byte *status);
void lmm_translate_message(mm_message *m_in, mm_message *m_out, byte *status);

/* Object class literals.                                                   */

#define OBJ_OBJECT_CLASS        "OBJECT"                /* class names      */
#define OBJ_MODULE_CLASS        "MODULE"

/* Object class methods and instance variables.                             */

int d_obj_class(), d_obj_superclass();
extern char v_obj_class[];
```



```
extern char v_obj_superclass[];

/* Unit class literals.                                                     */

#define UNIT_NOT_INIT           1                       /* unit status      */
#define UNIT_IDLE               2
#define UNIT_OFF_LINE           3

/* Unit class methods and instance variables.                               */

int d_unit_name(), d_unit_reset();
extern char v_unit_name[];
extern int  v_unit_reset;

/* Indicates the total number of functions in the dictionary.              */
/* Variable is declared and assigned a value in each individual unit.       */

extern int lmm_num_funcs;

/* Unit dictionary is an array of function (method) pointers.               */
/* Variable is declared and initialized in each individual unit.            */

extern int (*lmm_dictionary[]) ();

#endif
```

```
/**********************************************************************\
*                               LMM.C                                  *
************************************************************************
*                                                                      *
* CPCI:         IED90-MRA-COM-MMS-LMM-C-ROCO                           *
*                                                                      *
* Description:  MMS local method manager functions.                   *
*               Implements the MRA standard Local Method Manager (LMM) *
*               module. This module contains the functions required to *
*               process incoming messages that represent local method  *
*               activation requests from external processes. It manages *
*               the MPU Dictionary data structure that contains available *
*               local methods as executable functions.                 *
*                                                                      *
*               Module LMM exports the following variables/functions:  *
*                                                                      *
*               int lmm_error;                                         *
*                                                                      *
*               lmm_init();                                            *
*               lmm_process();                                         *
*               lmm_fire();                                            *
*               lmm_generate_message();                                *
*               lmm_translate_message();                               *
*                                                                      *
* Notes:        1) The LMM functions are implementation independent.   *
*               2) This module is NOT a stand-alone compilation unit.  *
*                  It is included by the module MM.C and is compiled there. *
*                  It is assumed that the file LMM.H is included before it. *
*                                                                      *
* Edit History: 12/20/90 - Written by Robin T. Laird.                  *
*                                                                      *
\**********************************************************************/

/* Public Variables:                                                   */

/* Global module error variable, lmm_error.                            */
/* lmm_error contains code of last error occurence.                    */
/* Should be set to AOK after each successful function call.           */
/* Variable can be examined by other software after each function call. */

XDATA int lmm_error = LMM_ERR_NOT_INIT; /* Global module error variable. */

/* Local method manager trigger condition table.                       */
/* Holds method activation conditions and corresponding method to activate. */
/* Var lmm_num_methods holds count of methods in trigger condition table. */

#define LMM_MAX_METHODS          1       /* Number of methods we can hold. */

typedef struct { mm_message    method;
                 word          period;
                 unsigned long fireat;
               } lmm_cond;

static XDATA int lmm_num_methods = 0;
static XDATA lmm_cond lmm_trigger[LMM_MAX_METHODS];


/**********************************************************************\
*                              lmm_init                                *
************************************************************************
*                                                                      *
* Function:     Initializes the Local Method Manager software subsystems. *
*               This includes initializing the module-specific subsystems *
*               via a call to the system function d_unit_reset().      *
*                                                                      *
* Input:        lmm_in t();                                            *
*                                                                      *
* Output:       Nothing.                                               *
*                                                                      *
* Globals:      lmm_error       : LMM.C                                *
*               lmm_num_methods : LMM.C                                *
*                                                                      *
* Edit History: 12/20/90 - Written by Robin T. Laird.                  *
```



```
*                                                                      *
\**********************************************************************/

void lmm_init()
{
  lmm_error = AOK;                        /* Assume function successful.   */

  /* Reset local method trigger table index to 0 (indicate table empty).   */

  lmm_num_methods = 0;

  /* Call module reset dictionary function (module specific initialization).*/

  d_unit_reset();
}


/**********************************************************************\
*                             lmm_process                              *
************************************************************************
*                                                                      *
* Function:     Processes the parameter message and updates the local method *
*               trigger table. Coordinates message translation and message *
*               generation. The input message is processed (translated) and *
*               any possible output or response message is generated for *
*               transmission to the originating unit.                  *
*                                                                      *
* Input:        lmm_process(                                           *
*                 mm_message *m_in;  pointer to decoded message to process. *
*                 mm_message *m_out; pointer to possible message to output. *
*                 byte *status;      pointer to method execution status. *
*               );                                                     *
*                                                                      *
* Output:       Nothing.                                               *
*                                                                      *
* Globals:      lmm_error       : LMM.C                                *
*               lmm_trigger     : LMM.C                                *
*               lmm_num_methods : LMM.C                                *
*                                                                      *
* Edit History: 12/20/90 - Written by Robin T. Laird.                  *
*                                                                      *
\**********************************************************************/

void lmm_process(m_in, m_out, status)
mm_message *m_in, *m_out;
byte *status;
{
  word period;

  lmm_error = AOK;                        /* Assume function successful...  */

  /* Check transaction category for periodic method activation.          */

  switch(m_in->trans_category)
  {
    case MM_PERIODIC_STATUS_REQUEST:

      /* Extract period from parameter field.                            */
      /* Period is one word long (ranges from 0 - 65535 ms).             */

      period = (word)m_in->parameter[MM_PPOS] << 8 | m_in->parameter[MM_PPOS+1]

      /* Shift remaining parameter bytes to reflect removal of period.   */
      /* This is done so that the local method doesn't see period.       */

      m_in->parameter_length -= MM_PLEN;
      memmove(m_in->parameter,
              &m_in->parameter[MM_PPOS+MM_PLEN],
              m_in->parameter_length);

      /* If the period is zero, remove method from trigger table.        */
      /* Currently, removing the method requires decrementing the index. */
      /* Later, we'll have to find the method first and then delete.     */
```

```
      /* The method would be looked up according to function and sequence #.*/

      if (period == 0)
      {
        lmm_num_methods = (lmm_num_methods ? lmm_num_methods-1 : 0);
        return;
      }
      else if (lmm_num_methods < LMM_MAX_METHODS)
      {
        /* Add method info to trigger table.                                  */

        lmm_trigger[lmm_num_methods].method = *m_in;
        lmm_trigger[lmm_num_methods].period = period;
        lmm_trigger[lmm_num_methods].fireat = period+rtc_time();
        lmm_num_methods++;
      }
      break;

    default:
      break;
  }

  /* Translate the message (activate function).                              */
  /* Input message, m_in,  contains translation information.                 */
  /* Output message, m_out, contains any results (in the parameter field).   */

  lmm_translate_message(m_in, m_out, status);

  /* Generate any required response using data from input message.           */
  /* Input message, m_in, contains data concerning required response.        */
  /* Output message, m_out, contains function results and message ACK, etc. */

  lmm_generate_message(m_in, m_out, status);
}

/**************************************************************************\
*                               lmm_fire                                   *
****************************************************************************
*                                                                          *
* Function:     Checks local method trigger table and fires appropriate    *
*               functions according to conditions set up in the table.     *
*               The conditions are set by external commands received from  *
*               other modules. The conditions are checked as often as      *
*               possible for possible execution of a function.             *
*                                                                          *
*               Currently, only temporal conditions are implemented.       *
*               This allows for periodic execution of functions.           *
*                                                                          *
* Input:        lmm_fire();                                                *
*                                                                          *
* Output:       Nothing.                                                   *
*                                                                          *
* Globals:      lmm_error       : LMM.C                                    *
*               lmm_trigger     : LMM.C                                    *
*               lmm_num_methods : LMM.C                                    *
*                                                                          *
* Edit History: 03/27/91 - Written by Robin T. Laird.                      *
*                                                                          *
\**************************************************************************/

#define LMM_SEND_ATTEMPTS       LCI_WAIT_FOREVER

void lmm_fire()
{
  int           i;
  byte          status;
  lci_message   l_out;
  mm_message    m_out;

  lmm_error = AOK;                        /* Assume function successful...    */

  /* Check number of methods in trigger table. If non-zero, continue.         */
```



```

  if (lmm_num_methods)
  {
    /* Check each method for activation.                                      */
    /* Compare current time to trigger time. If greater, then fire method.    */

    for (i = 0; i < lmm_num_methods; i++)
    {
      if (rtc_time() > lmm_trigger[i].fireat)
      {
        /* Process method as usual.                                            */
        /* Update next fire time.                                              */

        lmm_trigger[i].fireat = lmm_trigger[i].period + rtc_time();
        lmm_translate_message(&lmm_trigger[i].method, &m_out, &status);
        lmm_generate_message(&lmm_trigger[i].method, &m_out, &status);

        /* If response required, encode message, and send via LCI.            */

        if (status == MM_RESPONSE_REQUIRED)
        {
          mm_encode_message(&m_out, l_out);
          lci_send_message(l_out, LMM_SEND_ATTEMPTS);
        }
      }
    }
  }
}


/**************************************************************************\
*                          lmm_generate_message                            *
****************************************************************************
*                                                                          *
* Function:     Generates a response message, m_out, based on the parameter *
*               input message, m_in. The transaction category and the      *
*               function status indicate how to respond to the input       *
*               message. The transaction disposition field is modified to  *
*               either command unknown, executed or failure depending on   *
*               the function status value. Since the output message is a   *
*               response to the input message the source and destination   *
*               address are swapped in the output message. Unchanged fields *
*               are copied from the input message to the output message.   *
*                                                                          *
* Input:        lmm_decode(                                                *
*                 mm_message *m_in;   pointer to (old) translated message. *
*                 mm_message *m_out;  pointer to response/results message. *
*                 byte *status;       pointer to translated function status. *
*               );                                                         *
*                                                                          *
* Output:       Nothing.                                                   *
*                                                                          *
* Globals:      lmm_error : LMM.C                                          *
*                                                                          *
* Edit History: 12/20/90 - Written by Robin T. Laird.                      *
*                                                                          *
\**************************************************************************/

void lmm_generate_message(m_in, m_out, status)
mm_message *m_in, *m_out;
byte *status;
{
  /* First check and see if the input message requires a response.           */
  /* Certain transaction categories require responses, others don't.         */
  /* If status from translated function indicates no response, just return. */

  if (*status==MM_SUPPRESS_OUTPUT || m_in->trans_category==MM_CONTROL_NO_ACK)
  {
    *status = MM_NO_RESPONSE;
    lmm_error = AOK;
    return;
  }
```

```c
    /* Otherwise we have to generate a response message...                 */

    /* Swap source and destination addresses of output and input messages.  */

    m_out->dest_modbot_id = m_in->src_modbot_id;
    m_out->dest_unit_id   = m_in->src_unit_id;

    m_out->src_modbot_id  = m_in->dest_modbot_id;
    m_out->src_unit_id    = m_in->dest_unit_id;

    /* Sequence number of output message remains unchanged.                 */

    m_out->sequence_number = m_in->sequence_number;

    /* Transaction disposition of output message is set to function status.
    /* Status is one of:
    /* MM_COMMAND_RECEIVED : Message received OK, no results generated.     */
    /* MM_COMMAND_EXECUTED : Method executed OK, results in output message. */
    /* MM_COMMAND_UNKNOWN  : Invalid function ID, no method executed.       */
    /* MM_COMMAND_EXECUTION_FAILURE : Invalid parameter (bad parameter, etc.).*/

    m_out->trans_disposition = *status;

    /* Transaction category of output message remains unchanged.            */

    m_out->trans_category = m_in->trans_category;

    /* Function ID of output message remains unchanged.                     */

    m_out->function_id = m_in->function_id;

    /* Parameter length and paramater buffer are already in output message. */
    /* Set status to indicate that a response message is required.          */

    *status = MM_RESPONSE_REQUIRED;
    lmm_error = AOK;
  }


/*************************************************************************\
*                           lmm_translate_message                          *
***************************************************************************
*                                                                         *
* Function:     Translates the parameter input message (m_in) into a local *
*               method (function) activation based on the function ID. The *
*               parameter passing field is modified to hold function results *
*               (if any). The status variable indicates the success of the *
*               activated method. A status value of MM_COMMAND_EXECUTED   *
*               means all went well. If the function failed, then the exact *
*               reason for failure is encoded in the parameter field as a  *
*               byte value. The local method manager global error variable, *
*               lmm_error, is set to LMM_ERR_MSG_TRANSLATION if the local *
*               method failed.                                            *
*                                                                         *
* Input:        lmm_translate_message(                                    *
*                 mm_message *m_in;  pointer to message to translate.     *
*                 mm_message *m_out; pointer to message holding results.  *
*                 byte *status;      pointer to translated function status. *
*               );                                                        *
*                                                                         *
* Output:       Nothing.                                                  *
*                                                                         *
* Globals:      lmm_error     : LMM.C                                     *
*               lmm_num_funcs : DICTIONARY                                *
*                                                                         *
* Edit History: 01/04/91 - Written by Robin T. Laird.                     *
*                                                                         *
\*************************************************************************/

void lmm_translate_message(m_in, m_out, status)
mm_message *m_in, *m_out;
byte *status;
{
```



```c
    /* If function ID invalid, indicate command unknown and return.         */

    if (m_in->function_id > lmm_num_funcs)
    {
      *status   = MM_COMMAND_UNKNOWN;
      lmm_error = LMM_ERR_MSG_TRANSLATION;
    }
    else
    {
      /* Copy parameters (if any) to standard parameter input buffer.       */
      /* Execute function indicated by function ID in input message.        */
      /* Copy results from standard parameter output buffer to message buffer.*/
      /* Note that we have to adjust for bit output by testing bit index.   */

      memcpy(mm_stdin.buffer, m_in->parameter, (int)m_in->parameter_length);
      mm_stdin.index     = 0;
      mm_stdin.bitindex  = 0;
      mm_stdout.index    = 0;
      mm_stdout.bitindex = 0;

      *status = (*lmm_dictionary[m_in->function_id])();

      if (mm_stdout.bitindex) mm_stdout.index++;
      memcpy(m_out->parameter, mm_stdout.buffer, mm_stdout.index);
      m_out->parameter_length = mm_stdout.index;
      mm_stdin.index     = 0;
      mm_stdin.bitindex  = 0;
      mm_stdout.index    = 0;
      mm_stdout.bitindex = 0;

      /* If there is an error in executing the function above, then:        */
      /*   1. Status variable will be set to MM_COMMAND_EXECUTION_FAILURE.  */
      /*   2. Source of error will be encoded in parameter output buffer.   */
      /* A NULL status indicates that no response message is to be generated. */

      switch(*status)
      {
        case MM_INITIATING:
        case MM_COMMAND_RECEIVED:
        case MM_COMMAND_EXECUTED:
        case MM_SUPPRESS_OUTPUT:
          lmm_error = AOK;
          break;

        case MM_COMMAND_UNKNOWN:
        case MM_COMMAND_EXECUTION_FAILURE:
          lmm_error = LMM_ERR_MSG_TRANSLATION;
          break;

        default:
          break;
      }
    }
  }
```

```
/**********************************************************************\
*                              LMMDCT.C                                *
************************************************************************
*                                                                      *
* CPCI:          IED90-MRA-COM-MMS-LMMDCT-C-ROCO                       *
*                                                                      *
* Description:   Local Method Manager (LMM) dictionary functions.      *
*                Implements the inherited ("built-in") local methods.  *
*                                                                      *
*                Module LMMDCT.C exports the following variables/functions: *
*                                                                      *
*                int lmm_num_funcs;                                    *
*                                                                      *
*                OBJECT (STATUS_REQUEST):                              *
*                d_obj_class();                                        *
*                d_obj_superclass();                                   *
*                                                                      *
*                UNIT (STATUS_REQUEST):                                *
*                d_unit_name();                                        *
*                                                                      *
*                UNIT (CONTROL, CONTROL_WITH_ACK):                     *
*                d_unit_reset();                                       *
*                                                                      *
* Notes:         None.                                                 *
*                                                                      *
* Edit History: 01/09/91 - Written by Robin T. Laird.                  *
*                                                                      *
\**********************************************************************/

#include <sysdefs.h>                    /* System constants and types.   */
#include "mm.h"                         /* Method manager literals.      */
#include "lmm.h"                        /* Local method manager.         */

/* Object class methods...                                               */

int d_obj_class()
{
  mm_sprintfb(&mm_stdout, "%s", v_obj_class);
  return(MM_COMMAND_EXECUTED);
}

int d_obj_superclass()
{
  mm_sprintfb(&mm_stdout, "%s", v_obj_superclass);
  return(MM_COMMAND_EXECUTED);
}

/* Unit class methods...                                                 */

int d_unit_name()
{
  mm_sprintfb(&mm_stdout, "%s", v_unit_name);
  return(MM_COMMAND_EXECUTED);
}

/* Prototype is given below (all reset functions return an int).         */
/* The reset() function is defined in each MODBOT module dictionary (file). */

int reset();
int d_unit_reset()
{
  v_unit_reset = UNIT_IDLE;
  reset();
  return(MM_COMMAND_EXECUTED);
}
```

```
/************************************************************************\
*                               MM.H                                     *
**************************************************************************
*                                                                        *
* CPCI:         IED90-MRA-COM-MMS-MM-H-R0C2                              *
*                                                                        *
* Description:  Method Manager (MM) variables and functions.             *
*               Contains constant function parameter declarations (#defines) *
*               as well as function return values (for success and failure *
*               of all operations). Contains the function prototypes for the *
*               MM.C module.                                             *
*                                                                        *
*               Module MM exports the following types/variables/functions: *
*                                                                        *
*               typedef mm_pbuffer;                                      *
*                                                                        *
*               int mm_error;                                            *
*               mm_pbuffer mm_stdin;                                     *
*               mm_pbuffer mm_stdout;                                    *
*                                                                        *
*               mm_init();                                               *
*               mm_cycle();                                              *
*               mm_encode_message();                                     *
*               mm_decode_message();                                     *
*               mm_sprintfb();                                           *
*               mm_sscanfb();                                            *
*               mm_service_event();                                      *
*               mm_terminate_event();                                    *
*               mm_check_event();                                        *
*                                                                        *
* Notes:        1) See SDS pp. 5-6 through 5-x for more information.     *
*                                                                        *
* Edit History: 10/01/90 - Written by Robin T. Laird.                    *
*                                                                        *
\************************************************************************/

#ifndef MM_MODULE_CODE
#define MM_MODULE_CODE                      7000

#include <lci.h>                            /* Local communications interface.  */

/* Public Data Structures:                                                  */

#define MM_ERR_NOT_INIT                     1+MM_MODULE_CODE
#define MM_ERR_MESSAGE_NOT_AVAILABLE        2+MM_MODULE_CODE
#define MM_ERR_MESSAGE_DISPOSITION          3+MM_MODULE_CODE
#define MM_ERR_LOCAL_METHOD_INIT            4+MM_MODULE_CODE
#define MM_ERR_SYSTEM_METHOD_INIT           5+MM_MODULE_CODE
#define MM_ERR_IN_PHONEBOOK                 6+MM_MODULE_CODE
#define MM_ERR_NO_EVENT_PENDING             7+MM_MODULE_CODE

#define MM_CYCLE_FOREVER                    32767
#define MM_RESPONSE_REQUIRED                0
#define MM_NO_RESPONSE                      1

/* Events are assigned unique numbers as identifiers for later cross-ref.   */
/* Invalid method activations return value MM_NULL_EVENT.                   */

#define MM_NULL_EVENT                       -1
#define MM_AWAITING_EVENT                   0

/* Inter-module message format (IMMF) (ISO OSI presentation layer).         */
/* IMMF transaction disposition values.                                     */

#define MM_INITIATING                       1
#define MM_COMMAND_RECEIVED                 2
#define MM_COMMAND_EXECUTED                 3
#define MM_COMMAND_UNKNOWN                  4
#define MM_COMMAND_EXECUTION_FAILURE        5
#define MM_SUPPRESS_OUTPUT                  6

/* IMMF transaction category values.                                        */
```



```
#define MM_CONTROL_WITH_ACK                 1
#define MM_CONTROL_NO_ACK                   2
#define MM_STATUS_REQUEST                   3
#define MM_PERIODIC_STATUS_REQUEST          4
#define MM_QUERY_CONTROL                    5
#define MM_SET_ALARM_LIMITS                 6
#define MM_QUERY_ALARM_LIMITS               7
#define MM_SET_OPERATING_LIMITS             8
#define MM_QUERY_OPERATING_LIMITS           9
#define MM_INDICATION                       10
#define MM_ALARM_ACTIVATED                  11
#define MM_ALARM_RETIRED                    12
#define MM_COMMAND_EXECUTION_INDICATION     13
#define MM_COMMAND_FAILED_INDICATION        14

/* Type definition for IMMF, used to exchange info between functions.       */

#define MM_MAX_PARAM_SIZE                   SYS_MAX_PACKET_SIZE

typedef struct { byte dest_modbot_id;
                 byte dest_unit_id;
                 byte message_length;
                 byte src_modbot_id;
                 byte src_unit_id;
                 byte sequence_number;
                 byte trans_disposition;
                 byte trans_category;
                 byte function_id;
                 byte parameter_length;
                 byte parameter[MM_MAX_PARAM_SIZE];
               } mm_message;

/* Type definition for parameter buffer, used for formatted I/O of data.    */
/* Index gives the next byte position within the buffer.                    */
/* Bit index gives next bit position within the current byte position.      */
/* Variables mm_stdin and mm_stdout are available for standard data I/O.    */

#define MM_PBUFFER_SIZE                     MM_MAX_PARAM_SIZE

typedef struct { byte buffer[MM_PBUFFER_SIZE];
                 int  index;
                 byte bitindex;
               } mm_pbuffer;

extern mm_pbuffer mm_stdin, mm_stdout;

/* External module global error variable.                                   */

extern int mm_error;

/* Public Functions:                                                        */

void mm_init();
void mm_cycle(int iterations);
void mm_encode_message(mm_message *to_encode, lci_message encoded);
void mm_decode_message(lci_message to_decode, mm_message *decoded);
void mm_sprintfb(mm_pbuffer *p, char *format, ...);
void mm_sscanfb(mm_pbuffer *p, char *format, ...);
void mm_service_event(int event, mm_message *m_out);
void mm_terminate_event(int event);
int  mm_check_event(int event);

#endif
```

```
/*************************************************************************\
*                                   MM.C                                  *
***************************************************************************
*                                                                         *
* CPCI:          IED90-MRA-COM-MMS-MM-C-R0C2                              *
*                                                                         *
* Description:   MMS (system) method manager functions.                   *
*                Implements the MRA standard Method Manager (MM) module.  *
*                This module contains the functions that control the local *
*                and remote method manager subsystems.                    *
*                                                                         *
*                Message format at this level (ISO OSI presentation layer) is:*
*                                                                         *
*                  |byte 0|byte 1|byte 2| byte 3 | byte 4 |byte 5|byte 6| *
*                  |----------------------------------------------------| *
*                  | DEST |      | SORC |        | TRANSC |      |      | *
*                  |DEVICE| MESG |DEVICE|SEQ NUMB|DISPOST/| FN ID|PARAMS| ... *
*                  |  ID  |LENGTH|  ID  |        | TRANSC |      |LENGTH| *
*                  |      |      |      |        |CATEGORY|      |      | *
*                                                                         *
*                Module MM exports the following variables/functions:     *
*                                                                         *
*                int mm_error;                                            *
*                mm_pbuffer mm_stdin;                                     *
*                mm_pbuffer mm_stdout;                                    *
*                                                                         *
*                mm_init();                                               *
*                mm_cycle();                                              *
*                mm_encode_message();                                     *
*                mm_decode_message();                                     *
*                mm_sprintfb();                                           *
*                mm_sscanfb();                                            *
*                mm_service_event();                                      *
*                mm_terminate_event();                                    *
*                mm_check_event();                                        *
*                                                                         *
* Notes:         1) The MM functions are implementation independent.      *
*                                                                         *
* Edit History: 12/20/90 - Written by Robin T. Laird.                     *
*                                                                         *
\*************************************************************************/

#include <string.h>                  /* Standard string functions.        */
#include <sysdefs.h>                 /* System constants and types.       */
#include <stdarg.h>                  /* Variable argument macros.         */
#include <rtc.h>                     /* Real-time clock functions.        */
#include "mm.h"                      /* Method manager.                   */
#include "pb.h"                      /* Method manager phone book.        */
#include "lmm.h"                     /* Local method manager.             */
#include "smm.h"                     /* System method manager.            */

#if defined(DEBUG)
#include <debug.h>
#endif

/* Public Variables:                                                      */

/* Global module error variable, mm_error.                                */
/* mm_error contains code of last error occurence.                        */
/* Should be set to AOK after each successful function call.              */
/* Variable can be examined by other software after each function call.   */

XDATA int mm_error = MM_ERR_NOT_INIT;   /* Global module error variable.  */

/* Global standard input and output parameter (passing) buffers.          */

XDATA mm_pbuffer mm_stdin, mm_stdout;

/* Definitions for field sizes/positions. Maintain with care!             */

#define MM_MODBOT_ID_BIT_SHIFT          5
#define MM_UNIT_ID_MASK                 0x1F
#define MM_TRANS_DISPOSIT_BIT_SHIFT     4
```



```
#define MM_TRANS_CATEGORY_MASK          0x0F

/* Position of period (in ms) in parameter field.                         */
/* Length of period in bytes.                                             */

#define MM_PPOS                         0
#define MM_PLEN                         2

/* Position of destination address in message.                            */
/* Number of overhead bytes at this communications level.                 */

#define MM_DEST_ADDR_POS                0
#define MM_COM_OVERHEAD_BYTES           7

/* Local method manager functions are included here.                      */

#include "lmm.c"

/* System method manager functions are included here.                     */

#include "smm.c"


/*************************************************************************\
*                                 mm_init                                 *
***************************************************************************
*                                                                         *
* Function:      Initializes the Local Method Manager software subsystems. *
*                This includes initializing data structures such as the   *
*                system phone book and the message print I/O buffers.     *
*                                                                         *
* Input:         mm_init();                                               *
*                                                                         *
* Output:        Nothing.                                                 *
*                                                                         *
* Globals:       mm_error   : MM.C                                        *
*                lmm_error  : LMM.C                                       *
*                smm_error  : SMM.C                                       *
*                mm_stdin   : MM.C                                        *
*                mm_stdout  : MM.C                                        *
*                                                                         *
* Edit History: 12/20/90 - Written by Robin T. Laird.                     *
*                                                                         *
\*************************************************************************/

void mm_init()
{
  int i;

  /* Initialize the local method manager (LMM).                           */

  lmm_init();
  if (lmm_error != AOK)
  {
    mm_error = MM_ERR_LOCAL_METHOD_INIT;
    return;
  }

  /* Initialize the system method manager (SMM).                          */

  smm_init();
  if (smm_error != AOK)
  {
    mm_error = MM_ERR_SYSTEM_METHOD_INIT;
    return;
  }

  /* Initialize the standard parameter input and output buffers.          */

  for (i = 0; i < MM_PBUFFER_SIZE; i++)
  {
    mm_stdin.buffer[i]  = 0;
    mm_stdout.buffer[i] = 0;
```

```
  }
  mm_stdin.index = mm_stdout.index = 0;
  mm_stdin.bitindex = mm_stdout.bitindex = 0;

  /* Initialize the system phone book data structure (NULL names, etc.).   */
  /* Update (create) phone book with current MODBOT/unit information.      */
  /* Reset mm_error if pb_init went OK (pb_init() affects mm_error).       */

  pb_init();
  if (pb_error != AOK)
  {
    mm_error = MM_ERR_IN_PHONEBOOK;
    return;
  }

  mm_error = AOK;                          /* Function successful.         */
}


/**************************************************************************\
*                                mm_cycle                                  *
****************************************************************************
*                                                                          *
* Function:     Processes the next message (or next several messages) from *
*               the local communications interface. The parameter iterations *
*               specifies how many messages to process. Messages are taken *
*               from the LCI, decoded, and then processed accordingly.     *
*               Messages that are initiating action are passed to the local *
*               method manager (LMM), while all others are passed to the   *
*               system method manager (SMM). Always cycles at least once.  *
*               If the number of iterations is MM_CYCLE_FOREVER then this  *
*               routine never returns (cycles continously processing local *
*               messages).                                                 *
*                                                                          *
*               Also checks local and system trigger conditions for possible *
*               firing of methods, which may cause additional messages to be *
*               output.                                                    *
*                                                                          *
* Input:        mm_cycle(                                                  *
*                 int iterations; number of messages to process.          *
*               );                                                         *
*                                                                          *
* Output:       Nothing.                                                   *
*                                                                          *
* Globals:      mm_error  : MM.C                                           *
*               lci_error : LCI.C                                          *
*                                                                          *
* Edit History: 12/20/90 - Written by Robin T. Laird.                      *
*                                                                          *
\**************************************************************************/

#define MM_SEND_ATTEMPTS        LCI_WAIT_FOREVER
#define MM_RECV_ATTEMPTS        8

void mm_cycle(iterations)
int iterations;
{
  int          i, inc;
  byte         status;
  lci_message  l_in, l_out;
  mm_message   m_in, m_out;

  /* Repeat iterations times...                                           */
  /* If iterations = MM_CYCLE_FOREVER then repeat forever (never return). */

  if (iterations == MM_CYCLE_FOREVER)
    inc = 0;
  else
    inc = 1;

  /* Get a message from the local communications subsystem (if available). */
  /* Decode message (network layer --> presentation layer).               */
  /* If the message is initiating action pass to local method manager.    */
```



```
  /* Else pass message to system method manager.                          */
  /* Ignore errors on multiple iterations.                                */

  i = 0;
  do {
    lci_receive_message(l_in, MM_RECV_ATTEMPTS);
    if (lci_error != AOK)
    {
      mm_error = MM_ERR_MESSAGE_NOT_AVAILABLE;
    }
    else
    {
      mm_error = AOK;

      mm_decode_message(l_in, &m_in);
      switch(m_in.trans_disposition)
      {
        case MM_INITIATING:

          /* Process the message as a local function activation.           */
          /* Status variable indicates whether or not response msg needed. */

          lmm_process(&m_in, &m_out, &status);
          break;

        case MM_COMMAND_RECEIVED:
        case MM_COMMAND_EXECUTED:
        case MM_COMMAND_UNKNOWN:
        case MM_COMMAND_EXECUTION_FAILURE:

          /* Process message as response to previously sent command.       */
          /* Status variable indicates whether or not response msg needed. */

          smm_process(&m_in, &m_out, &status);
          break;

        default:
          mm_error = MM_ERR_MESSAGE_DISPOSITION;
          continue;
      }

      /* If response required to above, encode message, send via LCI.     */

      if (status == MM_RESPONSE_REQUIRED)
      {
        mm_encode_message(&m_out, l_out);
        lci_send_message(l_out, MM_SEND_ATTEMPTS);
      }
    }

    /* Check local and system trigger conditions.                         */
    /* Fire appropriate methods. May cause messages to be sent.           */

    lmm_fire();
    smm_fire();

  } while ((i+=inc) < iterations);
}


/**************************************************************************\
*                             mm_encode_message                            *
****************************************************************************
*                                                                          *
* Function:     Encodes the information contained in the mm message        *
*               variable as a message of type lci_message. Each field of   *
*               the input structure is encoded in the correct position in  *
*               the output message as defined by the message format (below). *
*               The encoded message can then be transmitted to the local   *
*               communications device (by the LCI subsystem).              *
*                                                                          *
*                  MM message in --> ENCODE --> LCI message out            *
*                                                                          *
```

```
 *                The functions mm_decode_message() and mm_encode_message()  *
 *                define the message format at this communications level.   *
 *                Changes to this format must be reflected by changes to these *
 *                two functions (AND ONLY THESE TWO FUNCTIONS).              *
 *                                                                           *
 * Input:         mm_encode_message(                                         *
 *                  mm_message *to_encode; pointer to message to encode (in). *
 *                  lci_message encoded;   pointer to encoded message (out).  *
 *                );                                                         *
 *                                                                           *
 * Output:        Nothing.                                                   *
 *                                                                           *
 * Globals:       mm_error : MM.C                                            *
 *                                                                           *
 * Edit History: 12/20/90 - Written by Robin T. Laird.                       *
 *                                                                           *
\****************************************************************************/

void mm_encode_message(to_encode, encoded)
mm_message *to_encode;
lci_message encoded;
{
  byte i, j, l;

  /* Inter-module message format (IMMF):                                    */

  /*   DEST               SORC                                              */
  /*  DEVICE    MESG    DEVICE   SEQ   TRANSC     TRANSC    FN  PARAMS      */
  /*    ID     LENGTH     ID     NUM  DISPOSIT   CATEGORY   ID  LENGTH  PARAMS */
  /* ---------------------------------------------------------------------- */
  /*    8        8        8       8      4          4       8     8      8n  */

  mm_error = AOK;                         /* Assume function successful.   */

  i = MM_DEST_ADDR_POS;                   /* Index of first encoded byte.  */

  encoded[i]     = to_encode->dest_modbot_id << MM_MODBOT_ID_BIT_SHIFT;
  encoded[i++]  |= to_encode->dest_unit_id & MM_UNIT_ID_MASK;

  encoded[i++]   = to_encode->parameter_length + MM_COM_OVERHEAD_BYTES;

  encoded[i]     = to_encode->src_modbot_id << MM_MODBOT_ID_BIT_SHIFT;
  encoded[i++]  |= to_encode->src_unit_id & MM_UNIT_ID_MASK;

  encoded[i++]   = to_encode->sequence_number;

  encoded[i]     = to_encode->trans_disposition << MM_TRANS_DISPOSIT_BIT_SHIFT;
  encoded[i++]  |= to_encode->trans_category & MM_TRANS_CATEGORY_MASK;

  encoded[i++]   = to_encode->function_id;

  encoded[i++]   = l = to_encode->parameter_length;
  for (j = 0; j < l; j++) encoded[i++] = to_encode->parameter[j];
}


/****************************************************************************\
*                             mm_decode_message                              *
******************************************************************************
*                                                                            *
* Function:      Decodes the information contained in the mm_message         *
*                variable as a message of type lci_message. Each field of    *
*                the input structure is encoded in the correct position in   *
*                the output message as defined by the message format (below). *
*                                                                            *
*                     LCI message in --> DECODE --> MM message out           *
*                                                                            *
*                The functions mm_decode_message() and mm_encode_message()   *
*                define the message format at this communications level.     *
*                Changes to this format must be reflected by changes to these *
*                two functions (AND ONLY THESE TWO FUNCTIONS).               *
*                                                                            *
* Input:         mm_decode_message(                                          *
```



```
 *                  lci_message to_decode; pointer to message to decode (in). *
 *                  mm_message *decoded;   pointer to message to encode (out). *
 *                );                                                         *
 *                                                                           *
 * Output:        Nothing.                                                   *
 *                                                                           *
 * Globals:       mm_error : MM.C                                            *
 *                                                                           *
 * Edit History: 12/20/90 - Written by Robin T. Laird.                       *
 *                                                                           *
\****************************************************************************/

void mm_decode_message(to_decode, decoded)
lci_message to_decode;
mm_message *decoded;
{
  byte i, j, l;

  /* Inter-module message format (IMMF):                                    */

  /*   DEST               SORC                                              */
  /*  DEVICE    MESG    DEVICE   SEQ   TRANSC     TRANSC    FN  PARAMS      */
  /*    ID     LENGTH     ID     NUM  DISPOSIT   CATEGORY   ID  LENGTH  PARAMS */
  /* ---------------------------------------------------------------------- */
  /*    8        8        8       8      4          4       8     8      8n  */

  mm_error = AOK;                         /* Assume function successful.   */

  i = MM_DEST_ADDR_POS;                   /* Index of first decoded byte.  */

  decoded->dest_modbot_id    = to_decode[i] >> MM_MODBOT_ID_BIT_SHIFT;
  decoded->dest_unit_id      = to_decode[i++] & MM_UNIT_ID_MASK;

  decoded->message_length    = to_decode[i++];

  decoded->src_modbot_id     = to_decode[i] >> MM_MODBOT_ID_BIT_SHIFT;
  decoded->src_unit_id       = to_decode[i++] & MM_UNIT_ID_MASK;

  decoded->sequence_number   = to_decode[i++];

  decoded->trans_disposition = to_decode[i] >> MM_TRANS_DISPOSIT_BIT_SHIFT;
  decoded->trans_category    = to_decode[i++] & MM_TRANS_CATEGORY_MASK;

  decoded->function_id       = to_decode[i++];

  decoded->parameter_length  = l = to_decode[i++];
  for (j = 0; j < l; j++) decoded->parameter[j] = to_decode[i++];
}


/****************************************************************************\
*                                mm_sprintfb                                 *
******************************************************************************
*                                                                            *
* Function:      Provides formatted output of binary data as in sprintf().  *
*                Used to place data into the parameter passing portion of   *
*                a message (i.e., the end of the message). Variables are    *
*                output according to special % flags as in printf() but    *
*                as binary values NOT ASCII. A maximum number of values can *
*                be passed as parameters and is compiler dependent. The only *
*                "flags" that are supported indicate the types and length of *
*                parameters as in %10s. The format is similar to sprintf(). *
*                                                                            *
*                Only the type flags are currently supported.                *
*                                                                            *
*                Care must be taken to set/reset the buffer indexes between  *
*                function calls. Otherwise the buffer may overflow...        *
*                                                                            *
*                The types currently supported are:                          *
*                                                                            *
*                Character      Type             Output Length               *
*                -------------------------------------------------           *
*                   %y          byte             1 bit                       *
```

```
 *                      %b          byte            8  bits        *
 *                      %c          char            8  bits        *
 *                      %u          unsigned int   16  bits        *
 *                      %d          int            16  bits        *
 *                      %l          long int       32  bits        *
 *                      %s          pointer        8n  bits        *
 *                                                                 *
 * Input:        mm_sprintfb(                                      *
 *                 mm_pbuffer;    pointer to output buffer (where data goes).  *
 *                 char *format;  pointer to data output format string.        *
 *                 ...;           data values to be output.                    *
 *               );                                                *
 *                                                                 *
 * Output:       Nothing.                                          *
 *                                                                 *
 * Globals:      mm_error : MM.C                                   *
 *                                                                 *
 * Edit History: 12/20/90 - Written by Robin T. Laird.             *
 *                                                                 *
\*****************************************************************************/

void mm_sprintfb(p, format, a1, a2, a3)
mm_pbuffer *p;
char *format;
double a1, a2, a3;
{
  va_list ap;
  char    *c, *sval;
  int     ival;
  long    lval;

  /* Initialize the variable argument macro pointers.                    */

  va_start(ap, format);

  /* Loop through the format string, process according to control chars.   */
  /* Access arguments using va_arg(), inc index and bit index accordingly.  */
  /* Index and bitindex indicate number of bytes and bits in last byte.     */

  for (c = format; *c; c++)
  {
    if (*c != '%')
    {
      if (p->bitindex)
      {
        p->index++;
        p->bitindex = 0;
      }
      p->buffer[p->index++] = *c;
      continue;
    }

    switch(*++c)
    {
      case 'y': /* bit */
        if (p->bitindex == 0) p->buffer[p->index] = 0;

        /* Franklin and Microsoft differ in way bytes are managed.          */

        #if defined(MSDOS)
        p->buffer[p->index] |= (va_arg(ap, int) << p->bitindex);
        #else
        p->buffer[p->index] |= (va_arg(ap, byte) << p->bitindex);
        #endif

        if (++p->bitindex == 8)
        {
          p->bitindex = 0;
          p->index++;
        }
        break;

      case 'b': /* byte */
```



```
      case 'c': /* char */
        if (p->bitindex)
        {
          p->index++;
          p->bitindex = 0;
        }

        /* Franklin and Microsoft differ in way bytes are managed.          */

        #if defined(MSDOS)
        p->buffer[p->index++] = va_arg(ap, int);
        #else
        p->buffer[p->index++] = va_arg(ap, byte);
        #endif

        break;

      case 'u': /* unsigned int */
      case 'd': /* int */
        if (p->bitindex)
        {
          p->index++;
          p->bitindex = 0;
        }
        ival = va_arg(ap, int);
        p->buffer[p->index++] = ival >> 8;
        p->buffer[p->index++] = ival & 0x00FF;
        break;

      case 'l': /* long */
        if (p->bitindex)
        {
          p->index++;
          p->bitindex = 0;
        }
        lval = va_arg(ap, long);
        p->buffer[p->index++] = lval >> 24;
        p->buffer[p->index++] = lval >> 16;
        p->buffer[p->index++] = lval >>  8;
        p->buffer[p->index++] = lval & 0x000000FFL;
        break;

      case 's': /* pointer (string) */
        if (p->bitindex)
        {
          p->index++;
          p->bitindex = 0;
        }
        sval = va_arg(ap, char*);
        do {
          p->buffer[p->index++] = (byte)*sval;
        } while (*sval++);
        break;

      default:
        if (p->bitindex)
        {
          p->index++;
          p->bitindex = 0;
        }
        p->buffer[p->index++] = *c;
    }
  }

  /* Clean up after moving argument pointer.                              */

  va_end(ap);
}


/*****************************************************************************\
*                               mm_sscanfb                                    *
*******************************************************************************
```

```
 *
 * Function:      Provides formatted input of binary data as in sscanf().
 *                Used to extract data from the parameter passing portion of
 *                a message (i.e., the end of the message). Variables are
 *                input according to special % flags as in printf() but
 *                as binary values NOT ASCII. A maximum number of values can
 *                be passed as parameters and is compiler dependent. The only
 *                "flags" that are supported indicate the types and length of
 *                parameters as in %10s. The format is similar to sscanf().
 *
 *                Only the type flags are currently supported.
 *
 *                Care must be taken to set/reset the buffer indexes between
 *                function calls. Otherwise the buffer may overflow...
 *
 *                The types currently supported are:
 *
 *                Character        Type            Input Length
 *                --------------------------------------------------------
 *                   %y            byte            1  bit
 *                   %b            byte            8  bits
 *                   %c            char            8  bits
 *                   %u            unsigned int    16 bits
 *                   %d            int             16 bits
 *                   %l            long int        32 bits
 *                   %s            pointer         8n bits
 *
 * Input:         mm_sscanfb(
 *                  mm_pbuffer;   pointer to input buffer (source of data).
 *                  char *format; pointer to data input format string.
 *                  ...;          data values to be output.
 *                );
 *
 * Output:        Nothing.
 *
 * Globals:       mm_error : MM.C
 *
 * Edit History: 12/20/90 - Written by Robin T. Laird.
 *
\**********************************************************************/

void mm_sscan') .. format, a1, a2, a3)
mm_pbuffer *p;
char *format;
double a1, a2, a3;
{
  va_list ap;
  char    *c, *sval;
  int     ival;
  long    lval;

  /* Initialize the variable argument macro pointers.                  */

  va_start(ap, format);

  /* Loop through the format string, process according to control chars.   */
  /* Access arguments using va_arg(), inc index and bit index accordingly.  */
  /* Index and bitindex indicate number of bytes and bits in last byte.     */

  for (c = format; *c; c++)
  {
    if (*c != '%')
    {
      continue;
    }

    switch(*++c)
    {
      case 'y': /* bit */
        *(va_arg(ap, byte*)) = (p->buffer[p->index] >> p->bitindex) & 0x01;
        if (++p->bitindex == 8)
        {
          p->bitindex = 0;
```



```
          p->index++;
        }
        break;

      case 'b': /* byte */
      case 'c': /* char */
        if (p->bitindex)
        {
          p->index++;
          p->bitindex = 0;
        }
        *(va_arg(ap, byte*)) = p->buffer[p->index++];
        break;

      case 'u': /* unsigned int */
      case 'd': /* int */
        if (p->bitindex)
        {
          p->index++;
          p->bitindex = 0;
        }
        ival  = (int)p->buffer[p->index++] << 8;
        ival |= (int)p->buffer[p->index++];
        *(va_arg(ap, int*)) = ival;
        break;

      case 'l': /* long */
        if (p->bitindex)
        {
          p->index++;
          p->bitindex = 0;
        }
        lval  = (long)p->buffer[p->index++] << 24;
        lval |= (long)p->buffer[p->index++] << 16;
        lval |= (long)p->buffer[p->index++] <<  8;
        lval |= (long)p->buffer[p->index++];
        *(va_arg(ap, long*)) = lval;
        break;

      case 's': /* pointer (string) */
        if (p->bitindex)
        {
          p->index++;
          p->bitindex = 0;
        }
        sval = va_arg(ap, char*);
        do {
          *sval++ = (char)p->buffer[p->index];
        } while (p->buffer[p->index++]);
        break;

      default:
        break;
    }
  }

  /* Clean up after moving argument pointer.                             */

  va_end(ap);
}


/**********************************************************************\
 *                          mm_service_event                            *
 ************************************************************************
 *                                                                      *
 * Function:      Removes the completed event from the system method queue, *
 *                and returns the response message in the output parameter  *
 *                message (m_out). The parameter portion of the message is  *
 *                also copied to the standard input I/O buffer (mm_stdin).  *
 *                Periodic events are NOT removed from the queue, but the   *
 *                information portion of the event is still returned.       *
 *                                                                      *
```

```
 * Input:         mm_service_event(
 *                  Int event;          number of event to service.
 *                  mm_message *m_out; pointer to message holding response.
 *                );                                                          *
 *                                                                            *
 * Output:        Nothing.                                                    *
 *                                                                            *
 * Globals:       smm_error : module SMM.C                                    *
 *                item      : module SMM.C                                    *
 *                queue     : module SMM.C (side effect)                      *
 *                mm_error  : module MM.C                                     *
 *                mm_stdin  : module MM.C                                     *
 *                                                                            *
 * Edit History: 01/24/91 - Written by Robin T. Laird.                        *
 *                                                                            *
\****************************************************************************/

void mm_service_event(event, m_out)
int event;
mm_message *m_out;
{
  mm_error = AOK;                          /* Assume function successful.     */

  /* Remove event from method queue if event is NOT periodic.                 */
  /* Event is actually acting as a node pointer.                              */
  /* The event may not be at beginning of the queue.                          */
  /* Note that the parameter event is NOT checked for validity.               */

  if (item[event].info.trans_category == MM_PERIODIC_STATUS_REQUEST)
  {
    *m_out = item[event].info;

    /* Reset trans disposition to indicate no new message.                    */

    item[event].info.trans_disposition = MM_AWAITING_EVENT;
  }
  else
  {
    if (event == queue)
      smm_remove_q(&queue, m_out);
    else
      smm_remove_q(&event, m_out);
  }

  /* Copy parameter information from message to standard input buffer.        */

  memcpy(mm_stdin.buffer, m_out->parameter, (int)m_out->parameter_length);
  mm_stdin.index    = 0;
  mm_stdin.bitindex = 0;
}


/****************************************************************************\
 *                            mm_terminate_event                              *
 ******************************************************************************
 *                                                                            *
 * Function:      Removes the completed event from the system method queue.   *
 *                Periodic events are removed from the queue.                 *
 *                A message is sent to the event handler of the appropriate   *
 *                module to indicate that the specified periodic event should *
 *                be terminated (by setting its period to 0).                 *
 *                                                                            *
 * Input:         mm_terminate_event(                                         *
 *                  Int event; number of event to terminate.                  *
 *                );                                                          *
 *                                                                            *
 * Output:        Nothing.                                                    *
 *                                                                            *
 * Globals:       smm_error : module SMM.C                                    *
 *                queue     : module SMM.C (side effect)                      *
 *                mm_error  : module MM.C                                     *
 *                                                                            *
 * Edit History: 03/28/91 - Written by Robin T. Laird.                        *
```



```
 *                                                                            *
\****************************************************************************/

#define MM_TERM_ATTEMPTS                    LCI_WAIT_FOREVER

void mm_terminate_event(event)
int event;
{
  lci_message l_out;
  mm_message  m_out;

  mm_error = AOK;                          /* Assume function successful.     */

  /* Remove event from method queue.                                          */
  /* Event is actually acting as a node pointer.                              */
  /* The event may not be at beginning of the queue.                          */
  /* Note that the parameter event is NOT checked for validity.               */

  if (event == queue)
    smm_remove_q(&queue, &m_out);
  else
    smm_remove_q(&event, &m_out);

  /* Change message length accordingly (overhead plus period length).         */
  /* Change transaction disposition to indicate initiating command.           */
  /* Set parameter field to indicate zero (0) period.                         */

  m_out.message_length        = MM_COM_OVERHEAD_BYTES + MM_PLEN;
  m_out.trans_disposition     = MM_INITIATING;
  m_out.parameter_length      = MM_PLEN;
  m_out.parameter[MM_PPOS]    = 0;
  m_out.parameter[MM_PPOS+1]  = 0;

  /* Encode the above information for transmission via the LCI.               */
  /* And send it...                                                           */

  mm_encode_message(&m_out, l_out);
  lci_send_message(l_out, MM_TERM_ATTEMPTS);
}


/****************************************************************************\
 *                              mm_check_event                                *
 ******************************************************************************
 *                                                                            *
 * Function:      Returns the status of the parameter event. If the event     *
 *                has completed, then the transaction disposition is returned, *
 *                indicating the success of the method activation associated  *
 *                with the event. If the event has not completed, then the    *
 *                value MM_AWAITING_EVENT is returned. If the queue is empty  *
 *                then NULL is returned.                                      *
 *                                                                            *
 * Input:         mm_check_event(                                             *
 *                  Int event; number of event to check.                      *
 *                );                                                          *
 *                                                                            *
 * Output:        Disposition of completed event or MM_AWAITING_EVENT.        *
 *                                                                            *
 * Globals:       queue     : module SMM.C                                    *
 *                item      : module SMM.C                                    *
 *                mm_error  : module MM.C                                     *
 *                                                                            *
 * Edit History: 01/24/91 - Written by Robin T. Laird.                        *
 *                                                                            *
\****************************************************************************/

int mm_check_event(event)
int event;
{
  /* Make sure the queue is not empty (report error if it is).                */
  /* Then report AOK and return transaction disposition.                      */
  /* Note that the parameter event is NOT checked for validity.               */

```

```
    if (smm_empty_q(queue))
    {
      mm_error = MM_ERR_NO_EVENT_PENDING;
      return(NULL);
    }
    else
    {
      mm_error = AOK;
      return((int)item[event].info.trans_disposition);
    }
  }
```

```
/***********************************************************************\
*                              PB.H                                     *
*************************************************************************
*                                                                       *
* CPCI:         IED90-MRA-COM-MMS-PB-H-ROCO                             *
*                                                                       *
* Description:  Method Manager Phone Book variables and functions.      *
*               Contains constant function parameter declarations (#defines) *
*               as well as function return values (for success and failure *
*               of all operations). Contains the function prototypes for the *
*               PB.C module.                                            *
*                                                                       *
*               Module PB exports the following types/variables/functions: *
*                                                                       *
*               int pb_error;                                           *
*                                                                       *
*               pb_init();                                              *
*               pb_update_pb();                                         *
*               pb_lookup_pb();                                         *
*                                                                       *
* Notes:        1) See SDS pp. 5-6 through 5-x for more information.    *
*                                                                       *
* Edit History: 02/04/91 - Written by Robin T. Laird.                   *
*                                                                       *
\***********************************************************************/

#ifndef PB_MODULE_CODE
#define PB_MODULE_CODE                  11000

/* Public Data Structures:                                              */

#define PB_ERR_NOT_INIT                 1+PB_MODULE_CODE
#define PB_ERR_ADDING_EVENT             2+PB_MODULE_CODE
#define PB_ERR_UNIT_NOT_FOUND           3+PB_MODULE_CODE

/* External module global error variable.                               */

extern int pb_error;

/* Public Functions:                                                    */

void pb_init();
void pb_update(char *name);
int  pb_lookup(char *name, byte *modbot_addr, byte *unit_addr);

#endif
```

```
/**************************************************************************\
*                                 PB.C                                     *
****************************************************************************
*                                                                          *
* CPCI:         1ED90-MRA-COM-MMS-PB-C-ROCO                                *
*                                                                          *
* Description:  Phone Book manager functions.                              *
*               Implements the Method Manager Phone Book manager module.   *
*               This module contains the functions that create and update  *
*               the system Phone Book which contains the names and addresses *
*               of all units within the MODBOT system (along the MODBUS    *
*               copbunications network).                                   *
*                                                                          *
*               Module PB exports the following variables/functions:       *
*                                                                          *
*               int pb_error;                                              *
*                                                                          *
*               pb_init();                                                 *
*               pb_update_pb();                                            *
*               pb_lookup_pb();                                            *
*                                                                          *
* Notes:        1) The PB functions are implementation independent.        *
*                                                                          *
* Edit History: 02/04/91 - Written by Robin T. Laird.                      *
*                                                                          *
\**************************************************************************/

#include <string.h>                     /* Standard string functions.      */
#include <sysdefs.h>                    /* System constants and types.     */
#include <rtc.h>                        /* Real-time clock functions.      */
#include "mm.h"                         /* Method manager.                 */
#include "smm.h"                        /* System ethod manager.           */
#include "pb.h"                         /* System method manager.          */

#if defined(DEBUG)
#include <debug.h>
#endif

/* Public Variables:                                                       */

/* Global module error variable, pb_error.                                 */
/* pb_error contains code of last error occurence.                         */
/* Should be set to AOK after each successful function call.               */
/* Variable can be examined by other software after each function call.    */

XDATA int pb_error = PB_ERR_NOT_INIT;    /* Global module error variable.  */

/* The system phone book consists of listings for each unit on each MODBOT. */
/* Each MODBOT has an associated address and a number of unit entries.     */
/* Each phone book entry has a unit name and a unit address.               */
/* The phone book is sorted by unit name at system start-up (pb_update()).  */

#define MAX_NAME_LEN             8
#define MAX_UNITS                32
#define MAX_MODBOTS              1

typedef struct { char name[MAX_NAME_LEN];
                 byte unit_addr;
               } unit_entry;

typedef struct { unit_entry unit[MAX_UNITS];
                 byte       modbot_addr;
                 byte       num_entries;
               } modbot_entry;

static XDATA modbot_entry pb[MAX_MODBOTS];


/**************************************************************************\
*                                pb_init                                   *
****************************************************************************
*                                                                          *
* Function:     Initializes the Method Manager Phone Book data structure.  *
```



```
*               Number of phone book entries for each MODBOT is set to zero. *
*               Names and addresses are set to NULL.                       *
*                                                                          *
* Input:        pb_init();                                                 *
*                                                                          *
* Output:       Nothing.                                                   *
*                                                                          *
* Globals:      pb_error  : PB.C                                           *
*               pb        : PB.C                                           *
*                                                                          *
* Edit History: 12/20/90 - Written by Robin T. Laird.                      *
*                                                                          *
\**************************************************************************/

void pb_init()
{
  byte modbot, unit;

  pb_error = AOK;                            /* Assume function successful.     */

  /* Initialize the system phone book data structure (NULL names, etc.).   */
  /* Update (create) phone book with current MODBOT/unit information.      */

  for (modbot = 0; modbot < MAX_MODBOTS; modbot++)
  {
    pb[modbot].modbot_addr = 0;
    pb[modbot].num_entries = 0;
    for (unit = 0; unit < MAX_UNITS; unit++)
    {
      pb[modbot].unit[unit].name[0]   = '\0';
      pb[modbot].unit[unit].unit_addr = 0;
    }
  }
  pb_update(NULLPTR);
}


/**************************************************************************\
*                               pb_update                                  *
****************************************************************************
*                                                                          *
* Function:     Creates the system phone book (or updates the entry for    *
*               the unit that corresponds to the parameter name). If the   *
*               parameter name is NULLPTR, then each MODBOT is queried for *
*               available units connected to its local network. Units      *
*               that are capable of responding are added to the phone book *
*               along with their addresses. The function also updates an   *
*               entry for a given name (the same procedure is followed     *
*               except the function halts when the named unit is found     *
*               or when the maximum number of units has been searched).    *
*                                                                          *
* Input:        pb_update(                                                 *
*                 char *name; pointer to name of unit to update (or NULLPTR).*
*               );                                                         *
*                                                                          *
* Output:       Nothing.                                                   *
*                                                                          *
* Globals:      pb_error : PB.C                                            *
*               pb       : PB.C                                            *
*                                                                          *
* Edit History: 01/02/91 - Written by Robin T. Laird.                      *
*                                                                          *
\**************************************************************************/

#define ONE_TIME                 1      /* Cycle method manager once.      */
#define WAIT_TIME                1500   /* Time in ms to wait for response. */

void pb_update(name)
char *name;
{
  int           event, status;
  byte          modbot, unit;
  mm_message    m_in;
```

```
    unsigned long stop_at;

    pb_error = AOK;                          /* Assume function successful.     */

    /* Query each (even numbered) unit on each MODBOT (could take a while).    */
    /* Send "QUERY NAME" message to unit at current MODBOT/unit address.      */
    /* If the unit responds then add name to phone book at current address.   */
    /* If we're looking for a particular name, then stop when (if) found.     */
    /* Variable no_change indicates if anything changed in the phone book.    */

    for (modbot = 0; modbot < MAX_MODBOTS; modbot++)
      for (unit = 0; unit < MAX_UNITS; unit+=2)
      {
        if ((event = unit_name(modbot, unit)) == MM_NULL_EVENT)
        {
          pb_error = PB_ERR_ADDING_EVENT;
        }
        else
        {
          /* Check for event completion. Timeout period should be short.      */

          stop_at = rtc_time() + WAIT_TIME;
          do {
            mm_cycle(ONE_TIME);
            status = mm_check_event(event);
          } while (rtc_time() < stop_at && status == MM_AWAITING_EVENT);
          mm_service_event(event, &m_in);

          /* If event completed OK, then add entry to phonebook.              */

          if (status != MM_AWAITING_EVENT && mm_error == AOK)
          {
            mm_sscanfb(&mm_stdin, "%s", pb[modbot].unit[unit].name);
            pb[modbot].modbot_addr           = modbot;
            pb[modbot].unit[unit].unit_addr = unit;
            pb[modbot].num_entries++;

            /* If we're updating a particular unit, stop here if name match.  */

            if (name != NULLPTR && strcmp(name, pb[modbot].unit[unit].name) == 0)
            {
              pb_error = AOK;
              return;
            }
          }
          else
          {
            /* Delete (NULL) this entry since unit didn't respond.            */

            pb[modbot].unit[unit].name[0]    = '\0';
            pb[modbot].unit[unit].unit_addr = 0;
          }
        }
      }
}


/**************************************************************************\
*                               pb_lookup                                  *
****************************************************************************
*                                                                          *
* Function:      Looks up the parameter unit name in the sytem phonebook.  *
*                If the entry is found, then the corresponding MODBOT/unit *
*                address is returned. Otherwise, the global error variable *
*                pb_error is set to PB_ERR_UNIT_NOT_FOUND. A binary search *
*                method is used to locate the name in the phonebook.       *
*                                                                          *
* Input:         pb_lookup(                                                *
*                  char *name;         pointer to name of unit to look up. *
*                  byte *modbot_addr; pointer to returned MODBOT address.  *
*                  byte *unit_addr;   pointer to returned unit address.    *
*                );                                                        *
*                                                                          *
```



```
* Output:        Integer indicating if the named unit was found (TRUE/FALSE). *
*                                                                          *
* Globals:       pb_error : PB.C                                           *
*                pb       : PB.C                                           *
*                                                                          *
* Edit History: 01/22/91 - Written by Robin T. Laird.                      *
*                                                                          *
\**************************************************************************/

int pb_lookup(name, modbot_addr, unit_addr)
char *name;
byte *modbot_addr, *unit_addr;
{
  byte modbot, unit;
  byte low, high, mid, cond;

  pb_error = AOK;                            /* Assume function successful.   */

  /* Search each unit on each MODBOT. Linear search is used.                  */

  for (modbot = 0; modbot < MAX_MODBOTS; modbot++)
  {
    for (unit = 0; unit < MAX_UNITS; unit+=2)
    {
      if (strcmp(name, pb[modbot].unit[unit].name) == 0)
      {
        *modbot_addr = pb[modbot].modbot_addr;
        *unit_addr   = pb[modbot].unit[unit].unit_addr;
        return(TRUE);
      }
    }
  }

  /* Entry not found. Set error variable and return 0 for index.              */

  pb_error = PB_ERR_UNIT_NOT_FOUND;
  return(FALSE);
}
```

```
/***********************************************************************\
*                              SMM.H                                    *
*************************************************************************
*                                                                       *
* CPCI:         IED90-MRA-COM-MMS-SMM-H-ROC1                            *
*                                                                       *
* Description:  System Method Manager (SMM) variables and functions.    *
*               Contains constant function parameter declarations (#defines) *
*               as well as function return values (for success and failure *
*               of all operations). Contains the function prototypes for the *
*               SMM.C module.                                           *
*                                                                       *
*               Defines the library functions for the inherited classes. *
*               Each .LIB file specifies the functions available from the *
*               particular MODBOT unit. Collectively, these functions   *
*               represent all of the external functions (A.K.A., methods) *
*               that an application anticipates using.                  *
*                                                                       *
*               Module SMM exports the following variables/functions:   *
*                                                                       *
*               int smm error;                                          *
*                                                                       *
*               smm_init();                                             *
*               smm_process();                                          *
*               smm_fire();                                             *
*               smm_generate_message();                                 *
*               smm_translate_message();                                *
*                                                                       *
*               OBJECT (STATUS_REQUEST):                                *
*               obj_class();                                            *
*               obj_superclass();                                       *
*                                                                       *
*               UNIT (STATUS_REQUEST):                                  *
*               unit_name();                                            *
*                                                                       *
*               UNIT (CONTROL, CONTROL_WITH_ACK):                       *
*               unit_reset();                                           *
*                                                                       *
* Notes:        1) See SDS pp. 5-6 through 5-x for more information.    *
*                                                                       *
* Edit History: 10/01/90 - Written by Robin T. Laird.                   *
*                                                                       *
\***********************************************************************/

#ifndef SMM_MODULE_CODE
#define SMM_MODULE_CODE                   10000

/* Public Data Structures:                                              */

#define SMM_ERR_NOT_INIT                  1+SMM_MODULE_CODE
#define SMM_ERR_MSG_TRANSLATION           2+SMM_MODULE_CODE
#define SMM_ERR_MSG_GENERATION            3+SMM_MODULE_CODE
#define SMM_ERR_NO_EVENT_MATCH            4+SMM_MODULE_CODE

#define SMM_ERR_EMPTY_QUEUE               5+SMM_MODULE_CODE
#define SMM_ERR_FULL_QUEUE                6+SMM_MODULE_CODE

/* External module global error variable.                               */

extern int smm_error;

/* Public Functions:                                                    */

void smm_init();
void smm_process(mm_message *m_in, mm_message *m_out, byte *status);
void smm_fire(void);
void smm_generate_message(char *dest_unit_name, mm_message *m_in, int *event);
void smm_translate_message(mm_message *m_in, mm_message *m_out, byte *status);

/* Inherited Public Functions:                                          */

int obj_class(byte modbot, byte unit);
int obj_superclass(byte modbot, byte unit);
```



```
int unit_name(byte modbot, byte unit);
int unit_reset(byte modbot, byte unit);

/* Number of inherited functions (A.K.A., methods) is given below.      */
/* Includes object and unit class methods defined herein.               */

#define SMM_NUM_INHERITED_FNS   4

/* Function identification numbers for object class methods.            */

#define OBJ_FN_CLASS            0
#define OBJ_FN_SUPERCLASS       1

/* Function identification numbers for unit class methods.              */

#define UNIT_FN_NAME            2
#define UNIT_FN_RESET           3

#endif
```

```
/***********************************************************************\
*                              SMM.C                                    *
*************************************************************************
*                                                                       *
* CPCI:          IED90-MRA-COM-MMS-SMM-C-ROCI                           *
*                                                                       *
* Description:   MMS system method manager functions.                   *
*                Implements the MRA standard System Method Manager (SMM) *
*                module. This module contains the functions required to *
*                process incoming messages that represent system method *
*                activation requests from external processes. It manages *
*                the MPU method activation queue data structure that manages *
*                execution and response of MODBOT unit functions.       *
*                                                                       *
*                Module SMM exports the following variables/functions:  *
*                                                                       *
*                int smm_error;                                         *
*                                                                       *
*                smm_init();                                            *
*                smm_process();                                         *
*                smm_fire();                                            *
*                smm_generate_message();                                *
*                smm_translate_message();                               *
*                                                                       *
* Notes:         1) The SMM functions are implementation independent.   *
*                3) This module is NOT a stand-alone compilation unit.  *
*                   It is included by the module MM.C and is compiled there. *
*                   It is assumed that the file SMM.H is included before it. *
*                                                                       *
* Edit History: 12/20/90 - Written by Robin T. Laird.                   *
*                                                                       *
/***********************************************************************/

/* Public Variables:                                                    */

/* Global module error variable, smm_error.                             */
/* smm_error contains code of last error occurence.                     */
/* Should be set to AOK after each successful function call.            */
/* Variable can be examined by other software after each function call. */

XDATA int smm_error = SMM_ERR_NOT_INIT; /* Global module error variable. */

/* Local error values and list-specific literals.                       */

#define SMM_ERR_OVERFLOW              7+SMM_MODULE_CODE
#define SMM_ERR_INVALID_PTR           8+SMM_MODULE_CODE

#define SMM_NULL_PTR                  -1
#define SMM_MAX_NODES                 16

/* Node pointers for array-based queues are simply integer indexes.     */

typedef int nodeptr;

/* Node for doubly-linked queue has left, right pointers along with data.  */
/* The event identifiers match the node index (in item below) with the node.*/
/* So, for example, the event identifier for item[4] is 4.              */

typedef struct { mm_message info;
                 int event;
                 nodeptr left;
                 nodeptr right;
               } node;

/* Node "pool" is an array of uninitialized nodes.                      */

static XDATA node item[SMM_MAX_NODES];

/* Avail queue is initialized to hold all nodes.                        */
/* Only right pointers are maintained on avail list.                    */
/* Global queue contains events that require processing.                */

static XDATA nodeptr avail, queue;
```



```
/* System method manager trigger condition table.                          */
/* Holds method activation conditions and corresponding method to activate. */
/* Var smm_num_methods holds count of methods in trigger condition table.  */

#define SMM_MAX_METHODS          1        /* Number of methods we can hold.  */

typedef struct { mm_message    method;
                 word          period;
                 unsigned long fireat;
               } smm_cond;

static XDATA int smm_num_methods = 0;
static XDATA smm_cond smm_trigger[SMM_MAX_METHODS];


/***********************************************************************\
*                              smm_full_q                               *
*************************************************************************
*                                                                       *
* Function:     Function that returns the boolean of whether or not the *
*               parameter queue is full (TRUE if so, FALSE if not).     *
*                                                                       *
* Input:        smm_full_q(                                             *
*                 nodeptr p; pointer to (index of) beginning of queue.  *
*               );                                                      *
*                                                                       *
* Output:       Integer, TRUE if queue FULL, FALSE if queue not FULL.   *
*                                                                       *
* Globals:      avail : module SMM.C                                    *
*                                                                       *
* Edit History: 07/08/90 - Written by Robin T. Laird.                   *
*                                                                       *
\***********************************************************************/

static int smm_full_q(p)
nodeptr p;
{
  return (avail == SMM_NULL_PTR);
}


/***********************************************************************\
*                              smm_empty_q                              *
*************************************************************************
*                                                                       *
* Function:     Function that returns the boolean of whether or not the *
*               parameter queue is empty (TRUE if so, FALSE if not).    *
*                                                                       *
* Input:        smm_empty_q(                                            *
*                 nodeptr p; pointer to (index of) beginning of queue.  *
*               );                                                      *
*                                                                       *
* Output:       Integer, TRUE if queue EMPTY, FALSE if queue not EMPTY. *
*                                                                       *
* Globals:      None.                                                   *
*                                                                       *
* Edit History: 07/08/90 - Written by Robin T. Laird.                   *
*                                                                       *
\***********************************************************************/

static int smm_empty_q(p)
nodeptr p;
{
  return (p == SMM_NULL_PTR);
}


/***********************************************************************\
*                              smm_avail_init                           *
*************************************************************************
*                                                                       *
* Function:     Initializes the global available node list. The avail list *
```

```
 *                 is a singly-linked list that contains nodes available for    *
 *                 use on other list structures (like the global event queue).  *
 *                                                                               *
 * Input:          smm_avail_init(                                               *
 *                   nodeptr *p; pointer to beginning of avail list.             *
 *                 );                                                            *
 *                                                                               *
 * Output:         Nothing.                                                      *
 *                                                                               *
 * Globals:        smm_error : module SMM.C                                      *
 *                 item      : module SMM.C                                      *
 *                                                                               *
 * Edit History: 02/15/91 - Written by Robin T. Laird.                           *
 *                                                                               *
\*******************************************************************************/

static void smm_avail_init(p)
nodeptr *p;
{
  nodeptr q;

  smm_error = AOK;

  for (q = 0; q < SMM_MAX_NODES-1; q++)
  {
    item[q].event = q;
    item[q].left  = SMM_NULL_PTR;
    item[q].right = q+1;
  }
  item[SMM_MAX_NODES-1].event = q;
  item[SMM_MAX_NODES-1].left  = SMM_NULL_PTR;
  item[SMM_MAX_NODES-1].right = SMM_NULL_PTR;

  *p = 0;
}


/*******************************************************************************\
 *                                smm_get_node                                   *
 *********************************************************************************
 *                                                                               *
 * Function:       Function that returns a pointer to the next available node.  *
 *                 SMM_NULL_PTR is returned if the avail list is empty.          *
 *                                                                               *
 * Input:          smm_get_node();                                               *
 *                                                                               *
 * Output:         Pointer (index of) next available node (or SMM_NULL_PTR).    *
 *                                                                               *
 * Globals:        smm_error : module SMM.C                                      *
 *                 item      : module SMM.C                                      *
 *                 avail     : module SMM.C                                      *
 *                                                                               *
 * Edit History: 02/15/91 - Written by Robin T. Laird.                           *
 *                                                                               *
\*******************************************************************************/

static nodeptr smm_get_node()
{
  nodeptr p;

  if (avail == SMM_NULL_PTR)
  {
    smm_error = SMM_ERR_OVERFLOW;
    return(SMM_NULL_PTR);
  }
  else
  {
    smm_error = AOK;
    p = avail;
    avail = item[avail].right;
    return(p);
  }
}
```



```


/*******************************************************************************\
 *                                smm_free_node                                  *
 *********************************************************************************
 *                                                                               *
 * Function:       Returns the parameter node to the avail list.                 *
 *                 An error is generated if the node to be returned is NULL.     *
 *                                                                               *
 * Input:          smm_free_node(                                                *
 *                   nodeptr p; pointer to (index of) node to free.              *
 *                 );                                                            *
 *                                                                               *
 * Output:         Nothing.                                                      *
 *                                                                               *
 * Globals:        smm_error : module SMM.C                                      *
 *                 item      : module SMM.C                                      *
 *                 avail     : module SMM.C                                      *
 *                                                                               *
 * Edit History: 02/15/91 - Written by Robin T. Laird.                           *
 *                                                                               *
\*******************************************************************************/

static void smm_free_node(p)
nodeptr p;
{
  if (p == SMM_NULL_PTR)
  {
    smm_error = SMM_ERR_INVALID_PTR;
  }
  else
  {
    smm_error = AOK;
    item[p].right = avail;
    avail = p;
  }
}


/*******************************************************************************\
 *                                smm_clear_q                                    *
 *********************************************************************************
 *                                                                               *
 * Function:       Initializes the parameter queue and clears its contents.      *
 *                                                                               *
 * Input:          smm_clear_q(                                                  *
 *                   nodeptr *p; pointer to beginning of queue to clear.         *
 *                 );                                                            *
 *                                                                               *
 * Output:         Nothing.                                                      *
 *                                                                               *
 * Globals:        smm_error : module SMM.C                                      *
 *                                                                               *
 * Edit History: 07/09/90 - Written by Robin T. Laird.                           *
 *                                                                               *
\*******************************************************************************/

static void smm_clear_q(p)
nodeptr *p;
{
  smm_error = AOK;                        /* Assume function successful...    */
  *p = SMM_NULL_PTR;
}


/*******************************************************************************\
 *                                smm_print_q                                    *
 *********************************************************************************
 *                                                                               *
 * Function:       Prints the contents of the parameter queue on the local       *
 *                 output device (serial port). Only the event number and the    *
 *                 associated function ID are currently printed.                 *
 *                                                                               *
```

```
 * Input:          smm_print_q(
 *                   nodeptr *p; pointer to (index of) queue to be printed.
 *                 );
 *
 * Output:         Nothing.
 *
 * Globals:        item : module SMM.C
 *
 * Edit History: 02/15/91 - Written by Robin T. Laird.
 *
\*************************************************************************/

static void smm_print_q(p)
nodeptr p;
{
  nodeptr q;

  if (p == SMM_NULL_PTR)
  {
    #if defined(DEBUG)
    printf("smm_print_q: NULL list\n\r");
    #endif
  }
  else
  {
    q = p;
    do {
      #if defined(DEBUG)
      printf("smm_print_q: l[%02d] : event = %02d fid = %b02d\n\r", q, item[q].
      #endif

      q = item[q].right;
    } while (q != p && q != SMM_NULL_PTR);
  }
}


/*************************************************************************\
 *                           smm_insert_q                                *
***************************************************************************
 *                                                                       *
 * Function:       Inserts an event into the parameter queue if room.    *
 *                 An error is returned if the queue is already full.    *
 *                                                                       *
 * Input:          smm_insert_q(                                         *
 *                   nodeptr *p;         pointer to queue to insert into. *
 *                   mm_message *info; pointer to event to be inserted.  *
 *                 );                                                    *
 *                                                                       *
 * Output:         Nothing.                                              *
 *                                                                       *
 * Globals:        smm_error : module SMM.C                              *
 *                 item      : module SMM.C                              *
 *                                                                       *
 * Edit History: 07/09/90 - Written by Robin T. Laird.                   *
 *                                                                       *
\*************************************************************************/

static void smm_insert_q(p, info)
nodeptr *p;
mm_message *info;
{
  nodeptr q, r;

  if ((q = smm_get_node()) != SMM_NULL_PTR)
  {
    smm_error = AOK;

    item[q].info = *info;

    if (*p == SMM_NULL_PTR)
    {
      item[q].left = q;
```



```
      item[q].right = q;
      *p = q;
    }
    else
    {
      r = item[*p].left;
      item[q].left  = r;
      item[q].right = *p;
      item[r].right = q;
      item[*p].left = q;
    }
  }
}


/*************************************************************************\
 *                           smm_remove_q                                *
***************************************************************************
 *                                                                       *
 * Function:       Removes an event from the parameter queue if available. *
 *                 An error is returned if the queue is already empty.   *
 *                                                                       *
 * Input:          smm_remove_q(                                         *
 *                   nodeptr *p;         pointer to queue to remove from. *
 *                   mm_message *info; pointer to event that is removed. *
 *                 );                                                    *
 *                                                                       *
 * Output:         Nothing.                                              *
 *                                                                       *
 * Globals:        smm_error : module SMM.C                              *
 *                 item      : module SMM.C                              *
 *                                                                       *
 * Edit History: 07/09/90 - Written by Robin T. Laird.                   *
 *                                                                       *
\*************************************************************************/

static void smm_remove_q(p, info)
nodeptr *p;
mm_message *info;
{
  nodeptr q, r, s;

  if (*p == SMM_NULL_PTR)
  {
    smm_error = SMM_ERR_INVALID_PTR;
  }
  else
  {
    smm_error = AOK;

    *info = item[*p].info;

    q = item[*p].left;
    r = item[*p].right;
    s = *p;
    if (s == q)
    {
      *p = SMM_NULL_PTR;
    }
    else
    {
      *p = item[*p].right;
      item[q].right = r;
      item[r].left  = q;
    }
    smm_free_node(s);
  }
}


/*************************************************************************\
 *                           smm_findin_q                                *
***************************************************************************
```

```
 *                                                                       *
 * Function:      Locates the node in the parameter queue whose event   *
 *                identifier matches the parameter message's sequence number *
 *                and whose source address matches the message's destination. *
 *                The entire queue is searched in a linear fashion since *
 *                the queue is not ordered in any particular manner.    *
 *                If matching event IDs are not found, then SMM_NULL_PTR is *
 *                returned. Otherwise, the pointer of the node is returned. *
 *                                                                       *
 * Input:         smm_findin_q(                                          *
 *                  nodeptr p;    pointer to (index of) queue to be searched. *
 *                  mm_message m; message containing search key information. *
 *                );                                                     *
 *                                                                       *
 * Output:        Nothing.                                               *
 *                                                                       *
 * Globals:       smm_error : module SMM.C                               *
 *                item      : module SMM.C                               *
 *                                                                       *
 * Edit History: 07/09/90 - Written by Robin T. Laird.                   *
 *                                                                       *
\*************************************************************************/

static nodeptr smm_findin_q(p, m)
nodeptr p;
mm_message *m;
{
  nodeptr q;

  if (p == SMM_NULL_PTR)
  {
    smm_error = SMM_ERR_INVALID_PTR;
    return(SMM_NULL_PTR);
  }
  else
  {
    q = p;
    do {

      #if defined(DEBUG)
      printf("smm_findin_q: item[q].event          = %d\n", item[q].event);
      printf("smm_findin_q: item[q].dest_modbot_id = %d\n", (int)item[q].info.d
      printf("smm_findin_q: item[q].dest_unit_id   = %d\n", (int)item[q].info.d
      printf("smm_findin_q: m->sequence_number     = %d\n", (int)m->sequence_nu
      printf("smm_findin_q: m->src_modbot_id       = %d\n", (int)m->src_modbot_
      printf("smm_findin_q: m->src_unit_id         = %d\n", (int)m->src_unit_id
      #endif

      if (item[q].event == m->sequence_number &&
          item[q].info.dest_modbot_id == m->src_modbot_id &&
          item[q].info.dest_unit_id   == m->src_unit_id)
      {
        smm_error = AOK;
        return(q);
      }
      q = item[q].right;
    } while (q != p && q != SMM_NULL_PTR);
    smm_error = SMM_ERR_NO_EVENT_MATCH;
    return(SMM_NULL_PTR);
  }
}


/*************************************************************************\
 *                          smm_next_event                               *
 *************************************************************************
 *                                                                       *
 * Function:      Generates a unique sequence number used to identify a *
 *                system message. The sequence number must be different *
 *                than any other sequence number "currently in use". The *
 *                number is used to coorelate response messages to commands. *
 *                                                                       *
 * Input:         smm_next_event();                                      *
```



```
 *                                                                       *
 * Output:        Returns an int value representing the sequence number. *
 *                                                                       *
 * Globals:       avail : module SMM.C                                   *
 *                                                                       *
 * Edit History: 01/23/91 - Written by Robin T. Laird.                   *
 *                                                                       *
\*************************************************************************/

static int smm_next_event()
{
  return(avail);
}


/*************************************************************************\
 *                             smm_init                                  *
 *************************************************************************
 *                                                                       *
 * Function:      Initializes the Local Method Manager software subsystems. *
 *                This includes initializing the module-specific subsystems *
 *                via a call to the system function unit_reset().       *
 *                                                                       *
 * Input:         smm_init();                                            *
 *                                                                       *
 * Output:        Nothing.                                               *
 *                                                                       *
 * Globals:       smm_error       : module SMM.C                         *
 *                avail           : module SMM.C                         *
 *                queue           : module SMM.C                         *
 *                smm_num_methods : module SMM.C                         *
 *                                                                       *
 * Edit History: 12/20/90 - Written by Robin T. Laird.                   *
 *                                                                       *
\*************************************************************************/

void smm_init()
{
  smm_error = AOK;                      /* Assume function successful.    */

  /* Reset system method trigger table index to 0 (indicate table empty).  */

  smm_num_methods = 0;

  /* Initialize the free node pool (avail list).                          */

  smm_avail_init(&avail);

  /* Initialize the standard event queue.                                 */

  smm_clear_q(&queue);
}


/*************************************************************************\
 *                            smm_process                                *
 *************************************************************************
 *                                                                       *
 * Function:      Processes the parameter message and updates the local method *
 *                trigger table. Coordinates message translation and message *
 *                generation. The input message is processed (translated) and *
 *                any possible output or response message is generated for *
 *                transmission to the originating unit.                  *
 *                                                                       *
 * Input:         smm_process(                                           *
 *                  mm_message *m_in;  pointer to decoded message to process. *
 *                  mm_message *m_out; pointer to possible message to output. *
 *                  byte *status;      pointer to method execution status. *
 *                );                                                     *
 *                                                                       *
 * Output:        Nothing.                                               *
 *                                                                       *
 * Globals:       smm_error : module SMM.C                               *
```

```
 *
 * Edit History: 12/20/90 - Written by Robin T. Laird.                     *
 *                                                                          *
\**************************************************************************/

void smm_process(m_in, m_out, status)
mm_message *m_in, *m_out;
byte *status;
{
  smm_error = AOK;                        /* Assume function successful.   */

  /* Translate incoming message as response to previous command.           */
  /* Translation will try and associate sequence number with event number. */

  smm_translate_message(m_in, m_out, status);
}


/**************************************************************************\
*                               smm_fire                                   *
****************************************************************************
*                                                                          *
* Function:     Checks system method trigger table and fires appropriate   *
*               functions according to conditions set up in the table.     *
*               The conditions are set by internal commands issued by      *
*               other functions. The conditions are checked as often as    *
*               possible for possible execution of a function.             *
*                                                                          *
*               Currently, only temporal conditions are implemented.       *
*               This allows for periodic execution of functions.           *
*                                                                          *
* Input:        smm_fire();                                                *
*                                                                          *
* Output:       Nothing.                                                   *
*                                                                          *
* Globals:      smm_error       : module SMM.C                             *
*               smm_trigger     : module SMM.C                             *
*               smm_num_methods : module SMM.C                             *
*                                                                          *
* Edit History: 03/28/91 - Written by Robin T. Laird.                      *
*                                                                          *
\**************************************************************************/

#define SMM_SEND_ATTEMPTS       LCI_WAIT_FOREVER

void smm_fire()
{
  int  i;
  byte event;

  smm_error = AOK;                        /* Assume function successful... */

  /* Check number of methods in trigger table. If non-zero, continue.     */

  if (smm_num_methods)
  {
    /* Check each method for activation.                                  */
    /* Compare current time to trigger time. If greater, then fire method. */

    for (i = 0; i < smm_num_methods; i++)
    {
      if (rtc_time() > smm_trigger[i].fireat)
      {
        /* Process method as usual.                                       */
        /* Update next fire time.                                         */

        smm_trigger[i].fireat = smm_trigger[i].period+rtc_time();
      }
    }
  }
}
```



```
/**************************************************************************\
*                          smm_generate_message                            *
****************************************************************************
*                                                                          *
* Function:     Generates a message as the result of a library function call.*
*               The function call causes a "method activation record" to be *
*               placed on the system method queue (if space is available). *
*               A message is assembled and sent, and an event identifier   *
*               in the form of a newly generated sequence number is returned *
*               to the routine that invoked the library function.          *
*                                                                          *
*               If a destination unit name is supplied (NOT NULL), then the *
*               address of the named unit is looked up in the phone book and *
*               used in the generated message.                             *
*                                                                          *
* Input:        smm_generate_message(                                      *
*                 char *dest_unit_name; pointer to name of unit destination. *
*                 mm_message *m_in;     pointer to (partial) message input. *
*                 int *event;           pointer to returned event identifier.*
*               );                                                         *
*                                                                          *
* Output:       Nothing.                                                   *
*                                                                          *
* Globals:      smm_error : module SMM.C                                   *
*               queue     : module SMM.C                                   *
*               mm_stdout : module MM.C                                    *
*               lci_error : module LCI.C                                   *
*                                                                          *
* Edit History: 12/20/90 - Written by Robin T. Laird.                      *
*                                                                          *
\**************************************************************************/

#define SMM_SEND_ATTEMPTS       LCI_WAIT_FOREVER

void smm_generate_message(dest_unit_name, m_in, event)
char *dest_unit_name;
mm_message *m_in;
int *event;
{
  byte modbot_addr, unit_addr;
  lci_message l_out;

  smm_error = AOK;                        /* Assume function successful.   */

  /* Make sure we can add another event to the method manager queue.      */

  if (smm_full_q(queue))
  {
    smm_error = SMM_ERR_FULL_QUEUE;
    *event    = MM_NULL_EVENT;
    return;
  }

  /* If destination unit name is NULL, then address is supplied in m_in.  */
  /* Else look up destination unit name in the phonebook and get address.  */
  /* If we can't find it, then return NULL event indicating failure.       */

  if (dest_unit_name != NULLPTR)
  {
    if (!pb_lookup(dest_unit_name, &modbot_addr, &unit_addr))
    {
      smm_error = SMM_ERR_MSG_GENERATION;
      *event    = MM_NULL_EVENT;
      return;
    }
    else
    {
      m_in->dest_modbot_id = modbot_addr;
      m_in->dest_unit_id   = unit_addr;
    }
  }

  /* Generate the rest of the outgoing message.                            */
```

```
  /* Assumes the following fields have been filled accordingly:          */
  /*   dest_modbot_id       :       above                                 */
  /*   dest_unit_id         :       above                                 */
  /*   message_length       :       supplied (generated by mm_encode)     */
  /*   src_modbot_id        :       supplied (determined by GCI)          */
  /*   src_unit_id          :       supplied (determined by GCI)          */
  /*   sequence_number      :       generated below                       */
  /*   trans_disposition    :       supplied (m_in)                       */
  /*   trans_category       :       supplied (m_in)                       */
  /*   function_id          :       supplied (m_in)                       */
  /*   parameter_length     :       global (mm_stdout)                    */
  /*   parameter            :       global (mm_stdout)                    */
  /* Note that we have to adjust for bit output by testing bit index.     */

  *event = smm_next_event();

  if (mm_stdout.bitindex) mm_stdout.index++;
  m_in->sequence_number   = (byte)*event;
  m_in->parameter_length = mm_stdout.index;
  memcpy(m_in->parameter, mm_stdout.buffer, m_in->parameter_length);
  mm_stdout.index      = 0;
  mm_stdout.bitindex = 0;

  /* Encode the above information for transmission via the LCI.           */
  /* And send it...                                                       */

  mm_encode_message(m_in, l_out);
  lci_send_message(l_out, SMM_SEND_ATTEMPTS);

  /* Change disposition of message to indicate awaiting response.         */

  m_in->trans_disposition = MM_AWAITING_EVENT;

  /* If msg sent OK, then add method activation record to queue.          */

  if (lci_error == AOK) smm_insert_q(&queue, m_in);
}


/****************************************************************************\
*                          smm_translate_message                             *
******************************************************************************
*                                                                            *
* Function:      Translates (copies) the parameter input message (m_in) into *
*                the system method queue at the position indicated by the    *
*                event parameter. The method is then available for servicing *
*                by the application process (routine).                       *
*                                                                            *
* Input:         smm_translate_message(                                      *
*                  mm_message *m_in;  pointer to message to translate.       *
*                  mm_message *m_out; pointer to any response message.       *
*                  byte *status;      pointer to translation status.         *
*                );                                                          *
*                                                                            *
* Output:        Nothing.                                                    *
*                                                                            *
* Globals:       smm_error : module SMM.C                                    *
*                queue     : module SMM.C                                    *
*                item      : module SMM.C                                    *
*                                                                            *
* Edit History: 02/15/91 - Written by Robin T. Laird.                        *
*                                                                            *
\****************************************************************************/

void smm_translate_message(m_in, m_out, status)
mm_message *m_in, *m_out;
byte *status;
{
  nodeptr n;

  /* Associate sequence number of incoming msg with msgs in event queue.  */
  /* Search queue starting at front for matching sequence number.         */

```



```
  if ((n = smm_findin_q(queue, m_in)) == SMM_NULL_PTR)
  {
    smm_error = SMM_ERR_NO_EVENT_MATCH;
    *status   = MM_NO_RESPONSE;
  }
  else
  {
    /* Found matching sequence number, so translate input message.          */
    /* Translation will cause method queue to be updated.                   */
    /* This is where the transaction disposition changes.                   */

    /* Copy message info into method queue at position indicated by event.  */
    /* The item in the queue that corresponds to the event number - that is */
    /* based on the input message sequence number - will be changed.        */
    /* In particular, the transaction disposition and transaction category. */

    item[n].info.message_length    = m_in->message_length;
    item[n].info.trans_disposition = m_in->trans_disposition;
    item[n].info.trans_category    = m_in->trans_category;
    item[n].info.parameter_length  = m_in->parameter_length;
    memcpy(item[n].info.parameter,m_in->parameter,(int)m_in->parameter_length);

    smm_error = AOK;
    *status   = MM_NO_RESPONSE;
  }
}
```

```c
/**********************************************************************\
*                              SMMLIB.C                                *
************************************************************************
*                                                                      *
* CPCI:         IED90-MRA-COM-MMS-SMMLIB-C-ROCO                        *
*                                                                      *
* Description:  System Method Manager (SMM) library functions.         *
*               Implements the inherited ("built-in") system methods.  *
*                                                                      *
*               Module SMMLIB.C exports the following functions:       *
*                                                                      *
*               OBJECT (STATUS_REQUEST):                               *
*               obj_class();                                           *
*               obj_superclass();                                      *
*                                                                      *
*               UNIT (STATUS_REQUEST):                                 *
*               unit_name();                                           *
*                                                                      *
*               UNIT (CONTROL, CONTROL_WITH_ACK):                      *
*               unit_reset();                                          *
*                                                                      *
* Notes:        None.                                                  *
*                                                                      *
* Edit History: 01/22/91 - Written by Robin T. Laird.                  *
*                                                                      *
\**********************************************************************/

#include <sysdefs.h>                    /* System constants and types.    */
#include "mm.h"                         /* Method manager literals.       */
#include "smm.h"                        /* System method manager.         */

/* Object class library functions...                                      */

int obj_class(modbot, unit)
byte modbot, unit;
{
  int        event;
  mm_message m;

  m.dest_modbot_id    = modbot;
  m.dest_unit_id      = unit;
  m.trans_disposition = MM_INITIATING;
  m.trans_category    = MM_STATUS_REQUEST;
  m.function_id       = OBJ_FN_CLASS;
  smm_generate_message(NULLPTR, &m, &event);
  return(event);
}

int obj_superclass(modbot, unit)
byte modbot, unit;
{
  int        event;
  mm_message m;

  m.dest_modbot_id    = modbot;
  m.dest_unit_id      = unit;
  m.trans_disposition = MM_INITIATING;
  m.trans_category    = MM_STATUS_REQUEST;
  m.function_id       = OBJ_FN_SUPERCLASS;
  smm_generate_message(NULLPTR, &m, &event);
  return(event);
}

/* Unit class library functions...                                        */

int unit_name(modbot, unit)
byte modbot, unit;
{
  int        event;
  mm_message m;

  m.dest_modbot_id    = modbot;
  m.dest_unit_id      = unit;
```



```c
  m.trans_disposition = MM_INITIATING;
  m.trans_category    = MM_STATUS_REQUEST;
  m.function_id       = UNIT_FN_NAME;
  smm_generate_message(NULLPTR, &m, &event);
  return(event);
}

int unit_reset(modbot, unit)
byte modbot, unit;
{
  int        event;
  mm_message m;

  m.dest_modbot_id    = modbot;
  m.dest_unit_id      = unit;
  m.trans_disposition = MM_INITIATING;
  m.trans_category    = MM_CONTROL_WITH_ACK;
  m.function_id       = UNIT_FN_RESET;
  smm_generate_message(NULLPTR, &m, &event);
  return(event);
}
```

```
#***********************************************************************
#                              MAKEFILE                                #
#***********************************************************************
#                                                                      #
# CPCI:         IED90-MRA-ICN-AC-MAKEFILE-TXT-R0C0                     #
#                                                                      #
# Description:  Makefile for the Modular Robotic Architecture (MRA).   #
#               Makes the ICN applications controller subsystem.       #
#                                                                      #
#               Targets are available for the following systems/subsystems: #
#                                                                      #
#               iac       - ICN Applications Controller                #
#               iac.hex   - ICN Applications Controller Program (80152) #
#               icnmon.hex - ICN Network Monitor Program (80152)       #
#               print     - Print ICN AC source files                  #
#                                                                      #
# Notes:        1) The dependency and production rules are included here. #
#               2) See also \mra\makefile.                             #
#                                                                      #
# Edit History: 03/25/91 - Written by Robin T. Laird.                  #
#                                                                      #
#***********************************************************************



################################## RULES ##################################

.SUFFIXES : .hex .exe .obj .c .a51

# Control settings for Franklin 8031 development

CC              =c51
AS              =a51
LINK            =l51
OTOH            =ohs51
CFLAGS          =cd la db sb
ASFLAGS         =
LFLAGS          =
OFLAGS          =
STARTUP         =\c51\crom.obj
CODESEG         =00000h
XDATASEG        =00000h

# Control settings for Microsoft MS-DOS development

MSC             =cl
MSAS            =masm
MSLINK          =link
MSCFLAGS        =/AS /c /Ol /Zi /Od
MSASFLAGS       =
MSLNKFLGS       =/co
LOADLIBES       =

.c.obj :
        $(CC) $< $(CFLAGS)

.a51.obj :
        $(AS) $< $(ASFLAGS)

.obj.exe :
        $(LINK) $(STARTUP),$< TO $@ code ($(CODESEG)) xdata ($(XDATASEG)) ixref

.exe.hex :
        $(OTOH) $< $(OFLAGS)


############################## DEFINITIONS ##############################

# Project, system, and application level definitions

PROJ            = mra
APPSYS          = app
```



```
COMSYS          = com
ICNSYS          = icn
MPUSYS          = mpu

APPSRC          = \$(PROJ)\$(APPSYS)\src
COMSRC          = \$(PROJ)\$(COMSYS)\src
ICNSRC          = \$(PROJ)\$(ICNSYS)\src
MPUSRC          = \$(PROJ)\$(MPUSYS)\src

COMBIN152       = \$(PROJ)\$(COMSYS)\bin\80152

ICNBIN152       = \$(PROJ)\$(ICNSYS)\bin\80152
ICNBIN152_MON   = \$(PROJ)\$(ICNSYS)\bin\80152\mon

# Common subsystem level source directories

DEVSRC          = $(COMSRC)\dev
HDRSRC          = $(COMSRC)\hdr
LCSSRC          = $(COMSRC)\lcs
MMSSRC          = $(COMSRC)\mms

# ICN subsystem level source directories

GCSSRC          = $(ICNSRC)\gcs
IACSRC          = $(ICNSRC)\ac

# MPU subsystem level source directories

LDSSRC          = $(MPUSRC)\lds
MACSRC          = $(MPUSRC)\ac

# Common subsystem global include and compilation units

SYSDEFS         = $(HDRSRC)\sysdefs.h

# ICN subsystem compilation units

IAC             = $(ICNBIN152)\main.obj        $(ICNBIN152)\mra.obj    \
                  $(ICNBIN152)\gci.obj         $(ICNBIN152)\gcd.obj    \
                  $(COMBIN152)\lci.obj         $(COMBIN152)\lcd.obj

ICNMON          = $(ICNBIN152)\main.obj        $(ICNBIN152)\mra.obj    \
                  $(ICNBIN152)\gci.obj         $(ICNBIN152_MON)\gcd.obj\
                  $(COMBIN152)\lci.obj         $(COMBIN152)\lcd.obj


################################# TARGETS #################################

iac                         : $(ICNBIN152)\iac.hex $(ICNBIN152_MON)\icnmon.hex

$(ICNBIN152)\iac.hex        : $(ICNBIN152)\iac
                              $(OTOH) $< hex($@)

$(ICNBIN152)\iac            : $(IAC)
                              $(LINK) @$(IACSRC)\iac.res

$(ICNBIN152_MON)\icnmon.hex : $(ICNBIN152_MON)\icnmon
                              $(OTOH) $< hex($@)

$(ICNBIN152_MON)\icnmon     : $(ICNMON)
                              $(LINK) @$(IACSRC)\icnmon.res

print                       : $(IAC)
                              %-a2ps -nf main.c | post
                              %-a2ps -nf mra.h | post
                              %-a2ps -nf mra.c | post
                              touch print


########################### ICN IAC DEPENDENCIES ###########################
```

```
# ICN IAC main program dependencies

$(ICNBIN152)\main.obj        : $(SYSDEFS)             \
                               $(IACSRC)\mra.h        \
                               $(GCSSRC)\gci.h        \
                               $(GCSSRC)\gcd.h        \
                               $(LCSSRC)\lci.h        \
                               $(LCSSRC)\lcd.h        \
                               $(MMSSRC)\mm.h         \
                               $(MMSSRC)\lmm.h        \
                               $(MMSSRC)\smm.h        \
                               $(LDSSRC)\ldi.h        \
                               $(IACSRC)\main.c
    $(CC)  $(IACSRC)\$*.c $(CFLAGS) df(180152) pr($(IACSRC)\$*.152) oj($(ICNBIN152


# ICN IAC mra system dependencies

$(ICNBIN152)\mra.obj         : $(SYSDEFS)             \
                               $(IACSRC)\mra.h        \
                               $(GCSSRC)\gci.h        \
                               $(GCSSRC)\gcd.h        \
                               $(LCSSRC)\lci.h        \
                               $(LCSSRC)\lcd.h        \
                               $(MMSSRC)\mm.h         \
                               $(MMSSRC)\lmm.h        \
                               $(MMSSRC)\smm.h        \
                               $(LDSSRC)\ldi.h        \
                               $(IACSRC)\mra.c
    $(CC)  $(IACSRC)\$*.c $(CFLAGS) df(180152) pr($(IACSRC)\$*.152) oj($(ICNBIN152
```

```
\c51\crom.obj,                  &
\mra\icn\bin\80152\main.obj,    &
\mra\icn\bin\80152\mra.obj,     &
\mra\lib\mra_1521.lib           &
to \mra\icn\bin\80152\iac       &
pr(\mra\icn\src\ac\iac.m51) co(00000h) xd(00000h) ix
```

```
\c51\crom.obj,                  &
\mra\icn\bin\80152\main.obj,    &
\mra\icn\bin\80152\mra.obj,     &
\mra\icn\bin\80152\gci.obj,     &
\mra\icn\bin\80152\mon\gcd.obj, &
\mra\lib\mra_1521.lib           &
to \mra\icn\bin\80152\mon\icnmon&
pr(\mra\icn\src\ac\icnmon.m51) co(00000h) xd(00000h) ix
```

```
/*********************************************************************\
*                              MAIN.C                                *
***********************************************************************
*                                                                    *
* CPCI:          IED90-MRA-ICN-AC-MAIN-C-ROCO                        *
*                                                                    *
* Description:   ICN main program.                                   *
*                Implements the MRA ICN (user) application program.  *
*                This is the main program for the ICN system.        *
*                It simply calls the standard MRA function mra_init() which *
*                is responsible for initializing and coordinating the MRA *
*                subsystems for the ICN application.                 *
*                                                                    *
* Notes:         1) Only the LCS and GCS subsystems are currently required. *
*                2) Subsystems are selected by setting the associated USE_ss *
*                   variable to YES (1). This can be done from either the *
*                   compilation command line or by modifying the MRA.H file. *
*                                                                    *
* Edit History: 10/10/90 - Written by Robin T. Laird.                *
*                                                                    *
\*********************************************************************/

#include <sysdefs.h>                    /* MRA standard declarations.    */
#include "mra.h"                        /* MRA public literals/functions. */

void main()
{
  /* Initialize the MRA subsystems and call mra_main().                 */
  /* Control never returns...                                           */

  mra_init();
}
```

```c
/***********************************************************************\
*                               MRA.H                                   *
*************************************************************************
*                                                                       *
* CPCI:         IED90-MRA-ICN-AC-MRA-H-ROCO                             *
*                                                                       *
* Description:  System configuration and default controller definitions.
*               Contains external declarations for the system initialization *
*               function and default application controller, mra_main(). *
*                                                                       *
*               The user/developer should select/de-select those MRA    *
*               subsystems that are required or being used. Only those  *
*               subsystems that are selected are initialized by mra_init(). *
*                                                                       *
*               Selecting USE_MRA will cause the default system application *
*               controller, mra_main(), to be used. The init routine calls *
*               the default controller after all selected subsystems have *
*               been initialized. If selected, mra_main() takes control and *
*               does not return.                                        *
*                                                                       *
*               Module MRA exports the following variables/functions:   *
*                                                                       *
*               int mra_error;                                          *
*                                                                       *
*               mra_init();                                             *
*               mra_main();                                             *
*                                                                       *
* Notes:        1) This file should be included only by the main() program. *
*                                                                       *
* Edit History: 07/07/90 - Written by Robin T. Laird.                   *
*                                                                       *
\***********************************************************************/

#ifndef MRA_MODULE_CODE
#define MRA_MODULE_CODE         12000

/* Public Data Structures:                                              */

#define ERR_MRA_NOT_INIT        1+MRA_MODULE_CODE

#define USE_GCS                 YES     /* Global Communications Subsystem. */
#define USE_LCS                 YES     /* Local  Communications Subsystem. */
#define USE_MMS                 NO      /* Method Management Subsystem.     */
#define USE_LDS                 NO      /* Logical Device Subsystem.        */
#define USE_MRA                 YES     /* Modular Robotic Architecture AC. */

#if USE_GCS
#include <gci.h>                        /* Global Comms Interface.          */
#include <gcd.h>                        /* Global Comms Device Driver.      */
#endif

#if USE_LCS
#include <lci.h>                        /* Local Comms Interface.           */
#include <lcd.h>                        /* Local Comms Device Driver.       */
#endif

#if USE_MMS
#include <mm.h>                         /* Method Manager.                  */
#include <pb.h>                         /* Phone Book.                      */
#include <lmm.h>                        /* Local Method Manager.            */
#include <smm.h>                        /* System Method Manager.           */
#endif

#if USE_LDS
#include <ldi.h>                        /* Logical Device Interface.        */
#endif

/* External module global error variable.                               */

extern int mra_error;

/* Public Functions:                                                    */

```



```c
void mra_init(void);
void mra_main(void);

#endif
```

```c
/****************************************************************************\
*                                  MRA.C                                     *
******************************************************************************
*                                                                            *
* CPCI:          IED90-MRA-ICN-AC-MRA-C-ROCO                                 *
*                                                                            *
* Description:   MRA system initialization and default controller functions. *
*                Implements the MRA system initialization and default system *
*                application controller (AC) functions which represent the   *
*                highest level interface to the Modular Robotic Architecture *
*                software systems. The mra_init() function must be called to *
*                correctly initialize the various software subsystems. The   *
*                function mra_main() is the default AC and replaces the user *
*                application program (mra_main() never returns to the calling *
*                function).                                                  *
*                                                                            *
*                Module MRA exports the following variables/functions:       *
*                                                                            *
*                int mra_error;                                              *
*                                                                            *
*                mra_init();                                                 *
*                mra_main();                                                 *
*                                                                            *
* Notes:         1) The MRA functions are implementation independent.        *
*                2) Module MRA represents the Default Applications Controller.*
*                                                                            *
* Edit History: 07/28/90 - Written by Robin T. Laird.                        *
*                                                                            *
\****************************************************************************/

#include <sysdefs.h>                    /* System constants and types.     */
#include "mra.h"                        /* MRA public literals/functions.  */

/* Public Variables:                                                        */

/* Global module error variable, mra_error.                                 */
/* mra_error contains code of last error occurence.                         */
/* Should be set to AOK after each successful function call.                */
/* Variable can be examined by other software after each function call.     */

XDATA int mra_error = ERR_MRA_NOT_INIT;


/****************************************************************************\
*                                 mra_init                                   *
******************************************************************************
*                                                                            *
* Function:      Initializes the MRA software subsystems.                    *
*                The subsystems are selected in the file MRA.H and only those *
*                subsystems selected will be initialized. If the default     *
*                application controller mra_main() is selected then a call is *
*                made to that function and control never returns. The default *
*                AC can be called separately by a call to mra_main() after   *
*                mra_init() returns (the same result is acheived).           *
*                                                                            *
* Input:         mra_init();                                                 *
*                                                                            *
* Output:        Nothing.                                                    *
*                                                                            *
* Globals:       mra_error : module MRA.C                                    *
*                gci_error : module GCI.C                                    *
*                lci_error : module LCI.C                                    *
*                mm_error  : module MM.C                                     *
*                ldi_error : module LDI.C                                    *
*                                                                            *
* Edit History: 10/01/90 - Written by Robin T. Laird.                        *
*                                                                            *
\****************************************************************************/

void mra_init()
{
  /* Initialize selected subsystems, return upon detected failure.        */

```



```c
  #if USE_GCS
  gci_init();
  if (gci_error != AOK) return;
  #endif

  #if USE_LCS
  lci_init();
  if (lci_error != AOK) return;
  #endif

  #if USE_MMS
  mm_init();
  if (mm_error != AOK) return;
  #endif

  #if USE_LDS
  ldi_init();
  if (ldi_error != AOK) return;
  #endif

  /* Function successful...                                                 */
  /* Set error variable to OK and return only if USE_MRA not selected.      */

  mra_error = AOK;

  /* Call MRA main program and never return...                              */

  #if USE_MRA
  mra_main();
  #endif
}


/****************************************************************************\
*                                 mra_main                                   *
******************************************************************************
*                                                                            *
* Function:      Default Application Controller (AC) for the ICN system.     *
*                The main() C program calls mra_init() and which then calls  *
*                mra_main(). The default controller coordinates operation of *
*                the MRA subsystems to pass information from the LAN to the  *
*                module processor (MPU) and vice versa.                      *
*                                                                            *
* Input:         mra_main();                                                 *
*                                                                            *
* Output:        Nothing.                                                    *
*                                                                            *
* Globals:       None.                                                       *
*                                                                            *
* Edit History: 10/31/90 - Written by Robin T. Laird.                        *
*                                                                            *
\****************************************************************************/

#define MRA_LOCAL_SEND_ATTEMPTS          LCI_WAIT_FOREVER
#define MRA_GLOBAL_SEND_ATTEMPTS         8

void mra_main()
{
  #if USE_MRA

  gci_message g;
  lci_message l;
  lcd_state   s;
  int         gci_rcv_error, lci_rcv_error;

  /* Do forever...                                                          */

  /* Get message from network (ICN LAN).                                    */
  /* Function will return immediately if nothing available.                 */
  /* Save GCI global error state.                                           */

  /* Get message from module processor (MPU).                               */
  /* Function will return immediately if nothing available.                 */
```

```
    /* Save LCI global error state.                                        */

    /* If no error on receipt of network msg then pass it on to MPU.       */
    /* If no error on receipt of MPU msg then pass it on to ICN LAN.       */

    while (1)
    {
      gci_receive_message(g, GCI_DONT_WAIT);
      gci_rcv_error = gci_error;

      lci_receive_message(l, LCI_DONT_WAIT);
      lci_rcv_error = lci_error;

      if (gci_rcv_error == AOK)
      {
        lci_send_message((lci_message)g, MRA_LOCAL_SEND_ATTEMPTS);
      }

      if (lci_rcv_error == AOK)
      {
        gci_send_message((gci_message)l, MRA_GLOBAL_SEND_ATTEMPTS);
      }
    }

    #endif
  }
```

```
#*************************************************************************#
#                              MAKEFILE                                   #
#*************************************************************************#
#                                                                         #
# CPCI:         IED90-MRA-ICN-GCS-MAKEFILE-TXT-ROCO                       #
#                                                                         #
# Description:  Makefile for the Modular Robotic Architecture (MRA).      #
#               Makes the ICN global communications subsystem.            #
#                                                                         #
#               Targets are available for the following systems/subsystems: #
#                                                                         #
#               gcs   - ICN Global Communications Subsystem               #
#               lib   - Add modules to COM library                        #
#               print - Print ICN GCS source files                        #
#                                                                         #
# Notes:        1) The dependency and production rules are included here. #
#               2) See also \mra\makefile.                                #
#                                                                         #
# Edit History: 03/22/91 - Written by Robin T. Laird.                     #
#                                                                         #
#*************************************************************************#



##################################### RULES ##################################

.SUFFIXES : .hex .exe .obj .c .a51

# Control settings for Franklin 8031 development

CC              =c51
AS              =a51
LINK            =l51
OTOH            =ohs51
CFLAGS          =cd la db sb
ASFLAGS         =
LFLAGS          =
OFLAGS          =
STARTUP         =\c51\crom.obj
CODESEG         =00000h
XDATASEG        =00000h

# Control settings for Microsoft MS-DOS development

MSC             =cl
MSAS            =masm
MSLINK          =link
MSCFLAGS        =/AS /c /Oi /Zi /Od
MSASFLAGS       =
MSLNKFLGS       =/co
LOADLIBES       =

.c.obj :
        $(CC) $< $(CFLAGS)

.a51.obj :
        $(AS) $< $(ASFLAGS)

.obj.exe :
        $(LINK) $(STARTUP),$< TO $@ code ($(CODESEG)) xdata ($(XDATASEG)) ixref

.exe.hex :
        $(OTOH) $< $(OFLAGS)



################################## DEFINITIONS ###############################

# Project, system, and application level definitions

PROJ            = mra
APPSYS          = app
COMSYS          = com
```



```
ICNSYS          = icn
MPUSYS          = mpu

COMSRC          = \$(PROJ)\$(COMSYS)\src

ICNLIB          = \$(PROJ)\lib
ICNSRC          = \$(PROJ)\$(ICNSYS)\src
ICNBIN152       = \$(PROJ)\$(ICNSYS)\bin\80152
ICNBIN152_MON   = \$(PROJ)\$(ICNSYS)\bin\80152\mon

# Common subsystem level source directories

HDRSRC          = $(COMSRC)\hdr

# ICN subsystem level source directories

GCSSRC          = $(ICNSRC)\gcs

# Common subsystem global include and compilation units

SYSDEFS         = $(HDRSRC)\sysdefs.h

# ICN subsystem compilation units

GCS             = $(ICNBIN152)\gcd.obj  $(ICNBIN152)\gci.obj          \
                  $(ICNBIN152_MON)\gcd.obj



#################################### TARGETS #################################

gcs             : $(GCS)

lib             : $(GCS)
                  -lib51 delete $(ICNLIB)\mra_1521.lib (gcd, gci)
                  -lib51 add $(ICNBIN152)\gcd.obj to $(ICNLIB)\mra_1521.lib
                  -lib51 add $(ICNBIN152)\gci.obj to $(ICNLIB)\mra_1521.lib
                  touch lib

print           : $(GCS)
                  %-a2ps -nf bmf.h | post
                  %-a2ps -nf bmf.c | post
                  %-a2ps -nf gcd.h | post
                  %-a2ps -nf gcd.c | post
                  %-a2ps -nf gci.h | post
                  %-a2ps -nf gci.c | post
                  %-a2ps -nf gsc.h | post
                  %-a2ps -nf gsc.c | post
                  touch print



########################### ICN GCS DEPENDENCIES #########################

# ICN GCD module dependencies for generic ICN and ICN monitor

GCD = $(SYSDEFS)         $(GCSSRC)\bmf.h        $(GCSSRC)\bmf.c        \
      $(GCSSRC)\gsc.h    $(GCSSRC)\gsc.c        $(GCSSRC)\gcd.h        \
      $(GCSSRC)\gcd.c

$(ICNBIN152)\gcd.obj     : $(GCD)
  $(CC) $(GCSSRC)\$*.c $(CFLAGS) df(I80152) pr($(GCSSRC)\$*.152) oj($(ICNBIN152

$(ICNBIN152_MON)\gcd.obj : $(GCD)
  $(CC) $(GCSSRC)\$*.c $(CFLAGS) df(ICNMON) pr($(GCSSRC)\$*.mon) oj($(ICNBIN152


# ICN GCI module dependencies

$(ICNBIN152)\gci.obj     : $(SYSDEFS)                                  \
                           $(GCSSRC)\gcd.h                             \
                           $(GCSSRC)\gci.h                             \
                           $(GCSSRC)\gci.c
```

```
	$(CC) $(GCSSRC)\$*.c $(CFLAGS) df(180152) pr($(GCSSRC)\$*.152) oj($(ICNBIN152
```

```
/**********************************************************************\
*                                BMF.H                                 *
************************************************************************
*                                                                      *
* CPCI:          IED90-MRA-ICN-GCS-BMF-H-ROC1                          *
*                                                                      *
* Description:   Frame Buffer Management variables and definitions.    *
*                Contains constant declarations and type definitions for the *
*                circular frame buffer management functions. Used by the *
*                GCS communications device handler module.              *
*                                                                      *
*                Module BMF exports the following types/functions:     *
*                                                                      *
*                typedef buffer;                                       *
*                                                                      *
*                buf_full();                                           *
*                buf_empty();                                          *
*                buf_clear();                                          *
*                buf_insert();                                         *
*                buf_remove();                                         *
*                                                                      *
* Notes:         1) This file is included by GCD.C.                    *
*                2) Module BMF requires type definitions from GCD.H.   *
*                                                                      *
* Edit History: 06/28/90 - Written by Robin T. Laird.                  *
*                                                                      *
\**********************************************************************/

/* Private Data Structures:                                            */

#ifndef BMF_MODULE_CODE
#define BMF_MODULE_CODE          1100

#define ERR_FULL_BUFFER          1+BMF_MODULE_CODE
#define ERR_EMPTY_BUFFER         2+BMF_MODULE_CODE

#define ERR_SRC_EQUAL_DEST       3+BMF_MODULE_CODE

/* MAX_BUFFER_SIZE defines the number of receive/transmit frames/buffer.   */

#define MAX_BUFFER_SIZE          48

/* Circular buffer (queue) to hold incoming/outgoing data frames.      */
/* Must be initialized using buf_clear().                              */

typedef struct { gcd_frame item[MAX_BUFFER_SIZE];
                 byte      front;
                 byte      rear;
                 byte      empty;
                 byte      full;
               } buffer;

/* Private Functions:                                                  */

static int buf_full(buffer *b);
static int buf_empty(buffer *b);
static void buf_clear(buffer *b);
static void buf_insert(buffer *b, gcd_frame f);
static void buf_remove(buffer *b, gcd_frame f);

#endif
```

```
/****************************************************************************\
*                                   BMF.C                                    *
******************************************************************************
*                                                                            *
* CPCI:          IED90-MRA-ICN-GCS-BMF-C-ROC2                                *
*                                                                            *
* Description:   Frame Buffer Management functions.                          *
*                Contains functions for initializing and managing the        *
*                circular frame buffers for the GCS global communications    *
*                device handler.                                             *
*                                                                            *
*                Module BMF exports the following functions:                 *
*                                                                            *
*                buf_front() macro;                                          *
*                buf_rear()  macro;                                          *
*                buf_inc()   macro;                                          *
*                buf_full();                                                 *
*                buf_empty();                                                *
*                buf_clear();                                                *
*                buf_insert();                                               *
*                buf_remove();                                               *
*                                                                            *
* Notes:         1) This module is NOT a stand-alone compilation unit.       *
*                   It is included by the module GCD.C and is compiled there.*
*                   It is assumed that the file GCD.H is included before it. *
*                   Note that all of the functions herein are static.        *
*                                                                            *
* Edit History: 06/29/90 - Written by Richard P. Smurlo and Robin T. Laird.  *
*                                                                            *
\****************************************************************************/

/* Private Variables:                                                        */

/* Declarations for the module circular receive and transmit buffers.        */
/* These are global to the GSC.C module and to the GCD.C module.             */

static XDATA buffer xmt_buffer;
static XDATA buffer rcv_buffer;


/****************************************************************************\
*                                 buf_front                                  *
******************************************************************************
*                                                                            *
* Function:      Macro that returns a pointer to the front (current)         *
*                element in the parameter buffer. Used to obtain the         *
*                address of the next frame to be received/transmitted.       *
*                                                                            *
* Input:         buf_front(                                                  *
*                  buffer b; buffer structure (NOT a pointer to it).         *
*                );                                                          *
*                                                                            *
* Output:        Pointer to front (current) frame in the buffer.             *
*                                                                            *
* Globals:       None.                                                       *
*                                                                            *
* Edit History: 07/08/90 - Written by Robin T. Laird.                        *
*                                                                            *
\****************************************************************************/

#define buf_front(b) (b.item[b.front])


/****************************************************************************\
*                                 buf_rear                                   *
******************************************************************************
*                                                                            *
* Function:      Macro that returns a pointer to the rear (last)             *
*                element in the parameter buffer. Used to obtain the         *
*                address of the next frame to be received/transmitted.       *
*                                                                            *
* Input:         buf_rear(                                                   *
*                  buffer b; buffer structure (NOT a pointer to it).         *
```



```
*                );                                                          *
*                                                                            *
* Output:        Pointer to rear (last) frame in the buffer.                 *
*                                                                            *
* Globals:       None.                                                       *
*                                                                            *
* Edit History: 07/10/90 - Written by Robin T. Laird.                        *
*                                                                            *
\****************************************************************************/

#define buf_rear(b) (b.item[b.rear])


/****************************************************************************\
*                                  buf_inc                                   *
******************************************************************************
*                                                                            *
* Function:      Macro that increments either the front or rear index        *
*                of a circular buffer of max size MAX_BUFFER_SIZE.           *
*                Should be used with care, i.e., not nested within a         *
*                compound statement like if ... then ... else.               *
*                                                                            *
* Input:         buf_inc(                                                    *
*                  word x; index to increment.                               *
*                );                                                          *
*                                                                            *
* Output:        Nothing.                                                    *
*                                                                            *
* Globals:       None.                                                       *
*                                                                            *
* Edit History: 12/04/90 - Written by Robin T. Laird.                        *
*                                                                            *
\****************************************************************************/

#define buf_inc(x) (x = ((x == MAX_BUFFER_SIZE-1) ? 0 : x+1))


/****************************************************************************\
*                                 buf_full                                   *
******************************************************************************
*                                                                            *
* Function:      Function that returns the boolean of whether or not the     *
*                parameter buffer is full (TRUE if so, FALSE if not).        *
*                                                                            *
* Input:         buf_full(                                                   *
*                  buffer *b; pointer to buffer structure.                   *
*                );                                                          *
*                                                                            *
* Output:        Integer, TRUE if buffer FULL, FALSE if buffer not FULL.     *
*                                                                            *
* Globals:       None.                                                       *
*                                                                            *
* Edit History: 07/08/90 - Written by Robin T. Laird.                        *
*                                                                            *
\****************************************************************************/

static int buf_full(b)
buffer *b;
{
  gcd_error = AOK;                        /* Assume function successful...   */
  return(b->full);
}


/****************************************************************************\
*                                 buf_empty                                  *
******************************************************************************
*                                                                            *
* Function:      Function that returns the boolean of whether or not the     *
*                parameter buffer is empty (TRUE if so, FALSE if not).       *
*                                                                            *
* Input:         buf_empty(                                                  *
*                  buffer *b; pointer to buffer structure.                   *
```

```c
 *                 );                                                        *
 *                                                                           *
 * Output:        Integer, TRUE if buffer EMPTY, FALSE if buffer not EMPTY.  *
 *                                                                           *
 * Globals:       None.                                                      *
 *                                                                           *
 * Edit History: 07/08/90 - Written by Robin T. Laird.                       *
 *                                                                           *
\***************************************************************************/

static int buf_empty(b)
buffer *b;
{
  gcd_error = AOK;                        /* Assume function successful...   */
  return(b->empty);
}


/***************************************************************************\
 *                                buf_clear                                  *
 *****************************************************************************
 *                                                                           *
 * Function:      Initializes the parameter buffer and clears its contents.  *
 *                The front and rear pointers are reset and the boolean      *
 *                state flags (i.e., empty, full) are set accordingly.       *
 *                                                                           *
 * Input:         buf_clear(                                                 *
 *                  buffer *b; pointer to buffer structure to clear.         *
 *                );                                                         *
 *                                                                           *
 * Output:        Nothing.                                                   *
 *                                                                           *
 * Globals:       gcd_error : module GCD.C                                   *
 *                                                                           *
 * Edit History: 07/09/90 - Written by Robin T. Laird.                       *
 *                                                                           *
\***************************************************************************/

static void buf_clear(b)
buffer *b;
{
  word i, j;

  gcd_error = AOK;                        /* Assume function successful...   */

  /* Process each of the frames in the buffer.                               */
  /* Set all bytes in frame to 0.                                            */

  for (i = 0; i < MAX_BUFFER_SIZE; i++)
    for (j = 0; j < GCD_MAX_FRAME_LENGTH; j++) b->item[i][j] = 0x00;

  /* Set the front and rear indexes equal to indicate empty buffer.          */
  /* Set the empty and full flags as appropriate.                            */

  b->front = b->rear = 0;
  b->empty = TRUE;
  b->full  = FALSE;
}


/***************************************************************************\
 *                                buf_insert                                 *
 *****************************************************************************
 *                                                                           *
 * Function:      Inserts a frame into the parameter buffer if room.         *
 *                The frame is not inserted if the source address equals     *
 *                the destination - this avoids sending a frame to ones self.*
 *                An error is returned if the buffer is already full.        *
 *                                                                           *
 * Input:         buf_insert(                                                *
 *                  buffer *b;   pointer to buffer structure to insert into. *
 *                  gcd_frame f; frame to be inserted.                       *
 *                );                                                         *
```



```c
 *                                                                           *
 * Output:        Nothing.                                                   *
 *                                                                           *
 * Globals:       gcd_error : module GCD.C                                   *
 *                                                                           *
 * Edit History: 07/09/90 - Written by Robin T. Laird.                       *
 *               01/30/91 - Robin T. Laird added insertion of node ID into f.*
 *               02/27/91 - Robin T. Laird added check for src equal dest.   *
 *                                                                           *
\***************************************************************************/

static void buf_insert(b, f)
buffer *b;
gcd_frame f;
{
  byte node_id;
  word i, length;

  gcd_error = AOK;                        /* Assume function successful...   */

  /* Make sure buffer isn't already full.                                    */

  if (b->full)
  {
    gcd_error = ERR_FULL_BUFFER;
    return;
  }

  /* Make sure we aren't trying to send a frame to ourself.                  */
  /* Insert source address at appropriate place in frame.                    */
  /* Source is obtained from node ID function at lowest level.               */

  if ((node_id = gsc_node_id()) == f[GCD_DEST_ADDR_POS])
  {
    gcd_error = ERR_SRC_EQUAL_DEST;
    return;
  }
  else
    f[GCD_SRC_ADDR_POS] = node_id;

  /* Extract length of frame from frame data (indexed at GCD_LEN_POS).       */
  /* Copy element into buffer and set appropriate structure fields.          */
  /* Length of frame is included as part of frame data (f[GCD_LEN_POS]).     */

  length = f[GCD_LEN_POS];
  for (i = 0; i < length; i++) b->item[b->rear][i] = f[i];

  /* Adjust rear index (MAX_BUFFER_SIZE-1 is last element in buffer).        */

  buf_inc(b->rear);

  /* Check and see if we've filled the buffer (set full flag if so).         */
  /* Set empty flag to FALSE since we just inserted.                         */

  if ((b->front == b->rear) && !b->empty) b->full = TRUE;
  b->empty = FALSE;
}


/***************************************************************************\
 *                                buf_remove                                 *
 *****************************************************************************
 *                                                                           *
 * Function:      Removes a frame from the parameter buffer if available.    *
 *                An error is returned if the buffer is already empty.       *
 *                                                                           *
 * Input:         buf_remove(                                                *
 *                  buffer *b;   pointer to buffer structure to remove from. *
 *                  gcd_frame f; frame to be removed.                        *
 *                );                                                         *
 *                                                                           *
 * Output:        Nothing.                                                   *
 *                                                                           *
```

```
 * Globals:      gcd_error : module GCD.C                                  *
 *                                                                         *
 * Edit History: 07/09/90 - Written by Robin T. Laird.                     *
 *                                                                         *
\**************************************************************************/

static void buf_remove(b, f)
buffer *b;
gcd_frame f;
{
  word i, length;

  gcd_error = AOK;                        /* Assume function successful...    */

  /* Make sure buffer isn't already empty.                                    */

  if (b->empty)
  {
    gcd_error = ERR_EMPTY_BUFFER;
    return;
  }

  /* Extract length of frame from frame data (indexed at GCD_LEN_POS).        */
  /* Copy element into frame and set appropriate structure fields.            */
  /* Buffer can't be full since we just removed an element.                   */

  length = b->item[b->front][GCD_LEN_POS];
  for (i = 0; i < length; i++) f[i] = b->item[b->front][i];
  b->full = FALSE;

  /* Adjust front index (MAX_BUFFER_SIZE-1 is last element in buffer).        */

  buf_inc(b->front);

  /* Check and see if we've depleted the buffer (set empty flag if so).       */

  if (b->front == b->rear) b->empty = TRUE;
}
```

```c
/*********************************************************************\
*                              GCD.H                                  *
***********************************************************************
*                                                                     *
* CPCI:          IED90-MRA-ICN-GCS-GCD-H-ROCO                         *
*                                                                     *
* Description:   GCS communications device handler variables and functions.
*                Contains constant function parameter declarations as well
*                as function return values (for success and failure of all
*                operations). Contains the function prototypes for the GCD.C
*                module.                                              *
*                                                                     *
*                Module GCD exports the following types/variables/functions:
*                                                                     *
*                typedef gcd_frame;                                   *
*                typedef gcd_state;                                   *
*                                                                     *
*                int gcd_error;                                       *
*                                                                     *
*                gcd_init();                                          *
*                gcd_reset();                                         *
*                gcd_enable();                                        *
*                gcd_disable();                                       *
*                gcd_receive_frame();                                 *
*                gcd_transmit_frame();                                *
*                gcd_status();                                        *
*                                                                     *
* Notes:         1) See SDS pp. 5-6 through 5-x for more information. *
*                                                                     *
* Edit History: 06/29/90 - Written by Robin T. Laird.                 *
*                                                                     *
\*********************************************************************/

/* Public Data Structures:                                            */

#ifndef GCD_MODULE_CODE
#define GCD_MODULE_CODE          1000

#define GCD_ERR_NOT_INIT         1+GCD_MODULE_CODE
#define GCD_ERR_FRAME_LENGTH     2+GCD_MODULE_CODE
#define GCD_ERR_NUM_ATTEMPTS     3+GCD_MODULE_CODE

#define GCD_ERR_RECEIVE_FRAME    4+GCD_MODULE_CODE
#define GCD_ERR_TRANSMIT_FRAME   5+GCD_MODULE_CODE

#define GCD_FAIL_RECEIVER        6+GCD_MODULE_CODE
#define GCD_FAIL_TRANSMITTER     7+GCD_MODULE_CODE

#define GCD_WAIT_FOREVER         65535
#define GCD_DONT_WAIT            0

#define GCD_MAX_FRAME_LENGTH     SYS_MAX_PACKET_SIZE
#define GCD_MAX_ATTEMPTS         60000

#define GCD_DEST_ADDR_POS        0
#define GCD_LEN_POS              1
#define GCD_SRC_ADDR_POS         2

/* Type for receive/transmit data frame, simply a 256-element array.  */

typedef byte gcd_frame[GCD_MAX_FRAME_LENGTH];

/* Structure type for GCD module status (holds rcv/xmt error counts). */

typedef struct { word r_valid_cnt;
                 word r_err_cnt;
                 word r_crce_err_cnt;
                 word r_ae_err_cnt;
                 word r_rcabt_err_cnt;
                 word r_ovr_err_cnt;
                 word x_valid_cnt;
                 word x_err_cnt;
                 word x_noack_err_cnt;
```



```c
                 word x_ur_err_cnt;
                 word x_tcdt_err_cnt;
               } gcd_state;

/* External module global error variable.                             */

extern int gcd_error;

/* Public Functions:                                                  */

void gcd_init(void);
void gcd_reset(void);
void gcd_enable(void);
void gcd_disable(void);
void gcd_receive_frame(gcd_frame f, word retry);
void gcd_transmit_frame(gcd_frame f, word retry);
void gcd_status(gcd_state *s);

#endif
```

```
/*********************************************************************\
*                              GCD.C                                  *
***********************************************************************
*                                                                     *
* CPCI:          IED90-MRA-ICN-GCS-GCD-C-RGC1                         *
*                                                                     *
* Description:   GCS communications device handler functions.         *
*                Implements the MRA ICN standard Global Communications *
*                Device (GCD) Handler module. This module contains the *
*                standard device handler functions and must include the *
*                low-level data link layer functions for the actual hardware *
*                implementation (currently implemented for the 80C152 GSC). *
*                                                                     *
*                Message format at this level (ISO OSI data link layer) is: *
*                                                                     *
*                 | byte 0 | byte 1 | byte 2 | byte 3 | byte 4 | byte n| *
*                 |-----------------------------------------------------| *
*                 |  DEST  | LENGTH | SOURCE |  xxxx  |  xxxx  |  .... | *
*                                                                     *
*                Module GCD exports the following variables/functions: *
*                                                                     *
*                int gcd_error;                                       *
*                                                                     *
*                gcd_init();                                          *
*                gcd_reset();                                         *
*                gcd_enable();                                        *
*                gcd_disable();                                       *
*                gcd_receive_frame();                                 *
*                gcd_transmit_frame();                                *
*                gcd_status();                                        *
*                                                                     *
* Notes:         1) The files BMF.H and BMF.C contain the support functions *
*                   for managing the receive and transmit circular buffers. *
*                2) The files GSC.H and GSC.C contain the required support *
*                   functions for the Global Serial Channel hardware.  *
*                                                                     *
* Edit History: 06/29/90 - Written by Richard P. Smurlo and Robin T. Laird. *
*                                                                     *
\*********************************************************************/

#include <sysdefs.h>                    /* System constants and types.    */
#include "gcd.h"                        /* GCD public literals/functions. */
#include "bmf.h"                        /* Buffer management functions.   */
#include "gsc.h"                        /* Global Serial Channel functions. */

#if defined(DEBUG)
#include <debug.h>                      /* Definitions for printf(), etc.  */
#endif

/* Public Variables:                                                      */

/* Global module error variable, gcd_error.                               */
/* gcd_error contains code of last error occurence.                       */
/* Should be set to AOK after each successful function call.              */
/* Variable can be examined by other software after each function call.   */

XDATA int gcd_error = GCD_ERR_NOT_INIT; /* Global module error variable.  */

/* Global module state variable, _gcd_state.                              */
/* Holds GCD module error counts for frame reception/transmission.        */

static XDATA gcd_state _gcd_state;      /* Global module state variable.  */

/* Globals that tracks number of xmt/rcv attempts.                        */

static XDATA int gcd_tries = 0;         /* Number of xmt/rcv attempts.    */
static XDATA int gcd_times = 0;         /* Number of attempts to make.    */
static XDATA int gcd_stop  = FALSE;     /* Indicates when to stop trying. */

/* Buffer management functions should be included here.                   */

#include "bmf.c"                        /* Buffer management functions.   */
```



```
/* Low-level data link layer support functions should be included here.   */

#include "gsc.c"                        /* Low-level GSC functions.       */


/*********************************************************************\
*                              gcd_init                               *
***********************************************************************
*                                                                     *
* Function:      Initializes the Global Communications Device Handler. *
*                Clears the transmit and receive buffers, obtains the ICN *
*                node ID, initializes the Global Serial Channel and the *
*                associated transmit and receive DMA channels, and performs *
*                a self-test of the GSC. The GSC reception/transmission *
*                error and valid frame counters of the module state structure *
*                are cleared.                                         *
*                                                                     *
* Input:         gcd_init();                                          *
*                                                                     *
* Output:        Nothing.                                             *
*                                                                     *
* Globals:       gcd_error   : module GCD.C                           *
*                _gcd_state  : module GCD.C                           *
*                rcv_buffer  : module BMF.C                           *
*                xmt_buffer  : module BMF.C                           *
*                                                                     *
* Edit History: 07/08/90 - Written by Robin T. Laird.                 *
*                                                                     *
\*********************************************************************/

void gcd_init()
{
  gcd_error = AOK;                      /* Assume function successful...  */

  /* Reset all error counting registers in module state variable.         */
  /* Valid reception/transmission counters are also cleared.              */

  _gcd_state.r_valid_cnt      = 0;
  _gcd_state.r_err_cnt        = 0;
  _gcd_state.r_crce_err_cnt   = 0;
  _gcd_state.r_ae_err_cnt     = 0;
  _gcd_state.r_rcabt_err_cnt  = 0;
  _gcd_state.r_ovr_err_cnt    = 0;
  _gcd_state.x_valid_cnt      = 0;
  _gcd_state.x_err_cnt        = 0;
  _gcd_state.x_noack_err_cnt  = 0;
  _gcd_state.x_ur_err_cnt     = 0;
  _gcd_state.x_tcdt_err_cnt   = 0;

  /* Initialize the transmit and receive buffers.                         */

  buf_clear(&xmt_buffer);
  buf_clear(&rcv_buffer);

  /* Get the node ID for this particular ICN, and pass to GSC init routine. */
  /* No errors are currently fatal (they are only warnings).              */

  gsc_sys_init(gsc_node_id());
  if (gcd_error != AOK) return;

  /* Initialize the GSC DMA controllers.                                  */
  /* No errors are currently possible.                                    */

  gsc_dma_init();
  if (gcd_error != AOK) return;

  /* Set up the GSC DMA channel receive/transmit destination/source params. */
  /* The parameters include the DMA src/dest addresses and the byte count. */
  /* Incoming frames go into the front of the rcv_buffer.                 */
  /* Outgoing frames go into the front of the xmt_buffer.                 */

  gsc_set_rcv_dst(buf_rear(rcv_buffer), GCD_MAX_FRAME_LENGTH);
  gsc_set_xmt_src(buf_front(xmt_buffer), GCD_MAX_FRAME_LENGTH);
```

```
  /* Start the GSC DMA receive channel (allow DMA to function).          */
  /* Start the GSC receiver (begin receiving data frames).               */
  /* Start the GSC transmitter for hardware based acknowledges.          */
  /* Errors are non-fatal and would cause only degraded performance.     */

  gsc_rcv_go();
  gsc_start_rcv();
  if (gcd_error != AOK) return;
  gsc_start_xmt();

  /* Enable interrupts...                                                */

  gsc_enable_interrupts();
}


/*************************************************************************\
*                               gcd_reset                                 *
***************************************************************************
*                                                                         *
* Function:     Performs a soft reset of the GCD systems.                 *
*               The receive and transmit buffers are cleared a  the       *
*               receive and transmit destination and source addresses     *
*               for the GSC DMA channels are reset to the beginning of    *
*               the buffers. The GSC reception/transmission error and     *
*               valid frame counters of the module state structure are    *
*               cleared.                                                  *
*                                                                         *
* Input:        gcd_reset();                                              *
*                                                                         *
* Output:       Nothing.                                                  *
*                                                                         *
* Globals:      gcd_error  : module GCD.C                                 *
*               _gcd_state : module GCD.C                                 *
*               rcv_buffer : module BMF.C                                 *
*               xmt_buffer : module BMF.C                                 *
*                                                                         *
* Edit History: 07/09/90 - Written by Robin T. Laird.                     *
*                                                                         *
\*************************************************************************/

void gcd_reset()
{
  gcd_error = AOK;                       /* Assume function successful...   */

  /* Reset all error counting registers in module state variable.         */
  /* Valid reception/transmission counters are also cleared.              */

  _gcd_state.r_valid_cnt     = 0;
  _gcd_state.r_err_cnt       = 0;
  _gcd_state.r_crce_err_cnt  = 0;
  _gcd_state.r_ae_err_cnt    = 0;
  _gcd_state.r_rcabt_err_cnt = 0;
  _gcd_state.r_ovr_err_cnt   = 0;
  _gcd_state.x_valid_cnt     = 0;
  _gcd_state.x_err_cnt       = 0;
  _gcd_state.x_noack_err_cnt = 0;
  _gcd_state.x_ur_err_cnt    = 0;
  _gcd_state.x_tcdt_err_cnt  = 0;

  /* Re-initialize the transmit and receive buffers.                      */

  buf_clear(&xmt_buffer);
  buf_clear(&rcv_buffer);

  /* Set up the GSC DMA channel receive/transmit destination/source params. */
  /* The parameters include the DMA src/dest addresses and the byte count.  */
  /* Incoming frames go into the front of the rcv_buffer.                   */
  /* Outgoing frames go into the front of the xmt_buffer.                   */

  gsc_set_rcv_dst(buf_rear(rcv_buffer), GCD_MAX_FRAME_LENGTH);
  gsc_set_xmt_src(buf_front(xmt_buffer), GCD_MAX_FRAME_LENGTH);
```



```
}


/*************************************************************************\
*                               gcd_enable                                *
***************************************************************************
*                                                                         *
* Function:     Enables reception and transmission of data frames.        *
*               The GSC receiver and transmitter are re-enabled.          *
*               The function assumes that the GSC has been disabled       *
*               for some reason. It is normally not necessary to enable   *
*               the GSC after it has been initialized (by gcd_init()).    *
*                                                                         *
* Input:        gcd_enable();                                             *
*                                                                         *
* Output:       Nothing.                                                  *
*                                                                         *
* Globals:      gcd_error : module GCD.C                                  *
*                                                                         *
* Edit History: 07/09/90 - Written by Robin T. Laird.                     *
*                                                                         *
\*************************************************************************/

void gcd_enable()
{
  gcd_error = AOK;                       /* Assume function successful...   */

  /* Start both the receiver and transmitter.                             */

  gsc_start_rcv();
  gsc_start_xmt();
}


/*************************************************************************\
*                               gcd_disable                               *
***************************************************************************
*                                                                         *
* Function:     Disables reception and transmission of data frames.       *
*               The GSC receiver and transmitter are disabled.            *
*               Any incoming data will be lost (dropped). The GSC must    *
*               be re-enabled using gcd_enable() before frames may be     *
*               received or transmitted again.                            *
*                                                                         *
* Input:        gcd_disable();                                            *
*                                                                         *
* Output:       Nothing.                                                  *
*                                                                         *
* Globals:      gcd_error : module GCD.C                                  *
*                                                                         *
* Edit History: 07/09/90 - Written by Robin T. Laird.                     *
*                                                                         *
\*************************************************************************/

void gcd_disable()
{
  gcd_error = AOK;                       /* Assume function successful...   */

  /* Stop both the receiver and transmitter.                              */

  gsc_stop_rcv();
  gsc_stop_xmt();
}


/*************************************************************************\
*                            gcd_receive_frame                            *
***************************************************************************
*                                                                         *
* Function:     Removes a frame from the receive buffer if possible.      *
*               If a frame is available, it is removed from the receive   *
*               buffer, and returned in the parameter f. If a frame is not *
*               available from the receive buffer, the function will perform *
```

```
*                  as follows depending upon the re-try value:                 *
*                                                                              *
*                  GCD_DONT_WAIT     : remove frame if available, return if not. *
*                  GCD_WAIT_FOREVER : wait forever for frame to be received.   *
*                  retry             : try this many times to receive frame.   *
*                                                                              *
* Input:           gcd_receive_frame(                                          *
*                    gcd_frame f; frame to be received (destination).          *
*                    word retry;  number of times to try receiving frame.      *
*                  );                                                          *
*                                                                              *
* Output:          Nothing.                                                    *
*                                                                              *
* Globals:         gcd_error   : module GCD.C                                  *
*                  gcd_tries   : module GCD.C                                  *
*                  gcd_times   : module GCD.C                                  *
*                  rcv_buffer  : module BMF.C                                  *
*                                                                              *
* Edit History: 07/08/90 - Written by Robin T. Laird.                          *
*               01/30/91 - Robin T. Laird added re-receive count/stop.         *
*                                                                              *
\******************************************************************************/

void gcd_receive_frame(f, retry)
gcd_frame f;
word retry;
{

#if defined(DEBUG)
unsigned char prev_rstat;        /* Last stored value of RSTAT.              */
unsigned char prev_bcrh0;        /* Last BCR's (high and low bytes).         */
unsigned char prev_bcrl0;
#endif

  #if defined(DEBUG)
  prev_rstat = 0xFF;
  prev_bcrh0 = 0xFF;
  prev_bcrl0 = 0xFF;
  #endif

  gcd_error = AOK;                          /* Assume function successful...  */

  gcd_tries = 0;                            /* Number of attempted rcvs.      */
  gcd_times = retry;                        /* Number of times to try.        */

  /* If we don't want to wait (GCD_DONT_WAIT):                                */
  /*   Try and remove frame from buffer.                                      */
  /*   If one not available then gcd_error = ERR_BUF_EMPTY.                   */
  /*   Else, we've successfully removed a received frame.                     */

  /* If we want to wait forever (GCD_WAIT_FOREVER):                           */
  /*   Wait until there is a frame in the buffer, then remove it.             */
  /*   This function WILL wait forever, i.e., keep trying forever.            */

  /* If we want to try a certain number of times:                             */
  /*   Set up and zero auxilliary loo counter.                                */
  /*   Loop, checking to see if frame is available and counter not expired.   */
  /*   If counter expires, indicate buffer empty and return.                  */
  /*   Else, get incoming frame and return.                                   */

  if (retry == GCD_DONT_WAIT)
  {
    buf_remove(&rcv_buffer, f);
  }
  else if (retry == GCD_WAIT_FOREVER)
  {
    while (buf_empty(&rcv_buffer))

      #if defined(DEBUG)
      if ((prev_rstat != RSTAT)||(prev_bcrh0 != BCRH0)||(prev_bcrl0 != BCRL0))
      {
        printf("gcd_receive_frame: RSTAT=%02bx, BCR(HL)0=%02bx%02bx\n", RSTAT,
        prev_rstat = RSTAT;
```



```
        prev_bcrh0 = BCRH0;
        prev_bcrl0 = BCRL0;
      }
      #endif
      ;

    buf_remove(&rcv_buffer, f);
  }
  else if (retry <= GCD_MAX_ATTEMPTS)
  {
    while (buf_empty(&rcv_buffer))
      if (++gcd_tries >= gcd_times)
      {
        gcd_error = GCD_ERR_RECEIVE_FRAME;
        break;
      }
  }
  else
  {
    gcd_error = GCD_ERR_NUM_ATTEMPTS;
  }
}


/******************************************************************************\
*                              gcd_transmit_frame                              *
********************************************************************************
*                                                                              *
* Function:     Adds the parameter frame to the transmit buffer if possible.   *
*               If the transmit buffer is full an error is generated.          *
*               Otherwise, if the buffer is empty, the frame is transmitted    *
*               immediately, and the frame information is inserted into the    *
*               transmit buffer. Depending upon the re-try value, the          *
*               function will perform as follows:                              *
*                                                                              *
*               GCD_DONT_WAIT    : return immediately, frame added to buffer.*
*               GCD_WAIT_FOREVER : wait forever for frame to be transmitted.  *
*               retry            : try this many times to transmit frame.     *
*                                                                              *
*               Currently, only the GCD_WAIT_FOREVER option is supported.      *
*               This is equivalent to a one frame deep transmit buffer.        *
*                                                                              *
* Input:        gcd_transmit_frame(                                            *
*                 gcd_frame f; frame to be transmitted.                        *
*                 word retry;  number of times to try transmitting frame.      *
*               );                                                             *
*                                                                              *
* Output:       Nothing.                                                       *
*                                                                              *
* Globals:      gcd_error  : module GCD.C                                      *
*               _gcd_state : module GCD.C (DEBUG)                              *
*               gcd_tri·   : module GCD.C                                      *
*               gcd_tim· . : module GCD.C                                      *
*               gcd_stop   : module GCD.C                                      *
*               xmt_buffer : module BMF.C                                      *
*                                                                              *
* Edit History: 07/08/90 - Written by Robin T. Laird.                          *
*               01/30/91 - Robin T. Laird added re-transmit count/stop.        *
*                                                                              *
\******************************************************************************/

void gcd_transmit_frame(f, retry)
gcd_frame f;
word retry;
{

#if defined(DEBUG)
unsigned char   prev_tstat;     /* Last stored value of TSTAT.                */
unsigned char   prev_bcrh1;     /* Last BCR's (high and low bytes).           */
unsigned char   prev_bcrl1;
#endif

  #if defined(DEBUG)
```

```c
    prev_tstat = 0xFF;
    prev_bcrhl = 0xFF;
    prev_bcrll = 0xFF;
    #endif

    gcd_error   = AOK;                      /* Assume function successful...    */

    gcd_tries   = 0;                        /* Number of attempted xmts.        */
    gcd_times   = retry;                    /* Number of times to try.          */
    gcd_stop    = FALSE;                    /* Don't stop re-transmissions.     */

    /* Insert frame into transmit buffer and send it.                          */
    /* If the transmit buffer is empty then the GCS transmitter is disbaled:   */
    /*    Reset the DMA transmitter source address and byte count.             */
    /*    Re-start the GSC transmitter (it was turned off when buf empty).     */
    /*    Re-enable the DMA transmit channel (also turned off when buf empty). */
    /* Else, just insert frame into buffer for transmission (sometime later). */
    /* If the insert failed, abort transmission and return an error.           */

    if (buf_empty(&xmt_buffer))
    {
      buf_insert(&xmt_buffer, f);
      if (gcd_error != AOK)
      {
        gcd_error = GCD_ERR_TRANSMIT_FRAME;
        return;
      }
      else
      {
        gsc_set_xmt_src(buf_front(xmt_buffer), f[GCD_LEN_POS]);
        gsc_start_xmt();
        gsc_xmt_go();
      }
    }
    else
    {
      buf_insert(&xmt_buffer, f);
      if (gcd_error != AOK)
      {
        gcd_error = GCD_ERR_TRANSMIT_FRAME;
        return;
      }
    }

    /* At a later date, it will be possible to add a frame to the transmit    */
    /* buffer and then return to the calling function immediately, the frame */
    /* would be transmitted when it moved to the front of the buffer.         */
    /* A re-try of zero would indicate that the frame is to be transmitted    */
    /* in the above manner (i.e., don't wait for frame to be transmitted).    */

    if (retry == GCD_DONT_WAIT)
    {
      gcd_error = GCD_ERR_TRANSMIT_FRAME;
    }
    else if (retry == GCD_WAIT_FOREVER)
    {
      while (!buf_empty(&xmt_buffer))
      {
        #if defined(DEBUG)
        if ((prev_tstat != TSTAT)||(prev_bcrhl != BCRHl)||(prev_bcrll != BCRLl))
        {
          printf("gcd_transmit_frame: TSTAT=%02bx, BCR(HL)]=%02bx%02bx\n", TSTAT,
          prev_tstat = TSTAT;
          prev_bcrhl = BCRHl;
          prev_bcrll = BCRLl;
        }
        printf("noack = %u, ur = %u, tcdt = %u, xmt_valid = %u, rcv_valid = %u\n"
          _gcd_state.x_noack_err_cnt,
          _gcd_state.x_ur_err_cnt,
          _gcd_state.x_tcdt_err_cnt,
          _gcd_state.x_valid_cnt,
          _gcd_state.r_valid_cnt);
        #endif
```



```c
      }
    }
    else if (retry <= GCD_MAX_ATTEMPTS)
    {
      while (!buf_empty(&xmt_buffer))
        if (gcd_stop)
        {
          buf_remove(&xmt_buffer, f);
          gcd_error = GCD_ERR_TRANSMIT_FRAME;
          break;
        }
    }
    else
    {
      gcd_error = GCD_ERR_NUM_ATTEMPTS;
    }
  }


/***************************************************************************\
*                                gcd_status                                 *
*****************************************************************************
*                                                                           *
* Function:     Returns the operational status of the GCD subsystem.       *
*               This is accomplished by returning the contents of the      *
*               module state structure that records various operational    *
*               parameters such as the number of valid frames received, etc. *
*                                                                           *
* Input:        gcd_status(                                                 *
*                 gcd_state *s; pointer to module state structure.         *
*               );                                                          *
*                                                                           *
* Output:       Nothing.                                                    *
*                                                                           *
* Globals:      gcd_error : module GCD.C                                    *
*                                                                           *
* Edit History: 07/09/90 - Written by Richard P. Smurlo.                    *
*                                                                           *
\***************************************************************************/

void gcd_status(s)
gcd_state *s;
{
  gcd_error = AOK;                          /* Assume function successful...    */

  *s = _gcd_state;
}
```

```c
/***************************************************************************\
*                                 GCI.H                                     *
*****************************************************************************
*                                                                           *
* CPCI:          IED90-MRA-ICN-GCS-GCI-H-ROCO                               *
*                                                                           *
* Description:   GCS communications interface variables and functions.      *
*                Contains constant function parameter declarations as well  *
*                as function return values (for success and failure of all  *
*                operations). Contains the function prototypes for the GCI.C *
*                module.                                                    *
*                                                                           *
*                Module GCI exports the following types/variables/functions: *
*                                                                           *
*                typedef gci_message;                                       *
*                                                                           *
*                int gci_error;                                             *
*                                                                           *
*                gci_init();                                                *
*                gci_receive_message();                                     *
*                gci_send_message();                                        *
*                                                                           *
* Notes:         1) See SDS pp. 5-6 through 5-x for more information.       *
*                                                                           *
* Edit History: 06/29/90 - Written by Robin T. Laird.                       *
*                                                                           *
\***************************************************************************/

/* Public Data Structures:                                                  */

#ifndef GCI_MODULE_CODE
#define GCI_MODULE_CODE          2000

#define GCI_ERR_NOT_INIT         1+GCI_MODULE_CODE
#define GCI_ERR_RECEIVE_MESSAGE  2+GCI_MODULE_CODE
#define GCI_ERR_SEND_MESSAGE     3+GCI_MODULE_CODE

/* Maximum and minimum retry values for send/receive of packets.            */
/* Values must correspond with related definitions in module GCD.H.         */

#define GCI_WAIT_FOREVER         65535
#define GCI_DONT_WAIT            0
#define GCI_MAX_ATTEMPTS         60000

/* Maximum message length SHOULD be two less than maximum frame length.     */

#define GCI_MAX_MESSAGE_LENGTH   SYS_MAX_PACKET_SIZE

/* MRA inter-module message type defined as a sequence of bytes.            */

typedef byte gci_message[GCI_MAX_MESSAGE_LENGTH];

/* External module global error variable.                                   */

extern int gci_error;

/* Public Functions:                                                        */

void gci_init();
void gci_receive_message(gci_message m, word retry);
void gci_send_message(gci_message m, word retry);

#endif
```

```
/***********************************************************************\
*                                GCI.C                                  *
*************************************************************************
*                                                                       *
* CPCI:          IED90-MRA-ICN-GCS-GCI-C-ROCO                           *
*                                                                       *
* Description:   GCS communications interface functions.                *
*                Implements the MRA ICN standard Global Communications  *
*                Interface (GCI) module. This module contains the standard *
*                communications interface functions that provide higher-level *
*                software access to the Intelligent Communications Node (ICN) *
*                Local Area Network (LAN).                              *
*                                                                       *
*                Message format at this level (ISO OSI network layer) is: *
*                                                                       *
*                 | byte 0 | byte 1 | byte 2 | byte 3 | byte 4 | byte n| *
*                 |----------------------------------------------------| *
*                 |  DEST  | LENGTH | SOURCE |  xxxx  |  xxxx  |  .... | *
*                                                                       *
*                Module GCI exports the following variables/functions:  *
*                                                                       *
*                int gci_error;                                         *
*                                                                       *
*                gci_init();                                            *
*                gci_receive_message();                                 *
*                gci_send_message();                                    *
*                                                                       *
* Notes:         1) The GCI functions are implementation independent.   *
*                2) Module GCI represents the MRA Communications Level. *
*                                                                       *
* Edit History: 07/28/90 - Written by Robin T. Laird.                   *
*                                                                       *
\***********************************************************************/

#include <sysdefs.h>                     /* System constants and types.      */
#include "gcd.h"                         /* GCD public literals/functions.   */
#include "gci.h"                         /* GCI public literals/functions.   */

/* Public Variables:                                                         */

/* Global module error variable, gci_error.                                  */
/* gci_error contains code of last error occurence.                          */
/* Should be set to AOK after each successful function call.                 */
/* Variable can be examined by other software after each function call.      */

XDATA int gci_error = GCI_ERR_NOT_INIT; /* Global module error variable.     */


/***********************************************************************\
*                               gci_init                                *
*************************************************************************
*                                                                       *
* Function:      Initializes the Global Communications Interface.       *
*                The Global Communications Devic Handler (GCD) subsystem *
*                along with all module variables are initialized. Any errors *
*                are examined for severity and an attempt is made to recover *
*                from non-fatal conditions. If initialization is unsuccessful *
*                then the error GCI_ERR_NOT_INIT is returned in gci_error. *
*                                                                       *
* Input:         gci_init();                                            *
*                                                                       *
* Output:        Nothing.                                               *
*                                                                       *
* Globals:       gci_error : module GCI.C                               *
*                gcd_error : module GCD.C                               *
*                                                                       *
* Edit History: 07/28/90 - Written by Robin T. Laird.                   *
*                                                                       *
\***********************************************************************/

void gci_init()
{
  gci_error = AOK;                      /* Assume function successful...    */
```



```

  /* Initialize the GCD subsystem (sets up ICN LAN hardware and software).  */

  gcd_init();
  if (gcd_error != AOK)
  {
    switch(gcd_error)
    {
      case GCD_FAIL_RECEIVER:
      case GCD_FAIL_TRANSMITTER:
        gci_error = GCI_ERR_NOT_INIT;
        break;
      default:
        gci_error = gcd_error;
    }
  }
}


/***********************************************************************\
*                          gci_receive_message                          *
*************************************************************************
*                                                                       *
* Function:      Receives the latest (oldest) message from the ICN LAN. *
*                If a message is not immediately available, the function *
*                attempts a specific number of recpetions (as given below) *
*                and then returns regardless. If a message is never received, *
*                then the global variable gci_error is set to the literal *
*                GCI_ERR_RECEIVE_MESSAGE. If a message is available then it *
*                is returned immediately.                               *
*                                                                       *
*                The number of receive errors is tracked so that if it  *
*                exceeds a maximum value over time, the GCS device handler *
*                is reset to try and remedy the problem.                *
*                                                                       *
* Input:         gci_receive_message(                                   *
*                  gci_message m; received message.                     *
*                  word retry;    number of times to try receiving message. *
*                );                                                     *
*                                                                       *
* Output:        Nothing.                                               *
*                                                                       *
* Globals:       gci_error : module GCI.C                               *
*                gcd_error : module GCD.C                               *
*                                                                       *
* Edit History: 08/11/90 - Written by Robin T. Laird.                   *
*                                                                       *
\***********************************************************************/

#define MAX_RECV_ERRORS         1000    /* On > 1000 errors, reset GCD.     */

void gci_receive_message(m, retry)
gci_message m;
word retry;
{
  gcd_state status;                     /* Holds GCD status.                */

  gci_error = AOK;                      /* Assume function successful...    */

  /* Request frame. Iterate retry number of times.                          */

  gcd_receive_frame(m, retry);

  /* If a frame is not available, check integrity of receiver.              */
  /* If number of receive errors is excessive then reset the GCD subsystem. */
  /* Currently only counting CRC and alignment errors.                      */

  /* If a frame is available then return it.                                */

  if (gcd_error != AOK)
  {
    gcd_status(&status);
    if (status.r_crce_err_cnt+status.r_ae_err_cnt > MAX_RECV_ERRORS) gcd_reset(
```

```
    gci error = GCI_ERR_RECEIVE_MESSAGE;
  }
}


/*********************************************************************\
*                         gci_send_message                            *
***********************************************************************
*                                                                     *
* Function:      Sends the parameter message to the ICN LAN.          *
*                If the message cannot be sent immediately, the function *
*                attempts a specific number of re-transmissions (given below) *
*                then returns regardless. If the message is never sent, then *
*                the global variable gci_error is set to GCI_ERR_SEND_MESSAGE.*
*                                                                     *
*                The number of send errors is tracked so that if it   *
*                exceeds a maximum value over time, the GCS device handler *
*                is reset to try and remedy the problem.              *
*                                                                     *
* Input:         gci_send_message(                                    *
*                  gci_message m; message to be sent.                 *
*                  word retry;    number of times to try sending message. *
*                );                                                   *
*                                                                     *
* Output:        Nothing.                                             *
*                                                                     *
* Globals:       gci error : module GCI.C                             *
*                gcd error : module GCD.C                             *
*                                                                     *
* Edit History: 08/11/90 - Written by Robin T. Laird.                 *
*                                                                     *
\*********************************************************************/

#define MAX_SEND_ERRORS        1000    /* > 1000 errors, reset GCD.       */

void gci_send_message(m, retry)
gci_message m;
word retry;
{
  gcd_state status;                    /* Holds GCD status.               */

  gci_error = AOK;                     /* Assume function successful...   */

  /* Send frame. Iterate retry number of times.                           */

  gcd_transmit_frame(m, retry);

  /* If frame could not be transmitted, check integrity of transmitter.   */
  /* If number of transmit errors is excessive then reset the GCD s bsystem.*/
  /* Currently only counting no acknowledgement errors.                   */

  if (gcd_error != AOK)
  {
    gcd_status(&status);
    if (status.x_noack_err_cnt > MAX_SEND_ERRORS) gcd_reset();
    gci_error = GCI_ERR_SEND_MESSAGE;
  }
}
```

```
/**************************************************************************\
*                              GSC.H                                       *
****************************************************************************
*                                                                          *
* CPCI:         IED90-MRA-ICN-GCS-GSC-H-ROCO                               *
*                                                                          *
* Description:  Global Serial Channel (GSC) variables and functions.       *
*               Contains constant function parameter and hardware register *
*               declarations for initializatin and control of the Intel    *
*               80C152 GSC. Implements the low-level data link layer portion *
*               of the MRA GCS communications device handler functions.    *
*                                                                          *
*               Module GSC exports the following functions:                *
*                                                                          *
*               gsc_node_id();                                             *
*               gsc_sys_init();                                            *
*               gsc_dma_init();                                            *
*               gsc_set_rcv_dst();                                         *
*               gsc_set_xmt_src();                                         *
*               gsc_start_rcv();                                           *
*               gsc_start_xmt();                                           *
*               gsc_rcv_go();                                              *
*               gsc_xmt_go();                                              *
*               gsc_stop_rcv();                                            *
*               gsc_stop_xmt();                                            *
*               gsc_enable_interrupts();                                   *
*               gsc_rcv_valid() interrupt;                                 *
*               gsc_xmt_valid() interrupt;                                 *
*               gsc_rcv_error() interrupt;                                 *
*               gsc_xmt_error() interrupt;                                 *
*                                                                          *
* Notes:        1) See the Intel 8-Bit Embedded Controller Handbook for    *
*                  more information (No. 270645-002, pp. 9-1 - 9-87).      *
*                                                                          *
* Edit History: 06/28/90 - Written by Richard P. Smurlo.                   *
*                                                                          *
\**************************************************************************/

/* Private Data Structures:                                                */

#ifndef GSC_MODULE_CODE
#define GSC_MODULE_CODE          1200

#define ERR_NODE_ID_TOO_LARGE    1+GSC_MODULE_CODE
#define ERR_RFIFO_NOT_CLEAR      2+GSC_MODULE_CODE

/* GSC Constants:                                                          */

#define ADR_TFIFO            0x85 /* Transmit FIFO Address.                 */
#define ADR_RFIFO            0xF4 /* Receive FIFO Address.                  */
#define GSC_BAUD             0x01 /* GSC Baud Rate = (OSCf)/((GSC_BAUD+1)*8). */
                                  /*   GSC_BAUD = 1 > Baud Rate = 921.6 KBPS. */
#define CLOCK_MASK           0x00 /* GMOD = 0b0XXX|XXXX for Internal clock. */
#define D_BKOFF_GMOD         0x60 /* GMOD = 0bX11X|XXXX for alternate BKOFF. */
#define ADDR_MASK            0x00 /* GMOD = 0bXXX0|XXXX for 8-bit Addresses. */
#define CRC_MASK             0x00 /* GMOD = 0bXXXX|0XXX for 16-bit CRC.     */
#define PREAMBLE_MASK        0x02 /* GMOD = 0bXXXX|X01X for 8-bit preamble. */
#define PROTOCOL_MASK        0x00 /* GMOD = 0bXXXX|XXX0 for CSMA/CD protocol. */

#define JAM_MASK             0x80 /* MYSLOT = 0b1XXX|XXXX for DC Jam Signal. */
#define D_BKOFF_SLOT         0x40 /* MYSLOT = 0bX1XX|XXXX for deterministic */
                                  /*   resolution.                          */
#define GSC_IFS              0    /* IFS = Number of Machine Cycles for     */
                                  /*   longest Receive service routine      */
#define GSC_SLOTTM           216  /* 256-SLOTTM = # of bit times for round  */
                                  /*   trip propagation and CRC jam time.   */
#ifdef ICNMON
#define GSC_HABEN            0    /* ICN Monitor should not Acknowledge.    */
#define GSC_AMSKO            0xFF /* AMSKO = 0xFF - Mask off ADRO Node ID.  */
#else
#define GSC_HABEN            1    /* HABEN = 1 for Hardware Based Acknowledges*/
#define GSC_AMSKO            0x00 /* AMSKO = 0x00 - Don't mask ADRO Node ID. */
#endif
```



```

#define GSC_AMSK1            0x00 /* AMSK1 = 0x00 - Don't mask off any part  */
                                  /*   of Address 1.                          */
#define GSC_DMA              1    /* DMA = 1 for DMA control of GSC.          */

#define NO_SLOT              0x00 /* MYSLOT = 0bXX00|0000 if Node ID > 32.    */
#define MAX_SLOTS            32   /* Maximum number of slots used.            */

#define R_OVR_ERR_MASK       0x80 /* Receive over-run Error mask.             */
#define R_RCABT_ERR_MASK     0x40 /* Receive Abort Error mask.                */
#define R_AE_ERR_MASK        0x20 /* Receive Alignment Error mask.            */
#define R_CRCE_ERR_MASK      0x10 /* Receive CRC Error mask.                  */
#define X_NOACK_ERR_MASK     0x40 /* Transmit No Acknowledge Error mask.      */
#define X_UR_ERR_MASK        0x20 /* Transmit Under-run Error mask.           */
#define X_TCDT_ERR_MASK      0x10 /* Transmit Collision Detect Error mask.    */

/* DMA Constants:                                                             */

#define HLDHLDA_DISABLE      0x9F /* PCON = 0bX00X|XXXX to disable the Hold/  */
                                  /*   Hold Acknowledge Logic.                */
#define R_XFERMODE_MASK      0x00 /* DCONO = 0bXXXX|X0XX for response to GSC  */
                                  /*   interrupts, alternate cycle mode.      */
#define R_DMNDMODE_MASK      0x08 /* DCONO = 0bXXXX|1XXX to enable demand mode*/
#define R_SRC_AUTOINC_MASK   0x00 /* DCONO = 0bXXX0|XXXX to clear auto incre- */
                                  /*   ment option for source addr.           */
#define R_SOURCE_MASK        0x20 /* DCONO = 0bXX1X|XXXX to define DMA source */
                                  /*   as SFR.                                */
#define R_DST_AUTOINC_MASK   0x40 /* DCONO = 0bX1XX|XXXX to set auto increment*/
                                  /*   option for receive dest. address.      */
#define R_DEST_MASK          0x00 /* DCONO = 0b0XXX|XXXX for DMA dest as ex-  */
                                  /*   ternal memory.                         */

#define X_XFERMODE_MASK      0x00 /* DCON1 = 0bXXXX|X0XX for response to GSC  */
                                  /*   interrupts, alternate cycle mode.      */
#define X_DMNDMODE_MASK      0x08 /* DCON1 = 0bXXXX|1XXX to enable demand mode*/
#define X_SRC_AUTOINC_MASK   0x10 /* DCON1 = 0bXXX1|XXXX to auto increment    */
                                  /*   source address.                        */
#define X_SOURCE_MASK        0x00 /* DCON1 = 0bXX0X|XXXX to select External   */
                                  /*   RAM as source.                         */
#define X_DST_AUTOINC_MASK   0x00 /* DCON1 = 0bX0XX|XXXX to clear auto incre- */
                                  /*   ment option for dest. addr.            */
#define X_DEST_MASK          0x80 /* DCON1 = 0b1XXX|XXXX to define DMA dest.  */
                                  /*   as SFR.                                */

/* Private Functions:                                                         */

static byte gsc_node_id(void);
static void gsc_sys_init(byte node_id);
static void gsc_dma_init(void);
static void gsc_set_rcv_dst(gcd_frame f, word length);
static void gsc_set_xmt_src(gcd_frame f, word length);
static void gsc_start_rcv(void);
static void gsc_start_xmt(void);
static void gsc_rcv_go(void);
static void gsc_xmt_go(void);
static void gsc_stop_rcv(void);
static void gsc_stop_xmt(void);
static void gsc_stop_xmt(void);
static void gsc_enable_interrupts();

#endif
```

```c
/***************************************************************************\
*                                GSC.C                                     *
****************************************************************************
*                                                                          *
* CPCI:          IED90-MRA-ICN-GCS-GSC-C-R0C2                              *
*                                                                          *
* Description:   Global Serial Channel (GSC) functions.                    *
*                Implements the MRA ICN low-level data link layer functions *
*                for the GCS global communications device handler.         *
*                Currently supports the 80C152 Global Serial Channel (GSC). *
*                                                                          *
*                Module GSC exports the following functions:               *
*                                                                          *
*                xptr_lo_offset() macro;                                   *
*                xptr_hi_offset() macro;                                   *
*                gsc_node_id();                                            *
*                gsc_sys_init();                                           *
*                gsc_dma_init();                                           *
*                gsc_set_rcv_dst();                                        *
*                gsc_set_xmt_src();                                        *
*                gsc_start_rcv();                                          *
*                gsc_start_xmt();                                          *
*                gsc_rcv_go();                                             *
*                gsc_xmt_go();                                             *
*                gsc_stop_rcv();                                           *
*                gsc_stop_xmt();                                           *
*                gsc_enable_interrupts();                                  *
*                gsc_rcv_valid() interrupt;                                *
*                gsc_xmt_valid() interrupt;                                *
*                gsc_rcv_error() interrupt;                                *
*                gsc_xmt_error() interrupt;                                *
*                                                                          *
* Notes:         1) This module is NOT a stand-alone compilation unit.     *
*                   It is included by the module GCD.C and is compiled there. *
*                   It is assumed that the file GCD.H is included before it. *
*                   Note that all of the functions herein are static.      *
*                                                                          *
* Edit History: 07/10/90 - Written by Richard P. Smurlo and Robin T. Laird. *
*                                                                          *
\***************************************************************************/

#include <reg152.h>                          /* 8031 and 80152 definitions.      */


/***************************************************************************\
*                             xptr_lo_offset                               *
****************************************************************************
*                                                                          *
* Function:      Obtains the low-offset portion of an external data (XDATA) *
*                pointer. Specific to the Franklin 8031 C compiler (V2.4). *
*                                                                          *
* Input:         xptr_lo_offset(                                           *
*                  xdata void *xp; XDATA pointer.                          *
*                );                                                        *
*                                                                          *
* Output:        Returns the low-offset portion of the external data pointer. *
*                                                                          *
* Globals:       None.                                                     *
*                                                                          *
* Edit History: 07/07/90 - Written by Robin T. Laird.                      *
*                                                                          *
\***************************************************************************/

#define xptr_lo_offset(xp)  (((unsigned int)xp)&0x00FF)


/***************************************************************************\
*                             xptr_hi_offset                               *
****************************************************************************
*                                                                          *
* Function:      Obtains the hi-offset portion of an external data (XDATA) *
*                pointer. Specific to the Franklin 8031 C compiler (V2.4). *
*                                                                          *
```



```c
* Input:         xptr_hi_offset(                                           *
*                  xdata void *xp; XDATA pointer.                          *
*                );                                                        *
*                                                                          *
* Output:        Returns the hi-offset portion of the external data pointer. *
*                                                                          *
* Globals:       None.                                                     *
*                                                                          *
* Edit History: 07/07/90 - Written by Robin T. Laird.                      *
*                                                                          *
\***************************************************************************/

#define xptr_hi_offset(xp)  ((((unsigned int)xp)>>8)&0x00FF)


/***************************************************************************\
*                               gsc_node_id                                *
****************************************************************************
*                                                                          *
* Function:      Determines node ID number for the particular ICN.         *
*                The ID number is currently read from the P6 80C152        *
*                parallel port which is connected to an 8-position switch. *
*                                                                          *
* Input:         gsc_node_id();                                            *
*                                                                          *
* Output:        Returns the node identification number (address) as a byte. *
*                Valid range is 0 <= node ID <= 0xFF.                      *
*                                                                          *
* Globals:       gcd_error    : module GCD.C                               *
*                80C152 regs : module REG152.H                             *
*                                                                          *
* Edit History: 06/28/90 - Written by Robin T. Laird.                      *
*                                                                          *
\***************************************************************************/

static byte gsc_node_id()
{
  gcd_error = AOK;                         /* Assume function successful...    */

  return ((byte)P6);
}


/***************************************************************************\
*                               gsc_sys_init                               *
****************************************************************************
*                                                                          *
* Function:      Initializes the Intel 80C152 Global Serial Channel (GSC). *
*                The GSC is set up for operation as follows:               *
*                                                                          *
*                BAUD RATE        = 1.8432 MBPS                            *
*                CLOCK SOURCE     = internal baud rate generator           *
*                PROTOCOL         = CSMA/CD                                *
*                PREAMBLE LENGTH  = 8-bits                                 *
*                BACKOFF MODE     = deterministic                          *
*                CRC LENGTH       = 16-bits                                *
*                IFS              = 256 B/Ts                               *
*                JAMMING SIGNAL   = D.C.                                   *
*                SLOT TIME        = 40 B/Ts                                *
*                ADDRESSING MODE  = 8-bits                                 *
*                HBA              = enabled                                *
*                                                                          *
*                Deterministic collision resolution requires that slot     *
*                numbers be used, and limits this implementation to 64 slots *
*                or ICN nodes. Since hardware based acknowledge is being used *
*                we are also forcing use of even addresses. We have chosen *
*                to base the slot number on the address number, which gives *
*                only 32 possible unique 8-bit addresses. If the node ID is *
*                greater than 32, then the GSC slot number is set to 0 and *
*                a warning is signaled (ERR_NODE_ID_TOO_LARGE).            *
*                                                                          *
* Input:         gsc_sys_init(                                             *
*                  byte node_id; node address (which equals slot number).  *
```

```
*                  );                                                  *
*                                                                      *
* Output:          Nothing.                                            *
*                                                                      *
* Globals:         gcd_error    : module GCD.C                         *
*                  80C152 regs : module REG152.H                       *
*                                                                      *
* Edit History: 07/10/90 - Written by Richard P. Smurlo.               *
*                  03/28/91 - Mofidied by Robin T. Laird added EA = 0. *
*                                                                      *
\**********************************************************************/

static void gsc_sys_init(node_id)
byte node_id;
{
  gcd_error = AOK;                          /* Assume function successful...    */

  /* Disable interrupts while we're changing things.                            */

  EA = 0;

  /* Initialize all GSC related Registers to 0x00.                              */

  GMOD    = 0x00;
  MYSLOT  = 0x00;

  /* Protocol Dependent Initialization.                                         */

  BAUD    = GSC_BAUD;                       /* GSC Baud Rate Select.            */
  GMOD    = GMOD   | PROTOCOL_MASK;         /* CSMA/CD or SDLC Protocol.        */
  GMOD    = GMOD   | CLOCK_MASK;            /* Int. or Ext. clock.              */
  GMOD    = GMOD   | PREAMBLE_MASK;         /* Message Preamble.                */
  GMOD    = GMOD   | D_BKOFF_GMOD;          /* Deterministic Backoff.           */
  MYSLOT  = MYSLOT | D_BKOFF_SLOT;          /* Ditto                            */
  GMOD    = GMOD   | CRC_MASK;              /* CRC definition.                  */
  IFS     = GSC_IFS;                        /* Inter Frame Space.(LSBit ignored)*/
  MYSLOT  = MYSLOT | JAM_MASK;              /* Jam Signal Mask.                 */
  SLOTTM  = GSC_SLOTTM;                     /* Slot Time.                       */
  GMOD    = GMOD   | ADDR_MASK;             /* Addressing.                      */

  /* The address mask registers are explicitly initialized to values that      */
  /* are "safe" to gaurd against false recognition of data frames. Only        */
  /* addresses that equal the node_id will be recognized. An un-advertised     */
  /* feature is that a node will actually recognize the node_id address       */
  /* (which is supposed to be an even value) AND it will recognize the         */
  /* address that is one greater than the node_id. The odd address is used     */
  /* when the sender does not want the HBA to be used.                          */

  ADR0    = node_id;                        /* Addressing. (Node ID # 1)        */
  ADR1    = node_id+1;                      /* Addressing. (Node ID # 2)        */
  ADR2    = node_id;                        /* Addressing. (Node ID # 1)        */
  ADR3    = node_id+1;                      /* Addressing. (Node ID # 2)        */
  AMSK0   = GSC_AMSK0;                      /* AMSK0 masks ADR0 for ICNMON (FF).*/
  AMSK1   = GSC_AMSK1;                      /* AMSK1 doesn't mask anything (00).*/
  HABEN   = GSC_HABEN;                      /* Hardware Based Ack.              */

  /* Deterministic Resolution Initialization.                                   */

  TCDCNT  = MAX_SLOTS;                      /* Maximum number of slots.         */
  PRBS    = 0xFF;                           /* Disable the PRBS for deter-      */
                                            /*    ministic resolution.          */

  /* Set up slot number of a given node. Higher number = higher priority.       */
  /* Slot numbers are used in deterministic collision resolution.               */
  /* Slot numbers will be based upon (equal to) node ID (address).              */
  /* Currently, an error is flagged if the node_id > MAX_SLOTS.                 */

  if (node_id <= MAX_SLOTS)
    MYSLOT = MYSLOT | node_id;
  else
  {
    MYSLOT = MYSLOT | NO_SLOT;              /* Slot number 0 if Node ID > MAX.  */
    gcd_error = ERR_NODE_ID_TOO_LARGE;      /* Warn that node ID is too big.    */
```



```
  }

  #if defined(DEBUG)
  printf("gsc_sys_init: GMOD   = %02bx\n", GMOD);
  printf("gsc_sys_init: BAUD   = <can't read>");
  printf("gsc_sys_init: MYSLOT= %02bx\n", MYSLOT);
  printf("gsc_sys_init: IFS    = %02bx (LSB indeterminate)\n", IFS);
  printf("gsc_sys_init: ADR0   = %02bx\n", ADR0);
  printf("gsc_sys_init: ADR1   = %02bx\n", ADR1);
  printf("gsc_sys_init: ADR2   = %02bx\n", ADR2);
  printf("gsc_sys_init: ADR3   = %02bx\n", ADR3);
  printf("gsc_sys_init: SLOTTM= %02bx\n", SLOTTM);
  printf("gsc_sys_init: AMSK0  = %02bx\n", AMSK0);
  printf("gsc_sys_init: AMSK1  = %02bx\n", AMSK1);
  printf("gsc_sys_init: RSTAT  = %02bx\n", RSTAT);
  printf("gsc_sys_init: TCDCNT= %02bx\n", TCDCNT);
  printf("gsc_sys_init: PRBS   = %02bx\n", PRBS);
  #endif
}


/***********************************************************************\
*                              gsc_dma_init                             *
*************************************************************************
*                                                                       *
* Function:     Initializes the Intel 80C152 Global Serial Channel (GSC) *
*               DMA controller. DMA channel 0 is used for receiving frames *
*               while DMA channel 1 is used for transmitting frames.    *
*               The DMA channels are initialized as follows:            *
*                                                                       *
*               DMA channel 0 (receiver):                               *
*                                                                       *
*               TRANSFER MODE     = GSC interrupts used to initiate transfer *
*               DMA CHANNEL MODE = demand mode                          *
*               INCREMENT SOURCE = no                                   *
*               SOURCE ADDRESS    = RFIFO SFR                           *
*               INCREMENT DEST    = yes          (done in receive routine) *
*               DEST ADDRESS      = rcv_buffer[] (done in receive routine) *
*                                                                       *
*               DMA channel 1 (transmitter):                            *
*                                                                       *
*               TRANSFER MODE     = GSC interrupts used to initiate transfer *
*               DMA CHANNEL MODE = demand mode                          *
*               INCREMENT DEST    = no                                  *
*               DEST ADDRESS      = TFIFO SFR                           *
*               INCREMENT SOURCE = yes          (done in transmit routine) *
*               SOURCE ADDRESS    = xmt_buffer[] (done in transmit routine) *
*                                                                       *
*               The channels are not "active" until their GO bits are set. *
*                                                                       *
* Input:        gsc_dma_init();                                         *
*                                                                       *
* Output:       Nothing.                                                *
*                                                                       *
* Globals:      gcd_error    : module GCD.C                             *
*               80C152 regs : module REG152.H                           *
*                                                                       *
* Edit History: 07/10/90 - Written by Richard P. Smurlo.                *
*                                                                       *
\***********************************************************************/

static void gsc_dma_init()
{
  gcd_error = AOK;                          /* Assume function successful...    */

  /* DMA Initialization:                                                        */

  DMA     = GSC_DMA;                        /* CPU or DMA control?              */
  PCON    = PCON & HLDHLDA_DISABLE;         /* Disable the Hold/Hold Ack. Logic */

  /* DMA Channel 0 Init. - Receiver:                                            */

  DCON0   = 0x00;
```

```
    DCON0    = DCON0 | R_XFERMODE_MASK;     /* Set Transfer Mode.                */
    DCON0    = DCON0 | R_DMNDMODE_MASK;     /* Set Demand Mode.                  */
    DCON0    = DCON0 | R_SRC_AUTOINC_MASK;  /* Set up Auto Increment Option.     */
    DCON0    = DCON0 | R_SOURCE_MASK;       /* Define DMA Source.                */
    DCON0    = DCON0 | R_DST_AUTOINC_MASK;  /* Auto inc destination address.     */
    DCON0    = DCON0 | R_DEST_MASK;         /* Define DMA Destination.           */

    SARL0    = ADR RFIFO;                   /* Source Address = RFIFO.           */
    SARH0    = 0x00;

  /* DMA Channel 1 Init. - Transmitter:                                          */

    DCON1    = 0x00;
    DCON1    = DCON1 | X_XFERMODE_MASK;     /* Transmit Transfer Mode.           */
    DCON1    = DCON1 | X_DMNDMODE_MASK;     /* Set Demand Mode.                  */
    DCON1    = DCON1 | X_SRC_AUTOINC_MASK;  /* Set auto increment option.        */
    DCON1    = DCON1 | X_SOURCE_MASK;       /* Define DMA Source.                */
    DCON1    = DCON1 | X_DST_AUTOINC_MASK;  /* Auto inc source address.          */
    DCON1    = DCON1 | X_DEST_MASK;         /* Define DMA Destination.           */

    DARL1    = ADR TFIFO;                   /* Dest. Addr. = TFIFO.              */
    DARH1    = 0x00;

    #if defined(DEBUG)
    printf("gsc_dma_init: TSTAT = %02bx\n", TSTAT);
    printf("gsc_dma_init: PCON  = %02bx\n", PCON);
    printf("gsc_dma_init: DCON0 = %02bx\n", DCON0);
    printf("gsc_dma_init: SARL0 = %02bx\n", SARL0);
    printf("gsc_dma_init: SARH0 = %02bx\n", SARH0);
    printf("gsc_dma_init: DCON1 = %02bx\n", DCON1);
    printf("gsc_dma_init: DARL1 = %02bx\n", DARL1);
    printf("gsc_dma_init: DARH1 = %02bx\n", DARH1);
    #endif
  }


/*****************************************************************************\
 *                               gsc_set_rcv_dst                              *
 ******************************************************************************
 *                                                                            *
 * Function:    Sets the DMA receiver destination address and byte count.     *
 *              DMA channel 0 is the receive channel. The destination for     *
 *              the reception is the parameter frame. Special care must be    *
 *              taken so that data in the destination is not over written     *
 *              (or processed) before the entire frame is received. The       *
 *              destination reception addr MUST be of type XDATA.             *
 *                                                                            *
 * Input:       gsc_set_rcv_dst(                                              *
 *                gcd_frame f; frame to be received (must be type XDATA).     *
 *                word length; frame length (0 < l <= GCD_MAX_FRAME_LENGTH).  *
 *              );                                                            *
 *                                                                            *
 * Output:      Nothing.                                                      *
 *                                                                            *
 * Globals:     gcd_error    : module GCD.C                                   *
 *              80C152 regs : module REG152.H                                 *
 *                                                                            *
 * Edit History: 06/28/90 - Written by Robin T. Laird.                        *
 *                                                                            *
\*****************************************************************************/

static void gsc_set_rcv_dst(f, length)
gcd_frame f;
word length;
{
  gcd_error = AOK;                          /* Assume function successful...   */

  /* Set up destination and byte count registers.                              */

  DARL0 = xptr_lo_offset(f);                /* Set DMA reception dest addr.    */
  DARH0 = xptr_hi_offset(f);
  BCRL0 = length & 0xFF;                    /* Set frame length hi/lo byte cnt. */
  BCRH0 = (length >> 8) & 0xFF;
```



```

  #if defined(DEBUG)
  printf("gsc_set_rcv_dst: DAR(HL)0 = %bx %bx\n", DARH0, DARL0);
  printf("gsc_set_rcv_dst: BCR(HL)0 = %bx %bx\n", BCRH0, BCRL0);
  #endif
}


/*****************************************************************************\
 *                               gsc_set_xmt_src                              *
 ******************************************************************************
 *                                                                            *
 * Function:    Sets the DMA transmitter source address and byte count.       *
 *              DMA channel 1 is the transmit channel. The source for the     *
 *              transmission is the parameter frame. Special care must be     *
 *              taken so that the source data is not over written before      *
 *              the transmission is complete. The transmission source addr    *
 *              MUST be of type XDATA. The first byte of the frame holds      *
 *              the transmission destination node address.                    *
 *                                                                            *
 * Input:       gsc_set_xmt_src(                                              *
 *                gcd_frame f; frame to be received (must be type XDATA).     *
 *                word length; frame length (0 < l <= GCD_MAX_FRAME_LENGTH).  *
 *              );                                                            *
 *                                                                            *
 * Output:      Nothing.                                                      *
 *                                                                            *
 * Globals:     gcd_error    : module GCD.C                                   *
 *              80C152 regs : module REG152.H                                 *
 *                                                                            *
 * Edit History: 06/28/90 - Written by Robin T. Laird.                        *
 *                                                                            *
\*****************************************************************************/

static void gsc_set_xmt_src(f, length)
gcd_frame f;
word length;
{
  gcd_error = AOK;                          /* Assume function successful...   */

  /* Set up source and byte count registers.                                   */

  SARL1 = xptr_lo_offset(f);                /* Set DMA tranmission source addr. */
  SARH1 = xptr_hi_offset(f);
  BCRL1 = length & 0xFF;                    /* Set frame length hi/lo byte cnt. */
  BCRH1 = (length >> 8) & 0xFF;

  #if defined(DEBUG)
  printf("gsc_set_xmt_src: SAR(HL)1 = %bx %bx\n", SARH1, SARL1);
  printf("gsc_set_xmt_src: BCR(HL)1 = %bx %bx\n", BCRH1, BCRL1);
  #endif
}


/*****************************************************************************\
 *                                gsc_start_rcv                               *
 ******************************************************************************
 *                                                                            *
 * Function:    Initiates reception of a data frame.                          *
 *              DMA channel 0 is the receive channel. It is assumed that      *
 *              the data receive destination address is already initialized. *
 *              THIS FUNCTION MUST BE CALLED AFTER gsc_rcv_go(), as in:       *
 *                                                                            *
 *              gsc_rcv_go();    -- Set DMA GO bit.                           *
 *              gsc_start_rcv(); -- Start receiver and enable interrupts.     *
 *                                                                            *
 * Input:       gsc_start_rcv();                                              *
 *                                                                            *
 * Output:      Nothing.                                                      *
 *                                                                            *
 * Globals:     gcd_error    : module GCD.C                                   *
 *              80C152 regs : module REG152.H                                 *
 *                                                                            *
```

```
 * Edit History: 07/10/90 - Written by Richard P. Smurlo and Robin T. Laird.  *
 *                                                                            *
\****************************************************************************/

#define MAX_RFIFO_READS    5              /* Read RFIFO 5 times to clear.    */

static void gsc_start_rcv()
{
  char temp;
  int  i;

  gcd_error = AOK;                        /* Assume function successful...   */

  /* Clear the receiver FIFO. Must read bytes until RFNE = 0.                */

  for (i = 0; RFNE != 0 && i < MAX_RFIFO_READS; i++) temp = RFIFO;
  if (i >= MAX_RFIFO_READS)
  {
    gcd_error = ERR_RFIFO_NOT_CLEAR;
    return;
  }

  /* Start reception...                                                      */

  GREN  = 1;                              /* Enable the GSC receiver.        */
  FGSRE = 1;                              /* Enable Receive Error ISR.       */
  FGSRV = 1;                              /* Enable Receive Valid ISR.       */
}


/****************************************************************************\
 *                              gsc_start_xmt                                 *
 ******************************************************************************
 *                                                                            *
 * Function:     Sets up transmission of data frames.                         *
 *               DMA channel 1 is the transmit channel. It is assumed that    *
 *               data transmit source address is already initialized.         *
 *               THIS FUNCTION MUST BE CALLED BEFORE gsc_xmt_go(), as in:     *
 *                                                                            *
 *               gsc_start_xmt(); -- Start transmitter and enable interrupts. *
 *               gsc_xmt_go();    -- Set DMA GO bit.                          *
 *                                                                            *
 * Input:        gsc_start_xmt();                                             *
 *                                                                            *
 * Output:       Nothing.                                                     *
 *                                                                            *
 * Globals:      gcd_error    : module GCD.C                                  *
 *               80C152 regs : module REG152.H                                *
 *                                                                            *
 * Edit History: 07/10/90 - Written by Richard P. Smurlo and Robin T. Laird.  *
 *                                                                            *
\****************************************************************************/

static void gsc_start_xmt()
{
  gcd_error = AOK;                        /* Assume function successful...   */

  /* Start transmission...                                                   */

  TEN   = 1;                              /* TEN = 1 to enable the xmitter.  */
  EGSTE = 1;                              /* Enable Transmit Error ISR.      */
  EGSTV = 1;                              /* Enable Transmit Valid ISR.      */
}


/****************************************************************************\
 *                              gsc_rcv_go                                    *
 ******************************************************************************
 *                                                                            *
 * Function:     Sets the DMA GO bit.                                         *
 *               DMA channel 0 is the receive channel. It is assumed that     *
 *               the data receive destination address is already initialized. *
 *               THIS FUNCTION MUST BE CALLED BEFORE gsc_start_rcv().         *
```



```
 *                                                                            *
 * Input:        gsc_rcv_go();                                                *
 *                                                                            *
 * Output:       Nothing.                                                     *
 *                                                                            *
 * Globals:      gcd_error    : module GCD.C                                  *
 *               80C152 regs : module REG152.H                                *
 *                                                                            *
 * Edit History: 07/10/90 - Written by Richard P. Smurlo and Robin T. Laird.  *
 *                                                                            *
\****************************************************************************/

static void gsc_rcv_go()
{
  gcd_error = AOK;                        /* Assume function successful...   */

  /* Start reception...                                                      */

  DCON0 = DCON0 | 0x01;                   /* Set DMA GO bit.                 */
}


/****************************************************************************\
 *                              gsc_xmt_go                                    *
 ******************************************************************************
 *                                                                            *
 * Function:     Starts transmission of data frames.                          *
 *               DMA channel 1 is the transmit channel. It is assumed that    *
 *               data transmit source address is already initialized.         *
 *               THIS FUNCTION MUST BE CALLED AFTER gsc_start_xmt().          *
 *                                                                            *
 * Input:        gsc_xmt_go();                                                *
 *                                                                            *
 * Output:       Nothing.                                                     *
 *                                                                            *
 * Globals:      gcd_error    : module GCD.C                                  *
 *               80C152 regs : module REG152.H                                *
 *                                                                            *
 * Edit History: 07/10/90 - Written by Richard P. Smurlo and Robin T. Laird.  *
 *                                                                            *
\****************************************************************************/

static void gsc_xmt_go()
{
  gcd_error = AOK;                        /* Assume function successful...   */

  /* Start transmission...                                                   */

  DCON1 = DCON1 | 0x01;                   /* Set DMA GO bit.                 */
}


/****************************************************************************\
 *                              gsc_stop_rcv                                  *
 ******************************************************************************
 *                                                                            *
 * Function:     Disables the GSC receiver by turning off the enable bit.     *
 *               Both the receive error and receive valid interrupt routines  *
 *               are disabled. Any incoming frame is dropped which will cause *
 *               a hardware acknowledge error at the other end.               *
 *                                                                            *
 * Input:        gsc_stop_rcv();                                              *
 *                                                                            *
 * Output:       Nothing.                                                     *
 *                                                                            *
 * Globals:      gcd_error    : module GCD.C                                  *
 *               80C152 regs : module REG152.H                                *
 *                                                                            *
 * Edit History: 07/10/90 - Written by Richard P. Smurlo.                     *
 *                                                                            *
\****************************************************************************/

static void gsc_stop_rcv()
```

```c
  {
    gcd_error = AOK;                        /* Assume function successful...   */

    GREN  = 0;                              /* GREN = 0 to disable the recvr.  */
    EGSRE = 0;                              /* Disable Receive Error ISR.      */
    EGSRV = 0;                              /* Disable Receive Valid ISR.      */
    DCON0 = DCON0 & 0xFE;                   /* Re-set DMA GO bit.              */
  }


/*****************************************************************************\
 *                              gsc_stop_xmt                                 *
 *****************************************************************************
 *                                                                           *
 * Function:     Disables the GSC transmitter by turning off the enable bit. *
 *               Both the transmit error and transmit valid interrupt        *
 *               routines are disabled. Any outgoing frame is terminated     *
 *               will cause a CRC or alignment reception error at the other  *
 *               end.                                                        *
 *                                                                           *
 * Input:        gsc_stop_xmt();                                             *
 *                                                                           *
 * Output:       Nothing.                                                    *
 *                                                                           *
 * Globals:      gcd_error    : module GCD.C                                 *
 *               80C152 regs  : module REG152.H                              *
 *                                                                           *
 * Edit History: 07/10/90 - Written by Richard P. Smurlo.                    *
 *                                                                           *
\*****************************************************************************/

static void gsc_stop_xmt()
  {
    gcd_error = AOK;                        /* Assume function successful...   */

    TEN   = 0;                              /* TEN = 0 to disable transmitter. */
    EGSTE = 0;                              /* Disable Transmit Error ISR.     */
    EGSTV = 0;                              /* Disable Transmit Valid ISR.     */
    DCON1 = DCON1 & 0xFE;                   /* Re-set DMA GO bit.              */
  }


/*****************************************************************************\
 *                          gsc_enable_interrupts                            *
 *****************************************************************************
 *                                                                           *
 * Function:     Enables processor interrupts for the 8031/80152.            *
 *                                                                           *
 * Input:        gsc_enable_interrupts();                                    *
 *                                                                           *
 * Output:       Nothing.                                                    *
 *                                                                           *
 * Globals:      gcd_error    : module GCD.C                                 *
 *               80C152 regs  : module REG152.H                              *
 *                                                                           *
 * Edit History: 03/09/91 - Written by Robin T. Laird.                       *
 *                                                                           *
\*****************************************************************************/

static void gsc_enable_interrupts()
  {
    gcd_error = AOK;                        /* Assume function successful...   */
    EA = 1;                                 /* Enable all interrupts.          */
  }


/*****************************************************************************\
 *                              gsc_rcv_valid                                *
 *****************************************************************************
 *                                                                           *
 * Function:     Processes a valid receive interrupt (EGSRV) from the GSC.   *
 *               This routine is called upon the completion of a valid frame *
 *               reception (i.e., no OVERRUN, ABORT, ALIGNMENT, nor CRC      *
```



```c
 *               error). If HBA is being used, then an ACK will be sent in   *
 *               response to receiving this data frame.                      *
 *                                                                           *
 *               This interrupt routine services the global rcv_buffer.      *
 *               As valid frames are received, they are placed in the next   *
 *               available position in the receive buffer. If the buffer is  *
 *               full, then the last (current) frame is overwritten until    *
 *               the buffer is no longer full and room is available for the  *
 *               frame being received. The DMA channel receive destination   *
 *               address is set to point to the next available location in   *
 *               the receive buffer.                                         *
 *                                                                           *
 *               The valid frame received counter of the global module state *
 *               structure is incremented. The GSC receiver is re-enabled    *
 *               and the DMA receive channel GO bit is set for reception of  *
 *               the next frame.                                             *
 *                                                                           *
 * Input:        gsc_rcv_valid() interrupt;                                  *
 *                                                                           *
 * Output:       Nothing.                                                    *
 *                                                                           *
 * Globals:      rcv_buffer   : module BMF.C                                 *
 *               gcd_state    : module GCD.C                                 *
 *               80C152 regs  : module REG152.H                              *
 *                                                                           *
 * Edit History: 07/10/90 - Written by Richard P. Smurlo.                    *
 *                                                                           *
\*****************************************************************************/

static void gsc_rcv_valid() interrupt 5 using 2
  {
    /* Make sure buffer isn't already full.                                   */
    /* If buffer full, incoming frames will be dropped, no ACK will be sent.  */

    if (rcv_buffer.full)
      {
        GREN = 0;                           /* Turn receiver off - NO ACK.     */
        return;
      }

    /* Determine length of received frame from DMA byte count registers.      */
    /* The DMA receive byte count is initialized to GCD_MAX_FRAME_LENGTH.     */
    /* As bytes are received, the byte count registers are decremented.       */

    rcv_buffer.item[rcv_buffer.rear][GCD_LEN_POS] =
      GCD_MAX_FRAME_LENGTH - ((BCRH0 << 8) | BCRL0);

    /* Adjust rear index (MAX_BUFFER_SIZE-1 is last element in buffer).       */
    /* Buffer can't be empty since we just received a frame.                  */

    buf_inc(rcv_buffer.rear);

    if (rcv_buffer.front == rcv_buffer.rear) rcv_buffer.full = TRUE;
    rcv_buffer.empty = FALSE;

    /* Increment valid reception count variable of module state structure.    */
    /* Rolls over after 65535 frames received (if someone is keeping track).  */

    _gcd_state.r_valid_cnt++;

    /* Set receive destination to newly updated rear of buffer.               */
    /* Reset DMA byte count registers for next reception.                     */

    DARL0 = xptr_lo_offset(buf_rear(rcv_buffer));
    DARH0 = xptr_hi_offset(buf_rear(rcv_buffer));
    BCRL0 = GCD_MAX_FRAME_LENGTH & 0xFF;
    BCRH0 = (GCD_MAX_FRAME_LENGTH >> 8) & 0xFF;

    /* Start next reception...                                                */

    DCON0 = DCON0 | 0x01;                   /* Set DMA GO bit.                 */
    GREN  = 1;                              /* Re-enable the GSC receiver.     */
  }
```

```c


/***************************************************************************\
*                              gsc_xmt_valid                                *
*****************************************************************************
*                                                                           *
* Function:     Processes a valid transmit interrupt (EGSTV) from the GSC.  *
*               This routine is called upon the completion of a valid frame *
*               transmission (i.e., no COLLISION DETECT, UNDERRUN, nor      *
*               ACKNOWLEDGE error). If HBA is being used, then an ACK would *
*               already have been received when we entered this routine.    *
*                                                                           *
*               This interrupt routine services the global xmt_buffer.      *
*               The current frame is removed from the transmit buffer. If   *
*               the transmit buffer is empty, transmission interrupts and   *
*               the transmitter are disabled. Otherwise, the transmitter is *
*               set up to transmit the next frame in the buffer. The DMA    *
*               channel transmit source address is set to point to the next *
*               location occupied in the transmit buffer.                   *
*                                                                           *
*               The valid frame transmit counter of the global module state *
*               structure is incremented. The GSC transmitter is re-enabled *
*               (so we can still respond to HBA) but the DMA transmit       *
*               channel GO bit is cleared since (currently) the transmit    *
*               buffer is conceptually only one deep.                       *
*                                                                           *
* Input:        gsc_xmt_valid() interrupt;                                  *
*                                                                           *
* Output:       Nothing.                                                    *
*                                                                           *
* Globals:      xmt_buffer  : module BMF.C                                  *
*               gcd_state   : module GCD.C                                  *
*               80C152 regs : module REG152.H                               *
*                                                                           *
* Edit History: 07/10/90 - Written by Richard P. Smurlo.                    *
*                                                                           *
\***************************************************************************/

static void gsc_xmt_valid() interrupt 8 using 2
{
  /* Buffer can't be full since we just removed an element.            */

  xmt_buffer.full = FALSE;

  /* Adjust front index (MAX_BUFFER_SIZE-1 is last element in buffer).  */

  buf_inc(xmt_buffer.front);

  /* Check and see if we've depleted the buffer (set empty flag if so). */

  if (xmt_buffer.front == xmt_buffer.rear) xmt_buffer.empty = TRUE;

  /* Increment valid transmission count variable of module state structure. */
  /* Rolls over after 65535 frames xmitted (if someone is keeping track).   */

  _gcd_state.x_valid_cnt++;

  /* If more than one element in transmit buffer...                     */
  /*   Set transmit source to newly updated front of buffer.            */
  /*   Reset DMA byte count registers for next transmission.            */
  /*   Re-enable the transmitter (must be done BEFORE GO bit is set).   */
  /*   Set DMA GO bit to start next transmission.                       */
  /* Otherwise, we're done transmitting for now, so turn things off...  */
  /*   Disable transmit valid and error interrupts.                     */
  /*   Disable the DMA channel.                                         */
  /*   But still enable the transmitter for HBA.                        */

  if (!xmt_buffer.empty)
  {
    SARL1 = xptr_lo_offset(buf_front(xmt_buffer));
    SARH1 = xptr_hi_offset(buf_front(xmt_buffer));
    BCRL1 = xmt_buffer.item[xmt_buffer.front][GCD_LEN_POS] & 0xFF;
    BCRH1 = (xmt_buffer.item[xmt_buffer.front][GCD_LEN_POS] >> 8) & 0xFF;
```



```c
    TEN   = 1;
    DCON1 = DCON1 | 0x01;
  }
  else
  {
    EGSTV = 0;                       /* Disable transmit valid interrupts.   */
    EGSTE = 0;                       /* Disable transmit error interrupts.   */
    DCON1 = DCON1 & 0xFE;            /* Clear the DMA GO bit.                */
    TEN   = 1;                       /* Re-enable the transmitter.           */
  }
}


/***************************************************************************\
*                              gsc_rcv_error                                *
*****************************************************************************
*                                                                           *
* Function:     Processes a receive error interrupt (EGSRE) from the GSC.   *
*               The module state structure error variables are updated.     *
*               The DMA receive destination pointers and associated byte    *
*               count registers are re-initialized, and the GSC is set      *
*               to try and receive the frame again.                         *
*                                                                           *
* Input:        gsc_rcv_error() interrupt;                                  *
*                                                                           *
* Output:       Nothing.                                                    *
*                                                                           *
* Globals:      gcd_state   : module GCD.C                                  *
*               80C152 regs : module REG152.H                               *
*                                                                           *
* Edit History: 07/10/90 - Written by Richard P. Smurlo.                    *
*                                                                           *
\***************************************************************************/
static void gsc_rcv_error() interrupt 6 using 2
{
  int   i;
  char  temp;

  /* Log the type of receive error.                                        */
  /* The sequence of checking is critical to correct error interpretation. */

  _gcd_state.r_err_cnt++;
  if (RSTAT & R_OVR_ERR_MASK)
    _gcd_state.r_ovr_err_cnt++;
  else
    if (RSTAT & R_RCABT_ERR_MASK)
      _gcd_state.r_rcabt_err_cnt++;
    else
      if (RSTAT & R_AE_ERR_MASK)
        _gcd_state.r_ae_err_cnt++;
      else
        if (RSTAT & R_CRCE_ERR_MASK)
          _gcd_state.r_crce_err_cnt++;

  /* Just had a bad reception, so set things up to receive again.          */
  /* Must clear the receiver FIFO (read bytes until RFNE = 0).             */
  /* Must re-initialize the receive destination DMA pointers and byte count.*/
  /* Incoming frames still go to the end of the receive buffer.            */

  for (i = 0; RFNE != 0 && i <= MAX_RFIFO_READS; i++) temp = RFIFO;

  DARLO = xptr_lo_offset(buf_rear(rcv_buffer));
  DARHO = xptr_hi_offset(buf_rear(rcv_buffer));
  BCRLO = GCD_MAX_FRAME_LENGTH & 0xFF;
  BCRHO = (GCD_MAX_FRAME_LENGTH >> 8) & 0xFF;

  DCONO = DCONO | 0x01;                 /* Set DMA Go bit.                 */
  GREN  = 1;                            /* Re-enable the GSC receiver.     */
}


/***************************************************************************\
*                              gsc_xmt_error                                *
```

```
*************************************************************************
*                                                                       *
* Function:      Processes a transmit error interrupt (EGSTE) from the GSC.  *
*                The module state structure error variables are updated. *
*                The DMA transmit source pointers and associated byte count *
*                registers are re-initialized, and the GSC is set to try *
*                and transmit the frame again.                          *
*                                                                       *
* Input:         gsc_xmt_error() interrupt;                             *
*                                                                       *
* Output:        Nothing.                                               *
*                                                                       *
* Globals:       _gcd_state   : module GCD.C                            *
*                gcd_tries    : module GCD.C                            *
*                gcd_times    : module GCD.C                            *
*                gcd_stop     : module GCD.C                            *
*                80C152 regs  : module REG152.H                         *
*                                                                       *
* Edit History: 07/10/90 - Written by Richard P. Smurlo.                *
*                01/30/91 - Robin T. Laird added re-transmit count/stop. *
*                                                                       *
\***********************************************************************/

static void gsc_xmt_error() interrupt 9 using 2
{
  /* Log the type of transmit error.                                    */
  /* The sequence of checking is critical to correct error interpretation. */

   gcd_state.x_err_cnt++;
  if (TSTAT & X_TCDT_ERR_MASK)
    _gcd_state.x_tcdt_err_cnt++;
  else
    if (TSTAT & X_UR_ERR_MASK)
      _gcd_state.x_ur_err_cnt++;
    else
      if (TSTAT & X_NOACK_ERR_MASK)
        _gcd_state.x_noack_err_cnt++;

  /* Check number of attempts made so far. If gcd_times == 0 not counting. */
  /* Variable gcd_times == 0 if attempt re-transmission forever.        */

  if (gcd_times)
  {
    if (gcd_stop)
    {
      return;
    }
    else if (++gcd_tries >= gcd_times)
    {
      gcd_stop = TRUE;                  /* Hit re-transmit limit.          */
      BCRH1 = 0;                        /* Set high byte count to 0.       */
      BCRL1 = 0;                        /* Set low  byte count to 0.       */
      TEN   = 1;                        /* Re-enable transmitter for ACKs. */
      return;
    }
  }

  /* Just had a bad transmission, so set things up to transmit again.   */
  /* Must re-initialize the transmit source DMA pointers and byte count. */
  /* Outgoing frame is still at the front of the transmit buffer.       */

  SARL1 = xptr_lo_offset(buf_front(xmt_buffer));
  SARH1 = xptr_hi_offset(buf_front(xmt_buffer));
  BCRL1 = xmt_buffer.item[xmt_buffer.front][GCD_LEN_POS] & 0xFF;
  BCRH1 = (xmt_buffer.item[xmt_buffer.front][GCD_LEN_POS] >> 8) & 0xFF;

  TEN   = 1;                            /* TEN = 1 to enable the xmitter.  */
  while(!TEN);                          /* Wait for TEN to actually be set. */
  DCON1 = DCON1 | 0x01;                 /* Set DMA GO bit.                 */
}
```

```
#*******************************************************************#
#                             MAKEFILE                              #
#*******************************************************************#
#                                                                   #
# CPCI:          IED90-MRA-MPU-AC-MAKFFILE-TXT-ROCO                 #
#                                                                   #
# Description:   Makefile for the Modular Robotic Architecture (MRA).#
#                Makes the MPU applications controller subsystem.   #
#                                                                   #
#                Targets are available for the following systems/subsystems: #
#                                                                   #
#                mac   - MPU Applications Controller                #
#                print - Print MPU AC source files                  #
#                                                                   #
# Notes:         1) The dependency and production rules are included here. #
#                2) See also \mra\makefile.                         #
#                                                                   #
# Edit History: 03/25/91 - Written by Robin T. Laird.               #
#                                                                   #
#*******************************************************************#



################################### RULES ###################################

.SUFFIXES : .hex .exe .obj .c .a51

# Control settings for Franklin 8031 development

CC          =c51
AS          =a51
LINK        =l51
OTOH        =ohs51
CFLAGS      =cd la db sb
ASFLAGS     =
LFLAGS      =
OFLAGS      =
STARTUP     =\c51\crom.obj
CODESEG     =00000h
XDATASEG    =00000h

# Control settings for Microsoft MS-DOS development

MSC         =cl
MSAS        =masm
MSLINK      =link
MSCFLAGS    =/AL /c /O1 /Z1 /Od
MSASFLAGS   =
MSLNKFLGS   =/co
LOADLIBES   =

.c.obj :
        $(CC) $< $(CFLAGS)

.a51.obj :
        $(AS) $< $(ASFLAGS)

.obj.exe :
        $(LINK) $(STARTUP),$< TO $@ code ($(CODESEG)) xdata ($(XDATASEG)) ixref

.exe.hex :
        $(OTOH) $< $(OFLAGS)



################################ DEFINITIONS ################################

# Project, system, and application level definitions

PROJ            = mra
APPSYS          = app
COMSYS          = com
ICNSYS          = icn
```



```
MPUSYS          = mpu

APPSRC          = \$(PROJ)\$(APPSYS)\src
COMSRC          = \$(PROJ)\$(COMSYS)\src
ICNSRC          = \$(PROJ)\$(ICNSYS)\src
MPUSRC          = \$(PROJ)\$(MPUSYS)\src

COMBINMS        = \$(PROJ)\$(COMSYS)\bin\msdos

MPUBIN31        = \$(PROJ)\$(MPUSYS)\bin\8031
MPUBIN152       = \$(PROJ)\$(MPUSYS)\bin\80152
MPUBINMS        = \$(PROJ)\$(MPUSYS)\bin\msdos
MPUBINSBC8      = \$(PROJ)\$(MPUSYS)\bin\sbc8

# Common subsystem level source directories

DEVSRC          = $(COMSRC)\dev
HDRSRC          = $(COMSRC)\hdr
LCSSRC          = $(COMSRC)\lcs
MMSSRC          = $(COMSRC)\mms

# ICN subsystem level source directories

GCSSRC          = $(ICNSRC)\gcs
IACSRC          = $(ICNSRC)\ac

# MPU subsystem level source directories

LDSSRC          = $(MPUSRC)\lds
MACSRC          = $(MPUSRC)\ac

# Common subsystem global include and compilation units

SYSDEFS         = $(HDRSRC)\sysdefs.h

# MPU subsystem compilation units

MAC             = $(MPUBIN31)\main.obj      $(MPUBIN31)\mra.obj      \
                  $(MPUBINMS)\main.obj      $(MPUBINMS)\mra.obj



################################### TARGETS ###################################

mac   : $(MAC)

print : $(MAC)
        $-a2ps -nf main.c | post
        $-a2ps -nf mra.h | post
        $-a2ps -nf mra.c | post
        touch print



########################### MPU MAC DEPENDENCIES ###########################

# MPU MAC main program dependencies

MPUMAIN = $(SYSDEFS)            $(MACSRC)\mra.h         $(GCSSRC)\gci.h \
          $(GCSSRC)\gcd.h       $(LCSSRC)\lci.h         $(LCSSRC)\lcd.h \
          $(MMSSRC)\mm.h        $(MMSSRC)\lmm.h         $(MMSSRC)\smm.h \
          $(LDSSRC)\ldi.h       $(DEVSRC)\rtc.h         $(MACSRC)\main.c

$(MPUBIN31)\main.obj            : $(MPUMAIN)
  $(CC) $(MACSRC)\$*.c $(CFLAGS) df(I8031) pr($(MACSRC)\$*.31) oj($(MPUBIN31)\$

$(MPUBINMS)\main.obj            : $(MPUMAIN)
  $(MSC) $(MSCFLAGS) /DIBMAT /Fs$(MACSRC)\$*.at /Fo$(MPUBINMS)\$* $(MACSRC)\$*.


# MPU MAC mra system dependencies

MPUMRA = $(SYSDEFS)             $(MACSRC)\mra.h         $(GCSSRC)\gci.h \
```

```
            $(GCSSRC)\gcd.h          $(LCSSRC)\lci.h          $(LCSSRC)\lcd.h \
            $(MMSSRC)\mn.h           $(MMSSRC)\lmm.h          $(MMSSRC)\smm.h \
            $(LDSSRC)\ldi.h          $(MACSRC)\mra.c

$(MPUBIN31)\mra.obj                  : $(MPUMRA)
  $(CC) $(MACSRC)\$*.c $(CFLAGS) df(18031) pr($(MACSRC)\$*.31) oj($(MPUBIN31)\$

$(MPUBINMS)\mra.obj                  : $(MPUMRA)
  $(MSC) $(MSCFLAGS) /DIBMAT /Fs$(MACSRC)\$*.at /Fo$(MPUBINMS)\$* $(MACSRC)\$*.
```

```
/****************************************************************************\
*                                  MAIN.C                                    *
******************************************************************************
*                                                                            *
* CPCI:          IED90-MRA-MPU-AC-MAIN-C-R1C1                                *
*                                                                            *
* Description:   MPU main program.                                           *
*                Implements the MRA MPU (user) application program.          *
*                This is the main program for the MPU system.                *
*                It simply calls the standard MRA function mra_init() which  *
*                is responsible for initializing and coordinating the MRA    *
*                subsystems for the MPU application.                         *
*                                                                            *
* Notes:         1) Subsystems are selected by setting the associated USE_ss *
*                   variable to YES (1). This can be done from either the    *
*                   compilation command line or by modifying the MRA.H file. *
*                                                                            *
* Edit History: 10/10/90 - Written by Robin T. Laird.                        *
*                                                                            *
\****************************************************************************/

#include <sysdefs.h>                   /* MRA standard declarations.      */
#include <rtc.h>                       /* MRA real-time clock functions.  */
#include "mra.h"                       /* MRA public literals/functions.  */

#include <debug.h>

#define TWO_SECONDS     2000L          /* Wait two seconds before start.  */

void main()
{
  /* Wait for a few seconds before fi:      ip.                           */
  /* This gives the ICN time to init :     ubsystems, buffers, etc.       */

  rtc_init();
  rtc_wait(TWO_SECONDS);

  /* Initialize the MRA subsystems and call mra_main().                   */
  /* Control never returns for systems with USE_MRA set to YES.           */

  mra_init();
}
```

```c
/***********************************************************************\
*                              MRA.H                                     *
*************************************************************************
*                                                                        *
* CPCI:          IED90-MRA-MPU-AC-MRA-H-ROCO                             *
*                                                                        *
* Description:   System configuration and default controller definitions. *
*                Contains external declarations for the system initialization *
*                function and default application controller, mra_main(). *
*                                                                        *
*                The user/developer should select/de-select those MRA    *
*                subsystems that are required or being used. Only those  *
*                subsystems that are selected are initialized by mra_init(). *
*                                                                        *
*                Selecting USE_MRA will cause the default system application *
*                controller, mra_main(), to be used. The init routine calls *
*                the default controller after all selected subsystems have *
*                been initialized. If selected, mra_main() takes control and *
*                does not return.                                        *
*                                                                        *
*                Module MRA exports the following variables/functions:   *
*                                                                        *
*                int mra_error;                                          *
*                                                                        *
*                mra_init();                                             *
*                mra_main();                                             *
*                                                                        *
* Notes:         1) This file should be included only by the main() program. *
*                                                                        *
* Edit History: 07/07/90 - Written by Robin T. Laird.                    *
*                                                                        *
\***********************************************************************/

#ifndef MRA_MODULE_CODE
#define MRA_MODULE_CODE          13000

/* Public Data Structures:                                               */

#define ERR_MRA_NOT_INIT         1+MRA_MODULE_CODE

#define USE_GCS                  NO      /* Global Communications Subsystem. */
#define USE_LCS                  YES     /* Local  Communications Subsystem. */
#define USE_MMS                  YES     /* Method Management Subsystem.     */
#define USE_LDS                  NO      /* Logical Device Subsystem.        */
#define USE_MRA                  YES     /* Modular Robotic Architecture AC. */

#if USE_GCS
#include <gci.h>                         /* Global Comms Interface.          */
#include <gcd.h>                         /* Global Comms Device Driver.      */
#endif

#if USE_LCS
#include <lci.h>                         /* Local Comms Interface.           */
#include <lcd.h>                         /* Local Comms Device Driver.       */
#endif

#if USE_MMS
#include <mm.h>                          /* Method Manager.                  */
#include <pb.h>                          /* Phone Book.                      */
#include <lmm.h>                         /* Local Method Manager.            */
#include <smm.h>                         /* System Method Manager.           */
#endif

#if USE_LDS
#include <ldi.h>                         /* Logical Device Interface.        */
#endif

/* External module global error variable.                               */

extern int mra_error;

/* Public Functions:                                                     */

```



```c
void mra_init(void);
void mra_main(void);

#endif
```

```c
/***********************************************************************\
*                                MRA.C                                  *
*************************************************************************
*                                                                       *
* CPCI:          IED90-MRA-MPU-AC-MRA-C-ROCO                            *
*                                                                       *
* Description:   MRA system initialization and default controller functions. *
*                Implements the MRA system initialization and default system *
*                application controller (AC) functions which represent the *
*                highest level interface to the Modular Robotic Architecture *
*                software systems. The mra_init() function must be called to *
*                correctly initialize the various software subsystems. The *
*                function mra_main() is the default AC and replaces the user *
*                application program (mra_main() never returns to the calling *
*                function).                                             *
*                                                                       *
*                Module MRA exports the following variables/functions:  *
*                                                                       *
*                int mra_error;                                         *
*                                                                       *
*                mra_init();                                            *
*                mra_main();                                            *
*                                                                       *
* Notes:         1) The MRA functions are implementation independent.   *
*                2) Module MRA represents the Default Applications Controller.*
*                                                                       *
* Edit History: 02/04/91 - Written by Robin T. Laird.                   *
*                                                                       *
\***********************************************************************/

#include <sysdefs.h>                    /* System constants and types.    */
#include "mra.h"                        /* MRA public literals/functions. */

/* Public Variables:                                                      */

/* Global module error variable, mra_error.                               */
/* mra_error contains code of last error occurence.                       */
/* Should be set to AOK after each successful function call.              */
/* Variable can be examined by other software after each function call.   */

XDATA int mra_error = ERR_MRA_NOT_INIT;


/***********************************************************************\
*                              mra_init                                 *
*************************************************************************
*                                                                       *
* Function:      Initializes the MRA software subsystems.               *
*                The subsystems are selected in the file MRA.H and only those *
*                subsystems selected will be initialized. If the default *
*                application controller mra_main() is selected then a call is *
*                made to that function and control never returns. The default *
*                AC can be called separately by a call to mra_main() after *
*                mra_init() returns (the same result is acheived).      *
*                                                                       *
* Input:         mra_init();                                            *
*                                                                       *
* Output:        Nothing.                                               *
*                                                                       *
* Globals:       mra_error : module MRA.C                               *
*                gci_error : module GCI.C                               *
*                lci_error : module LCI.C                               *
*                mm_error  : module MM.C                                *
*                ldi_error : module LDI.C                               *
*                                                                       *
* Edit History: 10/01/90 - Written by Robin T. Laird.                   *
*                                                                       *
\***********************************************************************/

void mra_init()
{
  /* Initialize selected subsystems, return upon detected failure.       */

```



```c
  #if USE_GCS
  gci_init();
  if (gci_error != AOK) return;
  #endif

  #if USE_LCS
  lci_init();
  if (lci_error != AOK) return;
  #endif

  #if USE_MMS
  mm_init();
  if (mm_error != AOK) return;
  #endif

  #if USE_LDS
  ldi_init();
  if (ldi_error != AOK) return;
  #endif

  /* Function successful...                                              */
  /* Set error variable to OK and return only if USE_MRA not selected.   */

  mra_error = AOK;

  /* Call MRA main program and never return...                           */

  #if USE_MRA
  mra_main();
  #endif
}


/***********************************************************************\
*                              mra_main                                 *
*************************************************************************
*                                                                       *
* Function:      Default Application Controller (AC) for the MPU system. *
*                The main() C program calls mra_init() and which then calls *
*                mra_main(). The default controller coordinates operation of *
*                the MRA subsystems to pass information from the LAN to the *
*                module processor (MPU) and vice versa.                 *
*                                                                       *
* Input:         mra_main();                                            *
*                                                                       *
* Output:        Nothing.                                               *
*                                                                       *
* Globals:       None.                                                  *
*                                                                       *
* Edit History: 02/04/91 - Written by Robin T. Laird.                   *
*                                                                       *
\***********************************************************************/

void mra_main()
{
  /* Cycle the method manager forever.                                   */
  /* Messages will be received via the LCI and processed according to type. */
  /* If dictionary functions activate system methods then they must call */
  /* mm_cycle() "occasionally" so that incoming messages are not dropped. */

  #if USE_MRA
  mm_cycle(MM_CYCLE_FOREVER);
  #endif
}
```

```
#*************************************************************************#
#                              MAKEFILE                                   #
#*************************************************************************#
#                                                                         #
# CPCI:          IED90-MRA-MPU-LDS-MAKEFILE-TXT-R0C0                      #
#                                                                         #
# Description:   Makefile for the Modular Robotic Architecture (MRA).     #
#                Makes the logical device interface subsystem.            #
#                                                                         #
#                Targets are available for the following systems/subsystems: #
#                                                                         #
#                lds   - Logical Device Interface Subsystem               #
#                lib   - Add modules to MRA library                       #
#                print - Add modules to MRA library                       #
#                                                                         #
# Notes:         1) The dependency and production rules are included here. #
#                2) See also \mra\makefile.                               #
#                                                                         #
# Edit History: 05/30/91 - Written by Robin T. Laird.                     #
#                                                                         #
#*************************************************************************#



##################################### RULES #####################################

.SUFFIXES : .hex .exe .obj .c .a51

# Control settings for Franklin 8031 development

CC          =c51
AS          =a51
LINK        =l51
OTOH        =ohs51
CFLAGS      =cd la db sb
ASFLAGS     =
LFLAGS      =
OFLAGS      =
STARTUP     =\c51\crom.obj
CODESEG     =00000h
XDATASEG    =00000h

# Control settings for Microsoft MS-DOS development

MSC         =cl
MSAS        =masm
MSLINK      =link
MSCFLAGS    =/AS /c /Oi /Zi /Od
MSASFLAGS   =
MSLNKFLGS   =/co
LOADLIBES   =

.c.obj :
        $(CC) $< $(CFLAGS)

.a51.obj :
        $(AS) $< $(ASFLAGS)

.obj.exe :
        $(LINK) $(STARTUP),$< TO $@ code ($(CODESEG)) xdata ($(XDATASEG)) ixref

.exe.hex :
        $(OTOH) $< $(OFLAGS)


################################## DEFINITIONS ##################################

# Project, system, and application level definitions

PROJ            = mra
APPSYS          = app
COMSYS          = com
```



```
MPUSYS          = mpu

MPULIB          = \$(PROJ)\lib
COMSRC          = \$(PROJ)\$(COMSYS;\src
MPUSRC          = \$(PROJ)\$(MPUSYS)\src

MPUBIN31        = \$(PROJ)\$(MPUSYS)\bin\8031
MPUBIN152       = \$(PROJ)\$(MPUSYS)\bin\80152
MPUBINMS        = \$(PROJ)\$(MPUSYS)\bin\msdos
MPUBINSBC8      = \$(PROJ)\$(MPUSYS)\bin\sbc8

# Common subsystem level source directories

HDRSRC          = $(COMSRC)\hdr

# Logical device interface subsystem level source directories

LDSSRC          = $(MPUSRC)\lds

# Common subsystem global include and compilation units

SYSDEFS         = $(HDRSRC)\sysdefs.h

# MPU subsystem compilation units

LDS             = $(MPUBIN152)\ldi.obj        $(MPUBIN31)\ldi.obj     \
                  $(MPUBINMS)\ldi.obj         $(MPUBINSBC8)\ldi.obj



################################### TARGETS ###################################

lds             : $(LDS)

lib             : $(LDS)
                  -lib51 delete $(MPULIB)\mra_1521.lib (ldi)
                  -lib51 add $(MPUBIN152)\ldi.obj to $(MPULIB)\mra_1521.lib
                  -lib51 delete $(MPULIB)\mra_311.lib (ldi)
                  -lib51 add $(MPUBIN31)\ldi.obj to $(MPULIB)\mra_311.lib
                  -lib $(MPULIB)\mra_mss.lib -ldi.obj+$(MPUBINMS)\ldi.obj;
                  -lib $(MPULIB)\mra_sbc8.lib -ldi.obj+$(MPUBINSBC8)\ldi.obj;
                  touch lib

print           : $(LDS)
                  %-a2ps -nf ldi.h | post
                  %-a2ps -nf ldi.c | post
                  touch print



########################### MPU DEV DEPENDENCIES ###########################

# MPU Logical Device Interface module dependencies for 80152, 8031, MS-DOS, SBC

LDI = $(SYSDEFS) $(LDSSRC)\ldi.h $(LDSSRC)\ldi.c

$(MPUBIN152)\ldi.obj                : $(LDI)
  $(CC) $(LDSSRC)\$*.c $(CFLAGS) df(I80152) pr($(LDSSRC)\$*.152) oj($(MPUBIN152

$(MPUBIN31)\ldi.obj                 : $(LDI)
  $(CC) $(LDSSRC)\$*.c $(CFLAGS) df(I8031)  pr($(LDSSRC)\$*.31)  oj($(MPUBIN31)

$(MPUBINMS)\ldi.obj                 : $(LDI)
  $(MSC) $(MSCFLAGS) /DIBMAT /Fs$(LDSSRC)\$*.at /Fo$(MPUBINMS)\$* $(LDSSRC)\$*.

$(MPUBINSBC8)\ldi.obj               : $(LDI)
  $(MSC) $(MSCFLAGS) /DSBC8 /Fs$(LDSSRC)\$*.sbc /Fo$(MPUBINSBC8)\$* $(LDSSRC)\$
```

```
/**************************************************************************\
*                                LDI.H                                     *
****************************************************************************
*                                                                          *
* CPCI:          IED90-MRA-MPU-LDS-LDI-H-ROCO                              *
*                                                                          *
* Description:   Logical Device Interface (LDI) variables and functions.   *
*                Contains constant function parameter declarations (#defines) *
*                as well as function return values (for success and failure *
*                of all operations). Contains the function prototypes for the *
*                LDI.C module.                                             *
*                                                                          *
*                Module LDI exports the following types/variables/functions: *
*                                                                          *
*                int ldi_error;                                            *
*                                                                          *
*                ldi_init();                                               *
*                                                                          *
* Notes:         1) See SDS pp. 5-6 through 5-x for more information.      *
*                                                                          *
* Edit History: 03/25/91 - Written by Robin T. Laird.                      *
*                                                                          *
\**************************************************************************/

#ifndef LDI_MODULE_CODE
#define LDI_MODULE_CODE                 14000

/* Public Data Stiuctures:                                                 */

#define LDI_ERR_NOT_INIT                1+LDI_MODULE_CODE

/* External module global error variable.                                  */

extern int ldi_error;

/* Public Functions:                                                       */

void ldi_init();

#endif
```

```
/**********************************************************************\
*                              LDI.C                                   *
************************************************************************
*                                                                      *
* CPCI:         IED90-MRA-MPU-LDS-LDI-C-ROCO                           *
*                                                                      *
* Description:  Logical device interface (LDI) functions.              *
*               Implements the standard MRA logical device interface module. *
*               The LDI provides functions for creating, deleting, and *
*               manipulating abstract data types that represent logical *
*               devices such as logical actuators or sensors.          *
*                                                                      *
*               Module LDI exports the following types/variables/functions: *
*                                                                      *
*               int ldi_error;                                         *
*                                                                      *
*               ldi_init();                                            *
*                                                                      *
* Notes:        1) The LDI is implemented as a blackboard data structure. *
*                                                                      *
* Edit History: 03/25/91 - Written by Robin T. Laird.                  *
*                                                                      *
\**********************************************************************/

#include <sysdefs.h>                    /* System constants and types.     */
#include "ldi.h"                        /* Logical device interface.       */

/* Public Variables:                                                      */

/* Global module error variable, ldi_error.                               */
/* ldi_error contains code of last error occurence.                       */
/* Should be set to AOK after each successful function call.              */
/* Variable can be examined by other software after each function call.   */

XDATA int ldi_error = LDI_ERR_NOT_INIT; /* Global module error variable.   */


/**********************************************************************\
*                              ldi_init                                *
************************************************************************
*                                                                      *
* Function:     Initializes the Logical Device Interface software subsystems.*
*               This includes initializing data structures such as the *
*               system blackboard and the memory allocation routines.  *
*                                                                      *
* Input:        ldi_init();                                            *
*                                                                      *
* Output:       Nothing.                                               *
*                                                                      *
* Globals:      ldi_error : LDI.C                                      *
*                                                                      *
* Edit History: 03/25/91 - Written by Robin T. Laird.                  *
*                                                                      *
\**********************************************************************/

void ldi_init()
{
  ldi_error = AOK;                      /* Assume function successful.     */
}
```

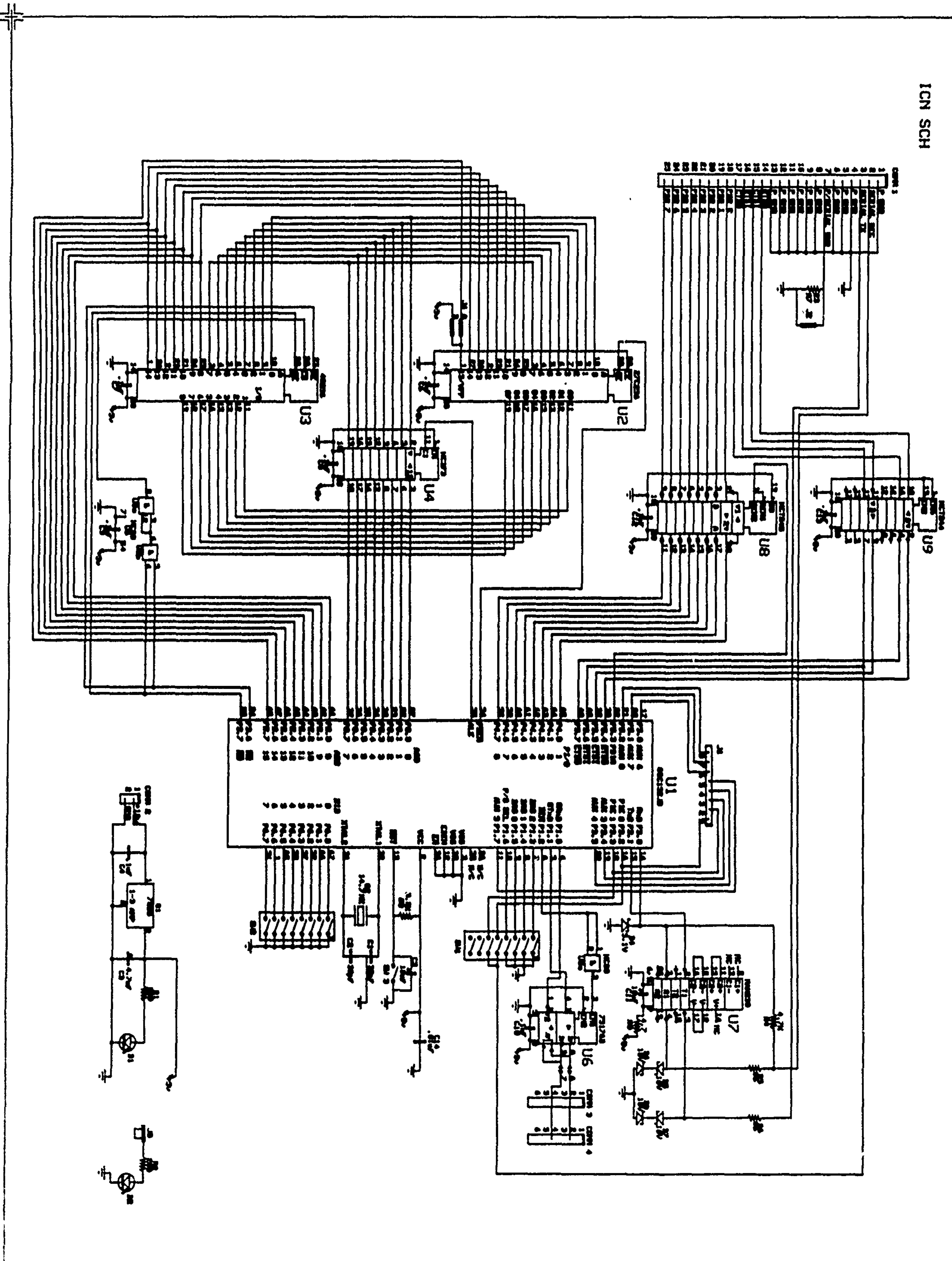
APPENDIX B

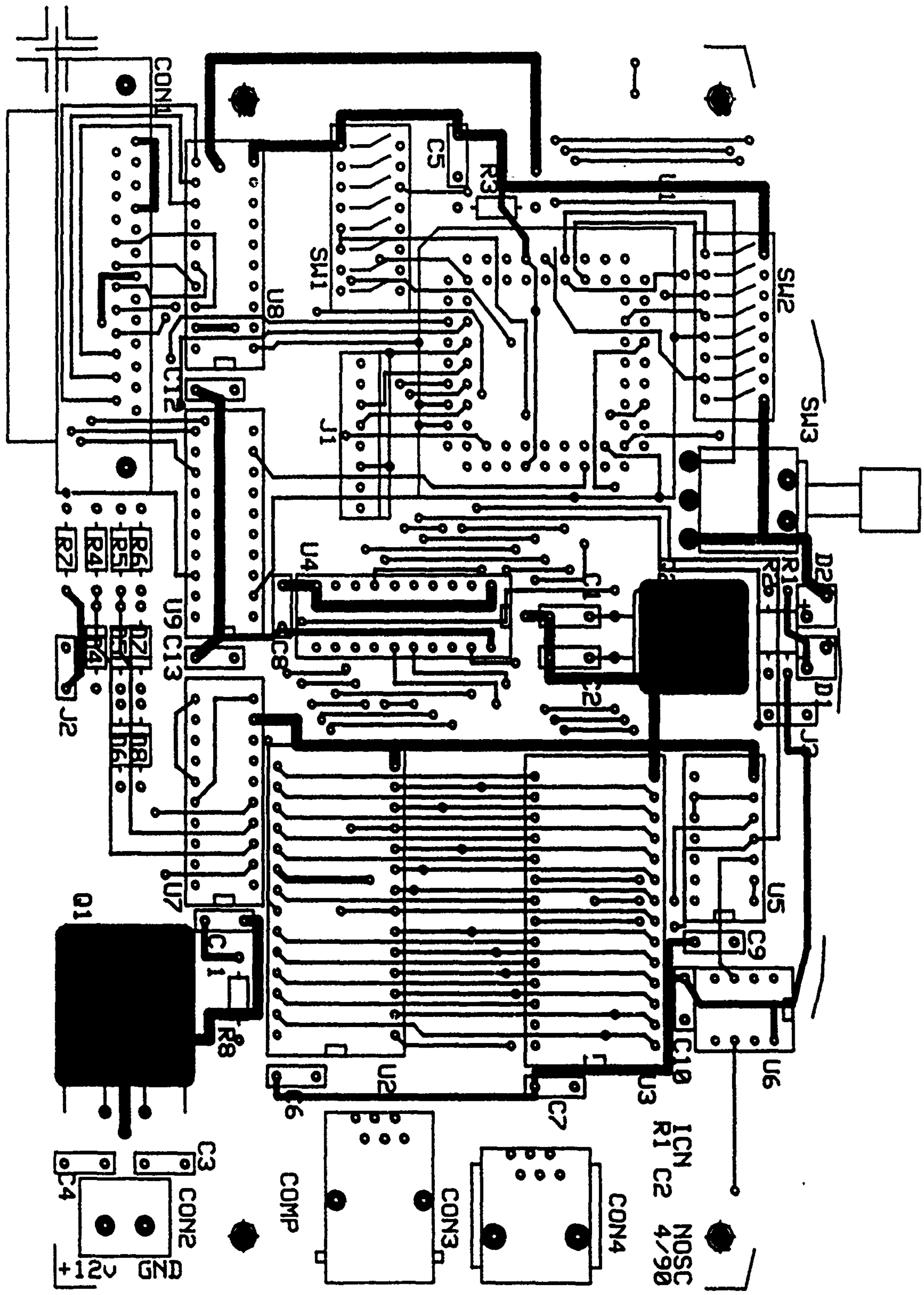
HARDWARE SYSTEM IMPLEMENTATION

Appendix B. Hardware System Implementation

The following section provides details on the standard hardware subsystems of the MRA. The information provided herein is sufficient to implement the standard hardware components that each robot module possesses.

For each of the standard components (i.e., ICN, PDN, and the PPCU), a schematic, layout diagram, and a parts list are included.


ICN SCH
U1
U2
U3
U4
U6
U7
U8
U9


CON1
C5
R3
U1
SW1
SW2
SW3
U8
C12
J1
U4
U9
C13
C8
C1
C2
R1
R2
D2
D1
J3
R7
R4
R5
R6
J2
U7
Q1
R8
U2
U3
U5
U6
C7
C9
C10
C3
C4
CON2
CON3
CON4
COMP
+12v
GND
ICN
R1 C2
NOSC
4/90

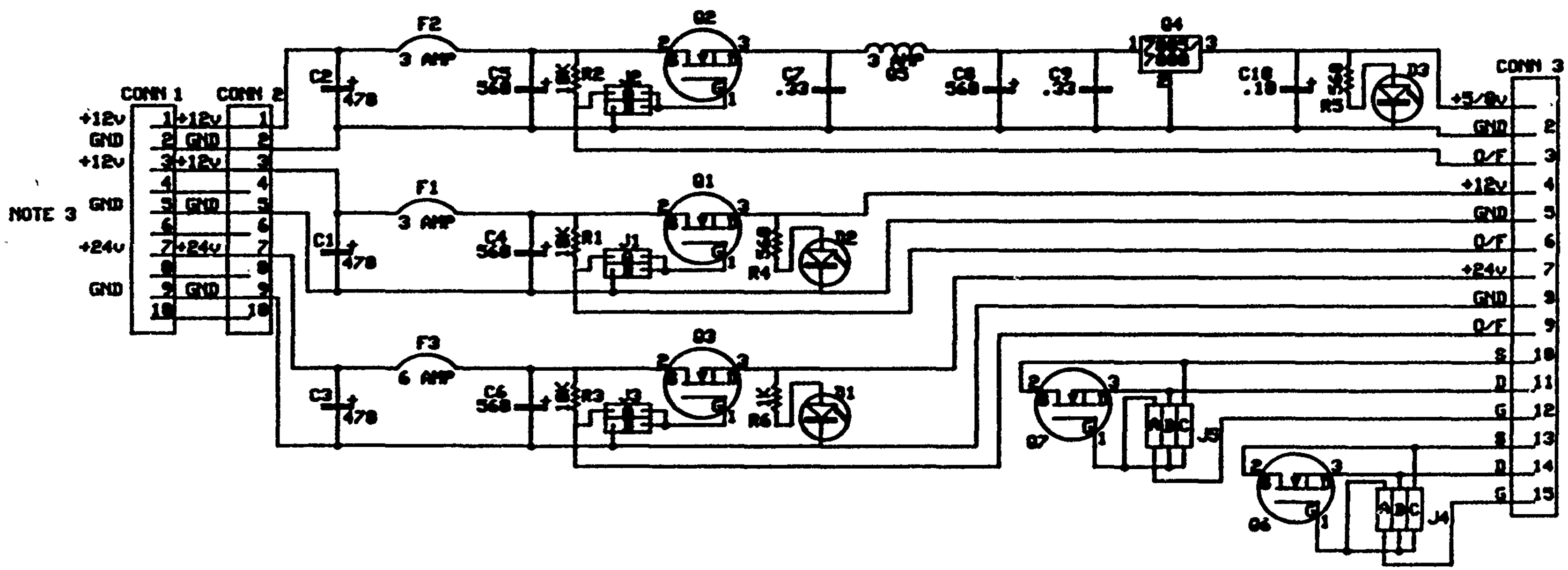
Intelligent Communications Node (ICN) Parts List

Board Ref.	Quan.	P/N - Discription
U1	1	N80C152JB-1, MPU
U2	1	27C256, EPROM
U3	1	43256AC-10L, RAM
U4	1	74HC373, LATCH
U5	1	74HC00, NAND
U6	1	SN75176B, BUS XCEIVER
U7	1	MAX233CPP, RS-232 DRIVER
U8	1	74HCT245, LINE DRIVER
U9	1	74HCT244, LINE DRIVER
C1,C2	2	33pF, 7V, (ceramic)
C3	1	4.7uF, 25V, (elec.)
C4	1	0.1uF, 16V, (tant.)
C5,C11	2	10uF, 16V, (elec.)
C6,C7,C8,C9, C10,C12,C13	7	0.1uF, 7V, (ceramic)
C14	1	.01uF, 16V, (tant.)
R1,R2	2	200 Ohm
R3	1	8.2K Ohm
R4	1	4.7K Ohm
R5,R6	2	86 Ohm
R7	1	23 Ohm
R8	1	4.7 Ohm
D1,D2	2	LED, 2.3v Green, PC Mount, 550 Series
D4	1	Zener Diode, 5.1V, 0.25W
D5,D6,D7,D8	4	Zener Diode, 10V, 0.25W

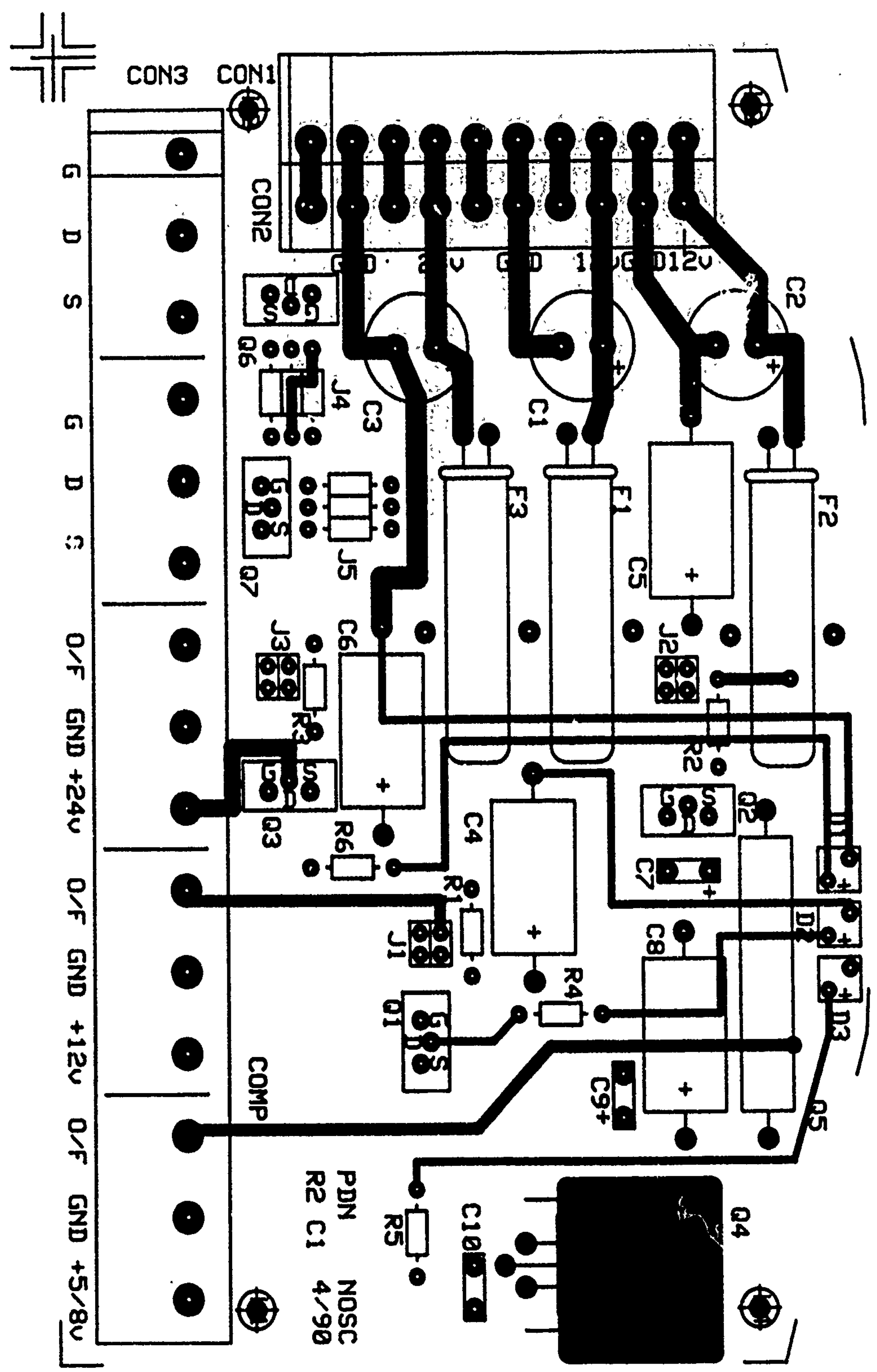
Board Ref.	Quan.	P/N - Discription
Q1	1	Voltage Regulator, 5V, 1A
Q2	1	Crystal Osc., 14.7456 MHz
SW1,SW2	2	Dip Switch, 8-pin, PC Mount, SPST
SW3	1	Momentary Push Button, PC Mount, NO
J1	1	Header, 8-pin, PC Mount, Vertical
CONN 1	1	DB-25 Male, PC Mount, Right Angle
CONN 2	1	2 Pin Screw Terminal, PC Mount
CONN 3	1	Phone Jack, 6-pin, PC Mount, Right Angle
CONN 4	1	Phone Jack, 6-pin, PC Mount, Vertical

file: icnparts.doc
last revised: 6/25/91

PDN SCH



NOTES: 1. ALL CAPACITOR VALUES ARE IN uF
2. ALL RESISTORS ARE 1/4 WATT, 5 %
3. FROM PPCU CONN 2 or PREVIOUS PDN CONN 2
4. J1,J2,J3 B-JUMPER TURNS POWER ON AT CONN 3
J1,J2,J3 A-JUMPER TURNS POWER OFF AT CONN 3 AND VOLTAGE BECOMES SWITCHABLE VIA. THE O/F PIN ON CONN 3
5. F1,F2,F3 ARE AUTO-RESET THERMAL BREAKERS, VALUES SHOWN ARE MAX.

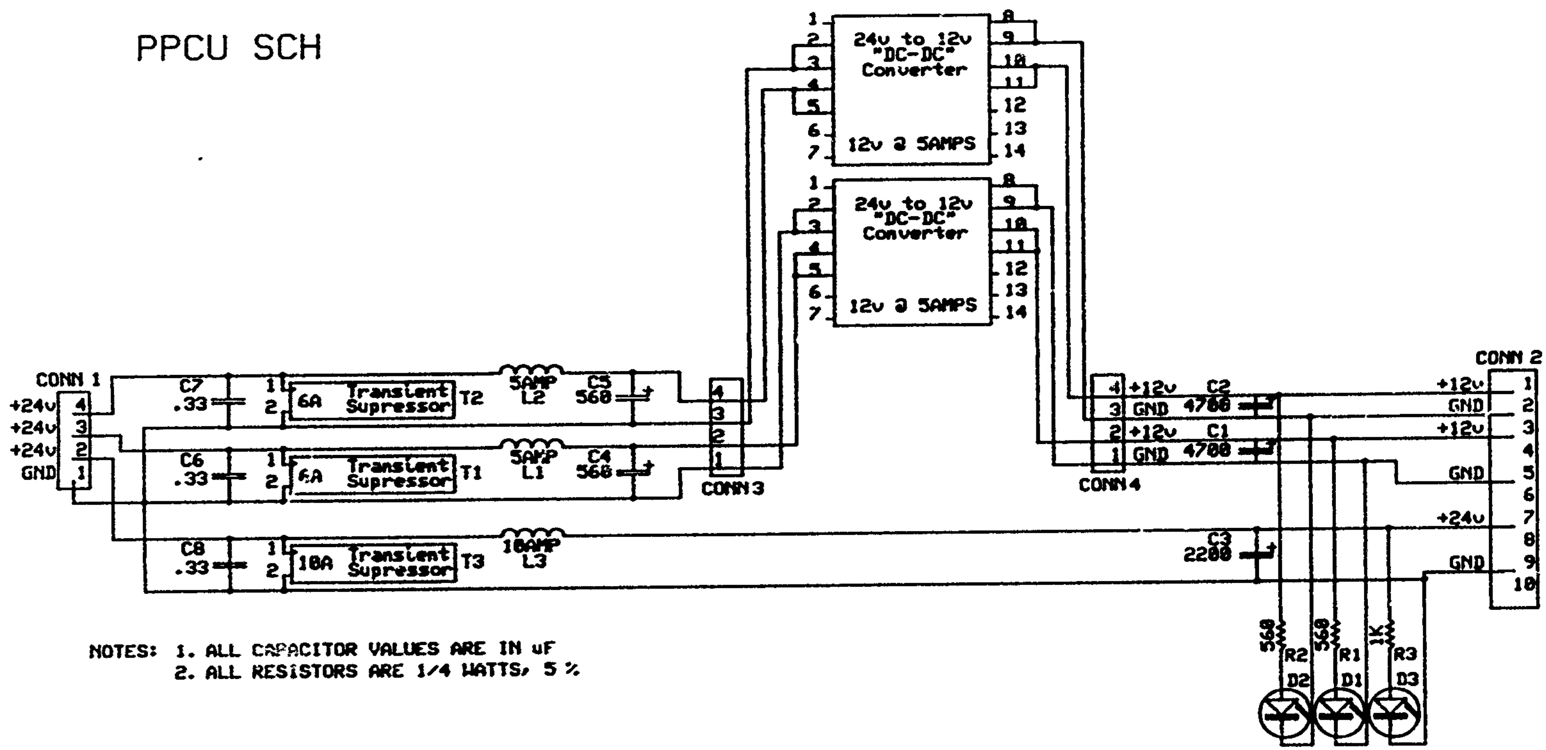

CON3
CON1
CON2
G D S
G D S
O/F GND +24v
O/F GND +12v
O/F GND +5/8v
COMP
PDN
R2 C1
NOSC
4/90

Power Distribution Node (PDN) Parts List

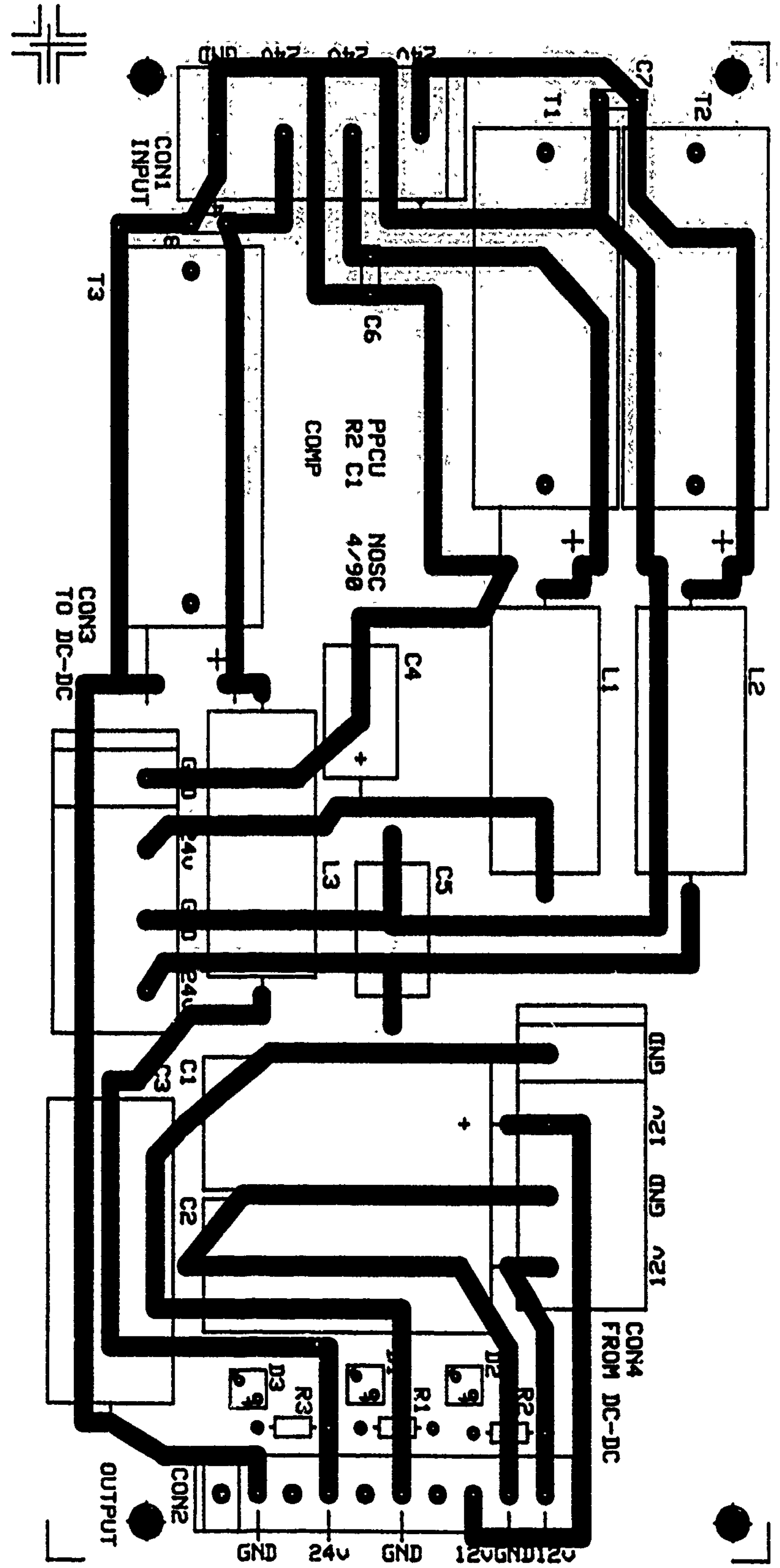
Board Ref	Quan.	P/N - Discription
C3	1	470uF (elec.),35V
C1,C2	2	470uF (elec.),25V
C4,C5,C6,C8	4	560uF (elec./tant.),25V
C7,C9	2	0.33uF (tant.)
C10	1	0.1uF (tant.)
R1,R2,R3	3	10K Ohm
R4,R5	2	560 Ohm
R6	1	1K Ohm
D1,D2,D3	3	LED, 2.3V Green, PC Mount, 550 Series
J1,J2,J3	3	2x2 PC Mount Jumpers
F1,F2	2	3A Resetable Circuit Breaker
F3	1	6A Resetable Circuit Breaker
Q1,Q2,Q3,Q6, Q7	5	IRF 9531, P-Channel MOSFET's, TO-220 style
Q4	1	7805/7808 Voltage Regulator
CONN 2	1	10 pin Vertical Terminal Strip, 8213 Series
CONN 1	1	10 pin Horizontal Terminal Strips, 8213 Series
CONN 3	1	15 Conductor Screw Terminal

file: pdnparts.doc
last revised: 6/10/90

PPCU SCH



NOTES: 1. ALL CAPACITOR VALUES ARE IN uF
2. ALL RESISTORS ARE 1/4 WATTS, 5 %


CON1
INPUT
GND
24v
24v
24v
T1
T2
C7
T3
C6
PPCU
R2 C1
COMP
NOSC
4/90
CON3
TO DC-DC
C4
L1
L2
L3
C5
GND
24v
GND
24v
C1
C3
C2
GND
12v
GND
12v
CON4
FROM DC-DC
D1
D2
D3
R1
R2
R3
CON2
OUTPUT
GND
24v
GND
12v
GND
12v

Platform Power Conditioning Unit (PPCU) Parts List

Board Ref.	Quan.	P/N - Discription
C1,C2	2	4700uF (elec.)
C3	1	2200uF (elec.)
C4,C5	2	560 uF (tant.)
C6,C7,C8	3	0.33 uF (tant.)
R1,R2	2	560 Ohm
R3	1	1K Ohm
L1,L2	2	5A Current Choke, 5200 Series
L3	1	10A Current Choke, 5200 Series
T1,T2,T3	3	15A, 28V Transient Suppressor
D1,D2,D3	3	LED, 2.3V Green, PC Mount, 550 Series
CONN 2	1	10 Conductor Pluggable Terminal Strip, PC Mount, 8213 Series
CONN 1,3,4	3	4 Conductor Screw Terminal Strips, PC Mount
	2	24-12V Dc-DC Converter, WR24S12/60K3

file: ppcuparts.doc
last revised: 6/12/91

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	September 1991	Final

4. TITLE AND SUBTITLE
DEVELOPMENT OF A MODULAR ROBOTIC ARCHITECTURE

5. FUNDING NUMBERS
PE: 0602936N
WU: DN300029

6. AUTHOR(S)
R. T. Laird, R. P. Smurlo, and S. R. Timmer

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Ocean Systems Center
San Diego, CA 92152-5000

8. PERFORMING ORGANIZATION REPORT NUMBER
NOSC TD 2171

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Office of Chief of Naval Research
Arlington, VA 22217

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This objective of this project is to develop the hardware and software components for constructing and controlling reconfigurable, modular robots.

14. SUBJECT TERMS
modular robots
mobile security robots
independent modules
control station
remote platform

15. NUMBER OF PAGES
200

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	SAME AS REPORT

UNCLASSIFIED

21a. NAME OF RESPONSIBLE INDIVIDUAL	21b. TELEPHONE (include Area Code)	21c. OFFICE SYMBOL
R. T. Laird	(619) 553-3667	Code 535

NSN 7540-01-280-5500

Standard form 298

UNCLASSIFIED

INITIAL DISTRIBUTION

Code 0012	Patent Counsel	(1)
Code 0144	R. November	(1)
Code 535	R. T. Laird	(15)
Code 952B	J. Puleo	(1)
Code 961	Archive/Stock	(6)
Code 964B	Library	(3)

Defense Technical Information Center
Alexandria, VA 22304-6145 (4)

NCCOSC Washington Liaison Office
Washington, DC 20363-5100

Center for Naval Analyses
Alexandria, VA 22302-0268

Navy Acquisition, Research & Development
Information Center (NARDIC)
Alexandria, VA 22333

Navy Acquisition, Research & Development
Information Center (NARDIC)
Pasadena, CA 91106-3955