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.
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
APPENDICES

A: SOFTWARE SYSTEM IMPLEMENTATION .................................. A-1
B: HARDWARE SYSTEM IMPLEMENTATION ................................. B-1

FIGURES

1. Robot module configurations. ............................................. 3
2. The relationship between components of a modular-robotic system. .... 4
3. MOSER remote-platform (RP) module configuration. .................... 8
4. Overall GRPA architecture for all levels (Aviles, Laird, & Myers, 1988.) . 10
5. Distributed computer architecture for ROBART II (Everett et al., 1990.) . 11
6. RCS architecture components (Barbera, Albus, & Fitzgerald, 1982). .... 13
7. NASREM (RCS) hierarchical control system architecture (Albus, McGain, & Lumina, 1984.) ....................................................... 15
8. Subsumption architecture (Brooks, 1986). ................................ 16
9. The Nested-Hierarchical Controller (NHC) architecture (Meystel, 1988). 18
10. Control-station (CS) configuration showing external device connections. . . 24
11. Generalized robot module "bus" (MODBUS). ................................ 26
12. ICN block diagram. ......................................................... 27
13. PDN block diagram. .......................................................... 28
14. PPCU block diagram. .......................................................... 30
15. Generic robot module. The communications link to the external environment (environment interface) and the sensors and actuators (real-world interface) are optional. .......................................................... 31
16. Telemetry link connecting the control-station (CS) processing unit and the remote-platform (RP) module. ............................................. 32
17. A single control-station (CS) controlling multiple remote platforms (RP) (note the addressing). ....................................................... 33
18. MRA system image (N,N-k architecture). ................................ 34
19. MRA-software block diagram. (Global-Communications Subsystem and Logical-Device Interface not shown.) ........................................... 36
20. MODBOT communications protocol (multiple layers). .................... 41
21. Example of an MRA definition for an I/R proximity module .................. 41
22. Communication paths between modular robot components .................. 43
23. MODBOT teams (one or more MODBOTs) and divisions
   (one or more teams) ................................................... 45
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 MIRA 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 submersible 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.

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.

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).
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.
REASON ABOUT BEHAVIOR OF OBJECTS

PLAN CHANGES TO THE WORLD

IDENTIFY OBJECTS

MONITOR CHANGES

BUILD MAPS

EXPLORE

WANDER

AVOID OBJECTS

(a) vertical decomposition of control system into horizontal slices

(b) layered control levels subsumed by higher levels

Figure 8. Subsumption architecture (Brooks, 1986).
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 capabilities is limited only by available technology and, as technology advances, so too will the abilities of a modular robot incorporating new technology as an add-on 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 MIRA 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 MouseStick 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 in-r 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.
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

![PDN block diagram](image)

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
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.](image-url)
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).

![Diagram of MRA software architecture]

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.
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.

<table>
<thead>
<tr>
<th>DEST MODBOT ID</th>
<th>DEST UNIT ID</th>
<th>SOURCE MODBOT ID</th>
<th>SOURCE UNIT ID</th>
<th>SEQUENCE NUMBER</th>
<th>TRANSACTION DISPOSITION</th>
<th>TRANSACTION CATEGORY</th>
<th>FUNCTION ID</th>
<th>PARAMETER LENGTH</th>
<th>PARAMETER</th>
</tr>
</thead>
<tbody>
<tr>
<td>DESTINATION ADDRESS</td>
<td>SOURCE ADDRESS</td>
<td>MESSAGE COORDINATION</td>
<td>FUNCTION SELECTION</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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.

```c
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

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMR</td>
<td>Autonomous Mobile Robot</td>
</tr>
<tr>
<td>AMRF</td>
<td>Automated Manufacturing Research Facility</td>
</tr>
<tr>
<td>ARDEC</td>
<td>Armament Research, Development, and Engineering Center</td>
</tr>
<tr>
<td>BIT</td>
<td>Built-in Test</td>
</tr>
<tr>
<td>CMOS</td>
<td>Complementary Metal-Oxide Semiconductor</td>
</tr>
<tr>
<td>CS</td>
<td>Control Station</td>
</tr>
<tr>
<td>CRC</td>
<td>Cyclic-Redundancy Check</td>
</tr>
<tr>
<td>CSMA/CD</td>
<td>Carrier-Sense Multiaccess with Collision Detection</td>
</tr>
<tr>
<td>GATERS</td>
<td>Ground-Air Telerobotic Systems</td>
</tr>
<tr>
<td>GRPA</td>
<td>Generic Robotic Processing Architecture</td>
</tr>
<tr>
<td>GSC</td>
<td>Global Serial Channel (of the Intel 80C152)</td>
</tr>
<tr>
<td>HDLC</td>
<td>High-level Data-Link Control</td>
</tr>
<tr>
<td>HDS</td>
<td>Hardware-Design Specification</td>
</tr>
<tr>
<td>ICN</td>
<td>Intelligent-Communication Node</td>
</tr>
<tr>
<td>IED</td>
<td>Independent Exploratory Development</td>
</tr>
<tr>
<td>I/R</td>
<td>Infrared</td>
</tr>
<tr>
<td>IRS</td>
<td>Interface-Requirements Specification</td>
</tr>
<tr>
<td>ISO</td>
<td>International Standards Organization</td>
</tr>
<tr>
<td>LAN</td>
<td>Local Area Network</td>
</tr>
<tr>
<td>LAWN</td>
<td>Local-Area Wireless Network</td>
</tr>
<tr>
<td>MB</td>
<td>Megabytes</td>
</tr>
<tr>
<td>Mbps</td>
<td>Megabits-per-second</td>
</tr>
<tr>
<td>MODBOT</td>
<td>Modular Robot</td>
</tr>
<tr>
<td>MODBUS</td>
<td>Robot Module Bus</td>
</tr>
<tr>
<td>MOSER</td>
<td>Mobile Security Robot (first application of the MRA)</td>
</tr>
<tr>
<td>MPU</td>
<td>Module Processing Unit</td>
</tr>
<tr>
<td>MRA</td>
<td>Modular Robotic Architecture</td>
</tr>
<tr>
<td>NASREM</td>
<td>NASA/NBS Standard Reference Model</td>
</tr>
<tr>
<td>NBS</td>
<td>National Bureau of Standards (a.k.a. NIST)</td>
</tr>
<tr>
<td>NHC</td>
<td>Nested-Hierarchical Controller</td>
</tr>
<tr>
<td>NIST</td>
<td>National Institute of Standards and Technology</td>
</tr>
<tr>
<td>NOSC</td>
<td>Naval Ocean Systems Center</td>
</tr>
<tr>
<td>OSI</td>
<td>Open-Systems Interconnection</td>
</tr>
<tr>
<td>PDN</td>
<td>Power-Distribution Node</td>
</tr>
<tr>
<td>PPCU</td>
<td>Platform-Power-Conditioning Unit</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>RCS</td>
<td>Realtime Control System</td>
</tr>
<tr>
<td>RP</td>
<td>Remote Platform</td>
</tr>
<tr>
<td>RVMF</td>
<td>Robotic-Vehicle Message Format</td>
</tr>
<tr>
<td>SBIR</td>
<td>Small-Business Innovative Research</td>
</tr>
<tr>
<td>SDLC</td>
<td>Synchronous Data-Link Control</td>
</tr>
<tr>
<td>SDS</td>
<td>Software-Design Specification</td>
</tr>
<tr>
<td>SPEC</td>
<td>System Specification</td>
</tr>
<tr>
<td>UGV/TOV</td>
<td>Unmanned Ground Vehicle/Teleoperated Vehicle</td>
</tr>
<tr>
<td>VSI</td>
<td>Virtual Systems Interface</td>
</tr>
<tr>
<td>WAN</td>
<td>Wide Area Network</td>
</tr>
</tbody>
</table>
8.0 REFERENCES


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.

1  LD  
2  A  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  59 directories
CPCl: IED90-MRA-MAKEFILE.TXT-R1CD

Description: Makefile for the Modular Robotic Architecture (MRA). Updates are all of the programming modules that reside under the individual systems and subsystems.

Targets are available for the following systems/subsystems:

mra - MRA subsystem (make all)
dev - COM Standard Hardware Device Driver Subsystem
ics - COM Local Communications Subsystem
rms - COM Method Manager Subsystem
qcs - ICM Global Communications Subsystem
lac - ICM Applications Controller
lds - MPU Logic Device Subsystem
cac - MPU Applications Controller
av - MPU Audio/Visual Module
calr - MPU Collision Avoidance 1R Module
caus - MPU Collision Avoidance 1RCS Module
cs - MPU Control Stations Module
hlp - MPU High-Level Processing Module
mob - MPU Mobility Module
nav - MPU Navigation Module
watch - MPU Clock/Watch Module

Compiler literal definitions and their meanings follow:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>CF-31</td>
<td>CF-31 running at 11.0592 MHz</td>
</tr>
<tr>
<td>BMAT</td>
<td>BMAT running at 11.0592 MHz</td>
</tr>
<tr>
<td>BSCB</td>
<td>BSCB running at 11.0592 MHz</td>
</tr>
<tr>
<td>ICM152</td>
<td>ICM running at 11.0592 MHz</td>
</tr>
<tr>
<td>ICMON</td>
<td>ICM running at 11.0592 MHz</td>
</tr>
</tbody>
</table>

Notes:
1. The dependency and production rules are included here.
2. Linkage parameters for RAM-based systems using the CF-31:
   - CODE SEGMENT = 00000h
   - XDATA SEGMENT = 00000h
3. Linkage parameters for ROM-based systems using the CF-31:
   - CODE SEGMENT = 00000h
   - XDATA SEGMENT = 00000h
4. Linkage parameters for ROM-based systems using the ICM:
   - CODE SEGMENT = 00000h
   - XDATA SEGMENT = 00000h
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 files to be maintained by separating the binaries.

Target System Processor Binary Directory

<table>
<thead>
<tr>
<th>System</th>
<th>Processor</th>
<th>Binary Directory</th>
</tr>
</thead>
<tbody>
<tr>
<td>CF-31</td>
<td>8031</td>
<td>bin\8031</td>
</tr>
<tr>
<td>CF-31/35</td>
<td>8031/8035</td>
<td>bin\8031/8035</td>
</tr>
<tr>
<td>ICM</td>
<td>8031</td>
<td>bin\8031</td>
</tr>
<tr>
<td>ICMON</td>
<td>8031</td>
<td>bin\8031</td>
</tr>
<tr>
<td>BSCB</td>
<td>8088/V20</td>
<td>bin\8088/V20</td>
</tr>
<tr>
<td>BSCB</td>
<td>8088/V20</td>
<td>bin\8088/V20</td>
</tr>
</tbody>
</table>

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...
<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>147</td>
<td>MPUSYS = mpu</td>
</tr>
<tr>
<td>148</td>
<td>MKALIB = $(PROJ)/lib</td>
</tr>
<tr>
<td>149</td>
<td># Library binaries</td>
</tr>
<tr>
<td>150</td>
<td>APPSRC = $(PROJ)/APPsys/src</td>
</tr>
<tr>
<td>151</td>
<td># Source files</td>
</tr>
<tr>
<td>152</td>
<td>COMSRCS = $(PROJ)/COMSYS/src</td>
</tr>
<tr>
<td>153</td>
<td>ICMSRCS = $(PROJ)/ICNSYS/src</td>
</tr>
<tr>
<td>154</td>
<td>MUPUSRC = $(PROJ)/MPUSYS/src</td>
</tr>
<tr>
<td>155</td>
<td>APPBIN31 = $(PROJ)/APPsys/bin/31binaries</td>
</tr>
<tr>
<td>156</td>
<td># 31 binaries</td>
</tr>
<tr>
<td>157</td>
<td>COMBIN31 = $(PROJ)/COMSYS/bin/31binaries</td>
</tr>
<tr>
<td>158</td>
<td>ICMBIN31 = $(PROJ)/ICNSYS/bin/31binaries</td>
</tr>
<tr>
<td>159</td>
<td>MUPBIN31 = $(PROJ)/MPUSYS/bin/31binaries</td>
</tr>
<tr>
<td>160</td>
<td>APPBIN152 = $(PROJ)/APPsys/bin/152binaries</td>
</tr>
<tr>
<td>161</td>
<td># 152 binaries</td>
</tr>
<tr>
<td>162</td>
<td>COMBIN152 = $(PROJ)/COMSYS/bin/152binaries</td>
</tr>
<tr>
<td>163</td>
<td>ICMBIN152 = $(PROJ)/ICNSYS/bin/152binaries</td>
</tr>
<tr>
<td>164</td>
<td>MUPBIN152 = $(PROJ)/MPUSYS/bin/152binaries</td>
</tr>
<tr>
<td>165</td>
<td>ICHBIN152_MON = $(PROJ)/ICNSYS/bin/152_mon</td>
</tr>
<tr>
<td>166</td>
<td>APPBIN512 = $(PROJ)/APPsys/bin/512binaries</td>
</tr>
<tr>
<td>167</td>
<td># 512 binaries</td>
</tr>
<tr>
<td>168</td>
<td>COMBIN512 = $(PROJ)/COMSYS/bin/512binaries</td>
</tr>
<tr>
<td>169</td>
<td>ICMBIN512 = $(PROJ)/ICNSYS/bin/512binaries</td>
</tr>
<tr>
<td>170</td>
<td>MUPBIN512 = $(PROJ)/MPUSYS/bin/512binaries</td>
</tr>
<tr>
<td>171</td>
<td>APPBINSRCB = $(PROJ)/APPsys/bin/absb</td>
</tr>
<tr>
<td>172</td>
<td># IFS-SRCB binaries</td>
</tr>
<tr>
<td>173</td>
<td>COMBINSRCB = $(PROJ)/COMSYS/bin/absb</td>
</tr>
<tr>
<td>174</td>
<td>ICMBINSRCB = $(PROJ)/ICNSYS/bin/absb</td>
</tr>
<tr>
<td>175</td>
<td>MUPBINSRCB = $(PROJ)/MPUSYS/bin/absb</td>
</tr>
<tr>
<td>176</td>
<td># Common subsystem level source directories</td>
</tr>
<tr>
<td>177</td>
<td>DEVSRCS = $(COMSRC)/dev</td>
</tr>
<tr>
<td>178</td>
<td>JDRSRCS = $(COMSRC)/ldr</td>
</tr>
<tr>
<td>179</td>
<td>LCSRCS = $(COMSRC)/lce</td>
</tr>
<tr>
<td>180</td>
<td>MMSRCS = $(COMSRC)/mms</td>
</tr>
<tr>
<td>181</td>
<td># IFS subsystem level source directories</td>
</tr>
<tr>
<td>182</td>
<td>GCSRCS = $(IFSRC)/gcs</td>
</tr>
<tr>
<td>183</td>
<td>IACSRCS = $(IFSRC)/iac</td>
</tr>
<tr>
<td>184</td>
<td># MPU subsystem level source directories</td>
</tr>
<tr>
<td>185</td>
<td>LDSSRCS = $(MPUSRCS)/ld</td>
</tr>
<tr>
<td>186</td>
<td>MACSRCS = $(MPUSRCS)/mac</td>
</tr>
<tr>
<td>187</td>
<td># Application subsystem level source directories</td>
</tr>
<tr>
<td>188</td>
<td>AVSRCS = $(APPSRC)/av</td>
</tr>
<tr>
<td>189</td>
<td>CAILSRC = $(APPSRC)/cair</td>
</tr>
<tr>
<td>190</td>
<td>CAUSSRCS = $(APPSRC)/caus</td>
</tr>
<tr>
<td>191</td>
<td>CGSRCS = $(APPSRC)/cas</td>
</tr>
<tr>
<td>192</td>
<td>HFSRCS = $(APPSRC)/hfs</td>
</tr>
<tr>
<td>193</td>
<td>HOBSRCS = $(APPSRC)/hob</td>
</tr>
<tr>
<td>194</td>
<td>NAVSRCS = $(APPSRC)/nav</td>
</tr>
<tr>
<td>195</td>
<td>MATCHSRC = $(APPSRC)/watch</td>
</tr>
</tbody>
</table>

# Common system targets

```bash
dev:
cd $(DEVSRC)
```

```bash
lic:
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```

```bash
make lic
```
 Jan 22 1992 08:28:21 makefile Page 1

1 |
2 |
3 | MAKEFILE
4 |
5 | CPU: IED90-MRA-APP-CAIR-MAKEFILE-TEXT-MOCO
6 |
7 | Description: Makefile for the Mobile Security Robot application.
8 | Makes the Collision Avoidance IR module subsystems.
9 |
10 | Targets are available for the following systems/subsystems:
11 | cair - MPU Collision Avoidance IR Module
12 | cair.hex - MPU Collision Avoidance IR Program (B031)
13 | print - Print Collision Avoidance IR source files
14 |
15 | Notes:
16 | 1) The dependency and production rules are included here.
17 | 2) See also `mrmakefile`
18 |
19 | Edit History: 03/25/91 - Written by Robin T. Laird.
20 |
21 |
22 |
23 |
24 |
25 |
26 | RULES
27 |
28 |
29 |
30 |
31 |
32 |
33 |
34 |
35 |
36 |
37 |
38 |
39 |
40 |
41 |
42 |
43 |
44 |
45 |
46 |
47 |
48 |
49 |
50 |
51 |
52 |
53 |
54 |
55 |
56 |
57 |
58 |
59 |
60 |
61 |
62 |
63 |
64 |
65 |
66 |
67 |
68 |
69 |
70 |
71 | PROJ = mra
72 | APPSYS = app
73 | COMSYS = com
6 Jan 22 1992 08:28:21 makefile Page 2
74 | ICN.sys = icn
75 | MIPsys = mpu
76 | APPSRC = $(PROJ)/$(APPsys)/src
77 | COMSRC = $(PROJ)/$(COMSYS)/src
78 | ICNSRC = $(PROJ)/$(ICNSRC)/src
79 | MFPSRC = $(PROJ)/$(MPFSPSRC)/src
80 |
81 | APPRRN31 = $(PROJ)/$(APPsys)/bin/0831
82 | APPRRN52 = $(PROJ)/$(APPsys)/bin/0852
83 | APPRRNMS = $(PROJ)/$(APPsys)/bin/msdos
84 | APPRRN8C = $(PROJ)/$(APPsys)/bin/8bc
85 |
86 | COMBRRN31 = $(PROJ)/$(COMSYS)/bin/0831
87 |
88 |
89 |
90 |
91 |
92 |
93 |
94 |
95 |
96 |
97 |
98 |
99 |
100 | GCSSRC = $(ICNSRC)/gcs
101 | IACSRC = $(ICNSRC)/iac
102 |
103 |
104 |
105 |
106 |
107 |
108 |
109 |
110 | CAIRSRC = $(APPsys)/cair
111 |
112 |
113 |
114 |
115 |
116 |
117 | CAIR = $(APPRRN31)/cair/d.obj
118 | $(MIP8N31)/cair/obj
119 | $(MIP8N31)/main/obj
120 | $(COMBRRN31)/rca/obj
121 | $(COMBRRN31)/msm/obj
122 | $(COMBRRN31)/mm/obj
123 | $(COMBRRN31)/msm/obj
124 |
125 |
126 |
127 |
128 |
129 |
130 |
131 | cair : $(APPRRN31)/cair.hex
132 |
133 |
134 |
135 |
136 | $(APPRRN31)/cair : $(CAIR) $(LINK) $(CAIRSRC)/cair/res
137 |
138 |
139 |
140 |
141 |
142 |
143 | print : $(CAIR)
144 | $(p2p) -nh cair/rd.h | post
145 | $(p2p) -nh cair/rd.c | post
146 |
147 |
# caird.h

```c
/* Error codes indicate source of function execution failure. */
#define CAILB_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])[] =
{
    obj_class,  
    obj_superclass,  
    d_unit_name,  
    d_unit_reset,  
    cair_max_update_rate,  
    cair_number_sensors,  
    cair_sensor_range,  
    cair_report_sensor,  
    cair_runtime_report_sensors,  
    cair_wait_delta_sense
};
#endif

# ifndef CAILB_MODULE_CODE
#define CAILB_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 CAILB_NUM_FUNCS 10
int lmm_num_funcs = CAILB_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_RESET;

/* 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_runtime_report_sensors();
int cair_wait_delta_sense();
```

---

**Edit History:** 01/04/91 - Written by Robin T. Laird.
/* * caird.c */

#include <stdio.h> /* Absolute addressing defs. */
#include <string.h> /* String constants and types. */
#include <sys/utsname.h> /* Method manager. */
#include <linux/irq.h> /* Thread manager. */
#include "caird.h" /* CAIR dictionary entries. */
#include <linux/xdata.h> /* Global "instance" variables. */

static XDATA v_cair_max_update_rate = 10; /* HZ. */
static XDATA byte v_cair_number_sensors = 15; /* Discrete. */
static XDATA int v_cair_sensors_range = 300; /* Tenth of a second. */

/* The IR sensors are connected to the 8255 of the 8031 (CP-31). */
/* IR sensors are 1-8 are in port A, sensor 9 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 1-9 are in port B, cont 3 at bit 0, cont 8 at bit 7. */
/* Continuity for IRs 9-11 are in port C, cont 9 at bit 0, 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 PC_BIND_8255 XBYTE[0x0003] /* 8255 control register. */
#define 8255 control values (see Intel books). */
#define ALL_INPUT 0x0B /* Ports A, B, and C as input. */
#define PC_CONT_SHIFT 4 /* Shift to make a word. */
#define PC_CONT_MASK 0x0070 /* Keep bits 4, 5, and 6. */
#define PC_BIND_CONT_UNUSED 0x07FF /* Mask out missing sensor bits. */
/* Continuity check bit shift values and masks. */
/* Continuity check mask global variable. */

/ * 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.N).

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)PB_8255 & PC_CONT_MASK) << PC_CONT_SHIFT | (word)PB_8255
    cont_mask = PC_CONT_UNUSED & -cont_mask;

    /**********************************************************/
    /* 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_stout). */
    /* Globals: mm_stout : 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_sprintf(mm_stout, "Ab", v_cair_max_update_rate);
    return(MM_COMMAND_EXECUTED);
}
}
int cair_number_sensors();

// Put number of sensors in the standard parameter output buffer.

int cair_report_sensor();

// Function: Returns the maximum range of the IR sensor (in tenths
// of inches). The value is returned in the standard parameter
// output buffer.

int cair_sensor_range();

// Output: Number of IR sensors as a byte value (mm_stdout).

// Global: mm_stdout : module MM.C
// v_Calr_number_sensors : module CAIR.C

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

/* Continuity mask is used to force disconnected sensors to 0-FF. */

/* Get the sensor value (0-0FF/not activated, 1-ON/activated). */

/* Continuity mask is used to force disconnected sensors to 0-FF. */

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

/*
 * Get the sensor numbers from the standard parameter input buffer.
 *
 * mm_ascanf(mm_stdin, "hh\b", &s1, &s2);
 *
 * Make sure requested sensors are in range.
 *
 * if (s1 > v_cair_number_sensors || s2 > v_cair_number_sensors)
 * return(NO_COMMAND_EXECUTION_FAILURE);
 *
 * Report invalid sensor range, and command failed.
 *
 * mm_sprintf(mm_stdout, "%h", CAIR_BAD_SENSOR_NUMBER);
 * return(NO_COMMAND_EXECUTED);
 *
 * Make sure first sensor is less than last sensor.
 *
 * if (s1 > s2) 
 * s1 = s2;
 * s2 = s1.
 *
 * Get all sensor values (0=OFF/not activated, 1=ON/activated).
 *
 * (val) = ((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 = (val) >> (i-1) & 1;
 *
 * Put sensor value in standard parameter output buffer.
 *
 * mm_sprintf(mm_stdout, "%y", v);
 * return(NO_COMMAND_EXECUTED);
 */

/*
 * Function:      cair_wait_delta_sensor
 *
 * 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_sensor();
 *
 * Output: States of all sensors as a word value (mm_stdout).
 *
 *Globals:  mm_stdout  ; variable MM.C
 *           cont_mask   ; variable CAIR.C
 *
 * Edit History: 05/24/90 - Original code written by Richard P. Smuro,
 * 03/06/91 - Dictionary implementation by Robin T. Laird.
 */

```c
/* cairl.h */

/* CAIRL.H */

* CPC: IED90-MRA-APP-CAIR-CAIRL-H-ROLO
* 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_num_sensor_sensors();
* cair_sensor_range();
* CAIR (STATUS_REQUEST, PERIODIC_STATUS_REQUEST);
* cair_report_sensor();
* cair_lri_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.

#include SMN.H /* System method manager. */

/* Public data structures: */
define CAIR_UNIT_NAME "CAIR"
define CAIR_FN_MAX_UPDATE_RATE (0+SMN.NUM_INHERITED_FUNS)
define CAIR_FN_NUM_SENSOR_SENSORS (1+SMN.NUM_INHERITED_FUNS)
define CAIR_FN_SENSOR_RANGE (2+SMN.NUM_INHERITED_FUNS)
define CAIR_FN_IR_SENSOR (3+SMN.NUM_INHERITED_FUNS)
define CAIR_FN_IR2_REPORT (4+SMN.NUM_INHERITED_FUNS)
define CAIR_FN_WAIT_DELTA (5+SMN.NUM_INHERITED_FUNS)

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

/* Public functions: */
int cair_max_update_rate(void);
int cair_num_sensor_sensors(void);
int cair_sensor_range(void);
int cair_report_sensor(byte n, word period);
int cair_lri_ir2_report_sensor(byte a1, byte a2, word period);
int cair_wait_delta_sense(word period);
#endif
```
CAIRL.C

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 (cairl...)

Parameters are passed to/from the functions via the standard I/O buffers (mm_message and mm_message). All local methods must return an integer value indicating success or failure as in PM_COMMAND_EXECUTED or PM_COMMAND_EXECUTION_FAILURE. Also, for commands that all the reason for failure must be put in the output buffer as a byte value.

Module CAIRL.C exports the following variables/functions:

- cairl_max_update_rate()
- cairl_number_sensors()
- cairl_report_sensor()
- cairl_wait_delta_sensor()
- cairl_wait_range()

Notes: 1) The calling sequence for each function is listed below.

Edit History: 01/10/91 - Written by Robin T. Laird.

#include <sysdeps.h> /* System constants and types. */
#include <errno.h> /* Method manager. */
#include <err.h> /* System method manager. */
#include "cairl.h" /* Unit name and function IDs. */

int cairl_max_update_rate() {
  int event;
  mm_message m;
  m_trans_disposition = MM_INITIATING;
  m_trans_category = MM_QUERY_OPERATING_LIMITS;
  m_function_id = CAIRL_MAX_UPDATE_RATE;
  mm_generate_message(CAIRL_UNIT_NAME, &m, event);
  return(event);
}

int cairl_number_sensors() {
  int event;
  mm_message m;
  m_trans_disposition = MM_INITIATING;
  m_trans_category = MM_QUERY_OPERATING_LIMITS;
  m_function_id = CAIRL_NUMBER_SENSORS;
  mm_generate_message(CAIRL_UNIT_NAME, &m, event);
  return(event);
}

int cairl_wait_delta_sensor(period) {
  int event;
  mm_message m;
  m_trans_disposition = MM_INITIATING;
  m_trans_category = MM_QUERY_OPERATING_LIMITS;
  m_function_id = CAIRL_WAIT_DELTA_SENSOR;
  mm_generate_message(CAIRL_UNIT_NAME, &m, event);
  return(event);
}

int cairl_wait_range() {
  int event;
  mm_message m;
  m_trans_disposition = MM_INITIATING;
  m_trans_category = MM_QUERY_OPERATING_LIMITS;
  m_function_id = CAIRL_WAIT_RANGE;
  mm_generate_message(CAIRL_UNIT_NAME, &m, event);
  return(event);
}
else
    m.trans_category = MM_PERIODIC_STATUS_REQUEST;
    mm_sprintf(mm_stdout, "tu", period);
    mm_generate_message(CAIR_UNIT_NAME, &m, event);
    return(event);
# MAKEFILE

rules

1.
2.
3.
4.
5.
6.
7.
8.
9.
10.
11.
12.
13.
14.
15.
16.
17.
18.
19.
20.
21.
22.
23.
24.
25.
26.
27.
28.
29.
30.
31.
32.
33.
34.
35.
36.
37.
38.
39.
40.
41.
42.
43.
44.
45.
46.
47.
48.
49.
50.
51.
52.
53.
54.
55.
56.
57.
58.
59.
60.
61.
62.
63.
64.
65.
66.
67.
68.
69.
70.
71.
72.
73.
74.
75.
76.
77.
78.
79.
80.
81.
82.
83.
84.
85.
86.
87.
88.
89.
90.
91.
92.
93.
94.
95.
96.
97.
98.
99.
100.
101.
102.
103.
104.
105.
106.
107.
108.
109.
110.
111.
112.
113.
114.
115.
116.
117.
118.
119.
120.
121.
122.
123.
124.
125.
126.
127.
128.
129.
130.
131.
132.
133.
134.
135.
136.
137.
138.
139.
140.
141.
142.
143.
144.
145.
146.

# Control settings for Franklin 8031 development

CFLAGS = -c
LDLIBS = -la
SUFFIXES = .o .c .d

# Control settings for Microsoft MS-DOS development

MSC = -c
MGRS = -ma
MCLink = -link
MSCFLAGS = -L /c /o1 /21 /2d
MSAFLAGS = -a
MSLIMFLCS = /co
LOADLIBS = -a

# Project, system, and application level definitions

PROJ = mra
APPSYS = app
COMSYS = com

# Common subsystem level source directories

COMLIB = $(COMSYS)\src
COMLIBS = $(COMSYS)\src
COMLIBINC = $(COMSYS)\inc
COMLIBSINC = $(COMSYS)\inc
COMLIBSINC86 = $(COMSYS)\inc\86

# Common subsystem global include and compilation units

DEVRC = $(COMRC)\dev
HDRSRC = $(COMRC)\hdr
SYSDefs = $(COMRC)\sysdefs.h

# Common subsystem include and compilation units

COMLIBRC = $(COMRC)\comlib.rc
COMLIBSRC = $(COMRC)\comlib.src
COMLIBSRC86 = $(COMRC)\comlib.src
COMLIBSINC86 = $(COMRC)\comlib.src

# COMDEV dependencies

COMDEV = $(COMDEV)\comdev.h $(COMDEV)\comdev.c

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

RTC = $(SYSDefs) $(DEVRC)\rtc.h $(DEVRC)\rtc.c

$(COMLIBSINC)\rtc.obj : $(RTC)
$(COMLIBSINC)\rtc.obj : $(RTC)
$(COMLIBSINC)\rtc.obj : $(RTC)
$(COMLIBSINC)\rtc.obj : $(RTC)
$(COMLIBSINC)\rtc.obj : $(RTC)
$(COMLIBSINC)\rtc.obj : $(RTC)
$(COMLIBSINC)\rtc.obj : $(RTC)
$(COMLIBSINC)\rtc.obj : $(RTC)
$(COMLIBSINC)\rtc.obj : $(RTC)
$(COMLIBSINC)\rtc.obj : $(RTC)
$(COMLIBSINC)\rtc.obj : $(RTC)
$(COMLIBSINC)\rtc.obj : $(RTC)

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

SIO = $(SYSDefs) $(DEVRC)\sio.h $(DEVRC)\sio.c

$(COMLIBSINC)\sio.obj : $(SIO)
$(COMLIBSINC)\sio.obj : $(SIO)
$(COMLIBSINC)\sio.obj : $(SIO)
$(COMLIBSINC)\sio.obj : $(SIO)
$(COMLIBSINC)\sio.obj : $(SIO)
$(COMLIBSINC)\sio.obj : $(SIO)
$(COMLIBSINC)\sio.obj : $(SIO)
$(COMLIBSINC)\sio.obj : $(SIO)
$(COMLIBSINC)\sio.obj : $(SIO)
$(COMLIBSINC)\sio.obj : $(SIO)
$(COMLIBSINC)\sio.obj : $(SIO)
$(COMLIBSINC)\sio.obj : $(SIO)

# MRA Application dependencies for 80152, 8031, MS-DOS

MRA = $(SYSDefs) $(DEVRC)\mra.h $(DEVRC)\mra.c

$(COMLIBSINC)\mra.obj : $(MRA)
$(COMLIBSINC)\mra.obj : $(MRA)
$(COMLIBSINC)\mra.obj : $(MRA)
$(COMLIBSINC)\mra.obj : $(MRA)
$(COMLIBSINC)\mra.obj : $(MRA)
$(COMLIBSINC)\mra.obj : $(MRA)
$(COMLIBSINC)\mra.obj : $(MRA)
$(COMLIBSINC)\mra.obj : $(MRA)
$(COMLIBSINC)\mra.obj : $(MRA)
$(COMLIBSINC)\mra.obj : $(MRA)
$(COMLIBSINC)\mra.obj : $(MRA)
$(COMLIBSINC)\mra.obj : $(MRA)

# COMDEV dependencies

COMDEV = $(COMDEV)\comdev.h $(COMDEV)\comdev.c

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

RTC = $(SYSDefs) $(DEVRC)\rtc.h $(DEVRC)\rtc.c

$(COMLIBSINC)\rtc.obj : $(RTC)
$(COMLIBSINC)\rtc.obj : $(RTC)
$(COMLIBSINC)\rtc.obj : $(RTC)
$(COMLIBSINC)\rtc.obj : $(RTC)
$(COMLIBSINC)\rtc.obj : $(RTC)
$(COMLIBSINC)\rtc.obj : $(RTC)
$(COMLIBSINC)\rtc.obj : $(RTC)
$(COMLIBSINC)\rtc.obj : $(RTC)
$(COMLIBSINC)\rtc.obj : $(RTC)
$(COMLIBSINC)\rtc.obj : $(RTC)
$(COMLIBSINC)\rtc.obj : $(RTC)
$(COMLIBSINC)\rtc.obj : $(RTC)

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

SIO = $(SYSDefs) $(DEVRC)\sio.h $(DEVRC)\sio.c

$(COMLIBSINC)\sio.obj : $(SIO)
$(COMLIBSINC)\sio.obj : $(SIO)
$(COMLIBSINC)\sio.obj : $(SIO)
$(COMLIBSINC)\sio.obj : $(SIO)
$(COMLIBSINC)\sio.obj : $(SIO)
$(COMLIBSINC)\sio.obj : $(SIO)
$(COMLIBSINC)\sio.obj : $(SIO)
$(COMLIBSINC)\sio.obj : $(SIO)
$(COMLIBSINC)\sio.obj : $(SIO)
$(COMLIBSINC)\sio.obj : $(SIO)
$(COMLIBSINC)\sio.obj : $(SIO)
$(COMLIBSINC)\sio.obj : $(SIO)

# MRA Application dependencies for 80152, 8031, MS-DOS

MRA = $(SYSDefs) $(DEVRC)\mra.h $(DEVRC)\mra.c

$(COMLIBSINC)\mra.obj : $(MRA)
$(COMLIBSINC)\mra.obj : $(MRA)
$(COMLIBSINC)\mra.obj : $(MRA)
$(COMLIBSINC)\mra.obj : $(MRA)
$(COMLIBSINC)\mra.obj : $(MRA)
$(COMLIBSINC)\mra.obj : $(MRA)
$(COMLIBSINC)\mra.obj : $(MRA)
$(COMLIBSINC)\mra.obj : $(MRA)
$(COMLIBSINC)\mra.obj : $(MRA)
$(COMLIBSINC)\mra.obj : $(MRA)
$(COMLIBSINC)\mra.obj : $(MRA)
$(COMLIBSINC)\mra.obj : $(MRA)
/* ***********************************************************************
rtc.h
------------------------------------------------------------------------*/

/*
  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: */

#ifdef RTC MODULE CODE
#define RTC MODULE CODE 5000
#endif

#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
#include "timeb.h"  /* MS-DOS time struct definitions. */
#include "reg51.h"   /* SFR definitions for 8031. */
#include "stdlib.h"  /* Random number generator fn. */
#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 occurrence. */
/* Should be set to A0H after each successful function call. */
/* Varible can be examined by other software after each function call. */

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

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

#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

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

void rtc_init() {
    /* Assume function successful... */
    if (!defined(INMAT)) {
        static void rtc_tickint() interrupt 1 using 2
        {
            tickcnt++;
        }
    }
    #ifndef
    #endif
}

/* 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. */
```

```
Jan 22 1992 07:39:00  rtc.c  Page 3

147    srand((int)rtc_time());
148 }
149
150 }----------rtc_wait-----------------------------------------
151
152 * Function:      waits for a specified period of time (in ms).
153 *                Function does not return until time expired.
154 *                If duration parameter is RTC_RANDOM_TIME then the
155 *                function waits a random amount of time between 2000
156 *                and 7000 ms.
157 *                Input: rtc_wait( unsigned long duration; time to wait or RTC_RANDOM_TIME; );
158 *                Output: Nothing.
159 *                Global: rtc_error : module RTC.C
160 * Edit History: 03/28/91 - Written by Robin T. Laird.
161
162 */
163
164 typedef MIN_RANDOM_TIME 2000 /* Wait no less than 2 seconds. */
165 typedef MAX_RANDOM_TIME 7000 /* Wait no more than 7 seconds. */
166
167 void rtc_wait(duration)
168     unsigned long duration;
169 {
170     int t;
171     unsigned long t0;
172
173     rtc_error = OK;  /* Assume function successful... */
174
175     t = rtc_time();
176     if (duration == RTC_RANDOM_TIME)
177     {
178     /* Gen a random time >= MIN_RANDOM_TIME and <= MAX_RANDOM_TIME. */
179     t = rand();
180     while (t < MIN_RANDOM_TIME || t > MAX_RANDOM_TIME)
181     { t = rand();
182     if (rtc_time() > t0 + MAX_RANDOM_TIME) return;
183     }
184     }
185
186     /* Wait that amount of time. */
187     while (rtc_time() < t0 + t);
188     }
189
190     else
191     { /* Wait amount of time specified by parameter duration. */
192     while (rtc_time() < t0 + duration);
193     }
194 }
195
196 }----------rtc_wait-----------------------------------------
197
198 */
199
200 }----------rtc_delay-----------------------------------------
201
202 * Function:      Routine to return time in milliseconds. The timer
203 *                period (as configured in the initialization routine
204 *                is approximately 8,868...ms for an 11,059 MHz CPU or
205 *                6,666...ms for 14,755 MHz CPU. Adding whole portion of ms
206 *                value derived from tick count and fraction of timer period
207 *                derives from timer 0 to calculate actual time. This gives
208 *                a time value that is accurate within +/- 1 ms.
209 */
210
211 #define OSQP_FSCALE_FACTOR 12 /* Reduced fraction equivalent. */
212 #define OSQP_SCALE_FACTOR 10 /* Reduced fraction equivalent. */
213 #define OSQP_PER_TIME 640 /* ms/count divisor. */
214 #define OSQP_PER_COUNT 5 /* ms/count diviend. */
215 #define OSQP_PER_MSB 4608 /* count/ms divisor. */
216
217 int rtc_delay()
218 { /* Calculate portion of timer period (in ms) based on current
219     *       timer value. */
220     int t0 = rtc_time();
221     int t1 = rtc_time();
222     int t2 = t1 - t0;
223     /* Add this value to time based on tick count to give actual time. */
224     int count = (((word)(t0<<8) | (word)t1) * OSQP_PER_COUNT) / OSQP_PER_MSB;
225     return((unsigned long)((tickcnt * OSQP_SCALE_FACTOR / OSQP_FSCALE_FACTOR) * count));
226     }
227
228 #else
229
230 /* Get time and store in timebuf structure (this includes ms portion). */
231 /* Return number of seconds * 1000 plus ms time for total ms time. */
232
233 t0 = time(timebuf);
234 t1 = time(timebuf)+((unsigned long)timebuf.time*1000+timebuf.mmillit);
235 #endif
236
237 #if defined(Rtc_Have_RTC)
238
239 #define MS_PER_COUNT 5 /* ms/count diviend. */
240 #define OSQP_PER_COUNT 5 /* ms/count diviend. */
241 #define OSQP_PER_MSB 5 /* ms/count diviend. */
242 #define OSQP_PER_MAX_MSB 5 /* ms/count diviend. */
243 #define OSQP_PER_LSB 5 /* ms/count diviend. */
244
245 #endif
246
247 unsigned long rtc_time()
248 { /* Get time from timer 0 and return it. */
249     return((unsigned long)((tickcnt * OSQP_FSCALE_FACTOR / OSQP_SCALE_FACTOR) * count));
250     }
251
252 #endif
253
254 static struct timebuf timebuf;
255 static char timebuf.name[10];
256 static struct timebuf timebuf2;
257 static char timebuf2.name[10];
258
259 char * timebuf.name[10] = "timebuf.name[10] = "timebuf.name[10];
260 char * timebuf2.name[10] = "timebuf2.name[10] = "timebuf2.name[10];
261
262 #define Rtc_Have_RTC 1
263 #endif
264
265 /* Get time from timer 0 and return it. */
266
```
#define CMS 0x2C  /* LFM-S104 J3. */

#define /* 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_bits);
void sio_putchar(int port, byte c);
byte sio_get_byte(int port);
int sio_putchar(int port, byte *s);
int sio_putchar(int port, byte *s);
void sio_putchar(int port, int num_bytes, byte *s);

#undef
```c
#define CC_MSK 0x0E /* Modem status register. */
#define CC_DATA_READY 0x01 /* LSR data ready bit. */
#define CC_RXERRORS 0x1E /* LSR error bits mask. */
#define CC_TRANSITIVE_HALT 0x20 /* LSR xmt holding register empty. */
#define CC_DLBA 0x00 /* DLAB set low (0). */
#define CC_DLBI 0x08 /* 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_MODBUS_ENABLE 0x08 /* DTR, RTS, and OUT2 active. */
#else defined(SCBC) /* */
/* */
#define CC_MSK 0x0E /* Modem status register. */
#define CC_DATA_READY 0x01 /* LSR data ready bit. */
#define CC_RXERRORS 0x1E /* LSR error bits mask. */
#define CC_TRANSITIVE_HALT 0x20 /* LSR xmt holding register empty. */
#define CC_DLBA 0x00 /* DLAB set low (0). */
#define CC_DLBI 0x08 /* 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_MODBUS_ENABLE 0x08 /* DTR, RTS, and OUT2 active. */
```
typedef struct {
    byte in[128];
} inputBuff;

typedef struct {
    byte out[128];
} outputBuff;

#define MAX_BUFFER_SIZE 128

static inputBuff inputBuffer;
static inputBuff outputBuffer;

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

#define SIO_Init(port, baud_rate, parity, word_len, stop_bits)

#define SIO_Error()

#define SIO_Error(1)
Jan 22 1992 07:42:35  sio.c  Page 5

293  
294  switch(word_len)  
295  
296  case NL6:  
297   case NL6:  
298   case NL7:  
299   case NL8:  
300   break;  
301   default:  
302   
303   sio_error = SIO_ERR_WORD_LENGTH;  
304   return;  
305   
306   switch(stop_bit)  
307   
308   case BR1:  
309   case BR2:  
310   break;  
311   default:  
312   
313   sio_error = SIO_ERR_STOP_BITS;  
314   return;  
315   
316   if(defined(I80152)) !defined(I8031))  
317   */ Local and auxiliary serial channels are initialized differently. */  
318   
319   if (port == ASCII)  
320   
321   /* Determine 8256 baud rate from parameter baud rate. */  
322   
323   switch(baud_rate)  
324   
325   case MAX_BAUD_RATE:  
326   
327   case BR1200:  
328   baud_rate = _BR1200;  
329   break;  
330   case BR600:  
331   baud_rate = _BR600;  
332   break;  
333   case BR2400:  
334   baud_rate = _BR2400;  
335   break;  
336   case BR2400:  
337   baud_rate = _BR2400;  
338   break;  
339   case BR1200:  
340   baud_rate = _BR1200;  
341   break;  
342   case BR600:  
343   baud_rate = _BR600;  
344   break;  
345   case BR2400:  
346   baud_rate = _BR2400;  
347   break;  
348   default:  
349   
350   baud_rate = _BR1200;  
351   break;  
352   
353   if (1)  
354   */ Init COMMAND REGISTER 1: stop bits, character length. */  
355   
356   */ Init COMMAND REGISTER 2: baud rate, parity. */  
357   
358   */ Init COMMAND REGISTER 3: receiver enable. */  
359   
360   /* Clear the receive buffer. */  
361   
362   MUART_CMD_1 = word_len | stop_bit;  
363   
364   MUART_CMD_2 = parity | baud_rate;  
365   
366   MUART_CMD_3 = MUART_RCV_ENABLE;  
367   
368   while((MUART_MSK & MUART_RBF_MSK) == MUART_RBF_MSK) i = MUART_RBR;  
369   
370   if (defined(MUART_POLLED))

Jan 22 1992 07:42:35  sio.c  Page 6

366  
367  /* init I/O data queue structures. */  
368  
369  wmt_buffer.front = wmt_buffer.rear = 0;  
370  wmt_buffer.empty = TRUE;  
371  wmt_buffer.full = FALSE;  
372  
373  rcv_buffer.front = rcv_buffer.rear = 0;  
374  rcv_buffer.empty = TRUE;  
375  rcv_buffer.full = FALSE;  
376  
377  /* Indicate low-level triggered interrupt from 8256. */  
378  
379  ITO = 0;  
380  EXO = 1;  
381  
382  /* Enable receive and transmit interrupts on the CP=31/535 8256. */  
383  
384  /* Assumes some other function enables processor interrupts. */  
385  
386  MUART_CMD_3 = MUART_NESTED_INTS;  
387  MUART_IER = MUART_RCV_INT_ENABLE;  
388  MUART_IER = MUART_XMT_INT_ENABLE;  
389  
390  endif

391  }

392  }

393  
394  /* Set registers according to chosen options. */  
395  
396  /* TIMER 1 is used in MODE 2 for variable baud rates. */  
397  
398  /* TH1 sets TIMER 1 re-load value. */  
399  
400  /* SCOM controls the serial port modes settings. */  
401  
402  /* TMOD controls the timer mode settings. */  
403  
404  /* TCON controls the timer itself (turns it on/off). */  
405  
406  /* TMOD and SMOD controls baud rate? */  
407  
408  /* FCOW = DOUBLE_BAUD_RATE; */  
409  
410  /* Set Power Control Mode register. */  
411  
412  /* TH1 = baud_rate; */  
413  
414  /* Set TIMER 1 re-load value for baud. */  
415  
416  /* TMOD = TIMER_1_MODE_2; */  
417  
418  /* TMOD = TIMER_1_MODE_2; */  
419  
420  /* SCON = SERIAL_PORT_MODE_1; */  
421  
422  /* SCON = SERIAL_PORT_MODE_1; */  
423  
424  /* TR1 = 1; */  
425  
426  if(defined(INMAT))

427  /* Initialize 8250, see "Technical Reference Personal Computer AT". */  
428  
429  /* Set baud rate (divisor latches). */  
430  
431  /* DELAB must be to write baud rate divisor latches. */  
432  
433  /* Parameter baud_rate is ignored. */  
434  
435  outport(CC_LCR, CC_DLAD1);  
436  outport(CC_DLAD2, baud_rate >> 8);  
437  outport(CC_DLAD3, baud_rate & 0x0ff);  
438  outport(CC_DLAD4, baud_rate & 0xff);  
439  
440  /* Set serial port operational parameters. */  
441  
442  /* DELAB must be to write/read LCR, IER, and data registers. */  
443  
444  outport(CC_LCR, CC_DLAD0 | parity | word_len | stop_bit);  
445  
446  /* Disable all 8250 interrupts on this port. */  
447  
448  outport(CC_IER, CCDISABLE_INT7);  
449  
450  /* Clear line status and modem status registers. */  
451  
452  inpport(CC_LSR);  
453  inpport(CC_MSR);  
454  
455  /* Clear chars from receive register. */  
456  
457  /* Line status register (bit 0) indicates if data ready. */  
458  
459  */
```c
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_WAIT_FLAG;
  return;
}
}

#endif defined(SIOCB)

/* Set 8255 baud rate divisor (and CC SELECT). */
/* Set 8255 UART control register. */
/* Ret-set 8255 modem control register. */
outp(port+CC_BSR, baud_rate);
outp(port+CC_HCR, parity | word_len | stop_bit);
outp(port+CC_MCR, CC_MCR_RESET);
#endif

/******************************************************************************
** sio_putchar **
*******************************************************************************/

typedef sio_putchar(s)

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(180152) || defined(18031)
    #if defined(MUART_POLLED)
      while (MUAR_MSR & MUAR_TBE_MASK) != MUAR_TBE_MASK;
    #else
      #if defined(MUART_TBE)
        while (MUAR_MSR & MUAR_TBE_MASK) != MUAR_TBE_MASK;
      #else
        while (sio_buffer.full);
      #endif
    #endif
  else
    #if defined(MUART_TBE)
      while (MUAR_MSR & MUAR_TBE_MASK) != MUAR_TBE_MASK;
    #else
      #if defined(MUART_TBE)
        while (sio_buffer.full);
      #endif
    #endif
  
  /* Local and auxiliary serial channels are managed differently. */
  if (port == ASC)
    #if defined(MUART_POLLED)
      while (MUAR_MSR & MUAR_TBE_MASK) != MUAR_TBE_MASK;
    #else
      #if defined(MUART_TBE)
        while (MUAR_MSR & MUAR_TBE_MASK) != MUAR_TBE_MASK;
      #else
        while (sio_buffer.full);
      #endif
    #endif
  else
    #if defined(MUART_TBE)
      while (MUAR_MSR & MUAR_TBE_MASK) != MUAR_TBE_MASK;
# sio.c

/* Wait until there is a character available to input. */
/* Get character from port. */
/* Return input character (as a byte). */
/* Assume function successful... */
if defined(180159) || defined(18031)
/* Local and auxiliary serial channels are managed differently. */
if (port == ASCII)
    #if defined(MUART_POLLED)
        while ((MMUART_MSR & MMUART_RBF_MASK) != MMUART_RBF_MASK); return(MMUART_RRR);
    else
        return(RCBUFFER.TAIL != FALSE);
    #else
        while( (RCV_BUFFER защиты (RCV_BUFFER.front = RCV_BUFFER.front) 
        1. if RCV_BUFFER.front == RCV_BUFFER.rear) RCV_BUFFER.empty = TRUE; 
        return(0); 
    endif

    endwhile
    while (RI != 0); /* Wait for receive interrupt flag to set. */ 
    RI = 0; /* Clear interrupt flag for next recev. */
    return(SBUF);
    #else
definition(SBCB)
/* 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 = inpport(CC_USR);
if (new USR = 0)
    if ((old USR & CC DR MASK) == CC DR MASK)
        old USR = 0;
    return(inpport(PIR RR));
    else
        while ((inpport(PIR USR) & CC DR MASK) != CC DR MASK); 
        return(inpport(PIR RR));
    else if (new USR & CC DR MASK == CC DR MASK)
        return(inpport(PIR RR));
    endif
#endif
while ((inpport(PIR USR) & CC DR MASK) != CC DR MASK); 
return(inpport(PIR RR));
#endif
endif
}
### sio.c

**Page 11**

```c
731 if (byte_avail)
732    return(TRUE);
733 else
734    new_usr = inp(port,CC_USR);
735    if (!old_usr && CC_DR_MASK == CC_DR_MASK)
736       byte_avail = TRUE;
737    else
738        old_usr = new_usr;
739        byte_avail = (new_usr & CC_DR_MASK) ? TRUE : FALSE;
740    return(byte_avail);
741 }
742
744 #endif
745
746 /*******************************************/
747 /* Function: Outputs a NULL terminated string to the serial port. */
748 /* Input: */
749 /*          sio_putchar() */
750 /*           int port; */
751 /*           byte *s; */
752 /* */
753 void sio_putchar(port, s)
754 {
755    int port;
756    byte *s;
757
758    // Wait until there's enough room for the string to be inserted into queue. */
759
760    while room_in_queue;
761
762    /* Output string, byte by byte until NULL char. */
763
764    if (port == ASC)
765        /* While buffer is full do: */
766
767    if (s == sio_putchar_iterator)
768        room_in_queue = MAX_BUFFER_SIZE - sio_putchar_iterator + 1;
769    else
770        room_in_queue = sio_putchar_iterator;
771    // While room in queue < strlen(char*s) */
772    while room_in_queue < strlen(char*s))
773        /* Copy output string to transmit queue (save original string ptr). */
774
775    step = s;
776    while(*s)
777
778    /* Set sio_error to SIO_ERR_STRING_LENGTH if the number of */
779    /* bytes to output is less than or equal to 0. */
780    /* Output: Nothing. */
```

### sio.c

**Page 12**

```c
804 /* If transmit buffer was empty, move byte to transmitter (gen INT); */
805 /* Buffer can't be empty since we just added a byte. */
806
807 if (xmt_buffer.empty)
808    xmt_buffer.empty = FALSE;
809    NTM_TRM = *step;
810
812 else
813    xmt_buffer.empty = FALSE;
814    return;
815
816 //send
817
818 while(*s) sio_putchar(port, *s);
820 }
821
822
823 /* Function: Inputs a NULL terminated string from the serial port. */
824 /* Input: */
825 /*          sio_getstr() */
826 /*           int port; */
827 /*           int port; */
828 /*           byte *s; */
829
830 if (port == ASC)
831    /* Stuff chars into s, using s as string pointer. */
832    /* Keep track of string length in s, return 1. */
833    /* Assume function successful... */
834
835 s = sio_getbyte(port);
836 ++*s;
837 while(*s++);
838 return(1);
839
840
842 int sio_getchar(port, s)
843 int port;
844 byte *s;
845 {
846  
847  
848  
849  
850  
851  
855  
857  
858  
859  
860  
861  
862  
863  
864  
865  
866  
867  
868  
869  
870  
871  
872  
873  
874  
875  
876  
```
void sio_putnstr(port, numbytes, s) 

int port;
int numbytes;
byte *s;

byte *steppy;
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;

/* 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 < numbytes);
}

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

steppy = s;

while(numbytes--)

  xmt_buffer.item[xmt_buffer.rear] = *steppy;

  xmt_buffer.rear++; 

  if (xmt_buffer.empty)

    xmt_buffer.empty = FALSE;

else 

  xmt_buffer.empty = FALSE;

return;

/* Function: Handles transmit/receive serial interrupts from the 8256. */

/* Assumes the use of the CP-31/535, and that serial rcv */

and xmt interrupts of the 8256 peripheral IC (level 14/15) 

are connected to the INT0 interrupt input of the 80535.

This requires jumping the bottom two pins of M3 on the 

cp-31/535 which connects 8256 INT to 80535 EXT INTO input. 

Serial INPUT interrupts (for char 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
	*ino_out_queue : module SIO.C
	sio_out_queue : module SIO.C

txt_history: 05/21/91 - Written by Robin T. Laird.

--------------------------------------------------------------------------------
xm_buffer.empty = TRUE;
else
    UART_TXR = xm_buffer.item[xm_buffer.front];
}

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

UART_CMD_3 = UART_EOI;
}

#endif
```c
I #ifndef IBBMAT
2 #include <req51.h>
3 #endif
4 #include <stdio.h>
5 #include <sio.h>
6
7 static XDATA char spf[80]; /* Buffer for printf() calls. */
```
```c
#include <sysdefs.h>

#define AOK 1
#define TRUE 1
#define FALSE 0
#define YES 1
#define NO 0

#define SYS_MAX_PACKET_SIZE 256

#define XDATA xdata

typedef unsigned char byte;
typedef signed int word;
```
# Description: Makefile for the Modular Robotic Architecture (MRA).
# Makes the common local communications subsystem.
# Targets are available for the following systems/subsystems:
# 1cs - COM Local Communications Subsystem
# 1ib - 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: 1991 - Written by Robin T. Laird.

## RULES

### Control settings for Franklin 8031 development

```makefile
MK = e51
AS = a51
LJN = i51
OTOH = oh51
CFLAGS = -cd la db sb
ASFLAGS =
LFLAGS =
OFLAGS =
STARTUP = (C)\$L\crom.obj
CODESEG = 00000h
```

### Control settings for Microsoft MS-DOS development

```makefile
MSC = \c\r
MSAS = \r\nMSLINK = \\r\nMCSDFLGS = /c /01 /Z1 /Od
MSASDFLAGS =
MLNKDFLAGS = /c\r
LOADLIBS =
.obj : $(CC) $(CFLAGS)
.o51.obj : $(AS) $(ASFLAGS)
.obj.exe : $(LJN) $(STARTUP),%e TO 58 code $(CODESEG) xdata $(XDATASEG) linkref
.obj.hex : $(OTOH) $(OFLAGS)
```

## DEFINITIONS

### Project, system, and application level definitions

```makefile
PROJ = \r\nAPP = app
CON = com
```
/***************

BMF.H

***************

#define BMP_MODULE_CODE 3100
#define BMP_FULL_BUFFER 1+BMP_MODULE_CODE
#define BMP_EMPTY_BUFFER 2+BMP_MODULE_CODE
#define BMP_BOP_NOT_FOUND 3+BMP_MODULE_CODE
#define BMP_CRC_INVALID 4+BMP_MODULE_CODE
#define BMP_INVALID_LENGTH 5+BMP_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 pl);
static void buf_remove(buffer *b, lcd_packet pl);

endif
```c
#define buf_front(b) (4b.item[b.front])

#define buf_rear(b) (4b.item[b.rear])
```

```c
#define ROP_LENGTH 3 /* Beginning-of-packet len. */
#define ADDR_LENGTH 1 /* Source address len. */
#define CRC_LENGTH 2 /* Checksum (CRC) len. */

#define PACKET_OVERHEAD (ROP_LENGTH + ADDR_LENGTH + CRC_LENGTH)
#define MIN_BYTES_IN_PACKET (PACKET_OVERHEAD + ADDR_LENGTH + LENGTH)

#define MIN_BYTES_IN_PACKET (PACKET_OVERHEAD + ADDR_LENGTH + LENGTH)
#define MIN_BYTES_IN_PACKET (PACKET_OVERHEAD + ADDR_LENGTH + LENGTH)
```
```c
/* Globals: None. */

/* Edit History: 07/08/90 - Written by Robin T. Laird. */
03/08/90 - Fixed bug when first packet bad, Robin T. Laird. */

static int buf_full(b);

buffer *b;

156 { /* Assume function successful... */
157 return(b->full); }

164 */
165 */ Function that returns the boolean of whether or not the *
166 parameter buffer is empty (TRUE if so, FALSE if not). *
167 Operation depends upon the parameter buffer as follows:
168 If buffer b == nxt_buffer then just returns empty flag.
169 If buffer b == scr_buffer then returns whether packet avail. *
170
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 *
umber indentad 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.

178 */
179 */ Input: *
180 buf_empty( buffer *b; pointer to buffer structure. */
181 */ Output: Integer, TRUE if buffer EMPTY, FALSE if buffer not EMPTY. */
182 */ Globals: None. */
183 /* Edit History: 12/10/90 - Written by Richard P. Smurlo and Robin T. Laird. */
184 /* 03/08/90 - Fixed bug when first packet bad, Robin T. Laird. */

195 static int buf_empty(b)
196 buffer *b;
197 { /* BOP sequence index. */
198 int bopix; /* Buffer full flag. */
199 int bfull; /* Number currently in buffer. */
200 word bytes_in_buffer; /* Actual calculated packet length. */
201 word packet_length; /* Static buffer front and rear. */
202
203 /* Assume function successful... */
204 led_error = AOK;
205 */
206 /* Just return structure flag for transmit buffer. */
207 /* Otherwise, determine if there are enough bytes for an entire packet. */
208 if (b == nxt_buffer)
209 return(nxt_buffer->empty);
210 else
211 { /* Quick check to see if buffer REALLY empty. */
212
213 if ((b->front == b->rear) && (b->full)) return(TRUE);
214
215 /* Set local buffer front, rear, full variables. */
216 */
217 /* Interrupts are happening, we don't want our vars changing as we go. */
```

```
```c
/* b->front points to beginning of BOP. */
/* b->front points to first byte in packet (past BOP). */
/* bytes_in_buff holds num bytes currently in buffer. */
/* b->empty 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 (b->front < MAX_BUFFER_SIZE - LCD_LEN_POS) {
    return(TRUE);
} else {
    packet_length = b->item[b->front + LCD_LEN_POS];
}

/* Packet INVALID if num bytes in buff < packet_length + overhead. */
if (bytes_in_buff < packet_length + PACKET_OVERHEAD) {
    return(FALSE);
}

buf_clear()

/* Initialize 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: */
/* b->front points 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 1;

lcd_error = OK;

/* Process each of the packets in the buffer. */
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;

/* Insert beginning of packet (BOP) header into buffer. */
for (i = 0; i < 1 < 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);
```
```c
449 static void buf_remove(b, p)
450 {  
451   word i, length;
452   word crc_cal, crc_revd, crc_hi, crc_lo;
453   lcd_error = ERR_OK;  /* Assume function successful... */
454   /* Make sure buffer isn't already empty. */
455   if (buf_empty(b))
456     {  
457       lcd_error = ERR_EMPTY_BUFFER;
458       return;
459      }
460   /* Valid packet in buffer (buffer not empty), proceed to remove. */
461   for (i = 0; i < BOP_LENGTH; i++) buf_inc(b->front);
462   /* Discard and skip past BOP. */
463   if (b->front < MAX_BUFFER_SIZE - LCD_LEN_POS)
464     length = b->item[b->front + LCD_LEN_POS];
465   else
466     length = b->item[b->front + LCD_LEN_POS - MAX_BUFFER_SIZE];
467   /* Copy length number of bytes. */
468   for (i = 0; i < length; i++)
469     {  
470       p[i] = b->item[b->front];
471       buf_inc(b->front);
472     }  
473 }
```
#include <stdio.h>

#define MAX_BAUD_RATE 38400

int main(void)
{
    int baud_rate = MAX_BAUD_RATE;
    printf("The baud rate is: %d
", baud_rate);
    return 0;
}
```c
#include <sys.h>  /* DOS IBM-AT service definitions. */
#define _include <sys.h>  /* P031 and 80152 definitions. */
#endif

/* Private Variables: */
#define COM_PORT  /* Variable that indicates operational mode (serial or parallel). */
#define _set by the lcc_sio_init() routine (default is serial mode). */
static XDATA int lcc_lo_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. */
#endif

/* 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_DLLA 0x09  /* Divisor latch LSR. */
#define CC_DLLB 0x09  /* Divisor latch MSR. */
#define CC_IMR 0x0A  /* Interrupt enable register. */
#define CC_LCR 0x0B  /* Line control reg. */
#define CC_MCR 0x0C  /* Modem control register. */
#define CC_LSR 0x0D  /* Line status register. */
```

```
/* define CC_MSR 0x0E */ /* Mode status register. */
/* define CC_DATA_READY 0x01 */ /* LSR data ready bit. */
/* define CC_RECEIVE_ERRS 0x0E */ /* LSR error bits mask. */
/* define CC_TRANSMIT_HOLD_EMPTY 0x20 */ /* LSR holding register empty. */
/* define CC_DLBA 0x00 */ /* DLB set low (0). */
/* define CC_DLBA 0x00 */ /* DLB set high (1). */
/* define CC_DISABLE_INT 0x00 */ /* Disable all interrupts. */
/* define CC_ENABLE_IRQ 0x01 */ /* Enable receive interrupts. */
/* define CC_DISABLE_INT 0x01 */ /* Enable transmit interrupts. */
/* define CC_ENABLE_MASTER 0x04 */ /* Enable line status interrupts. */
/* define CC_MODEM_BUS_ENABLE 0x0B */ /* DTR, RTS, and OUT active. */
/* define LINE_STATUS_ERR_INT 0x06 */ /* 8259 Interrupt controller (IC) register (absolute addresses). */
/* define IC_CM2 0x02 */ /* Operation control word 2. */
/* define IC_CM1 0x02 */ /* Operation control word 1. */
/* define IC_ENABLE_IRQ3 0x07 */ /* Enable int on IRQ3. */
/* define IC_ENABLE_IRQ4 0x0E */ /* Enable int on IRQ4. */
/* define IC_EOI 0x00 */ /* Non-specific end-of-interrupt. */
/* define IC_EOI */ /* Global that can be changed by user as desired BEFORE call to lcc_int(). */
/* define IC_COM PORT = COM; */
/* define IC_BAUD_RATE = 8015200; */
/* define lcc_sio_init() */
/* void (interrupt far *lcc_irq3_vector()); */
/* void (interrupt far *lcc_irq4_vector()); */
/* else */
/* Switch select bits for the ICH and MPU (80152 and 8031). */
/* Define I/O mode (parallel/serial select) special function bit variable. */
/* sbit PSEL = P1-6; */
/* sbit baud rate select special function bit variables. */
/* sbit BRS2 = P1-5; */
/* sbit BRS1 = P1-4; */
/* sbit BRS2 = P1-3; */
/* Define special function register bit/byte variables for LPC. */
/* define PIO 0x */ /* Parallel I/O (mt/car) data register. */
/* sbit PDIR = P5-3; */ /* Parallel data direction (0=IN, 1=OUT). */
/* sbit RST0 = P5-4; */ /* Request to send output (active low). */
/* sbit RST 0 = P5-4; */ /* Clear to send input (active low). */
/* sbit TS1 = P5-6; */ /* Request to send input (active low). */
/* sbit TS1 = P5-6; */ /* Clear to send output (active low). */

/* xptr_lo_offset */
/* * Input: xptr_lo_offset */
```
```c
/*
 * xdata void *xp; XDATA pointer.
 * Output: Returns the low-offset portion of the external data pointer.
 * Globals: None.
 */
define xptr_lo_offset(xp) ((unsigned int)xp<<0x0000)

/*
 * 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(xp); XDATA pointer.
 * Output:  Returns the low-offset portion of the external data pointer.
 * Globals: None.
 */
define xptr_hi_offset(xp) (((unsigned int)xp)>>0x0000)

/*
 * Function: Determines the /O mode for external communications.
 * For the 80152:
 * The mode is read from a switch connected to P1 bit 6 of the
 * 80152 parallel port. Low (0) selects serial mode, while high (1)
 * selects parallel mode. The mode bit corresponds to P/S SEL on the
 * ICM schematic.
 */

/*
 * Function: Determines the baud rate for external serial communications.
 * For the 80152:
 * The baud rate is read from a switch connected to P1 of
 * the 80152 parallel port. Bits 3, 4, and 5 select one of
 * eight available baud rates from 300 baud to 38400 baud.
 * Bit 3 corresponds to BS12 on the ICM schematic, 4 is BS11,
 * and bit 5 to BS0.
 */

/*
 * Bit 3, 4, and 5 are decoded as follows (see also ICC.H):
 */

/*
 * If a 14 MHz CPU is being used (1B0152), 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 38300 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:   loc_baud();
 * Output:  Returns the baud rate code for external serial
 *          communications as given by the table above.
 * Globals: loc_error : module LOC.C
 *          8031 reg : module REG152.H
 *          loc_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 loc_baud()
 * { int baud_rate;
 *   loc_baud_rate = module MAIN.C
 *   if (defined(1B0152))
 *     switch((byte)8031<2 | (byte)8031<1 | (byte)8030)
 *     { case 0:  
 *       baud_rate = BR300; 
 *       break;
 *     /* Convert the value found to the proper baud rate... */
 */
```
Jan 22 1992 07:54:00 icc.c Page 5

Jan 22 1992 07:54:00 icc.c Page 6
#icc.c

```c
#define DOUBLE_BAUD_RATE 0x00 */ * PCON = 0b0xxxxx SMOD=0 */
#define DOUBLE_BAUD_RATE 0x40 /* PCON = 0b1xxxxx SMOD=1 */
#define LCD_MODE 0x01

#define void Icc_sys_init(mode, baud_rate)
#define int mode;
#define int baud_rate;

'byte' 1, temp;

#define lcd_error = AOK; /* Assume function successful... */
#define * Initialize the I/O channel depending upon the mode... */
#define * For serial mode operation; */
#define * For 8031 or 8051; */
#define * TIMER 1 is used in MODE 2 for variable baud rates. */
#define * TH1 sets TIMER 1 re-load value. */
#define * SCCON controls the serial port mode settings. */
#define * TMOD controls the timer mode settings. */
#define * TCON controls the timer itself (turns it on/off). */
#define * SMOD controls baud rate doubling depending upon CPU speed. */
#define * Clear serial buffer by reading data. */
#define * Configure 8250 with baud rate and serial params defined elsewhere. */
#define * Disable serial interrupts for now. */
#define * Clear all registers (MSR, LSR, and data buffer). */
#define * Save old interrupt vectors (so someone else can restore). */
#define * Save old 8259 interrupt control word (so someone else can restore). */
#define * Enable processor bus interrupts via the MODEM control register. */
#define * For parallel operations; */
#define * Set port direction to input. */
#define * Clear data control lines. */
#define * Clear data port. */
#define switch(mode)

/* case LCC_SIO_MODE: */
ICC_Lo_mode = LCC_SIO_MODE;

#define if defined(INBMT) */
#define * Disable interrupts while we're changing vectors, ports, etc. */
#define * disable(); */
#define * Initialize 8250, see "Technical Reference Personal Computer AT". */
#define * Set baud rate (divisor latches). */
#define * DELAB must be 1 to write to baud rate divisor latches. */
#define * Parameter baud_rate is ignored. */
#define output(ICC_com_port(CC_LCR, CC_DLBAR));
#define output(ICC_com_port(CC_DLLSM, ICC_baud_rate >> 8) & 0xFF); output(ICC_com_port(CC_DLLSM, ICC_baud_rate & 0xFF));
#define output(ICC_com_port(CC_LCR, CC_DLBAR));
#define output(ICC_com_port(CC_LCR, CC_DLBAR) | ICC_tty_control); */
#define * Disable all 8250 interrupts for now. */
#define output(ICC_com_port(CC_IE, CC_DISABLE_INT));
#define output(ICC_com_port(CC_IER, CC_DISABLE_INT));
#define * Clear line status and modem status registers. */
#define input(ICC_com_port(CC_LSR));
#define input(ICC_com_port(CC_MSR));
```

```c
/* Disable processor interrupts while we're changing things. */

ISR - 0;

PDIR = P10_INPUT; /* Parallel I/O direction. */
RTSD = 0; /* Data control lines... */
CST1 = 0;
RTSI = 0;
CST0 = 0;
P10 = 0; /* Parallel I/O data port. */

#endif
break;

default:
    led_error = ERR_IO_MODE;
    return;
}

/**********************************************************

Function: Set 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
                    lcc_packet p; pointer to receive packet (dest).
                    word length; size of receive packet (in bytes).

Output: Nothing.

Globals:  led_error : module LCD.C
          lcc_io_mode : module LCC.C
          RTSD : module REG8250_H
          lcc_com_port : module MAIN.C

Edit History: 03/09/91 - Written by Robin T. Laird.

/**********************************************************/

static void lcc_set_rcv_dst(p, length)
    lcc_packet p;
    word length;
{
    led_error = AOK; /* Assume function successful... */
}

/**********************************************************

Function: Set 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
                    lcc_packet p; pointer to transmit packet.
                    word length; size of receive packet (in bytes).

/**********************************************************

Output: Nothing.

Globals:  led_error : module LCD.C
          lcc_io_mode : module LCC.C
          RTSD : module REG8250_H
          lcc_com_port : module MAIN.C

Edit History: 03/09/91 - Written by Robin T. Laird.

/**********************************************************/

static void lcc_start_rcv()
    lcc_packet p;
    word length;
{
    led_error = AOK; /* Assume function successful... */
}

/**********************************************************

Output: Nothing.

Globals:  led_error : module LCD.C
          lcc_io_mode : module LCC.C
          RTSD : module REG8250_H
          lcc_com_port : module MAIN.C

Edit History: 03/09/91 - Written by Robin T. Laird.

/**********************************************************/

static void lcc_start_xmt()
    lcc_packet p;
    word length;
{
    led_error = AOK; /* Assume function successful... */
}
```

```c
Jan 22 1992 07:54:00  icc.c  Page 11
711 case ICC_SIO_MODE:
712 PS: ;   /* Enable serial interrupts. */
713 break;
714 case ICC_PIO_MODE:
715 EX0 = 1; /* Enable external interrupt 0. */
716 EX1 = 1; /* Enable external interrupt 1. */
717 break;
718 default:
719 break;
720 #endif
721
722 /*****************************************
723 * Function: Initiates transmission of data packets.
724 * Enables the ICC/LPC interrupt service routines.
725 * Input:   icc_start_xmt();
726 * Output:  Nothing.
727 * Globals: led error : module LCD.C
728 ICC.io mode : module ICC.C
729 8031 Iregs : module REG152.H
730 ICC_com_port : module MAIN.C
731 ** Edit History: 07/10/90 - Written by Richard P. Smurlo and Robin T. Laird.
732 03/14/91 - Added code for IBM-AT, Robin T. Laird.
733 *****************************************************
734
735 static void icc_start_xmt()
736 {    /* Assume function successful... */
737    led error = AOK;    /* Assume function successful... */
738    /* Enable processor interrupts according to operational mode. */
739    if defined(1MMAT)
740    /* Enables interrupts on COM port IRQ 0 a bit to enable IRQ Int. */
741    /* Assumes that COM1/COM3 on IRQ4 and COM2/COM4 on IRQ3. */
742    switch(icc_com_port)
743 {    /* Enable transmit interrupts on the 8250. */
744    case COM1:
745    case COM3:
746    output(IC_OW1, inp(IC_OW1) & IC_ENABLE_IRQ4);
747    /* Call interrupt routine to start transmissions. */
748    break;
749    case COM2:
750    case COM4:
751    output(IC_OW1, inp(IC_OW1) & IC_ENABLE_IRQ3);
752    /* Call interrupt routine to start transmissions. */
753    break;
754    default:
755    break;
756    }
757
758 /*****************************************
759 * Function: Disables the ICC/LPC receiver by turning off the enable bit.
760 * Any incoming packet is dropped.
761 * Input:   icc_stop_rev();
762 * Output:  Nothing.
763 * Globals: led error : module LCD.C
764 ICC.io mode : module ICC.C
765 8031 Iregs : module REG152.H
766 ICC_com_port : module MAIN.C
767 ** Edit History: 07/10/90 - Written by Richard P. Smurlo.
768 03/14/91 - Added code for IBM-AT, Robin T. Laird.
769 *****************************************************
770
771 static void icc_stop_rev()
772 {    /* Assume function successful... */
773    led error = AOK;
774    /* Disable processor interrupts according to operational mode. */
775 /*****************************************
```
# ifndef(Eโรค

/* Disable receive and line status interrupts on the 8250. */

output(&ccm_port<cc1> & input(&ccm_port<cc1>) & -CC_ENABLE_RECV & -CC_ENABLE_STAT);

/* Disable IRQ3 and IRQ4 interrupts on 8259. */

output(&ccm_port<cc1>);  // Restore old 8259 interrupt control bit mask.

switch(&ccm_port<cc1> {  // Disable serial interrupts.

  case &CC_SIO_MODE:  // Disable serial interrupts.

    ES = 0;

    break;

  case &CC_PIO_MODE:  // Disable external interrupt 0.

    EX0 = 0;

    break;

  default:  // Disable external interrupt 1.

    break;

  }  // Restore old 8259 interrupt control bit mask.

// Disable serial interrupts.

switch(&ccm_port<cc1>)

  // Function: Enables processor interrupts for the 8031/8052.

  Input: &ccm_enable_interrupts();

  Output: Nothing.


    &ccm_port<cc1>;  // Edit History: 03/19/91 - Added code for IBM-AT, Robin T. Laird.

// Function: Disables the LCC/LPC transmitter by turning off enable bit.

  Any outgoing packet is terminated.

  Input: &ccm_stop_meter();

  Output: Nothing.

  Global:  // Edit History: 07/10/90 - Written by Richard P. Smurlo.

    &ccm_port<cc1>;  // 03/14/91 - Added code for IBM-AT, Robin T. Laird.

// Function: Processes a valid serial interrupt (ESV) from the LCC.

  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 occurred and proceeds accordingly.

  This interrupt routine services the global ccc buffer. As bytes are received, they are placed in the next position in the receive buffer, and the next 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.

// Function: Enables processor interrupts for the 8031/8052.

  Input: &ccm_enable_interrupts();

  Output: Nothing.


    &ccm_port<cc1>;  // Edit History: 03/19/91 - Added code for IBM-AT, Robin T. Laird.

  // Function: Disables the LCC/LPC transmitter by turning off enable bit.

  Any outgoing packet is terminated.

  Input: &ccm_stop_meter();

  Output: Nothing.

  Global:  // Edit History: 07/10/90 - Written by Richard P. Smurlo.

    &ccm_port<cc1>;  // 03/14/91 - Added code for IBM-AT, Robin T. Laird.

// Function: Processes a valid serial interrupt (ESV) from the LCC.

  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 occurred and proceeds accordingly.

  This interrupt routine services the global ccc buffer. As bytes are received, they are placed in the next position in the receive buffer, and the next 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.
```c
/* 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)
  rcv_buffer.rear = b;
  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 beginning of buffer. */
  /* Increment front pointer. */
  /* Buffer can't be full since we just transmitted a byte. */
  /* Reading IIR or writing to transmit buffer req 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;

break;
```
case LINE_STATUS_ERR_INT:
    if (inp(COM2+CC_LSR) & CC_DATA_READY)
        inp(COM2+CC_RBR);
    *errcrt;
    break;

case RCV_DATA_AVAIL_INT:
    b = inp(COM2+CC_RBR);
    if (inp(COM2+CC_LSR) & CC_RECEIVE_ERRS) && rcv_buffer.full)
        rcv_buffer_item[rcv_buffer.rear] = b;
        buf_inc(rcv_buffer.front);
    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;
    break;

static void interrupt for icc_sio3_int() {
    /* Interrupt occasioned on COM4.
    */
    switch(inp(COM3+CC_LSR)) {
        case MODEM_INT:
            inp(COM3+CC_LSR);
            break;

        case LINE_STATUS_ERR_INT:
            if (inp(COM3+CC_LSR) & CC_DATA_READY)
                inp(COM3+CC_RBR);
                *errcrt;
            break;

        case RCV_DATA_AVAIL_INT:
            b = inp(COM3+CC_RBR);
            if (inp(COM3+CC_LSR) & CC_RECEIVE_ERRS) && rcv_buffer.full)
                rcv_buffer_item[rcv_buffer.rear] = b;
                buf_inc(rcv_buffer.front);
            if (rcv_buffer.front == rcv_buffer.rear) rcv_buffer.full = TRUE;
            break;
case XMT_FMPY_INT:
  default:
    if ((__com4 & CC_TRANS_HLD) & !xmt_buffer.empty)
      xmt_buffer.full = FALSE;
    outp(COM4 & CC_TRNS, xmt_buffer.item(xmt_buffer.front));
    buf inc(xmt_buffer.front);
    if (xmt_buffer.front == xmt_buffer.rear) xmt_buffer.empty = TRUE;
    else
      while ((__com4 & CC_TRNS) != 1);
      output(COM4, IC_TRNS);

#line 1334
static void lcc_pio_int() interrupt 4 using 2
/* Determine which interrupt occurred (receive or transmit). */

if (RI)
  /* For 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 (!xmt_buffer.full)
    xmt_buffer.item(xmt_buffer.rear) = SBUF;
    buf inc(xmt_buffer.rear);
    if (xmt_buffer.front == xmt_buffer.rear) xmt_buffer.full = TRUE;
    else
      else
        if (TI)
          /* For valid transmit interrupt: */
          /* Clear transmit interrupt flag. */
          /* Make sure buffer is 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;
            else
              #endif
              #if defined(180152)
              1401 static void lcc_pio_int() interrupt 0 using 2
              1404 }
              1406 }
              1407 #endif
              1388 /* Both parallel input and output requests are processed via */
              1389 /* this interrupt handler. */
              1390 /* */
              1391 /* Input: lcc_pio_int() interrupt; */
              1392 /* */
              1393 /* Output: Nothing. */
              1394 /* */
              1395 /*Globals: None. */
              1396 /* */
              1397 /* Edit History: 07/10/90 - Written by Richard P. Smuro. */
              1398 /* */
              1399 
              1400 #if defined(180152)
              1401 static void lcc_pio_int() interrupt 0 using 2
              1404 }
              1406 }
              1407 #endif

A-55
*/
* FILENAME: lcd.h
* 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
* functions. Contains the function prototypes for the LCD
* module.
* Module 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-w for more information.
* Edit History: 08/14/90 - Written by Robin T. Laird.
*/

/* Public Data Structures: */

#define LCD_MODULE_CODE 3000
#define LCD_ERR_NUM_INIT 1+LCD_MODULE_CODE
#define LCD_ERR_PACKET_LENGTH 2+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_SEND Whit 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_cccg_err_cnt;
                    word r_bop_err_cnt;
                    word x_valid_cnt;
                    word x_err_cnt;
               } lcd_state;

/* External module global error variable. */
/*
 * 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 B00 LCD and
 * the ROC12 LCD/LPC - Local Serial/Parallel Channel).
 *
 * Message format at this level (ISO OSI data link layer) is:
 * 1 byte 0 | byte 1 | byte 2 | byte 3 | byte 4 | byte 5 | ....
 * 1 | DST | LENGTH | SOURCE | 0000 | 0000 | ....
 * 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 BMP.h and BMP.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: 03/18/90 - 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: */

#define Global module error variable, lcd_error.
#define Global module state variable, lcd_state.
#define Global module state variable, lcd_state.
#define Global module state variable, lcd_state.
#define Buffer management functions should be included here.
#define Buffer management functions.
#define Low-level data link layer support functions should be included here.
#define Local communications.
#define Main function.

/* Function: Initializes the Local Communications Device Handler.
 * Clears the transmit and receive buffers, obtains the local
 * communications channel 1/0 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.
 */

void lcd_init()
{
    lcd_error = 0;
    /* 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.crc_err_cnt = 0;
    lcd_state.r_hap_err_cnt = 0;
    lcd_state.s_valid_cnt = 0;
    lcd_state.s_err_cnt = 0;
    /* Initialize the transmit and receive buffers. */
    buf.clear(&tmt_buffer);
    buf.clear(&rcv_buffer);
    /* Get the 1/0 mode and baud rate (if applicable) for this system. */
    /* Pass to LCC initialization function (which sets up 1/0 channel). */
    /* No errors are currently fatal (they are only warnings). */
    lcc_err_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_snd();
    /* Enable interrupts... */
    lcc_enable interrupts();
}


/* Function: Performs a soft reset of the LCD system.
 * The receive and transmit buffers are cleared and the
 * receive and transmit destination and source addresses
 * for the LCC 1/0 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.
 */

void lcd_reset()
{
    /*...*/
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.i_valid_cnt = 0;
    lcd_state.r_valid_cnt = 0;
    lcd_state.r_error_cnt = 0;
    lcd_state.x_valid_cnt = 0;
    lcd_state.x_error_cnt = 0;
    /* Re-initialize the transmit and receive buffers. */
    buf_clear(tmt_buffer);
    buf_clear(rcv_buffer);
}

void lcd_enable()
{
    lcd_enable();
    Output: Nothing.
    /* 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
    rcv_buffer : module BMF.C
    Edit History: 07/09/90 - Written by Robin T. Laird.
}

void lcd_disable()
{
    lcd_disable();
    Output: Nothing.
    /* 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();
    /* If we don't want to wait (LCC 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 (LCC WAIT FOREVER): */
    /* Wait until there is a packet in the buffer, then remove it. */
    /* This function WILL wait forever, i.e., indefinitely. */
    /* If we want to wait for a period of time: */
    /* Set up and zero auxiliary 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);
    }
```c
797    else if (retry == LCD_WAIT_FOREVER)
798        { 
799            while (!buf_empty(&recv_buffer)) 
800                buf_remove(&recv_buffer, pl);
801        } 
802        else if (retry <= LCD_MAX_ATTEMPTS) 
803        { 
804            while (!buf_empty(&recv_buffer)) 
805                if (retry-- == 0) 
806                    { 
807                        lcd_error = LCD_ERR_RECEIVE_PACKET;
808                        return;
809                    } 
810                buf_remove(&recv_buffer, pl);
811        } 
812        else 
813            { 
814                lcd_error = LCD_ERR_NUM_ATTEMPTS;
815            } 
816            /* If an error occurred while removing packet from buffer, log type. */
817            /* If no error occurred, log successful reception. */
818            if (lcd_error == ERR_BDP_NOT_FOUND) 
819            { 
820                lcd_state.r_bdp_err_cnt++;
821                lcd_state.r_err_cnt++;
822            } 
823            else if (lcd_error == ERR_CRC_INVALID) 
824            { 
825                lcd_state.r_crc_err_cnt++;
826                lcd_state.r_valid_cnt++;
827            } 
828            else if (lcd_error == LCD_ERR_RECEIVE_PACKET) 
829            { 
830                return;
831            }
832            else 
833            { 
834                lcd_insert(&xmt_buffer, pl);
835                if (lcd_error != LCD_OK) 
836                { 
837                    lcd_error = LCD_ERR_TRANSMIT_PACKET;
838                    return;
839                }
840            }
841            /* At a later date, it will be possible to add a packet to the transmit */
842            /* buffer and then return to the calling function immediately, the packet */
843            /* would be transmitted when it moved to the front of the buffer. */
844            /* A retry of zero would indicate that the packet is to be transmitted */
845            /* in the above manner (i.e., don't wait for packet to be transmitted). */
846            /* For now, always wait for completion. */
847            if (retry == LCD_WAIT_FOREVER) 
848            { 
849                lcd_error = LCD_ERR_TRANSMIT_PACKET;
850                return;
851            }
852            else if (retry == LCD_MAX_ATTEMPTS) 
853            { 
854                while (!buf_empty(&xmt_buffer));
855            } 
856            else 
857            { 
858                lcd_insert(&xmt_buffer, pl);
859                if (lcd_error != LCD_OK) 
860                { 
861                    lcd_error = LCD_ERR_TRANSMIT_PACKET;
862                    return;
863                }
864            }
865        }
866/* Function: Adds the parameter packet to the transmit buffer if possible. */
867/* If the transmit buffer is full an error is generated. */
868/* Otherwise, if the buffer is empty, the packet is transmitted immediately, and the packet information is inserted into the */
869/* transmit buffer. Depending upon the retry value, the */
870/* function will perform as follows: */
871/* LCD_WAIT_FOREVER : return immediately, pkt added to buffer. */
872/* LCD_DONT_WAIT : retry retry times to transmit packet. */
873/* Currently, only the LCD_WAIT_FOREVER option is supported. */
874/* This is equivalent to a one packet deep transmit buffer. */
875lcd_transmit_packet(p, retry)
876    } 
877    /* Output: */
878    lcd_transmit_packet(p); 
879    /* word retry: */
880    number of times to try transmitting packet. 
881    
882    *Globals: *
883    *lcd_error : module LCD-C
884    *xmt_buffer : module BMC-C
885    *lcd_state : module LCD-C
886    *Edit History: 07/06/90 - Written by Robin T. Laird.
887    */
888
889void lcd_transmit_packet(p, retry)
890    lcd_packet p;
891```
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:
led_status:
led_state *s pointer to module state structure,
Output:
Nothing,
Globals: led_error : module LCD.C
Edit History: 07/09/90 - Written by Richard P. Smurlo.

```
void led_status(s)
  {
    led_error = OK;       /* Assume function successful... */
    *s = _led_state;
  }
```
/* LCI.h */

#define LCI_MODULE_CODE 4000
#define LCI_ERR_NOT_INIT 1 + LCI_MODULE_CODE
#define LCI_ERR_RECEIVE_MESSAGE (1 + 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

#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
/* 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:
                lcd_error = "LCI_ERR_NOT_INIT;
                break;
        default:
            lcd_error = lcd_error;
    }
}

/****************************************************************************

lci_message

**************************************************************************/

Function:  Receives the latest 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 failures is tracked so that if it
exceeds a maximum value over time, the LCD device handler
will reset to try and remedy the problem.

Input:    lci_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: 08/11/89 - Written by Robin T. Laird.

****************************************************************************/

#define MAX_RECV_ERRORS 1000 /* On > 1000 errors, reset LCD. */

void lci_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. */

    lci_message(m, retry);
}

void lci_init()
{
    lci_error = AOK;
        /* Assume function successful... */

    { lci_message: module LCI.C
      lci_error = module LCD.C
    }

    Edit History: 07/28/90 - Written by Robin T. Laird.

****************************************************************************/
/**
 Function: Send 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.

 Global:
 lci_error : module LCI,C
 lci_send_message(): module LCD,C

 Edit History: 06/11/90 - 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;
     lci_state status; /* Holds LCD status. */
     lci_error = ACK; /* Assume function successful... */

   /* Send packet. Iterate retry number of times. */
   lci_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 (lci_error != ACK)
       if (lci_status(status);
       if (error_cnt > MAX_SEND_ERRORS) lci_reset();
   lci_error = LCI_ERR_SEND_MESSAGE;

)}
Jan 22 1992 08:10:02 makefile

1 | MAKEFILE
2 | # Description: Makefile for the Modular Robotic Architecture (MRA).
3 | Makes the common method manager subsystem.
4 | Targets are available for the following systems/subsystems:
5 | - mma - Common Method Manager Subsystem
6 | - lib - Add modules to MRA library
7 | Notes: 1) The dependency and production rules are included here.
8 | 2) See also mma\makefile.
9 | Edit History: 03/22/91 - Written by Robin T. Laird.

25 | RULES
26 | .SUFFIXES: .obj .exp .obj .c .s51
27 | # Control settings for Franklin 8031 development
28 | FC = -S
29 | AS = -S51
30 | LINK = -S51
31 | OTOH = -S51
32 | CFLAGS = -wcd Id db ab
33 | ASFLAGS =
34 | FLAGS =
35 | OFLAGS = -a51
36 | STARTUP =-a51\crom.obj
37 | CODESEG = 00000h
38 | KDATASEG = 00000h
39 | # Control settings for Microsoft MS-DOS development
40 | MSC = -c1
41 | MSAS = -maxm
42 | MLSINK = \ link
43 | MSCLFLAGS = /c /01 /Z1 /vold
44 | MSASLFLAGS = /co
45 | LONDLINES =
46 | .c.obj: $(CC) $C(CFLAGS)
47 | .s51.obj: $(AS) - (ASFLAGS)
48 | .obj.exp: $(LINK) $(STARTUP), $< TO $@ code $(CODESEG) xdata $(KDATASEG) $ref
49 | .exp.hex: $(OTOH) $< $(OFLAGS)

DEFINITIONS

Project, system, and application level definitions

PROJ = mma
APPSYS = - app
COMSYS = - ccm

COM MMS DEPENDENCIES
Jan 22 1992 08:10:02  makefile

147 1MMDC $ (SYSDFS) $ (MMSSRC) \mm.h $ (MMSSRC) \mm.c $ (MMSSRC) /1mmdc.c
148
149 $(COMBIN152)/1mmdc.obj : $(1MMDC)
150 $(CC) $(MMSSRC)/*.c $(CFLAGS) df(180152) pr$(MMSSRC)/*.152 oj$(COMBIN152)
151 $(COMBIN3) /1mmdc.obj : $(1MMDC)
152 $(CC) $(MMSSRC)/*.c $(CFLAGS) df(18033) pr$(MMSSRC)/*.31 oj$(COMBIN3)
153
154 $(COMINMS) /1mmdc.obj : $(1MMDC)
155 $(MSC) $(MSCFLAGS) /DIBMAT /Fo$(COMINMS)/*.at /Fo$(COMINMS)/*.s $(MMSSRC)/*.s.
156
157 # COM System Method Manager library module dependencies
158
159 $(COMIN52)/$(mmlib.obj) : $(COMIN3) mmlib.c
160 $(COMIN3)/$(mmlib.obj) : $(COMIN3) mmlib.c
161 $(COMIN3)/$(mmlib.obj) : $(COMIN3) mmlib.c
162 $(COMIN3)/$(mmlib.obj) : $(COMIN3) mmlib.c
163 $(COMIN3)/$(mmlib.obj) : $(COMIN3) mmlib.c
164
165 $(COMIN3) /$(mmlib.obj) : $(COMIN3) mmlib.c
166 $(COMIN3) /$(mmlib.obj) : $(COMIN3) mmlib.c
167 $(COMIN3) /$(mmlib.obj) : $(COMIN3) mmlib.c
168
169 $(COMIN3) /$(mmlib.obj) : $(COMIN3) mmlib.c
170 $(MSC) $(MSCFLAGS) /DIBMAT /Fo$(COMINMS)/*.at /Fo$(COMINMS)/*.s $(MMSSRC)/*.s.
171
172 # COM Phone Book module dependencies
173
174 $(PB) $ (SYSDFS) $ (MMSSRC) \pb.h $ (MMSSRC) \pb.c
175 $(COMIN152)/$(pb.obj) : $(PB)
176 $(COMIN3)/$(pb.obj) : $(PB)
177 $(COMIN3)/$(pb.obj) : $(PB)
178 $(COMIN3)/$(pb.obj) : $(PB)
179 $(COMIN3)/$(pb.obj) : $(PB)
180 $(COMIN3)/$(pb.obj) : $(PB)
181 $(COMIN3)/$(pb.obj) : $(PB)
182
183 $(COMIN3)/$(pb.obj) : $(PB)
184 $(MSC) $(MSCFLAGS) /DIBMAT /Fo$(COMINMS)/*.at /Fo$(COMINMS)/*.s $(MMSSRC)/*.s.
185
186 # COM Method Manager module dependencies
187
188 $(MM) $(SYSDFS) $(LCSSRC) \lc1.h $ (MMSSRC) \mm.h $ (MMSSRC) \mm.c
189 $(COMIN152)/$(mm.obj) : $(MM)
190 $(COMIN3)/$(mm.obj) : $(MM)
191 $(COMIN3)/$(mm.obj) : $(MM)
192 $(COMIN3)/$(mm.obj) : $(MM)
193 $(COMIN3)/$(mm.obj) : $(MM)
194 $(COMIN3)/$(mm.obj) : $(MM)
195 $(COMIN3)/$(mm.obj) : $(MM)
196 $(COMIN3)/$(mm.obj) : $(MM)
197 $(COMIN3)/$(mm.obj) : $(MM)
198 $(COMIN3)/$(mm.obj) : $(MM)
199 $(COMIN3)/$(mm.obj) : $(MM)
200 $(MSC) $(MSCFLAGS) /DIBMAT /Fo$(COMINMS)/*.at /Fo$(COMINMS)/*.s $(MMSSRC)/*.s.
/**
 * LMM.H
 */

#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[];
```c
/* Local method manager function. */
/* Implements the MRM 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 MRM trigger table 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 MRM.C and is compiled there. */
/* It is assumed that the file LMM.A is included before it. */

/* Public Variables: */
/* Global module error variable, lmm_error. */
/* lmm_error contains code of last error occurrence. */
/* lmm_error should be set to AOK after each 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. */
/* lmm_process(); */
/* lmm_trigger(); */
/* lmm_method(); */
/* lmm_method(); */

/* define LMM_MAX_METHODS 10 /* Number of methods we can hold. */

typedef struct { /* Name data */
    unsigned long long_t; /* Word period; */
} lmm_method_t;

static XDATA int lmm_num_methods = 0;
static XDATA lmm_method_t lmm_method[LMM_MAX_METHODS];

void lmm_main_Init()
{ /* Initialize the Local Method Manager software subsystems. */
    /* This includes initializing the module-specific subsystems */
    /* via a call to the system function d_unit_reset(). */
    lmm_init();
}

* 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(); */
  lmm_message *lmm_in; /* Pointer to decoded message to process. */
  lmm_message *lmm_out; /* Pointer to message to output. */
  byte *status; /* Pointer to method execution status. */
}

* Output: 
  AOK. */

* Globals: 
  lmm_error : LMM.C */
  lmm_init(); */
  lmm_process(); */
  lmm_method(); */

* Edit History: 12/20/90 - Written by Robin T. Laird. */
```
```c
Jan 22 1992 08:04:08 Imm.c Page 3

/* The method would be looked up according to function and sequence #.*
147 if (period == 0)
148 {
149   imm_num_methods = (imm_num_methods ? imm_num_methods + 1 : 0);
150   return;
151 }
152
153 /* Add method info to trigger table. */
154
155 imm_trigger[imm_num_methods].method = *m_in;
156 imm_trigger[imm_num_methods].period = period;
157 imm_trigger[imm_num_methods].fire = period + rtc_time();
158 imm_num_methods++;
159 break;
160
161 /* Translate the message (activate function). */
162
163 /* Output message, m_out, contains translation information. */
164
165 /* Output message, m_out, contains any results (in the parameter field). */
166
167 imm_translate_message(m_in, m_out, status);
168
169 /* Generate any required response using data from input message. */
170
171 /* Output message, m_out, contains data concerning required response. */
172
173 imm_generate_message(m_in, m_out, status);
174
175 
176 /*********************************************************************************/
177 /*
178   Function: Checks local method trigger table and fires appropriate
179   functions according to conditions set up in the table.
180   The conditions are set by external commands received from
181   other modules. The conditions are checked as often as
182   possible for possible execution of a function.
183   Currently, only temporal conditions are implemented.
184   This allows for periodic execution of functions.
185   
186   Input:  imm_fire();
187   Output: Nothing.
188 /*********************************************************************************/
189 globals: imm_error = IMM_C;
190 immTrigger = IMM_C;
191 imm_num_methods = IMM_C;
192
193 /* Edit History: 03/27/91 - Written by Robin T. Laird. */
194 
195 */
196 #define IMM_SEND_ATTEMPTS LCI_WAIT FOREVER
197 void imm_fire()
198 { int j;
199   byte status;
200   m_in_message m_in_out;
201   m_out_message m_out;
202   
203   /* Assume function successful... */
204   /* Check number of methods in trigger table. If non-zero, continue. */
```
/* Otherwise we have to generate a response message... */

m_out = dest_modid = m_in = src_modid;

m_out = dest_unit_id = m_in = src_unit_id;

m_out = src_modid = m_in = dest_modid;

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;

m_out = trans_category = m_in = trans_category;

m_out = function_id = m_in = function_id;

/* Parameter length and parameter buffer are already in output message. */

/* Set status to indicate that a response message is required. */

status = MM_RESPONSE_REQUIRED;

imm_error = MM_ERROR;

} /* 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, imm_error, is set to IMM_ERROR_MSG_TRANSLATION if the local method failed.
 */

/* Input: */

imm_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:

imm_error : IMM_ERROR

imm_numfuncs : DICTIONARY

/* Edit History: 01/04/91 - Written by Robin T. Laird. */

****************************************************************************/
/* Local Method Manager (LMM) dictionary functions. 
   Implements the inherited ("built-in") local methods. 

   Module LMMDCCT.C exports the following variables/functions:
   
   int lmm_mm_funcs;
   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 Rebin T. Laird.

#include <sysdef.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 sprintf(mm stdout, "%s", v_obj_class);
  return(MM COMMAND EXECUTED);
}

int d obj _superclass()
{
  mm sprintf(mm stdout, "%s", v_obj superclass);
  return(MM COMMAND EXECUTED);
}

/* Unit class methods... */

int d unit _name()
{
  mm sprintf(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 MODMCC module dictionary (file). */

int reset();
int d unit _reset()
{
  v_unit reset = UNIT_IDLE;
  reset();
  return(MM COMMAND EXECUTED);

typedef mm_buffer;
int mm_errno;
int mm_putchar mm_stdin;
int mm_putchar mm_stdout;
int mm_init();
int mm_cycle();
int mm_encode_message();
int mm_decode_message();
int mm_sprintf();
int mm_scannfl();
int mm_service_event();
int mmカードローン();
int mm_check_event();

/* Note: 1) See SDP pp. 5-6 through 5-9 for more information. */

/* Edit History: 10/01/89 - Written by Robin T. Laird. */
```c
#include <string.h>
#include <sysdev.h>
#include <stdio.h>
#include <ctype.h>
#include <time.h>
#include "m.h"
#include "m.m"

#ifdef (DEBUG) /* Public Variables:

extern int mm_error;

define MM_ERROR_NEW_SUBSYSTEM "0x0F"

/* Position of period (in ms) in parameter field. */
define MM_PERIOD_POS 0

/* Length of period in bytes. */
define MM_PERIOD_LEN 0

/* Position of destination address in message. */
define MM_DEST_ADDR_POS 0

/* Number of overhead bytes at this communications level. */
define MM_COM_OVERHEAD_BYTES 0

/* Local method manager functions are included here. */
define MM_LOCAL_MANAGER_FUNCS 0

/* System method manager functions are included here. */
define MM_SYSTEM_MANAGER_FUNCS 0

#define MM_INIT (mm_init();)

extern MM_C

/* Definitions for field sizes/positions. Maintain with care! */
define MM_MODROT_ID_BIT_SHIFT 5
#define MM_UNIT_ID_MASK 0x01
#define MM_TRANS_DISP_IN_BIT_SHIFT 4

... (remains the same as the previous page) ...
```
Jan 22 1992 08:05:02 mm.c Page 3

#define MM_SFND_ATTEMPTS LCI_WAIT_FOREVER
#define MM_RECV_ATTEMPTS 8

void mm_cycle(iterations)
{
    int i, inc;
    byte status;
    lci_message l_in, l_out;
    mm_message m_in, m_out;

    if (iterations == MM_CYCLE_FOREVER) repeat forever (never return).

    if (iterations == MM_CYCLE_FOREVER) repeat forever (never return).

    else
        for (i=0; i < iterations; i++)
            lci_send_message(l_out, MM_SEND_ATTEMPTS);

    while ((i++) < iterations);

Jan 22 1992 08:05:02 mm.c Page 4

/* 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 -> ENCODE -> lci message out */
Jan 22 1992 08:05:02 mm.c Page 5

```c
void mm_encode_message(void, encoded)
{
    message *to encode;
    lc_message encoded;
    /* Inter-module message format (IMMF): */
    /* DEST */
    /* SORC */
    /* DEVICE MESS DEVICE SEQ TRANSC TRANSC FN PARAMS */
    /* ID LENGTH ID NUM DISPOSIT CATEGORY ID LENGTH PARAMS */
    /* */
    /* */
    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->xrc_smodbot_id = MM_MODBOT_ID_BIT_SHIFT;
    encoded[i++] = to_encode->xrc_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] = 1 = to_encode->parameter_length;
    for (j = 0; j < i; j++) encoded[i++] * to_encode->parameter[j];
}
```

Jan 22 1992 08:05:02 mm.c Page 6

```c
/* Function: decode_message */

int decode_message
{
    * Input: mm_decode_message
    * Output: mm_message decoded: pointer to message to decode (out).
    * Global: mm_error = MMLC.
    * Edit History: 12/20/90 - Written by Robin T. Laird,
    * */
    mm_message *to decode;
    lc_message decoded;
    i = 0;
    /* Inter-module message format (IMMF): */
    /* DEST */
    /* SORC */
    /* DEVICE MESS DEVICE SEQ TRANSC TRANSC FN PARAMS */
    /* ID LENGTH ID NUM DISPOSIT CATEGORY ID LENGTH PARAMS */
    /* */
    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->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++];
    for (j = 0; j < i; j++) decoded->parameter[j] = to_decode[i++];
    mm噼噼啪啪
    */
    */ 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 %d. 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 | 1 bit |
```
419 * \b byte 8 bits *
420 * \c char 8 bits *
421 * \d unsigned int 16 bits *
422 * \e int 16 bits *
423 * \f long int 32 bits *
424 * \g pointer 8 bits *
425 *
426 input: * mm_printb;
427 * \m \n pointer to output buffer (where data goes).
428 * \n \c *format: pointer to output format string.
429 * ...
430 *
431 *
432 * \ Output: Nothing.
433 *
434 * \Globals: mm_error : MM_C
435 *
436 /* Edit History: 12/20/90 - Written by Robin T. Laird. *
437 *
438 \ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/
439 *
440 void mm_printb(p, format, a1, a2, a3)
441 mm_buffer *p;
442 char *format;
443 double a1, a2, a3;
444 {
445 va_list ap;
446 char *c, *val;
447 int [v1];
448 long [v2];
449 ...
450 /* Initialize the variable argument macro pointers. */
451 va_start(ap, format);
452 /* Loop through the format string, process according to control chars. */
453 /* Accesses arguments using va_arg(), inc index and bit index accordingly. */
454 /* Index and bit index indicate number of bytes and bits in last byte. */
455 for (c = format; *c; ++c)
456 {
457 if (*c == \d)
458 {
459 if (p->bitindex)
460 {
461 p->index++;  
462 p->bitindex = 0;
463 }
464 p->buffer[p->index++] = *c;
465 continue;
466 }
467 switch(*c)
468 {
469 case \d: /* bit */
470 if (p->bitindex == 0) p->buffer[p->index] = 0;
471 /* Franklin and Microsoft differ in byte way bits are managed. */
472 /* If defined(MS DOS)
473 p->buffer[p->index] = (va_arg(ap, int) << p->bitindex);
474 else
475 p->buffer[p->index] = (va_arg(ap, byte) << p->bitindex);
476 /*endif*/
477 break;
478 case \f: /* byte */
479 case \a: /* char */
480 case \f: /* char */
481 if (p->bitindex)
482 {
483 if (p->bitindex == 0)
484 p->index++;  
485 p->bitindex = 0;
486 p->buffer[p->index++] = *c;
487 break;
488 }
489 case \a: /* byte */
490 case \d: /* int */
491 if (p->bitindex)
492 {
493 p->index++;
494 p->bitindex = 0;
495 p->buffer[p->index++] = a1;
496 break;
497 }
Jan 22 1992 08:05:02 mm.c Page 9

585 # Function: Provides formatted input of binary data as in scanf().
586 Used to extract data from the parameter passing portion of 
587 a message (i.e., the end of the message). Variables are 
588 input according to special % flags as in printf() but 
589 as binary values NOT ASCII. A maximum number of values can 
590 be passed as parameters and scanf is compiler dependent. The only 
591 "flags" that are supported indicate the types and length of 
592 parameters as in %s. The format is similar to scanf().
593
594 Only the type flags are currently supported.
595
596 Care must be taken to set/reset the buffer indexes on 
597 function calls. Otherwise the buffer may overflow...
598
599 The types currently supported are:
600
<table>
<thead>
<tr>
<th>Character</th>
<th>Type</th>
<th>Input Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>%d</td>
<td>byte</td>
<td>1 bit</td>
</tr>
<tr>
<td>%c</td>
<td>char</td>
<td>8 bits</td>
</tr>
<tr>
<td>%u</td>
<td>int</td>
<td>16 bits</td>
</tr>
<tr>
<td>%l</td>
<td>long int</td>
<td>32 bits</td>
</tr>
<tr>
<td>%p</td>
<td>pointer</td>
<td>8 bits</td>
</tr>
</tbody>
</table>

611 **Input**:

612 mm_scann() mm_scann()

613 mm_scann() mm_scann() mm_scann()

616 **Output**: Nothing.

617 **Globals**: mm_error mm_limit

618 Edit History: 1/20/90 - Written by Robin T. Laid.

621 

624 `**************************************************************************`

625 void mm_scann() .. format, a1, a2, a3)
626 char *format;
627 double a1, a2, a3;
630 va_list ap;
637 char *c, *sval;
633 int iival;
634 long lival;
638 /* Initialize the variable argument macro pointers. */
639 va_start(ap, format);
640 /* Loop through the format string, process according to control chars. */
641 /* Access arguments using va_arg(), inc index and bit index accordingly. */
642 /* Index and bit index indicate number of bytes and bits in last byte. */
643 for (c = format; *c; c++)
644 { if (*c != '*')
645    continue;
646 } switch(*t-c) {
651    case 'y': /* bit */
652    case 'y': /* bit */
656 } p->bitindex = 0;

Jan 22 1992 08:05:02 mm.c Page 10

658 p->index++;
659 }
660 break;
661 case 'b': /* byte */
662 case 'c': /* char */
664 if (p->bitindex)
665 {
666 p->index++;
667 p->bitindex = 0;
668 */ (va_arg(ap, byte*)) = p->buffer[p->index++];
670 break;
671 case 'r': /* unsigned int */
672 case 'd': /* int */
674 if (p->bitindex)
675 {
676 p->index++;
677 p->bitindex = 0;
678 iival = (int)p->buffer[p->index++];
679 iival = (int)p->buffer[p->index++];
680 iival = (int)p->buffer[p->index++];
681 */ (va_arg(ap, int*)) = iival;
682 break;
683 case 'l': /* long */
684 if (p->bitindex)
685 {
686 p->index++;
687 p->bitindex = 0;
689 iival = (long)p->buffer[p->index++];
690 iival = (long)p->buffer[p->index++];
691 iival = (long)p->buffer[p->index++];
692 iival = (long)p->buffer[p->index++];
693 iival = (long)p->buffer[p->index++];
694 */ (va_arg(ap, long*)) = iival;
695 break;
697 case 's': /* pointer (string) */
699 if (p->bitindex)
700 {
701 p->index++;
702 p->bitindex = 0;
703 sval = va_arg(ap, char*);
704 do 
705   *sval++ = (char)p->buffer[p->index];
706    while (p->buffer[p->index++]);
707 break;
708 default:
709 break;
711 }
712 }
713 */ Clean up after moving argument pointer. */
714 va_end(ap);
```c
mm.c

* Input:
*  mm_service_event(int event);
*  number of event to service.
*  mm_message *m_out; pointer to message holding response.
*  }
*
* Output: Nothing.
*
* Global:
*  mm_error : module MM.C
*  item : module MM.C
*  queue : module MM.C (side effect)
*  mm_error : module MM.C
*  mm_attd: module MM.C
*  }
*
* Edit History: 01/24/91 - Written by Robin T. Laird.
*
* \***************************************************************************/

void mm_service_event(int 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_WAITING_EVENT;

else
  if (event == queue)
    mm_remove_q(queue, m_out);
  else
    mm_remove_q(event, m_out);

/* Copy parameter information from message to standard input buffer. */

memcpy(mm_attd, input, m_out->parameter, (int)m_out->parameter_length);

mm_attd_index = 0;

mm_attd.bit_index = 0;

(void) mm_attd;

/** 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.
*
* Global:
*  mm_error : module MM.C
*  queue : module MM.C (side effect)
*  mm_error : module MM.C
*  }
*
* Edit History: 03/28/91 - Written by Robin T. Laird.
```
PB.H

* 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 EDS pp. 5-6 through 5-x for more information.

* Edit History: 02/04/91 - Written by Robin T. Laird.

/*****************************************************************************
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
```c
#include <string.h> /* Standard string functions. */
#include <sysdef.h> /* System constants and types. */
#include <rtc.h> /* Real-time clock functions. */
#include "mm.h" /* Method manager. */
#include "scrn.h" /* System screen 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 occurrence. */
/* Should be set to OK after each successful function call. */
/* Variable can be examined by other software after each function call. */

XDATA int pb_error = PB_ERR_HOT_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

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];

/* Function: Initializes the Method Manager Phone Book data structure. */

void pb_init()
{
    /* 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.
}

#endif
```

```
#include "pb.h"
#include <debug.h>

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];

/* Function: Initializes the Method Manager Phone Book data structure. */

void pb_init()
{
    /* 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.
}

#endif
```
```c
Jan 22 1992 08:07:26 pb.c  Page 3

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

/**
 * 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;

/** */

/** */

ifdef SMM_MODULE_CODE
#endif

define SMX_MODULE_CODE 10000

/* Public Data Structures: */

define smm_err_not_init 1+SMX_MODULE_CODE
#define smm_err_msg_translation 2+SMX_MODULE_CODE
#define smm_err_msg_generation 3+SMX_MODULE_CODE
#define smm_err_no_event_match 4+SMX_MODULE_CODE
#define smm_err_empty_queue 5+SMX_MODULE_CODE
#define smm_err_full_queue 6+SMX_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_file(void);
void smm_generate_message(char *char_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);

```
/* 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 { smm_message method; word period; unsigned long fixed; 1 smm_cond; static XDATA int smm_num_methods = 0; static XDATA smm_cond smm_trigger[SMM_MAX_METHODS];

int smm_error;
smm_init();
smm_process();
smm_generate_message();
smm_translate_message(); Notes: 1) The smm functions are implementation independent.
2) This module is NOT a standalone compilation unit.
3) It is included by the module MM.C and is compiled there.
It is assumed that the file MM.H is included before it.

Edit History: 17/70/90 - Written by Robin T. Laird.

/* Public Variables: */
/* Global module error variable, smm_error. */
/* smm_error contains code of last error occurrence. */
/* Should be set to 0x0 after each successful function call. */
/* Variables may be examined by other software after each function call. */
/* XDATA int smm_error = SMM_ERROR_NOT_INIT; */ Global module error variable. */
/* Local error values and list-specific literals. */
#define SMM_ERROR_OVERFLOW +SMM_MODULE_CODE
#define SMM_ERROR_INVALID_PTR +SMM_MODULE_CODE
#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 { smm_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;
static void smm_avail_init(p)
nodeptr *p;
{
nodeptr q;

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;
}

/* Function: Function that returns a pointer to the next available node.
   SMM_NULL_PTR is returned if the avail list is empty.
*/

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);
  }
}

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;
  }
}

static void smm_CLEAR_q(p)
nodeptr p;
{
  if (p == SMM_NULL_PTR)
  {
    smm_error = AOK;
  } /* Assume function successful... */
  p = SMM_NULL_PTR;
}

/* 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.*/
```c
#define ITEM_MAX_CAPACITY 100
#define ITEM_MIN_CAPACITY 10

int itemCompare(item_t *item1, item_t *item2)
{
    // Compare function for item_list
}

void itemListInitialize(item_list_t *itemList)
{
    // Initialize function for item_list
}

item_t* itemListFind(item_list_t *itemList, int itemID)
{
    // Find function
}

void itemListInsert(item_list_t *itemList, int itemID, int *info)
{
    // Insert function
}

void itemListRemove(item_list_t *itemList, int itemID)
{
    // Remove function
}

int itemListIsEmpty(item_list_t *itemList)
{
    // Check if item_list is empty
}
```
Jan 22 1992 08:08:12 smm.c Page 7

439 * Function:
440 * Locate 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.
441 * If matching event IDs are not found, then SMM_NULL_PTR is returned. Otherwise, the pointer to the node is returned.
442 * Input: smm_findin_q(address of queue), m
443 * Output: Nothing.
444 * Global: smm_error : module SMM.C

Jan 22 1992 08:08:12 smm.c Page 8

512 * Function: Retrieves an int value representing the sequence number.
513 * Global: avail : module SMM.C
514 * Edit History: 01/23/91 - Written by Robin T. Laird.
515
516 static int smm_next_event()
517 { return(available);}
518
519 /**************************************************************************/
520 static nodeptr smm_findin_q(p, m)
521 nodeptr p;
522 smm_message *m;
523 ]
524 } 1
525 if (p == SMM_NULL_PTR)
526 smm_error = SMM_ERR_INVALID_PTR;
527 return(SMM_NULL_PTR);
528 }
529 else
530 q = p->right;
531 if (q != NULL)
532 smm_error = SMM_ERR_Q is not valid;
533 return(SMM_NULL_PTR);
534 }
535
536 /**************************************************************************/
537 void smm_init()
538 {
539 smm_error = AOK; /* Assume function successful. */
540 /* Reset system method trigger table index to 0 (indicate table empty). */
541 smm_num_methods = 0;
542 /* Initialize the free node pool (avail list). */
543 smm_avail_init(savall);
544 /* Initialize the standard event queue. */
545 smm_clear_q(savall);
546 }
547
548 /**************************************************************************/
549 /* 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.
550 * Input: smm_process()
551 * Global: smm_error : module SMM.C

A-86
```c
#define SMM_SEND_ATTEMPTS LCI_WAIT_FOREVER

void smm_send amattempts
char *dest_unit_name; m_in, m_out, event
char *dest_unit_name;

smm_message *m_in, *m_out;
smm_message *m_in, *m_out;

void smm_process(m_in, m_out, status)
smm_process(m_in, m_out, status);

/* Translate incoming message as response to previous command. */
smm_message *m_in, *m_out, status;

smm_process(m_in, m_out, status); /* Assume function successful. */

/* Translation will try and associate sequence number with event number. */
smm_translate_message(m_in, m_out, status);

smm_translate_message(m_in, m_out, status);

/* 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_message ; module SMM.C
smm_num_methods ; module SMM.C
smm_trigger ; module SMM.C

Edit History: 02/28/91 - Written by Robin T. Laird.

Edit History: 12/20/90 - Written by Robin T. Laird.

---

 Jan 22 1992 08:08:12  smm.c Page 9
---

 Jan 22 1992 08:08:12  smm.c Page 10
---

```
Assume the following fields have been filled accordingly:

```c
int dest_modnet_id; /* destination network */
int src_modnet_id; /* source network */
int dest_unit_id; /* destination unit */
int src_unit_id; /* source unit */
int trans_sequence_number; /* sequence number */
int trans_disposition; /* disposition */
int param_unit_id; /* parameter unit */
int param_sequence_number; /* parameter sequence number */
int param_length; /* parameter length */
int param_type; /* parameter type */
int mm_send_message(uint8_t *msg, uint8_t msg_length, uint8_t event);
```

```
void mm_translate_message(uint8_t *msg, uint8_t msg_length, uint8_t event);
```

The function `mm_translate_message` translates (copies) the parameter input message into the system method queue at the position indicated by the `mm_send_message`. The method is then available for servicing by the application process (routine).

```c
if (mm_send_message(msg, msg_length, event)) {
    mm_translate_message(msg, msg_length, event);
}
```

`mm_translate_message` is defined as follows:

```c
void mm_translate_message(uint8_t *msg, uint8_t msg_length, uint8_t event);
```
# smmlib.c

/**
 * smmlib.c
 *
 * CPC1: 1ED90-MSA-COM-MMS-SMMLIB-C-RDCC
 *
 * 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;
    ssn_generate_message(NULLPTR, &m, &event);
    return(event);
}

/* Object superclass library functions */

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;
    ssn_generate_message(NULLPTR, &m, &event);
    return(event);
}

/* Unit class library functions... */

int unit_name(modbot, unit)
{
    int event;
    mm_message m;
    m.dest_modbot_id = modbot;
    m.dest_unit_id = unit;
}
makfile

CPCST: IED9G-MRA-ICN-AC-MAKEFILE.TXT-80COU

Description: Makefile for the Modular Robotical Architecture (MRA).

Targets are available for the following systems/subsystems:

1. IAC = ICA Applications Controller
2. IAC.hex = ICA Applications Controller Program (80152)
3. ICMON.hex = ICMON Network Monitor Program (80152)
4. print = Print ICM AC source files

Notes:
1) The dependency and production rules are included here.
2) See also PMakefile.

Edit History: 03/25/91 - Written by Robin T. Laird.


************** RULES **************

.SUFFIXES : .hex .exe .obj 

# Control settings for Franklin 8031 development
CC = cc
AS = cc
LINK = cc
OTOH = otoh
CFLAGS = -c -Ia -db sb
ASIFLAGS =
LFLAGS =
UFLAGS =
STARTUP = \$(M)\crom.obj

CSEG = 00000h
XDATASEG = 00000h

# Control settings for Microsoft MS-DOS development
MSC = c
MSAS = nasm
MSLINK = link
MSFLAGS = /A/S/c /O1 /ZI /Od
MSASFLAGS =
MSLINKFLAGS = /c
LOADLIBES =
.obj : 
$(CC) $< $(CFLAGS)

.s51.obj : 
$(AS) $< $(ASFLAGS)

.obj, .exe, .hex : 
$(LINK) $(STARTUP), $< TO 80 code $(CODESEG) \ndata $(XDATASEG) \xref

.o : 
$(OTOH) $< $(CFLAGS)


************** TARGETS **************

IAC : $(ICN815I2)\iac.hex $(ICN815I2_MON)\icmon.hex

$(ICN815I2)\iac.hex : $(ICN815I2)\iac
$(OTOH) \c exe $(58)

$(ICN815I2)\iac : $(IAC)
$(LINK) $(IACSRC) \iac.res
$(ICN815I2_MON)\icmon.hex : $(ICN815I2_MON)\icmon
$(OTOH) \c hex $(58)

$(ICN815I2_MON)\icmon : $(ICNMN)
$(LINK) $(IACSRC) \icmon.res

print : $(IAC)
-2p -f main.c \ post
-2p -f mra.h \ post
-2p -f mra.c \ post
touch print

************** ICM IAC DEPENDENCIES **************

makfile

1. COMSYS = com
2. ICN815I2 = icn
3. MFUSYS = mp4
4. APPSRC = $(PROJ)\$(APPSYS)\src
5. COMSRC = $(PROJ)\$(COMSYS)\src
6. ICMON = $(PROJ)\$(ICN815I2)\src
7. MFUSRC = $(PROJ)\$(MFUSYS)\src
8. COMB1N159 = $(PROJ)\$(COMSYS)\bin\80152
9. ICMONB1N159 = $(PROJ)\$(ICN815I2)\bin\80152
10. BCMONB1N159 = $(PROJ)\$(ICN815I2_MON)\bin\80152
11. # Common subsystem level source directories
12. DEVSRC = $(COMSRC)\dev
13. HDR8SRC = $(COMSRC)\hdr
14. LCSRC = $(COMSRC)\ics
15. MESSRC = $(COMSRC)\mess
16. # ICN subsystem level source directories
17. CCSSRC = $(ICN815I2)\cex
18. IACSRC = $(ICN815I2)\acr
19. # MFU subsystem level source directories
20. LDSSRC = $(MFUSRC)\ds
21. MACSRC = $(MFUSRC)\acr
22. # Common subsystem global include and compilation units
23. SYSCDEF = $(HDR8SRC)\sysdefs.h
24. # ICM subsystem compilation units
25. IAC = $(ICN815I2)\main.obj $(ICN815I2)\main.obj
26. $(ICN815I2)\main.obj $(ICN815I2)\main.obj
27. $(ICN815I2)\main.obj $(ICN815I2)\main.obj
28. $(ICN815I2_MON)\main.obj $(ICN815I2_MON)\main.obj
29. $(ICN815I2_MON)\main.obj $(ICN815I2_MON)\main.obj
30. $(ICN815I2_MON)\main.obj $(ICN815I2_MON)\main.obj

************** DEFINITIONS **************

# Project, system, and application level definitions
PROJ = mra
APPSYS = app
### makefile

**Page 3**

<table>
<thead>
<tr>
<th>Line</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>148</td>
<td># ICH IAC main program dependencies</td>
</tr>
<tr>
<td>149</td>
<td>$(ICBIN152)/main.o : $(SYSDEFS) $(IASRC)/mra.h</td>
</tr>
<tr>
<td>150</td>
<td>$(ICBIN152)/mra.h $(GCSSRC)/qcl.h $(GCSSRC)/qcd.h</td>
</tr>
<tr>
<td>151</td>
<td>$(GCSSRC)/qcl.h $(GCSSRC)/qcd.h $(LCSSRC)/led.h</td>
</tr>
<tr>
<td>152</td>
<td>$(LCSSRC)/led.h $(MMSSRC)/mm.h $(MMSSRC)/mm.h</td>
</tr>
<tr>
<td>153</td>
<td>$(MMSSRC)/mm.h $(MMSSRC)/mm.h $(LDSRC)/dl.h</td>
</tr>
<tr>
<td>154</td>
<td>$(LDSRC)/dl.h $(ICASRC)/main.c</td>
</tr>
<tr>
<td>160</td>
<td>$(ICASRC)/main.c $(CFLAGS) df(180152) $(ICASRC)/main.o $(ICBIN152)</td>
</tr>
<tr>
<td>161</td>
<td># ICH IAC mra system dependencies</td>
</tr>
<tr>
<td>162</td>
<td>$(ICBIN152)/mra.o : $(SYSDEFS) $(ICASRC)/mra.h</td>
</tr>
<tr>
<td>163</td>
<td>$(ICASRC)/mra.h $(GCSSRC)/qcl.h $(GCSSRC)/qcd.h</td>
</tr>
<tr>
<td>164</td>
<td>$(GCSSRC)/qcl.h $(GCSSRC)/qcd.h $(LCSSRC)/led.h</td>
</tr>
<tr>
<td>165</td>
<td>$(LCSSRC)/led.h $(MMSSRC)/mm.h $(MMSSRC)/mm.h</td>
</tr>
<tr>
<td>166</td>
<td>$(MMSSRC)/mm.h $(MMSSRC)/mm.h $(LDSRC)/dl.h</td>
</tr>
<tr>
<td>167</td>
<td>$(LDSRC)/dl.h $(ICASRC)/mra.c</td>
</tr>
<tr>
<td>174</td>
<td>$(ICASRC)/mra.c $(CFLAGS) df(180152) $(ICASRC)/main.o $(ICBIN152)</td>
</tr>
</tbody>
</table>
/* -------------------------------------------------------------
  MAIN.C
-------------------------------------------------------------*/

/* CPC1: IFD90-MRA-ICW-AC-MAIN-C-ROCO */

/* Description: ICN main 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_x 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();
}
#include <errno.h>
#define MRN_MODULE_CODE 17000

/* Public Data Structures: */
#define ERR_MRN_NOT_INIT 1+MRN_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. */
#define MRN_GCS () /* Global Comms Interface. */
#define MRN_LCS () /* Local Comms Interface. */
#define MRN_MMS () /* Method Manager. */
#define MRN_LDS () /* Phone Book. */
#define MRN_MRA () /* Local Method Manager. */
#define MRN_SMH () /* System Method Manager. */

/* External module global error variable. */
extern int mra_error;

/* Public Functions: */
```c
#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 occurrence. */
/* Should be set to AOK after each successful function call. */
/* Variable can be examined by other software after each function call. */

/* Define int mra_error = ERR_MRA_NOT_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 achieved). */

int mra_init();

void mra_main();

void mra_init();

#define MRA_LOCAL_SEND_ATTEMPTS 5
#define MRA_GLOBAL_SEND_ATTEMPTS 5

/* Define int mra_error = module MRA.C */
#define gci_error = module GCI.C
#define lci_error = module LCI.C
#define mm_error = module MM.C
#define ld_error = module LD.C

/* Edit History: 10/01/99 - Written by Robin T. Laird. */

/* Get message from network (ICW LAN). */
/* Function will return immediately if nothing available. */
/* Save GCI global error state. */
/* Get message from module processor (M PU). */
/* 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 ICW LAN. */

while (1)
{
    qcl_receive_message(q, LCI_DONT_WAIT);
    qcl_recv_error = qcl_error;
    lci_receive_message(l, LCI_DONT_WAIT);
    lci_recv_error = lci_error;

    if (qcl_recv_error == AOK)
    {
        lci_send_message((lci_message)q, MRA_LOCAL_SEND_ATTEMPTS);
    }

    if (lci_recv_error == AOK)
    {
        qcl_send_message((qcl_message)l, MRA_GLOBAL_SEND_ATTEMPTS);
    }

    send();
}
# MAKEFILE

CPCl: IED90-MRA-ICN-GCS-MAKEFILE-TXT-A9C

## Description
Makefile for the Modular Robotic Architecture (MRA).

### Targets
MRA makes the ICN global communications subsystem available for the following systems/subsystems:

- qcs: 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

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.SUFFIXES: .hex .exe .obj .c .a51</td>
<td>Applies to the following file extensions.</td>
</tr>
</tbody>
</table>

### Control settings for Franklin 8031 development

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CC</td>
<td>-c51</td>
</tr>
<tr>
<td>AS</td>
<td>-a51</td>
</tr>
<tr>
<td>LINK</td>
<td>-I31</td>
</tr>
<tr>
<td>OTOH</td>
<td>-o a51</td>
</tr>
<tr>
<td>CFLAGS</td>
<td>-cd lo db sb</td>
</tr>
<tr>
<td>ARGL</td>
<td></td>
</tr>
<tr>
<td>OFLAGS</td>
<td></td>
</tr>
<tr>
<td>STARTUP</td>
<td>-s5ivrom.obj</td>
</tr>
<tr>
<td>CODESEG</td>
<td>+00000h</td>
</tr>
<tr>
<td>DATASEG</td>
<td>+00000h</td>
</tr>
</tbody>
</table>

### Control settings for Microsoft MS-DOS development

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSC</td>
<td>-c1</td>
</tr>
<tr>
<td>MSAS</td>
<td>-maam</td>
</tr>
<tr>
<td>MSILINK</td>
<td>-link</td>
</tr>
<tr>
<td>MCFILAGS</td>
<td>-HS -c /O1 /Z1 /Od</td>
</tr>
<tr>
<td>MSASFLAGS</td>
<td></td>
</tr>
<tr>
<td>MSILINKFLAGS</td>
<td>-/co</td>
</tr>
<tr>
<td>LOADLINES</td>
<td></td>
</tr>
</tbody>
</table>

### Targets

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.obj</td>
<td></td>
</tr>
<tr>
<td>$(CC)</td>
<td>$&lt; $(CFLAGS)</td>
</tr>
<tr>
<td>.o51b.obj</td>
<td></td>
</tr>
<tr>
<td>$(AS)</td>
<td>$&lt; $(ASFLAGS)</td>
</tr>
<tr>
<td>.obj:</td>
<td></td>
</tr>
<tr>
<td>$(LINK)</td>
<td>$(STARTUP),$&lt; TO $@ code $(CODESEG) xdata $(DATASEG)</td>
</tr>
<tr>
<td>.exe:</td>
<td></td>
</tr>
<tr>
<td>$(OTOH)</td>
<td>$&lt; $(OFLAGS)</td>
</tr>
</tbody>
</table>

### Definitions

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROJ</td>
<td>mra</td>
</tr>
<tr>
<td>APPSYS</td>
<td>app</td>
</tr>
<tr>
<td>COMSYS</td>
<td>com</td>
</tr>
</tbody>
</table>
```c
/* BMF.H */

/* 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 GCC.C. 
2) Module BMF requires type definitions from GCC.H. 

/* Edit History: 06/28/90 - Written by Robin T. Laird. 
*/

/* Private Data Structures: */

ifndef BMF_MODULE_CODE
#define BMF_MODULE_CODE 1100
#define LLC_FULL_BUFFER 1+BMF_MODULE_CODE
#define LL_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 {
    gsd_frame_item[MAX_BUFFER_SIZE];
    byte front;
    byte rear;
    byte empty;
    byte full;
    
    buffer;
} buffer;

/* Private Functions: */
static int buf_full(buffer *b);
static int buf_empty(buffer *b);
static void buf_insert(buffer *b, gsd_frame f);
static void buf_remove(buffer *b, gsd_frame f);
#endif
```
/* ...
 * 
 * Description: 
 * Contains functions for initializing and managing the 
 * circular frame buffers for the GCS global communications 
 * device handler.
 * Module BMC exports the following functions:
 * buf_front() macro;
 * buf_rear() macro;
 * buf_inc() macro;
 * buf_full() ;
 * buf_empty();
 * buf_clear();
 * buf_incr();
 * buf_removal();
 *
 * Notes: 
 * 1) This module is NOT a stand-alone compilation unit.
 * It is included by the module GCM.C and is compiled there.
 * It is assumed that the file GCM.C is included before it.
 * Note that all of the definitions herein are static.
 * 
 * Edit History: 06/29/90 - Written by Richard P. Smirlo and Robin T. Laird.
 */

/* Private Variables: */
/* Declarations for the module circular receive and transmit buffers. */
/* There are global to the GCS.C module and to the GCM.C module. */
static XDATA buffer wmt_buffer;
static XDATA buffer rev_buffer;

/*------------------------------------------------------------------------
 * 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: 
 * 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])

/*------------------------------------------------------------------------
 * 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: 
 * buffer b; buffer structure (NOT a pointer to it).
 * Output: 
 * Pointer to rear (last) frame in the buffer.
 *
 * Globals: None.
 * 
 * Edit History: 07/08/90 - Written by Robin T. Laird.
 */
#define buf_rear(b) (b.item[b.rear])

/*------------------------------------------------------------------------
 * 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 used within a 
 * compound statement like if ... then ... else. 
 * Input: 
 * buf_index 
 * 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))

/*------------------------------------------------------------------------
 * Function: 
 * Function that returns the boolean of whether or not the 
 * parameter buffer is full (TRUE if so, FALSE if not). 
 * Input: 
 * 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.
 */
#define buf_full(b) 
buffer *b; 
*(b.item[b.front]) ;

/*------------------------------------------------------------------------
 * Function: 
 * Function that returns the boolean of whether or not the 
 * parameter buffer is empty (TRUE if so, FALSE if not). 
 * Input: 
 * 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.
 */
#define buf_empty(b) 
buffer *b; 
*(b.item[b.front]) ;

}
static int buf_empty(b)
buffer *b;

qcd_error = OK; /* Assume function successful... */
return(b->empty);

/* 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: */
/* Output: Nothing. */
/*Globals: qcd_error: module GCD.C */
Edit History: 07/09/90 - Written by Robin T. Laird.

static void buf_clear(b)
buffer *b;

qcd_error = OK; /* 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. */

/* 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 oneself. */
/* An error is returned if the buffer is already full. */
/* Input: buf: */
/* Output: */

static void buf_remove(b)
buffer *b;

qcd_error = OK; /* Assume function successful... */

/* 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 (if[GCD_LEN_POS]). */

/* Function: Removes a frame from the parameter buffer if available. */
/* An error is returned if the buffer is already empty. */
/* Input: */
/* Output: Nothing. */

/* Global: */

/* 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 equal dest. */

"""
static void buf_remove(b, f)
buffer *b;
frame *f;
word i, length;
qud_error = AOK;
/* Assume function successful... */
/* Make sure buffer isn't already empty. */
if (!b->empty)
    { qud_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;
/* globals: qud_error : module GCD.C */
/* edit history: 07/09/90 - written by Robin T. Laird. */
/* */
/*************************************************************************/
/*------------------------------------------\n * GCD.h
 *------------------------------------------*/

#define GCD_MODULE_CODE 1000

#define GCD_ERR_NOT_INIT 3+GCD_MODULE_CODE
#define GCD_ERR_FRAMING_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 qcd_frame[GCD_MAX_FRAME_LENGTH];

/* Structure type for GCD module status (holds recv/mnt error counts). */
typedef struct {
    word r_valid_cnt;
    word r_err_cnt;
    word r_crc6_err_cnt;
    word r_as_err_cnt;
    word r_reMt_Err_cnt;
    word r_crrErr_cnt;
    word x_valid_cnt;
    word x_err_cnt;
    word x不再Err_cnt;
}
**gcd.c**

```
#include <sysdefs.h> /* System constants and types. */
#include "gcd.h" /* GCD public literals/functions. */
#include "bfm.h" /* Global Serial Channel functions. */
#define defined(DBUG) /* Definitions for print(), etc. */

/* Public Variables: */
*/
/* Global module error variable, gcd.error. */
/* gcd.error contains code of last error occurrence. */
/* Should be set to OK 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. */
*/
/* Global 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 "bfm.c" /* Buffer management functions. */
```

**gcd.c**

```
/* Low-level data link layer support functions should be included here. */

/*
* gcd
* 
* #include "gcd.c" /* Low-level GCD functions. */
*/

/*
* Function: Initializes the Global Communications Device Handler.
* Clears the transmit and receive buffers, obtains the ION
* 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 = 0; */ /* 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_cnt     = 0;
   _gcd_state_r_xmtErr_cnt     = 0;
   _gcd_state_xovErr_cnt     = 0;
   _gcd_state_x_valid_cnt = 0;
   _gcd_state.x_rErr_cnt   = 0;
   _gcd_state.x_noackErr_cnt   = 0;
   _gcd_state.x_urErr_cnt    = 0;
   _gcd_state.x_cdErr_cnt    = 0;
*/
/* 
   /* Initialize the transmit and receive buffers. */
   set_t蹩(xmt_buffer);
   set_r蹩(rcv_buffer);
   /* Get the node ID for this particular ION, and pass to GCD init routine. */
   /* // No errors are currently fatal (they are only warnings). */
   qsc_sys_init(qsc_node_id());
   if (gcd_error != 0) return;
   /* 
   /* Initialize the GSC DMA controllers. */
   /* /* No errors are currently possible. */
   /* qsc_dma_init();
   /* if (gcd_error != 0) return;
   /* 
   /* Set up the GSC DMA channel receive/transmit destination/source params. */
   /* The parameters include the DMA src/dst 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. */
   qsc_set_rcv_dst(buf_rex(rcv_buffer), GCD_MAX_FRAME_LENGTH);
   qsc_set_xmt_src(buf_front(xmt_buffer), GCD_MAX_FRAME_LENGTH);
*/
```
Jan 22 1992 08:14:30 gcd.c Page 3

141 /* Start the GSC DMA receive channel (s)low DMA to function. */
142 /* Start the GSC receiver (begin receiving data frames). */
143 /* Start the GSC transmitter for hardware based acknowledge. */
144 /* Issuing a non-fatal and would cause only degraded performance. */
145
gsc_start_retry();
146
gsc_start_smt();
147
/* Enable interrupts... */
148
gsc_enable_intrunments();
149
/* gsc reset */
150
/**
151 * Perform a soft reset of the GCD systems.
152 * The receive and transmit buffers are cleared (the
153 * receive and transmit destination and source addresses
154 * for the GSC DMA channels are reset to the beginning of
155 * the buffers. The GSC reception/transmission error and
156 * valid frame counters of the module state structure are
157 * cleared.
158 *
159 * Input:  gsc reset ();
160 * Output: Nothing.
161 *
162 * Global:  gsmr_reinit : module GCD.
163 */
164
gcd_reinit();
165
/**
166 * Reset all error counting registers in module state variable.
167 * Valid reception/transmission counters are also cleared.
168 */
169
gcd_state.r_valid cnt = 0;
170
/* gsmr_reinit */
171
gsmr_reinit();
172
/**
173 * Re-initialize the transmit and receive buffers.
174 */
175
clear_kmbuffer();
176
clear(rcv_buffer);
As follows depending upon the re-try value:

**gcd_wait_forever**: wait forever for frame to be received.

**retgy**: try this many times to receive frame.

**Input**: gcd_receive_frame();

**Output**: Nothing.

**Globals**:

- gcd_error: module GCD.c
- gcd_tries: module GCD.c
- gcd_times: module GCD.c
- recv_buffer: module BMP.C

**Edit History**: 07/08/90 - Written by Robin T. Laird.

**01/30/91** - Robin T. Laird added re-transmit count/stop.

```c
void gcd_receive_frame(f, retry)

{gcd.frame();

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
#endif

if defined(DEBUG)

prev_bcrh0 = BCRH;

prev_bcrl0 = BCRLO;

#endif
#endif

if (retry == GCD_DONT_WAIT)

buf_remove(recv_buffer, f);

else if (retry == GCD_MAX_ATTEMPTS)

while (buf_empty(recv_buffer))

if (++gcd_tries == gcd_times)

gcd_error = GCD_ERR_RECEIVE_FRAME;

break;

}

}

void gcd_transmit_frame(f, retry)

{gcd.frame();

word retry;

if defined(DEBUG)

unsigned char prev_tstat; /* Last stored value of TSTAT. */

unsigned char prev_bcrh; /* Last BCR's (high and low bytes). */

unsigned char prev_bcrl; /*

#endif
#endif

```

```c
/* Assume function successful... */

/* Number of attempted rcvs. */

/* Number of times to retry. */

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 auxiliary llo counter.

Loop, checking to see if frame is available and counter not expired.

else, get incoming frame and return.

if (retry == GCD_DONT_WAIT)

{buf.remove(recv_buffer, f);

}

else if (retry == GCD_WAIT_FOREVER)

{while (buf_empty(recv_buffer))

if defined(DEBUG)

ᡴᡩᡩᡨᠫᡯᡩᡨᡩᡨᡩᡩᡩᡯᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩᡩdbc

```
Jan 22 1992 08:14:30  gcd.c  Page 7
419  prev_tsstat = 0xFF;
420  prev_bcrh = 0xFF;
421  prev_bcll = 0xFF;
422  sendline;
423  gerd_error = ALW;  /* Assume function successfui... */
424  
425  gerd_tries = 0;  /* Number of attempted xmts.*/
426  gerd_times = retry;  /* Number of times to try.*/
427  gerd_stop = FALSE;
428  
429  /* Insert frame into transmit buffer and send it.*/
430  /* If the transmit buffer is empty then the GCD transmitter is disabled.*/
431  /* Reset the DMA transmitter source address and byte count.*/
432  /* Re-start the GCD transmitter (it was turned off when buf empty).*/
433  /* Re-enable the DMA transmit channel also turned off when buf empty.*/
434  /* Else, just insert frame into buffer for transmission (sometime later).*/
435  /* If the insert failed, abort transmission and return an error. */
436  if (buf_empty(NULL_buffer))
437     
438     buf_insert(NULL_buffer, f);
439     if (gerd_error != AOK)
440         
441         gerd_error = GCD_ERR_TRANSMIT_FRAME;
442         return;
443     
444     else
445         
446         gerd_src_xmt_src(NULL_buffer, f[GCD_LEN_POS]);
447         gerd_stat_wmt();
448         gerd_src_qd();
449     }
450  
451  else
452  
453  buf_insert(NULL_buffer, f);
454  if (gerd_error != AOK)
455         
456         gerd_error = GCD_ERR_TRANSMIT_FRAME;
457         return;
458  
459  while (!buf_empty(NULL_buffer))
460     
461     if (defined(DEBUG))
462       
463       if (prev_tsstat != TSTAT) || (prev_bcrh != BCRH) || (prev_bcll != BCLL)
464         
465         printf("gerd transm frame: TSTAT=%02x, BCRH(%d)=%02x, BCLL=%02x", TSTAT,
466             prev_bcrh, BCRH);  
467         prev_bcll = BCLL;
468         
469         printf("noac = %u, ur = %u, tcdr = %u, xmt_valid = %u,  
470             gerd_state.x.noac_err_cnt,  
471             gerd_state.x.ur_err_cnt,  
472             gerd_state.x.tcdr.err_cnt,  
473             gerd_state.x_valid_cnt,  
474             gerd_state.f_valid_cnt);  
475         
476         #endif

Jan 22 1992 08:14:30  gcd.c  Page 8
512  }
513  else if (retry < GCD_MAX_ATTEMPTS)
514  
515  while (!buf_empty(NULL_buffer))
516  
517  if (gerd_stop)
518     
519     buf_remove(NULL_buffer, f);
520     gerd_error = GCD_ERR_TRANSMIT_FRAME;
521     break;
522  
523  }
524  else
525  
526  gerd_error = GCD_ERR_NUM_ATTEMPTS;
527  
528  
529  /*****************************************************/
530  gerd_status;
531  /*****************************************************/
532  
533  Function:  Returns the operational status of the GCD subsystem.  
534  This is accomplished by returning the contents of the  
535  module state structure that records various operational  
536  parameters such as the number of valid frames received, etc.  
537  
538  Input:  gerd_status;  pointer to module state structure.  
539  
540  Output: Nothing.  
541  
542 Globals:  gerd_error; module GCD.C  
543  
544  Edit History: 07/09/90 - Written by Richard P. Smurlo.  
545  
546  1/1/90  
547  
548  void gerd_status();  
549  gerd_state *s;  
550  
551  
552  if (retry == GCD_DONT_WAIT)
553  
554  gerd_error = AOK;  /* Assume function successful... */
555  
556  *s = _ged_state;
557  
558  */
# GCI.H

* Description: GCI 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 type/variables/functions:

typedef gci_message;

int gci_error;

void gci_init();

void gci_receive_message();

void 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:

#define GCI_MODULE_CODE 2000

#define GCI_FRM_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_WAIT 0

#define GCI_MAX_ATTEMPTS 60000

* Maximum message length SHOULD be two less than maximum frame length.

#define GCI_MAX_MESSAGE_LENGTH 64000

* 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);
/* Initialize the GCD subsystem (sets up ICN LAN hardware and software). */
gcl_init();
if (gcl_error != OK)
    switch(gcl_error)
        case GCD_FAIL_RECEIVER:
            gcl_error = GCI_ERR_NOT_INIT;
            break;
        default:
            gcl_error = gcl_error;
        
    gcl_receive_message();

    gcl_message m; received message.
    word retry; number of times to try receiving message.
} 

/* Global module error variable, gcl_error. */
/* gcl_error contains code of last error occurrence. */
/* Should be set to OK after each successful function call. */
/* Variable can be examined by other software after each function call. */
#define GCI_ERR_NOT_INIT; /* Global module error variable. */

/* Function: Initializes the Global Communications Interface. */
The Global Communications Device 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 gcl_error.

/* Input:*/
gcl_init();
/* Output:*/
Nothing.

/*Globals:*/
gcl_error : module GCL.C
gcl_error : module GCD.C

/* Edit History: 07/28/90 - Written by Robin T. Laird. */
```c
#include <stdlib.h>

void gci_send_message(uint16_t m, uint8_t retry)
{
    gci_message m;
    word retry;
    qci_state status; /* Holds GCI status. */
    qci_error = AOK;
    /* Assume function successful... */
    qci_transmit_frame(m, retry);
    /* Send frame, iterate retry number of times. */
    if (qci_error != AOK)
    {
        qci_status(tstatus);
        if (tstatus.x_noack_err_cnt > MAX_SEND_ERRORS) gci_reset();
        qci_error = GCI_ERR_SEND_MESSAGE;
    }
    /* If frame could not be transmitted, check integrity of transmitter. */
    /* If number of transmit errors is excessive then reset the GCI's system. */
    /* Currently only counting no acknowledgement errors. */
    if (tstatus.x_noack_err_cnt > MAX_SEND_ERRORS)
    {
        gci_status(tstatus);
        if (tstatus.x_noack_err_cnt > MAX_SEND_ERRORS) gci_reset();
        qci_error = GCI_ERR_SEND_MESSAGE;
    }
}
```

CPCI: IDE90-MRA-ICM-GCS-GSC-H-8000

Description: Global Serial Channel (GSC) variables and functions. 
Contains constant function parameter and hardware register 
definitions for initialization and control of the Intel 
801525 GSC. Implements the low-level data link layer portion 
of the MRA GCS communications device handler functions.

Module GSC exports the following functions:

/* Private Data Structures: */

#define GSC_MODULE_CODE 1
#define GSC_MODULE_CODE 1200
#define ERR_NODE_ID_TOO_LARGE 1 + GSC_MODULE_CODE
#define ERR_BIFIO_HUP_CLEAR> GSC_MODULE_CODE

/* GSC Constants: */

#define ADDR_BIFIO 0xBS /* Transmit FIFO Address. */
#define ADDR_BIFIO 0xBS /* Receive FIFO Address. */
#define GSC_BAUD 0x05 / GSC Baud Rate = (GSL)/(GSC_BAUD+1)MB.
#define GSC_BAUD 1 GSC Baud Rate = 521.4 Kbps.
#define CLOCK_MXK 0x00 / GMS = 00XX111XXX for internal clock.
#define D_BIFIO_CMD 0x60 /* GMS = 00XX111XXX for alternate BIFIO. */
#define ADDR_MWK 0x00 /* GMS = 00XX111XXX for 8-bit Addresses. */
#define ADDR_MWK 0x00 /* GMS = 00XX111XXX for 8-bit Addresses. */
#define PREMMED_MK 0x00 /* GMS = 00XX111XXX for CSMA/CD protocol. */
#define JM_MK 0x00 /* GMS = 00XX111XXX for DC Jam Signal. */
#define D_BIFIO_SLOT 0x00 /* GMS = 00XX111XXX for DC Jam Signal. */
#define D_BIFIO_SLOT 0x00 /* GMS = 00XX111XXX for DC Jam Signal. */
#define GSC_IFS 0 /* IFN = Number of Machine Cycles for */
#define GSC_IFS 0 /* IFN = Number of Machine Cycles for */
#define GSC_SLOTS 216 /* longest receive service routine */
#define GSC_SLOTS 216 /* longest receive service routine */
#define IC_MK 0 /* ICN Monitor should not Acknowledge. */
#define GSC_AMSK 0xFF /* AMSK = 0xFF - Mask off ARDO Node ID. */
#define GSC_AMSK 0xFF /* AMSK = 0xFF - Mask off ARDO Node ID. */
#define HABEN 1 /* HABEN = 1 for Hardware Based Acknowledgments*/
#define GSC_AMSK 0x00 /* AMSK = 0x00 - Don't mask ARDO Node ID. */
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 *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}>>8}+0x00FF

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

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

Function: Determines node ID number for the particular ICH.

The input ID number is currently read from the P6 80C152 parallel port which is connected to an 8-position switch.

Input: xptr_node_id;

Output: Returns the node identification number (address) as a byte.

Valid range is 0 <= node ID <= 0xFF.

Globals: xcd: module GCD.C

80C152 reg: module REG152.M

Edit History: 06/28/90 - Written by Robin T. Laird.

static byte xcd_node_id;

$qcd_error = xcd;

/* Assume function successful... */

return ((byte)P6);
```c
/* Output: Nothing. */

* Global: qcd_error : module GCD.C

* / Init History: 07/10/90 - Written by Richard P. Smuro.

static void qcd_sys_init(node_id)
byte node_id;
{
    qcd_error = AOK; /* Assume function successful... */
    /* Disable interrupts while we're changing things. */
    EA = 0;
    /* Initialize all GSC related Registers to 0x00. */
    GMOD = 0x00;
    MYSLT = 0x00;
    /* Protocol Dependent Initialization. */
    BAUD = GSC_BAUD;
    GMOD = GMOD | PROTOCOL_MASK; /* GSC Baud Rate Select. */
    GMOD = GMOD | PROTOCOL_MASK; /* Int. or Ext. clock. */
    GMOD = GMOD | PROTOCOL_MASK; /* Message Preamble. */
    GMOD = GMOD | PROTOCOL_MASK; /* Deterministic Backoff. */
    MYSLT = MYSLT | D_BKOFF SLOT; /* Ditto. */
    MYSLT = MYSLT | D_BKOFF SLOT; /* CRC definition. */
    EMOD = EMOD | EMOD | JAM_MSK; /* Inter Frame Space. (LSBit Ignored) */
    SLOTTM = GSC SLOTTM; /* Slot Time. */
    EMOD = EMOD | EMOD | ADDR_MSK; /* Addressing. */

    /* The address mask registers are explicitly initialized to values that */
    /* are "safe" to use 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. */
    ADRO = node_id; /* Addressing. (Node ID # 1) */
    ADRO = node_id+1; /* Addressing. (Node ID # 2) */
    ADRO = node_id+1; /* Addressing. (Node ID # 1) */
    ADRO = node_id+1; /* Addressing. (Node ID # 1) */
    AMSK0 = GSC AMSK0; /* AMSK0 masks ADRO for ICHNOW (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 PRBS for determin- */
    /* resion. */

    /* 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) {
        MYSLT = MYSLT + node_id;
        else {
            MYSLT = MYSLT + NO SLOT; /* Slot number 0 if Node Id > MAX. */
            qcd_error = ERR_NODE_ID_TOO_LARGE; /* Warn that node ID is too big. */
        }
}
```

<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>291</td>
<td><code>DCON0 = DCON0 &amp; K_XFERMODE_MASK;</code></td>
<td>Set Transfer Mode.</td>
</tr>
<tr>
<td>292</td>
<td></td>
<td></td>
</tr>
<tr>
<td>293</td>
<td><code>DCON0 = DCON0 &amp; K_XFERMODE_MASK;</code></td>
<td>Set Demand Mode.</td>
</tr>
<tr>
<td>294</td>
<td></td>
<td></td>
</tr>
<tr>
<td>295</td>
<td><code>DCON0 = DCON0 &amp; K_SRC_AUTOINC_MASK;</code></td>
<td>Set up Auto Increment Option.</td>
</tr>
<tr>
<td>296</td>
<td></td>
<td></td>
</tr>
<tr>
<td>297</td>
<td><code>DCON0 = DCON0 &amp; K_SRC_AUTOINC_MASK;</code></td>
<td>Define DMA Source.</td>
</tr>
<tr>
<td>298</td>
<td><code>DCON0 = DCON0 &amp; K_DST_AUTOINC_MASK;</code></td>
<td>Define DMA destination address.</td>
</tr>
<tr>
<td>299</td>
<td><code>DCON0 = DCON0 &amp; K_DEST_MASK;</code></td>
<td>Define DMA Destination.</td>
</tr>
<tr>
<td>300</td>
<td>SARLO = ADDR_RFIPO;</td>
<td>Source Address = RFIPO.</td>
</tr>
<tr>
<td>301</td>
<td>SARHO = 0x00;</td>
<td></td>
</tr>
<tr>
<td>302</td>
<td><code>/* DMA Channel 1 Init. - Transmitter: */</code></td>
<td></td>
</tr>
<tr>
<td>303</td>
<td></td>
<td></td>
</tr>
<tr>
<td>304</td>
<td><code>DCON1 = 0x00;</code></td>
<td></td>
</tr>
<tr>
<td>305</td>
<td><code>DCON1 = DCON1 &amp; K_XFERMODE_MASK;</code></td>
<td>Transmit Transfer Mode.</td>
</tr>
<tr>
<td>306</td>
<td><code>DCON1 = DCON1 &amp; K_DEST_MASK;</code></td>
<td>Set Demand Mode.</td>
</tr>
<tr>
<td>307</td>
<td><code>DCON1 = DCON1 &amp; K_SRC_AUTOINC_MASK;</code></td>
<td>Set auto increment option.</td>
</tr>
<tr>
<td>308</td>
<td><code>DCON1 = DCON1 &amp; K_DST_AUTOINC_MASK;</code></td>
<td></td>
</tr>
<tr>
<td>309</td>
<td><code>DCON1 = DCON1 &amp; K_DEST_MASK;</code></td>
<td></td>
</tr>
<tr>
<td>310</td>
<td>DARLI = ADDR_TFIFO;</td>
<td>Dest. Addr. = TFIFO.</td>
</tr>
<tr>
<td>311</td>
<td>DARN1 = 0x00;</td>
<td></td>
</tr>
<tr>
<td>312</td>
<td><code>#if defined(DEBUG)</code></td>
<td></td>
</tr>
<tr>
<td>313</td>
<td><code>print(&quot;gsc_dma_init: TSTAT = \%02x\n&quot;, TSTAT);</code></td>
<td></td>
</tr>
<tr>
<td>314</td>
<td><code>print(&quot;gsc_dma_init: FCN = \%02x\n&quot;, FCN);</code></td>
<td></td>
</tr>
<tr>
<td>315</td>
<td><code>print(&quot;gsc_dma_init: SAR = \%02x\n&quot;, SAR);</code></td>
<td></td>
</tr>
<tr>
<td>316</td>
<td><code>print(&quot;gsc_dma_init: SAR = \%02x\n&quot;, SAR);</code></td>
<td></td>
</tr>
<tr>
<td>317</td>
<td><code>print(&quot;gsc_dma_init: DARN1 = \%02x\n&quot;, DARN1);</code></td>
<td></td>
</tr>
<tr>
<td>318</td>
<td><code>print(&quot;gsc_dma_init: DARN1 = \%02x\n&quot;, DARN1);</code></td>
<td></td>
</tr>
<tr>
<td>319</td>
<td><code>#endif</code></td>
<td></td>
</tr>
<tr>
<td>320</td>
<td></td>
<td></td>
</tr>
<tr>
<td>321</td>
<td><code>gsc_set_rcv_dat</code></td>
<td></td>
</tr>
<tr>
<td>322</td>
<td><code>/* Function: Sets the DMA receiver destination address and byte count. */</code></td>
<td></td>
</tr>
<tr>
<td>323</td>
<td><code>DMA channel 0 is the receive channel. The destination for the notion is the parameter frame. Special care must be taken so that data is not written (or processed) before the entire frame is received.</code></td>
<td></td>
</tr>
<tr>
<td>324</td>
<td><code>the destination address must be of type XDATA. */</code></td>
<td></td>
</tr>
<tr>
<td>325</td>
<td><code>Input: gsc_set_rcv_dat</code></td>
<td></td>
</tr>
<tr>
<td>326</td>
<td><code>gcd_frameceiver_frame to be received (must be type XDATA). */</code></td>
<td></td>
</tr>
<tr>
<td>327</td>
<td><code>word length; frame length (0 &lt; 1 &lt;= GCD_MAX_FRAME_LENGTH). */</code></td>
<td></td>
</tr>
<tr>
<td>328</td>
<td><code>*/ Output: Nothing. */</code></td>
<td></td>
</tr>
<tr>
<td>329</td>
<td><code>Globals: gcd_error = module GCD.C */ *80C152_regs = module REG512.H */</code></td>
<td></td>
</tr>
<tr>
<td>330</td>
<td><code>Edit History: 06/28/90 - Written by Robin T. Laird. */</code></td>
<td></td>
</tr>
<tr>
<td>331</td>
<td><code>*/ </code></td>
<td></td>
</tr>
<tr>
<td>332</td>
<td><code>static void gsc_set_rcv_dat(f, length) */ *80C152_regs = module REG512.H */</code></td>
<td></td>
</tr>
<tr>
<td>333</td>
<td><code>gsc_frame f; */ *80C152_regs = module REG512.H */</code></td>
<td></td>
</tr>
<tr>
<td>334</td>
<td><code>word length; */ *80C152_regs = module REG512.H */</code></td>
<td></td>
</tr>
<tr>
<td>335</td>
<td><code>gcd_error = AOK; */ *80C152_regs = module REG512.H */</code></td>
<td>Assume function successful... */</td>
</tr>
<tr>
<td>336</td>
<td><code>*/ *80C152_regs = module REG512.H */</code></td>
<td></td>
</tr>
<tr>
<td>337</td>
<td><code>*/ Set up destination and byte count registers. */ */ </code></td>
<td></td>
</tr>
<tr>
<td>338</td>
<td><code>DARLI = xptr_lo_offset(f); */ *80C152_regs = module REG512.H */</code></td>
<td>Set DMA reception dest addr.</td>
</tr>
<tr>
<td>339</td>
<td><code>DARN1 = xptr_hi_offset(f); */ *80C152_regs = module REG512.H */</code></td>
<td></td>
</tr>
<tr>
<td>340</td>
<td><code>BCRL0 = length &amp; 0xFF; */ *80C152_regs = module REG512.H */</code></td>
<td>Set frame length hi/lo byte cnt.</td>
</tr>
<tr>
<td>341</td>
<td><code>BCRH0 = length &gt;&gt; 8 &amp; 0xFF; */ *80C152_regs = module REG512.H */</code></td>
<td></td>
</tr>
</tbody>
</table>
```c
// Description: 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().
void gsc_start_xmt() {
    // Start transmitter and enable interrupts.
    gsc_xmt_go(); // Set DMA Go bit.
}

void gsc_xmt_go() {
    // Enable Transmit Error ISR.
    eneTransError = 1;
    // Enable Transmit Valid ISR.
    eneTransValid = 1;
    // Start transmission...
}
```

```c
// Description: 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_stop_rcv().
void 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_stop_rcv().
    gsc_stop_rcv();
```
Function: gsc_stop_xmt()

Input: gsc_stop_xmt()

Output: Nothing.

Globals: gsc_error = module GCD.C

80152 regs = module REG152.H

Edit History: 07/10/90 - Written by Richard P. Smurlo.

static void gsc_stop_xmt()
{
    gsc_error = AOK;
    /* Assume function successful... */
    TEN = 0;
    /* TEN = 0 to disable transmitter. */
    EGSTER = 0;
    /* Disable Transmit Error ISR. */
    EGSTY = 0;
    /* Disable Transmit Valid ISR. */
    DCONI = DCONI & 0xF0;
    /* Re-set DMA GO bit. */
}

gsc_enable_interrupts

Function: Enables processor interrupts for the 8031/80152.

Input: gsc_enable_interrupts()

Output: Nothing.

Globals: gsc_error = module GCD.C

80152 regs = module REG152.H

Edit History: 03/09/91 - Written by Robin T. Laird.

static void gsc_enable_interrupts()
{
    gsc_error = AOK;
    /* Assume function successful... */
    EA = 1;
    /* Enable all interrupts. */
    gsc_stop_xmt()
}

gsc_boost_xmt()

Function: Processes a valid receive interrupt (GSSRV) from the GSC.

This routine is called upon the completion of a valid frame reception (i.e., no OVERRUN, ABORT, ALIGNMENT, nor CRC error). If RBA 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 BMP.C

gcd_state : module GCD.C

80152 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 (rcv_buffer.full)
    {
        GREN = 0;
        /* Turn receiver off - NO ACK. */
        return;
    }

    /* Determine length of received frame from DMA byte count registers. */
    rcv_buffer.item[rcv_buffer.rear][GCD_LEN_POS] =
    gcd_max_frame_length - (BCRH < 8) ? BCRO16 : BCRO8;

    /* Adjust rear index (MAX_BUFFER_SIZE-1 is last element in buffer). */
    rcv_buffer.rear = (rcv_buffer.rear + 1) % MAX_BUFFER_SIZE;

    /* Buffer can't be empty since we just received a frame. */
    if (rcv_buffer.front == rcv_buffer.rear && rcv_buffer.full == TRUE)
        /* Buffer empty = FALSE; */
    else
        /* Increment valid reception count variable of module state structure. */
        /* Roll 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. */
    DCONO = DCONO & 0xF0;
    /* Set DMA GO bit. */
    GREN = 1;
    /* Re-enable the GSC receiver. */
}
static void qsc_xmt_valid() interrupt 8 using 2
717 */ Buffer can't be full since we just removed an element. */
718 xmt_buffer.full = FALSE;
719 /* Adjust front index (MAX_BUFFER_SIZE-1 is last element in buffer). */
720 buf_idx(xmt_buffer.front);
721 /* Check and see if we've depleted the buffer (set empty flag if so). */
722 if (xmt_buffer.front == xmt_buffer.rear) xmt_buffer.empty = TRUE;
723 /* Increment valid transmission count variable of module state structure. */
724 /* Rolls over after 65535 frames emitted (if someone is keeping track). */
725 qcd_state.x_valid_cnt++;
726 /* If more than one element in transmit buffer... */
727 /* Set transmit source to newly updated front of buffer. */
728 /* Reset DMA byte count registers for next transmission. */
729 /* Re-enable the transmitter (must be done BEFORE GO bit is set). */
730 /* DMA GO bit to start next transmission. */
731 /* Otherwise, we're done transmitting for now, so turn things off ... */
732 /* Disable transmit valid and error interrupts. */
733 /* Disable the DMA channel. */
734 /* But still enable the transmitter for HBA. */
735 if (!xmt_buffer.empty) 736 737 SARLI = xptr_lo_offset(buf_front(xmt_buffer));
738 SARHI = xptr_hi_offset(buf_front(xmt_buffer));
739 BCRLI = xmt_buffer_item[xmt_buffer.front][GCC_LEN_POS] & 0x0F;
740 BCRLH = xmt_buffer_item[xmt_buffer.front][GCC_LEN_POS >> 8] & 0xFF;
741 RCRLI = (xmt_buffer_item[xmt_buffer.front][GCC_LEN_POS] >> 8) & 0xFF;
742
Function: Processes a transmit error interrupt (EGST) 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.

Global:
- gsc_state : module GSC.c
- gsc_tries : module GSC.c
- gsc_times : module GSC.c
- gsc_stop : module GSC.c
- B0C152 regs : module REG152.h

Edit History: 07/01/90 - Written by Richard P. Smuro.
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. */

qsc_state.x.err_cnt++;
if (TSTAT & X_TCDT_ERR_MASK)
  qsc_state.x.tcdt_err_cnt++;
else
  if (TSTAT & X.UR_ERR_MASK)
    qsc_state.x.ur_err_cnt++;
  else
    if (TSTAT & X.NACK_ERR_MASK)
      qsc_state.x.nack.err_cnt++;

/* Check number of attempts made so far. If qsc_times == 0 not counting. */
/* Variable qsc_times = 0 if attempt re-transmit forever. */

if (qsc_times)
  if (qsc_stop)
    return;
  else if (++qsc_tries > qsc_times)

  qsc_stop = TRUE;
  /* Hit re-transmit limit. */
  BCRH1 = 0;
  /* Set high byte count to 0. */
  BCRH1 = 0;
  /* Set low byte count to 0. */
  TEN = 1;
  /* Re-enable transmitter for ACRs. */
  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));
BCRH1 = 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 xmtter. */
while(!TEN);
DCONI.DCONI = 0x01;
/* Wait for TEN to actually be set. */
DCONI = DCONI | 0x01;
/* Set DMA GO bit. */
# Rules

``` Makefile
MPUSYS = mpu
APPsrc = $(PROJ)/$(APPsrc)/src
COMSRC = $(PROJ)/$(COMSRC)/src
ICNsrc = $(PROJ)/$(ICNsrc)/src
MPUSRC = $(PROJ)/$(MPUSRC)/src
COMIN3 = $(PROJ)/$(COMIN3)/bin/mzos
MPUBIN31 = $(PROJ)/$(MPUBIN31)/bin/mzos
MPUBIN152 = $(PROJ)/$(MPUBIN152)/bin/mzos
MPUBINMS = $(PROJ)/$(MPUBINMS)/bin/mzos
MPUBINSCB = $(PROJ)/$(MPUBINSCB)/bin/mzos

# Common subsystem level source directories
DEVSRC = $(COMSRC)/dev
HDDSRC = $(COMSRC)/hddr
LCSSRC = $(COMSRC)/lc
MMSRC = $(COMSRC)/mms

# ICN subsystem level source directories
GCSSRC = $(ICNsrc)/gc
JACSRC = $(ICNsrc)/jac

# MPU subsystem level source directories
LDSSRC = $(MPUSRC)/ld
MACSRC = $(MPUSRC)/mac

# Common subsystem global include and compilation units
SYSDEFS = $(HDDSRC)/sysdefs.h

# MPU subsystem compilation units
MAC = $(MPUBIN31)/main.obj $(MPUBINMS)/main.obj

# Targets
mac : $(MAC)
print : $(MAC)

c , -2ps -nf main,c | post
-d , -2ps -nf main . h | post
d , -2ps -nf main . c | post
touch print

# MPU MAC dependencies

# MPU MAC main program dependencies
MPUMAIN = $(SYSDEFS) $(MACSRC)/main.c $(CFLAGS) $(1D303)/pr$(MACSRC)/main.c $(MPUBIN31)/$(MPUBINMS)/main.obj

# MPU MAC main system dependencies
MPUMRA = $(SYSDEFS) $(MACSRC)/main.c $(CFLAGS) $(1D303)/pr$(MACSRC)/main.c $(MPUBIN31)/$(MPUBINMS)/main.obj
```

# Definitions

``` Makefile
# Project, system, and application level definitions
PROJ = mra
APPsys = upp
COMsys = com
ICNsys = icn
```

# Control settings for Franklin 1D303 development
CC = c51
AS = a51
LINK = a51
OTON = op51
CFLAGS = -D -la db eb
ASFLAGS =
LFLAGS =
OFLAGS =
STARTUP = \c51\crom.obj
CODESEG = +000000h
DATASEG = +000000h

# Control settings for Microsoft MS-DOS development
MSC = c
MSAS = as
MSLINK = link
MCFLAGS = /A /c /O1 /Z1 /Od
MASFLAGS =
MSLINKFLAGS = /co
LOADLIBS =
.obj : $(CC) $< $(CFLAGS)
.obj : $(AS) $< $(ASFLAGS)

# Control settings for Franklin 1D303 development
CC = c51
AS = a51
LINK = a51
OTON = op51
CFLAGS = -D -la db eb
ASFLAGS =
LFLAGS =
OFLAGS =
STARTUP = \c51\crom.obj
CODESEG = +000000h
DATASEG = +000000h

# Control settings for Microsoft MS-DOS development
MSC = c
MSAS = as
MSLINK = link
MCFLAGS = /A /c /O1 /Z1 /Od
MASFLAGS =
MSLINKFLAGS = /co
LOADLIBS =
.obj : $(CC) $< $(CFLAGS)
.obj : $(AS) $< $(ASFLAGS)

# Control settings for Franklin 1D303 development
CC = c51
AS = a51
LINK = a51
OTON = op51
CFLAGS = -D -la db eb
ASFLAGS =
LFLAGS =
OFLAGS =
STARTUP = \c51\crom.obj
CODESEG = +000000h
DATASEG = +000000h

# Control settings for Microsoft MS-DOS development
MSC = c
MSAS = as
MSLINK = link
MCFLAGS = /A /c /O1 /Z1 /Od
MASFLAGS =
MSLINKFLAGS = /co
LOADLIBS =
.obj : $(CC) $< $(CFLAGS)
.obj : $(AS) $< $(ASFLAGS)

# Control settings for Franklin 1D303 development
CC = c51
AS = a51
LINK = a51
OTON = op51
CFLAGS = -D -la db eb
ASFLAGS =
LFLAGS =
OFLAGS =
STARTUP = \c51\crom.obj
CODESEG = +000000h
DATASEG = +000000h

# Control settings for Microsoft MS-DOS development
MSC = c
MSAS = as
MSLINK = link
MCFLAGS = /A /c /O1 /Z1 /Od
MASFLAGS =
MSLINKFLAGS = /co
LOADLIBS =
.obj : $(CC) $< $(CFLAGS)
.obj : $(AS) $< $(ASFLAGS)

# Control settings for Franklin 1D303 development
CC = c51
AS = a51
LINK = a51
OTON = op51
CFLAGS = -D -la db eb
ASFLAGS =
LFLAGS =
OFLAGS =
STARTUP = \c51\crom.obj
CODESEG = +000000h
DATASEG = +000000h

# Control settings for Microsoft MS-DOS development
MSC = c
MSAS = as
MSLINK = link
MCFLAGS = /A /c /O1 /Z1 /Od
MASFLAGS =
MSLINKFLAGS = /co
LOADLIBS =
.obj : $(CC) $< $(CFLAGS)
.obj : $(AS) $< $(ASFLAGS)
```
```c
#include <sysdef.h> /* MRA standard declarations. */
#include <rtc.h> /* MRA real-time clock functions. */
#include "mra.h" /* MRA public literals/functions. */
#define TWO_SECONDS 7000L /* Wait two seconds before start. */

void main()
{
    /* Wait for a few seconds before fitting. */
    /* This gives the ICN time to initialize subsystems, 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();
}
```
/* 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/deselect 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: 07/04/90 - Written by Robin T. Laird.

*/

#include MRA_MODULE_CODE
#define MRA_MODULE_CODE 13000
#define ERR_MRA_NOT_INIT 1

/* Public Data Structures: */

#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 <gcs.h>  /* Global Comms Interface. */
#include <gds.h>  /* Global Comms Device Driver. */
#endif

#if USE_LCS
#include <lcs.h>  /* Local Comms Interface. */
#include <lcsd.h> /* Local Comms Device Driver. */
#endif

#if USE_MMS
#include <mms.h>  /* Method Manager. */
#include <mmh.h>  /* Phone Book. */
#include <lmh.h>  /* Local Method Manager. */
#include <smmh.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: */
# MRA.C

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:

- 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 occurrence. */
/* Should be set to OK after each successful function call. */
/* Variable can be examined by other software after each function call. */

#define 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 achieved). */

/* Input: mra_init(); */
/* Output: Nothing. */
/* Globals: */
/* mra_error : module MRA.C */
/* cbc_error : module CLI.C */
/* lcl_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_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 */
    /* m_cycle() "occasionally" so that incoming messages are not dropped. */
    if USE_MM
        m_cycle(MM_CYCLE_FOREVER);
    } 
```
Jan 22 1992 08:24:08 makefile Page 1

1  MAKEFILE
2  # Description: Makefile for the Modular Robotic Architecture (MRA).
3  # Makes the logical device interface subsystem.
4  # Targets are available for the following systems/subsystems:
5  # Ind - Logical Device Interface Subsystem
6  # 1ib - Add module to MRA library
7  # Notes: 1) The dependency and production rules are included here.
8  # 2) See also uMRA.makefile
9  
10  # Edit History: 05/30/91 - Written by Robin T. Laird.
11  
12  #*****************************************************************************
13  #*****************************************************************************
14  #*****************************************************************************
15  #*****************************************************************************
16  #*****************************************************************************
17  #*****************************************************************************
18  #*****************************************************************************
19  #*****************************************************************************
20  #*****************************************************************************
21  #*****************************************************************************
22  #*****************************************************************************
23  #*****************************************************************************
24  #*****************************************************************************
25  #*****************************************************************************
26  #*****************************************************************************
27  #*****************************************************************************
28  #*****************************************************************************
29  #*****************************************************************************
30  #*****************************************************************************
31  #*****************************************************************************
32  #*****************************************************************************
33  #*****************************************************************************
34  #*****************************************************************************
35  #*****************************************************************************
36  #*****************************************************************************
37  #*****************************************************************************
38  #*****************************************************************************
39  #*****************************************************************************
40  #*****************************************************************************
41  #*****************************************************************************
42  #*****************************************************************************
43  #*****************************************************************************
44  #*****************************************************************************
45  #*****************************************************************************
46  #*****************************************************************************
47  #*****************************************************************************
48  #*****************************************************************************
49  #*****************************************************************************
50  #*****************************************************************************
51  #*****************************************************************************
52  #*****************************************************************************
53  #*****************************************************************************
54  #*****************************************************************************
55  #*****************************************************************************
56  #*****************************************************************************
57  #*****************************************************************************
58  #*****************************************************************************
59  #*****************************************************************************
60  #*****************************************************************************
61  #*****************************************************************************
62  #*****************************************************************************
63  #*****************************************************************************
64  #*****************************************************************************
65  #*****************************************************************************
66  #*****************************************************************************
67  #*****************************************************************************
68  #*****************************************************************************
69  #*****************************************************************************
70  #*****************************************************************************
71  #*****************************************************************************
72  #*****************************************************************************
73  #*****************************************************************************
74  #*****************************************************************************
75  #*****************************************************************************
76  #*****************************************************************************
77  #*****************************************************************************
78  #*****************************************************************************
79  #*****************************************************************************
80  #*****************************************************************************
81  #*****************************************************************************
82  #*****************************************************************************
83  #*****************************************************************************
84  #*****************************************************************************
85  #*****************************************************************************
86  #*****************************************************************************
87  #*****************************************************************************
88  #*****************************************************************************
89  #*****************************************************************************
90  #*****************************************************************************
91  #*****************************************************************************
92  #*****************************************************************************
93  #*****************************************************************************
94  #*****************************************************************************
95  #*****************************************************************************
96  #*****************************************************************************
97  #*****************************************************************************
98  #*****************************************************************************
99  #*****************************************************************************
100 #*****************************************************************************
101 #*****************************************************************************
102 #*****************************************************************************
103 #*****************************************************************************
104 #*****************************************************************************
105 #*****************************************************************************
106 #*****************************************************************************
107 #*****************************************************************************
108 #*****************************************************************************
109 #*****************************************************************************
110 #*****************************************************************************
111 #*****************************************************************************
112 #*****************************************************************************
113 #*****************************************************************************
114 #*****************************************************************************
115 #*****************************************************************************
116 #*****************************************************************************
117 #*****************************************************************************
118 #*****************************************************************************
119 #*****************************************************************************
120 #*****************************************************************************
121 #*****************************************************************************
122 #*****************************************************************************
123 #*****************************************************************************
124 #*****************************************************************************
125 #*****************************************************************************
126 #*****************************************************************************
127 #*****************************************************************************
128 #*****************************************************************************
129 #*****************************************************************************
130 #*****************************************************************************
131 #*****************************************************************************
132 #*****************************************************************************
133 #*****************************************************************************
134 #*****************************************************************************
135 #*****************************************************************************
136 #*****************************************************************************
137 #*****************************************************************************
138 #*****************************************************************************
139 #*****************************************************************************
140 #*****************************************************************************

---

Jan 22 1992 08:24:08 makefile Page 2

74  MPUSYS = /mpu
75  76  MPPULIB = $(PROJ)\lib
77  COMSYC = $(PROJ)\COMSYS\src
78  MPUSYS = $(PROJ)\MPUSYS\src
79  80  MPUBIN31 = $(MPUSYS)\bin\R03
81  MPUBIN32 = $(MPUBIN31)\bin\R01
82  MPUBIN35 = $(MPUBIN31)\bin\��до
83  MPUBINSCB8 = $(MPUBIN31)\bin\abc8
84  85  # Common subsystem level source directories
86  HDSRC = $(COMSRC)\hr
87  88  # Logical device interface subsystem level source directories
89  LDSSRC = $(MPUSYS)\ld
90  91  # Common subsystem global include and compilation units
92  SYSDefs = $(HDSRC)\sysdefs.h
93  94  # MPU subsystem compilation units
95  96  97  98  99  100  #*****************************************************************************
101 #*****************************************************************************
102 #*****************************************************************************
103 #*****************************************************************************
104 #*****************************************************************************
105 #*****************************************************************************
106 #*****************************************************************************
107 #*****************************************************************************
108 #*****************************************************************************
109 #*****************************************************************************
110 #*****************************************************************************
111 #*****************************************************************************
112 #*****************************************************************************
113 #*****************************************************************************
114 #*****************************************************************************
115 #*****************************************************************************
116 #*****************************************************************************
117 #*****************************************************************************
118 #*****************************************************************************
119 #*****************************************************************************
120 #*****************************************************************************
121 #*****************************************************************************
122 #*****************************************************************************
123 #*****************************************************************************
124 #*****************************************************************************
125 #*****************************************************************************
126 #*****************************************************************************
127 #*****************************************************************************
128 #*****************************************************************************
129 #*****************************************************************************
130 #*****************************************************************************
131 #*****************************************************************************
132 #*****************************************************************************
133 #*****************************************************************************
134 #*****************************************************************************
135 #*****************************************************************************
136 #*****************************************************************************
137 #*****************************************************************************
138 #*****************************************************************************
139 #*****************************************************************************
140 #*****************************************************************************

---
/* DESCRIPTION: Logical Device Interface (LDI) variables and functions.
   Contains constant function parameter declarations (typedefs) 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 S5S pp. 5-6 through 5-x for more information.

   Edit History: 03/25/91 - Written by Robin T. Laird.

include LDI_MODULE_CODE
#define LDI_MODULE_CODE 14000

/* Public Data Structures: */

#define LDI_ERR_NOT_INIT 1+LDI_MODULE_CODE

/* External module global error variable. */
extern int ldi_error;

/* Public Functions: */
void ldi_init();

#endif
/* It is ndused by itself. */

#include <sysdefs.h>  // System constants and types.
#include "ldi.h"       // Logical device interface.

/* Public Variables */
/* Global module error variable, idl_error. */
/* idl_error contains code of last error occurrence. */
/* Should be set to OK after each successful function call. */
/* Variable can be examined by other software after each function call. */
XDATA int idl_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()
{
    if (idl_error == OK) { /* Assume function successful. */
        // Initialize subsystems...
    }
}
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.
## Intelligent Communications Node (ICN) Parts List

<table>
<thead>
<tr>
<th>Board Ref.</th>
<th>Quan.</th>
<th>P/N - Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>U1</td>
<td>1</td>
<td>N80C152JB-1, MPU</td>
</tr>
<tr>
<td>U2</td>
<td>1</td>
<td>27C256, EPROM</td>
</tr>
<tr>
<td>U3</td>
<td>1</td>
<td>43256AC-10L, RAM</td>
</tr>
<tr>
<td>U4</td>
<td>1</td>
<td>74HC373, LATCH</td>
</tr>
<tr>
<td>U5</td>
<td>1</td>
<td>74HC00, NAND</td>
</tr>
<tr>
<td>U6</td>
<td>1</td>
<td>SN75176B, BUS XCEIVER</td>
</tr>
<tr>
<td>U7</td>
<td>1</td>
<td>MAX233CPP, RS-232 DRIVER</td>
</tr>
<tr>
<td>U8</td>
<td>1</td>
<td>74HCT245, LINE DRIVER</td>
</tr>
<tr>
<td>U9</td>
<td>1</td>
<td>74HCT244, LINE DRIVER</td>
</tr>
<tr>
<td>C1, C2</td>
<td>2</td>
<td>33pF, 7V, (ceramic)</td>
</tr>
<tr>
<td>C3</td>
<td>1</td>
<td>4.7uF, 25V, (elec.)</td>
</tr>
<tr>
<td>C4</td>
<td>1</td>
<td>0.1uF, 16V, (tant.)</td>
</tr>
<tr>
<td>C5, C11</td>
<td>2</td>
<td>10uF, 16V, (elec.)</td>
</tr>
<tr>
<td>C6, C7, C8, C9, C10, C12, C13</td>
<td>7</td>
<td>0.1uF, 7V, (ceramic)</td>
</tr>
<tr>
<td>C14</td>
<td>1</td>
<td>0.01uF, 16V, (tant.)</td>
</tr>
<tr>
<td>R1, R2</td>
<td>2</td>
<td>200 Ohm</td>
</tr>
<tr>
<td>R3</td>
<td>1</td>
<td>8.2K Ohm</td>
</tr>
<tr>
<td>R4</td>
<td>1</td>
<td>4.7K Ohm</td>
</tr>
<tr>
<td>R5, R6</td>
<td>2</td>
<td>86 Ohm</td>
</tr>
<tr>
<td>R7</td>
<td>1</td>
<td>23 Ohm</td>
</tr>
<tr>
<td>R8</td>
<td>1</td>
<td>4.7 Ohm</td>
</tr>
<tr>
<td>D1, D2</td>
<td>2</td>
<td>LED, 2.3v Green, PC Mount, 550 Series</td>
</tr>
<tr>
<td>D4</td>
<td>1</td>
<td>Zener Diode, 5.1V, 0.25W</td>
</tr>
<tr>
<td>D5, D6, D7, D8</td>
<td>4</td>
<td>Zener Diode, 10V, 0.25W</td>
</tr>
<tr>
<td>Board Ref.</td>
<td>Quan.</td>
<td>P/N - Description</td>
</tr>
<tr>
<td>------------</td>
<td>-------</td>
<td>--------------------</td>
</tr>
<tr>
<td>Q1</td>
<td>1</td>
<td>Voltage Regulator, 5V, 1A</td>
</tr>
<tr>
<td>Q2</td>
<td>1</td>
<td>Crystal Osc., 14.7456 MHz</td>
</tr>
<tr>
<td>SW1,SW2</td>
<td>2</td>
<td>Dip Switch, 8-pin, PC Mount, SPST</td>
</tr>
<tr>
<td>SW3</td>
<td>1</td>
<td>Momentary Push Button, PC Mount, NO</td>
</tr>
<tr>
<td>J1</td>
<td>1</td>
<td>Header, 8-pin, PC Mount, Vertical</td>
</tr>
<tr>
<td>CONN 1</td>
<td>1</td>
<td>DB-25 Male, PC Mount, Right Angle</td>
</tr>
<tr>
<td>CONN 2</td>
<td>1</td>
<td>2 Pin Screw Terminal, PC Mount</td>
</tr>
<tr>
<td>CONN 3</td>
<td>1</td>
<td>Phone Jack, 6-pin, PC Mount, Right Angle</td>
</tr>
<tr>
<td>CONN 4</td>
<td>1</td>
<td>Phone Jack, 6-pin, PC Mount, Vertical</td>
</tr>
</tbody>
</table>
### Power Distribution Node (PDN) Parts List

<table>
<thead>
<tr>
<th>Board Ref</th>
<th>Quan.</th>
<th>P/N - Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>C3</td>
<td>1</td>
<td>470μF (elec.), 35V</td>
</tr>
<tr>
<td>C1, C2</td>
<td>2</td>
<td>470μF (elec.), 25V</td>
</tr>
<tr>
<td>C4, C5, C6, C8</td>
<td>4</td>
<td>560μF (elec./tant.), 25V</td>
</tr>
<tr>
<td>C7, C9</td>
<td>2</td>
<td>0.33μF (tant.)</td>
</tr>
<tr>
<td>C10</td>
<td>1</td>
<td>0.1μF (tant.)</td>
</tr>
<tr>
<td>R1, R2, R3</td>
<td>3</td>
<td>10K Ohm</td>
</tr>
<tr>
<td>R4, R5</td>
<td>2</td>
<td>560 Ohm</td>
</tr>
<tr>
<td>R6</td>
<td>1</td>
<td>1K Ohm</td>
</tr>
<tr>
<td>D1, D2, D3</td>
<td>3</td>
<td>LED, 2.3V Green, PC Mount, 550 Series</td>
</tr>
<tr>
<td>J1, J2, J3</td>
<td>3</td>
<td>2x2 PC Mount Jumpers</td>
</tr>
<tr>
<td>F1, F2</td>
<td>2</td>
<td>3A Resetable Circuit Breaker</td>
</tr>
<tr>
<td>F3</td>
<td>1</td>
<td>6A Resetable Circuit Breaker</td>
</tr>
<tr>
<td>Q1, Q2, Q3, Q6, Q7</td>
<td>5</td>
<td>IRF 9531, P-Channel MOSFET's, TO-220 style</td>
</tr>
<tr>
<td>Q4</td>
<td>1</td>
<td>7805/7808 Voltage Regulator</td>
</tr>
<tr>
<td>CONN 2</td>
<td>1</td>
<td>10 pin Vertical Terminal Strip, 8213 Series</td>
</tr>
<tr>
<td>CONN 1</td>
<td>1</td>
<td>10 pin Horizontal Terminal Strips, 8213 Series</td>
</tr>
<tr>
<td>CONN 3</td>
<td>1</td>
<td>15 Conductor Screw Terminal</td>
</tr>
</tbody>
</table>

file: pdnparts.doc
last revised: 6/10/90
<table>
<thead>
<tr>
<th>Board Ref.</th>
<th>Quan.</th>
<th>P/N - Discription</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1,C2</td>
<td>2</td>
<td>4700uF (elec.)</td>
</tr>
<tr>
<td>C3</td>
<td>1</td>
<td>2200uF (elec.)</td>
</tr>
<tr>
<td>C4,C5</td>
<td>2</td>
<td>560 uF (tant.)</td>
</tr>
<tr>
<td>C6,C7,C8</td>
<td>3</td>
<td>0.33 uF (tant.)</td>
</tr>
<tr>
<td>R1,R2</td>
<td>2</td>
<td>560 Ohm</td>
</tr>
<tr>
<td>R3</td>
<td>1</td>
<td>1K Ohm</td>
</tr>
<tr>
<td>L1,L2</td>
<td>2</td>
<td>5A Current Choke, 5200 Series</td>
</tr>
<tr>
<td>L3</td>
<td>1</td>
<td>10A Current Choke, 5200 Series</td>
</tr>
<tr>
<td>T1,T2,T3</td>
<td>3</td>
<td>15A, 28V Transient Suppressor</td>
</tr>
<tr>
<td>D1,D2,D3</td>
<td>3</td>
<td>LED, 2.3V Green, PC Mount, 550 Series</td>
</tr>
<tr>
<td>CONN 2</td>
<td>1</td>
<td>10 Conductor Pluggable Terminal Strip, PC Mount, 8213 Series</td>
</tr>
<tr>
<td>CONN 1,3,4</td>
<td>3</td>
<td>4 Conductor Screw Terminal Strips, PC Mount</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>24-12V DC-DC Converter, WR24S12/60K3</td>
</tr>
</tbody>
</table>

file: ppcuparts.doc
last revised: 6/12/91
DEVELOPMENT OF A MODULAR ROBOTIC ARCHITECTURE

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

Naval Ocean Systems Center
San Diego, CA 92152-5000

This objective of this project is to develop the hardware and software components for constructing and controlling reconfigurable, modular robots.
<table>
<thead>
<tr>
<th>214: NAME OF RESPONSIBLE INDIVIDUAL</th>
<th>215: TELEPHONE (Include Area Code)</th>
<th>216: OFFICE SYMBOL</th>
</tr>
</thead>
<tbody>
<tr>
<td>R. T. Laird</td>
<td>(619) 553-3667</td>
<td>Code 535</td>
</tr>
</tbody>
</table>
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