NRL Report 8925

A Demonstration of Surveillance-Radar Communication

J. O. COLEMAN AND J. J. ALTER

Radar Analysis Branch
Radar Division

October 4, 1985

NAVAL RESEARCH LABORATORY
Washington, D.C.

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; distribution unlimited.
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) NRL Report 8925	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Naval Research Laboratory 6b. OFFICE SYMBOL (*If applicable*) Code 5312	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (*City, State, and ZIP Code*) Washington, DC 20375-5000	7b. ADDRESS (*City, State, and ZIP Code*)
8a. NAME OF FUNDING / SPONSORING ORGANIZATION Naval Sea Systems Command 8b. OFFICE SYMBOL (*If applicable*)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (*City, State, and ZIP Code*) Washington, DC 20362-5101	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. 62712N; PROJECT NO. SF12131691; TASK NO.; WORK UNIT ACCESSION NO. DN080-003
11. TITLE (*Include Security Classification*) A Demonstration of Surveillance-Radar Communication	
12. PERSONAL AUTHOR(S) Coleman, J.O. and Alter, J.J.	
13a. TYPE OF REPORT Final 13b. TIME COVERED FROM 10/81 TO 9/84	14. DATE OF REPORT (*Year, Month, Day*) 1985 October 4 15. PAGE COUNT 17
16. SUPPLEMENTARY NOTATION	
17. COSATI CODES FIELD / GROUP / SUB-GROUP	18. SUBJECT TERMS (*Continue on reverse if necessary and identify by block number*) Computer communication networks Radar communication Tactical data transfer

19. ABSTRACT (*Continue on reverse if necessary and identify by block number*)

Success in multiplatform sensor integration will require a robust communication capability for transferring data. One way to communicate is to use the existing radar transmitters and antennas. Consequently, the Naval Research Laboratory (NRL) has investigated techniques for radar communications, and has constructed a simple radar communication demonstration system to illustrate a concept suitable both for new radar designs and for retrofitting onto existing radars.

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT ☐ UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL M.I. Skolnik	22b. TELEPHONE (*Include Area Code*) (202) 767-2936 22c. OFFICE SYMBOL Code 5312

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted.
All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE

CONTENTS

A DEMONSTRATION OF SURVEILLANCE-RADAR COMMUNICATION

INTRODUCTION

A system to demonstrate computer-to-computer communication was built by using the transmitter and rotating antenna of the Naval Research Laboratory's (NRL) experimental L-band surveillance radar. This communication capability can be used with only a small degradation in radar performance. The demonstration system illustrates concepts on which an operational radar communication system could be built.

NRL's work focused on radars with conventionally rotating antennas, rather than phased arrays, simply because they are more common. Although phased arrays, with their beam agility, are naturally suited to a part-time communication function, there simply are not many phased-array radars operational. This report first gives an overview of the demonstration, and describes in detail the communication-link architecture which features a layered structure. Finally, the demonstration system hardware and software are presented.

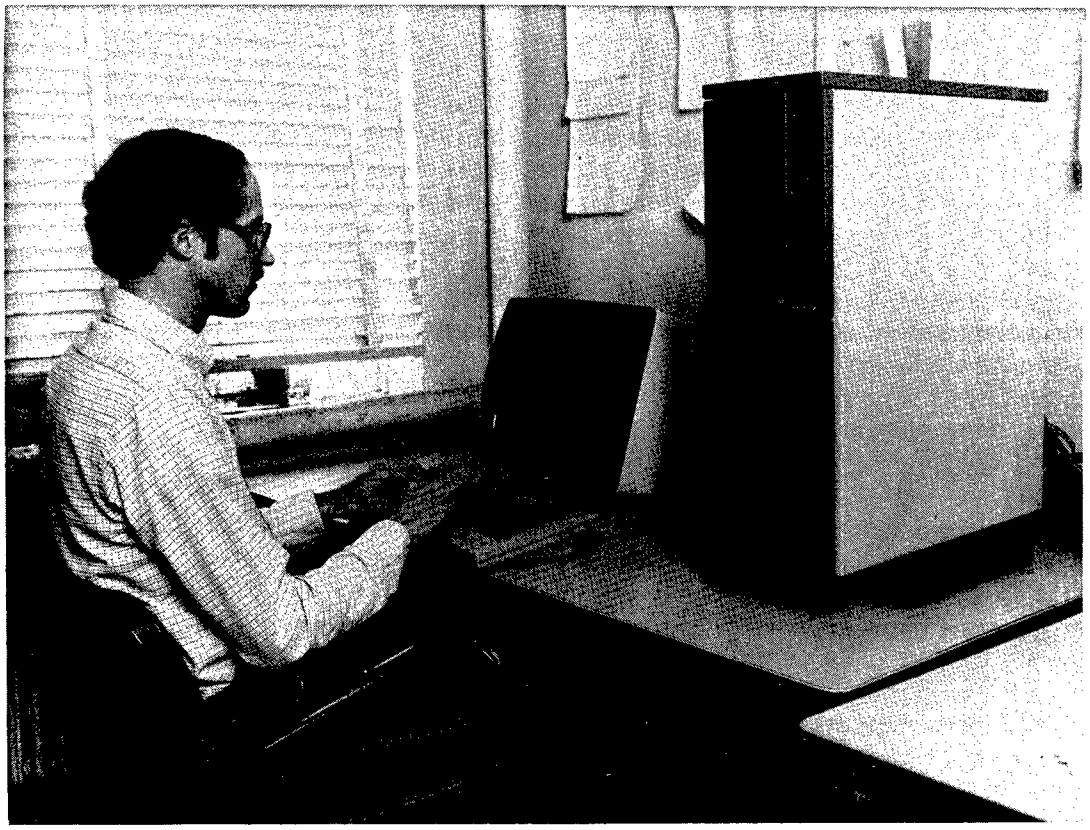
A DEMONSTRATION SYSTEM

NRL's radar communication demonstration system was built around an experimental surveillance radar located at NRL's Chesapeake Bay Detachment (CBD). A second site at CBD was chosen to complete the demonstration link. Because there is no radar at the second site suitable for the radar communication function, a simple "radar emulator" was built to substitute for a real radar at that site. The emulator cannot function as an actual radar, but it can transmit the communication waveforms. The discussion below centers on the operation of the radar-based system, with it understood that the operation of the system at the second site is similar.

The basic approach is to replace several of the radar's pulses with the output of a modem as the main beam of the radar's antenna sweeps over the receive site [1]. For the demonstration, messages can be typed into a small computer for transmission to the second site as the antenna position allows; see Fig. 1. In addition to preparing and formatting messages, the computer monitors the radar's antenna position and pulse timing. With this information, it controls the switching of the radar's high-power amplifier between radar and communication waveforms. The communication waveform is generated by a modem whose data input is provided by the computer at the appropriate times. The modem also converts incoming communication signals from the second site back into data, and passes the data into the computer for decoding and display. It is possible to observe the radar's performance in detecting aircraft while it is transmitting data across the link.

The demonstration system can be conceptually divided into two major components: the communication link and the demonstration software. Here the term communication link includes not only the communication hardware and the radar, but also all the software necessary to form a usable communication link that can be used by other software modules to communicate with each other. It therefore includes all the software that interacts with the radar or performs general communications functions. The demonstration software exists only to provide an interface to the communication link that is convenient for humans to use. It gathers messages from the computer keyboard and passes them with the appropriate control information to the communication link for transmission. It simultaneously accepts

Manuscript approved May 17, 1985.



81417(2)

Fig. 1 — Messages typed into a control computer can be transmitted through the radar

incoming messages from the communication link and presents them appropriately on the computer's video display.

The rest of the discussion of the demonstration system is divided into three parts. First, the link architecture is discussed in very general terms, the hardware components of the demonstration are described, and the structure of the system software is outlined.

COMMUNICATION-LINK ARCHITECTURE

The communication-link "architecture" is a collection of techniques used to control the interfacing with the radar, control system timing, and provide data framing and various levels of error control. Taken collectively, the services provided by the system components implemented according to this architecture allow the demonstration software to use the communication link without concern for the details of the link's operation. Ideally, the link should appear as a transparent path for data transferred between software modules at different sites.

The Layered Approach

Modern computer communication networks are generally designed around *layered architectures* [2,3]. The services needed to provide a communication path between two points are provided by entities structured into layers. The applications, which run on the computers at two sites, communicate through a communication path provided by the highest layer entity at each site. These two entities provide a predefined set of procedures called a *protocol.* The *protocol* allows the entities to communicate with each other through a more primitive communication path provided by the pair of entities of the

next lower layer at the two sites. In turn, the entities of this lower layer provide a path for the higher layer entities by using another protocol to communicate with each other through a still more primitive path provided by entities of an even lower layer. This layering continues until at the lowest layer, two entities communicate with each other by using a physical channel. To summarize, *peer* entities (entities at the same layer but at different sites) use a protocol to communicate through a path provided by peer entities of the next lower layer.

Figure 2 schematically illustrates this relationship among the various entities. The dashed lines represent communication paths that are not physical channels but are really provided by the pair of entities immediately below. For example, the path between the two "high" entities is provided by the "middle" entities. Therefore, the actual path of the data from the left "user" to the right user is down through the left "low" entity, across the physical channel to the right low entity, and up to the right user. A certain amount of overhead or control information is typically communicated between *peer* entities of the lower layers, and is never seen by the user entities.

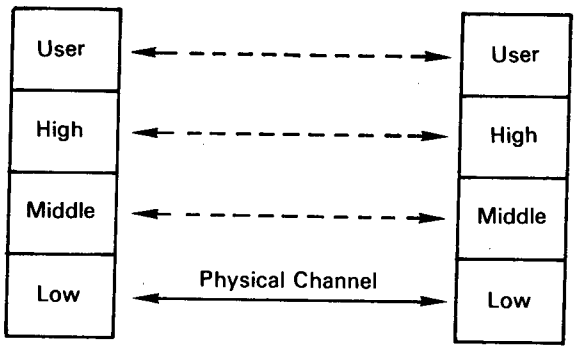


Fig. 2 — An example of layering

One interpretation of the structure in Fig. 2 is that entities at each successively higher layer add some capability, such as error correction, etc., to that provided by the entities at the layers below. The inverse viewpoint is that the entities at each layer exist to hide some information from the entities at a higher layer. The information hidden might be, for example, restrictive timing requirements or a substandard reliability level of the underlying, lower layer communication path. In the present context, the entities at each layer hide from entities at higher layers the internal details and methods of their operation, including the protocols they use to communicate with their counterparts at other sites. This allows many changes in the implementation of entities at one layer to proceed without the designer having to be concerned with effects on the operation of entities at other layers. In information-hiding terms, a layer provides the additional capability by recognizing that the entities at any one layer also hide from entities at a higher layer the "inadequacies of the services" provided by the entities at a lower layer.

Criteria for Layer Definition

Zimmerman [4] lists a set of principles to consider in defining a set of architectural layers. A subset of those principles reflects the philosophy of this design:

(1) Create a boundary at a point where the service description can be small and the number of interactions across the boundary is minimized.

(2) Create separate layers to handle functions that are manifestly different in the process performed or the technology involved.

(3) Enable changes of functions or protocols within a layer without affecting the other layers.

Other things being equal, systems of many small (in the sense of functions performed) layers tend to have more communication overhead than systems of a few large layers because the many separate layers must operate independently. A system structured into a few large layers can result in efficient operation; however, the operation of such large layers is often difficult to understand because of complexity. At the early stages of this experimental system, simplicity and reliability are more important than efficiency. In addition, we have generally tried to have the functionally different aspects of the interaction with the radar transmitter occur from within different layers. These factors have resulted in more layers in the architecture described here than are used to accomplish roughly the same objectives in various well-known communication systems [3]. The entire system described here corresponds to roughly the combination of the data-link layer and the physical layer in one representative architecture [4]. Fortunately, most of the tasks to be divided between the layers were inherently sufficiently independent such that little extraneous protocol overhead resulted from this approach.

The Layers of the Architecture

Table 1 summarizes the functions of the various layers of service in this architecture. (The warning-pulse feature of the group layer was designed into the protocol, but it was not actually implemented in the demonstration system); see Ref. 5 for more detail on the functions of the various layers and the protocols used. Figure 3 illustrates the relationship among the various entities. The labels shown in Fig. 3 on the dashed lines between *peer* entities refer to the designations we have given to the units of data that are passed through the associated communication paths. For example, the user-interface-modules at the top communicate units of data (referred to as low- and high-priority frames) back and forth through two separate paths provided by frame-service. In the physical operation of the system, the name was generally chosen to reflect where the units of data correspond to some particular unit of radar transmission.*

Table 1 — Layers of Architecture

The Layers	
User interface	Multiplexes multiple users (generally higher level protocols) through the link. Provides a convenient interface to user software modules.
Frame	Handles data frames of two priorities, providing frame demarcation and detecting and discarding frames received in error.
Codec	Does forward-error-correction to ensure data integrity if a group is lost or garbled.
Scan	Handles units of data corresponding to radar scans. Deals with addressees and antenna direction. Detects missing groups.
Group	Handles units of data corresponding to radar pulse groups. Uses a warning pulse in each group to prevent collisions with radar output at the receive site.
Pulse	Handles units of data corresponding to radar pulses, removing random trailing data added by modem. Provides data interface to hardware layers below.
Buffer	Handle bursts of data, transforming hardware data rates. Synchronizes outgoing bursts with the radar.
Modem	Handles high-speed bit streams, sending them through the radar and over the RF channel.

*Viewed from a particular receiving site, once per *scan* the radar sends several *groups*, each of which contains several *pulses*.

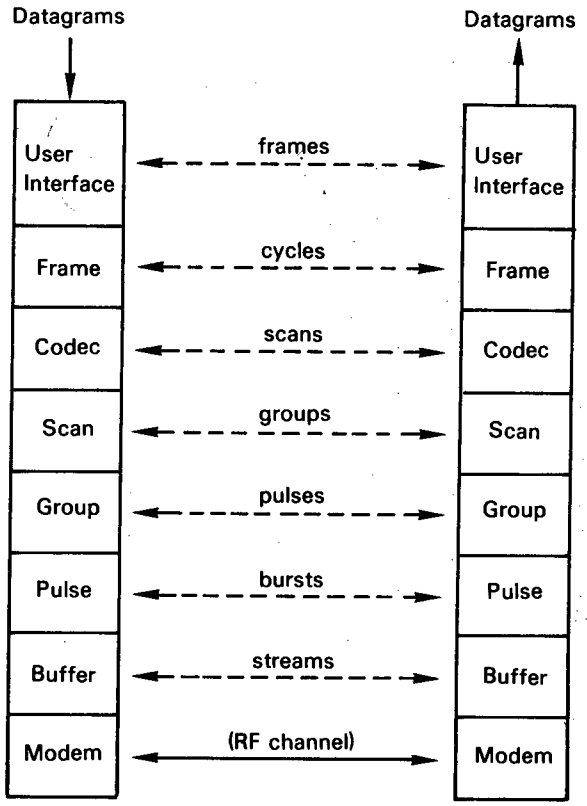


Fig. 3 — Relationships between the layers

Some of these layers are hardware, some software, and some bits of both. The modem and buffer layers are implemented in hardware. The scan, group, and pulse layers are largely implemented in software with hardware interfaces to the radar as well as to the buffer layer below. The user-interface, frame, and codec layers are entirely software with no hardware interfaces of any kind.

Although only a two-site demonstration system was built, there is no reason why this architecture cannot accommodate multiple two-site links by using a single radar transmitter. The transmit-site entities can be (and should be) shared between links to several receive sites. A similar statement applies to a receive site that has links to several transmit sites.

For a simple demonstration link, this set of layers is adequate. However, other layers, higher than those listed here (or, alternatively, between user-interface-service and frame-service), would need to be defined in an operational network of these links. The first such additional layer would almost certainly be a network layer. A network layer would handle forwarding of datagrams over multiple hops, making all the routing decisions. If this communication system were to be interoperable with other DoD computer-communication networks, a layer conforming to the DoD-standard *Internet Protocol* [6] specification should probably reside above the network layer. Further, the associated DoD-standard *Transmission Control* [7] and *User Datagram* [8] protocols may be appropriate to provide virtual-circuit and datagram service.

DEMONSTRATION SYSTEM HARDWARE

Figure 4 shows the major hardware components of the demonstration system at the radar site. The computer, a Convergent Technologies workstation, serves a dual role; it is host to the demonstration software, and it is the "link controller," running the communication-link software that implements

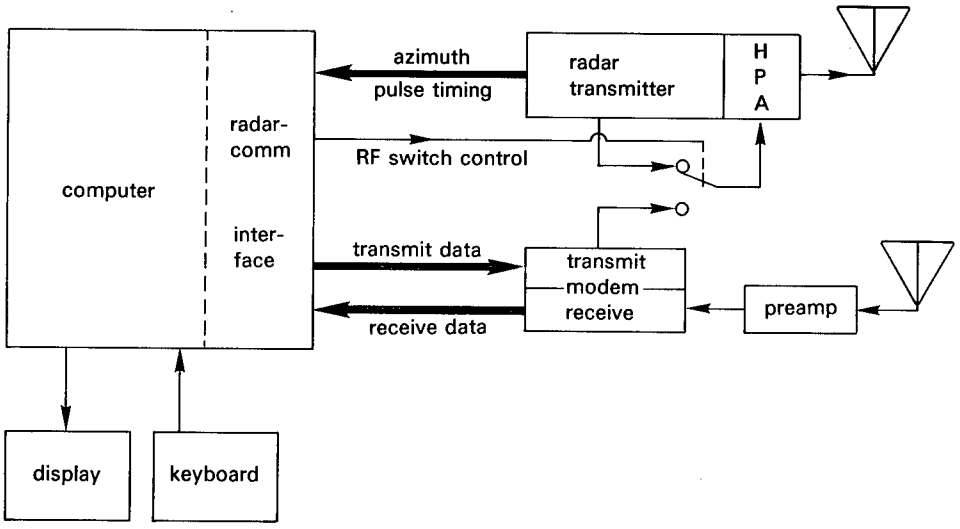


Fig. 4 — The demonstration-system hardware at the radar site

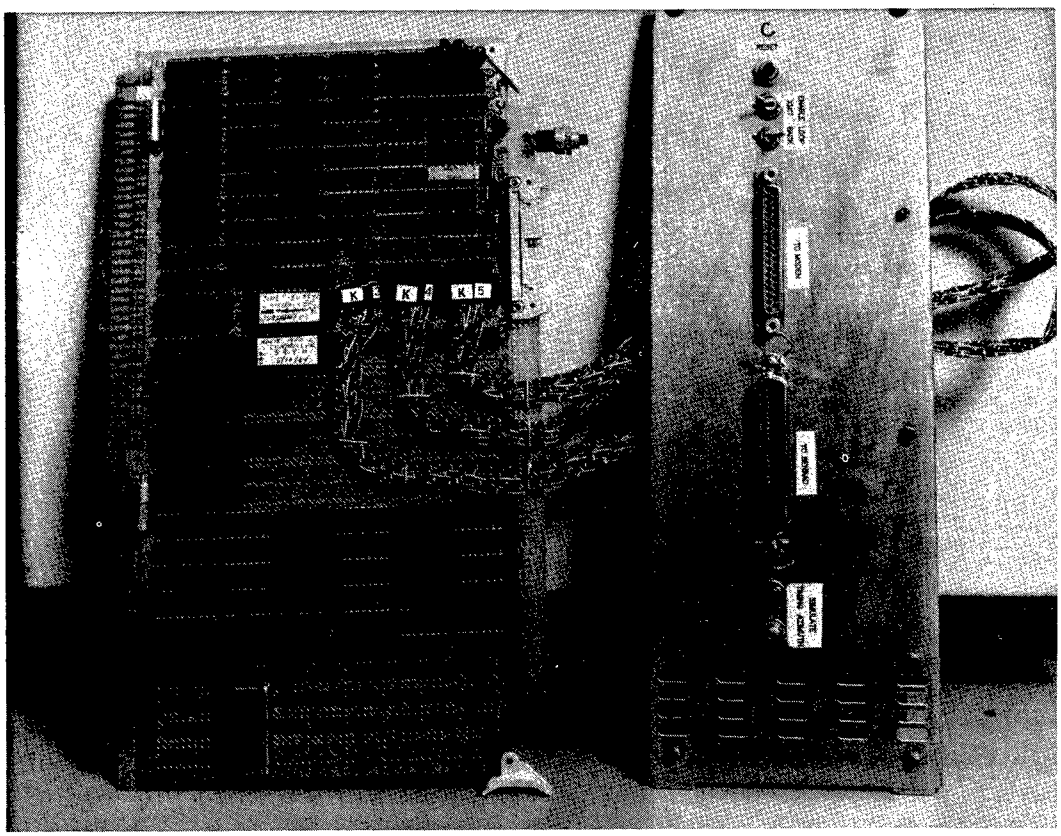
the communication-link architecture described above. The radar communications interface is a wire-wrapped card containing 124 ICs and plugs into the computer's multibus. It interacts with the software via DMA (direct memory access), port (programmed) I/O, and interrupts. This interface provides high-speed buffering of data (the buffer layer in the communication-link architecture) to and from the modem, and it controls the timing of transmit operations.

The Radar Communications Interface Card

Most of the hardware complexity of this system is in the interface card (pictured in Fig. 5). Its functions are described here in terms of the actions that take place in and around the interface to transmit and receive data.

The hardware sequence of events to transmit a communication burst is as follows. First, the communication software orders the interface to transmit a data burst by writing the memory address of an internal buffer that contains the outgoing burst to a particular I/O port on the interface card. The interface can then use DMA to copy the data from the computer's memory to an on-card high-speed buffer. Control information is copied with the data to indicate the antenna azimuth and the pulse ID (within the radar pulse group) at which the data is to be transmitted. The interface uses an interrupt to notify the software when DMA is complete. When the radar's antenna azimuth lines indicate the required antenna direction, and the pulse-timing lines from the radar transmitter indicate that the selected pulse is about to begin, the interface switches the radar's high-power amplifier (HPA) input from the radar's own waveform generator to the modem transmitter's output. The data burst is then passed to the modem transmitter where it is converted to an RF waveform for transmission through the radar's HPA and antenna. When transmission of the burst is complete, the HPA input is once again switched to the radar's waveform generator so that the radar waveform can be transmitted.

The interface simultaneously handles incoming data from the modem receiver. The software first writes the memory address of an empty buffer in the computer's memory to a particular I/O port on the interface card. The modem receiver passes the data burst to the interface as it is demodulated, and the data burst is then queued in an on-card high-speed buffer. DMA is used to transfer the data to the buffer in the computer's memory. An interrupt signals the software that the buffer has been filled.



81417(1)

Fig. 5 — Hardware interface to the modem

To allow multiple transmit and receive buffers to be processed by the interface in quick succession, the interface provides storage for multiple buffer addresses. Since all instructions to the interface from the software are provided by writing buffer addresses to I/O ports, this allows the relatively slow software to work ahead of the higher speed hardware to some extent. On transmit, this allows an entire scan's worth of bursts to be specified to the interface before the first is transmitted. On receive, it allows the interface to have enough buffer addresses on hand to handle an incoming scan of data without interacting with the software. This buffer-address queuing therefore greatly reduces the requirement for fast interaction between software and hardware. The software can interact with the interface on a scan-by-scan rather than a pulse-by-pulse basis.

The Modem

The modem for the demonstration system was designed to be simple and robust. We chose frequency-shift keying (FSK) as the modulation type for this demonstration because: (1) it is simple to implement; (2) it is so tolerant (at high-modulation indices) of various types of distortion that it was not even necessary to take the time to characterize the distortion levels in the transmitter of the particular radar we were using; and (3) for this demonstration at least, we could afford the high bandwidth per bit transmitted.*

The demonstration-system modem operates at an instantaneous data rate of 5 Mbit/s, transitting a lower frequency tone for 200 ns to send a zero, and transmitting a tone 12 MHz higher for 200 ns to

*We are not suggesting by this decision to use FSK for the demonstration system that FSK should be used operationally. FSK should probably *not* be used in an operational system because its spectral efficiency is so low.

send a one. The modem transmitter precedes each transmitted data burst with a preamble comprising a 17-bit prefix of alternating ones and zeros followed by a fixed 7-bit sync code. The alternating prefix allows the modem receiver time to discover the correct clock phase. The sync code indicates to the modem receiver that data are about to begin. The sync code can be recognized correctly even with a transmission error in one of it seven bits [9].

DEMONSTRATION SYSTEM SOFTWARE

The Software Interface to the Communication Link

The communication-link program and application programs that use the link can run simultaneously on the Convergent Technologies workstation that serves as the link controller. The communication-link program is set up in such a way that the link can be used by more than one application program at once. The link appears as a resource, much as do various operating system features.

An application program, for example, the demonstration software, interacts with the communication link by using the operating system to send "requests" to a particular "exchange" (an exchange serving the role of a mailbox). The communication-link program waits for requests to arrive at the exchange and acts on each in turn. Each request contains a request code, identifying the purpose of the request, and it also contains the ID of the "response exchange" to which the communication link should send a response to the request. The requesting program will wait at the specified response exchange for a response to its request.

In addition to a request code and response exchange ID, each request must include information specific to the particular type of request. For example, a request to transmit a datagram would include the ID of the site to which it is to be sent, the ID of the recipient (program) at that site, the priority, the address in memory where the datagram contents begin, and the length of the datagram. For a receive request, the address and length of an empty datagram buffer must be provided, along with the ID of the program making the request. A response to a receive request would be generated only when the buffer had been filled with data from a suitably addressed incoming datagram. The response would include the length of the datagram, its priority, and its site of origin. In addition to requests to transmit and receive datagrams, requests may be sent to update the stored azimuth of a receive site, to open or close an error-logging file, and to cancel a previous request (not yet responded to).

The Communication-Link Program

The communication-link program comprises roughly 5000 lines of Convergent Technologies Pascal divided into 21 independently compiled "units."* The program runs as seven independent processes (threads of control), which communicate and synchronize their activities by passing messages to each other via exchanges. An interrupt handler, triggered into execution by the modem interface, activates appropriate processes by passing messages to them.

Figure 6 shows the overall structure of the communication-link program. The seven units that implement the protocols are lined up, left to right, from the vertical dashed line on the left to the multibus on the right. The dashed line represents the boundary between the communication-link program and the application programs. Each of the units shown implements communication functions corresponding to the layer of the architecture associated with its name. In Fig. 6, a unit is represented as a tall rectangle with its name over the top and is subdivided by solid horizontal lines into its major routines. For example, the CodecUnit contains routines CodecSource and CodecSink. The intent here is to show the major structure of the program into processes, not to give a detailed picture of the entire program. Consequently, only routines that are called from outside the unit are shown. Most of these

*A Convergent Pascal "unit" is a collection of constants, types, variables, procedures, and functions that can be used by other programs or units through an explicitly provided interface. It is similar in concept to an Ada package.

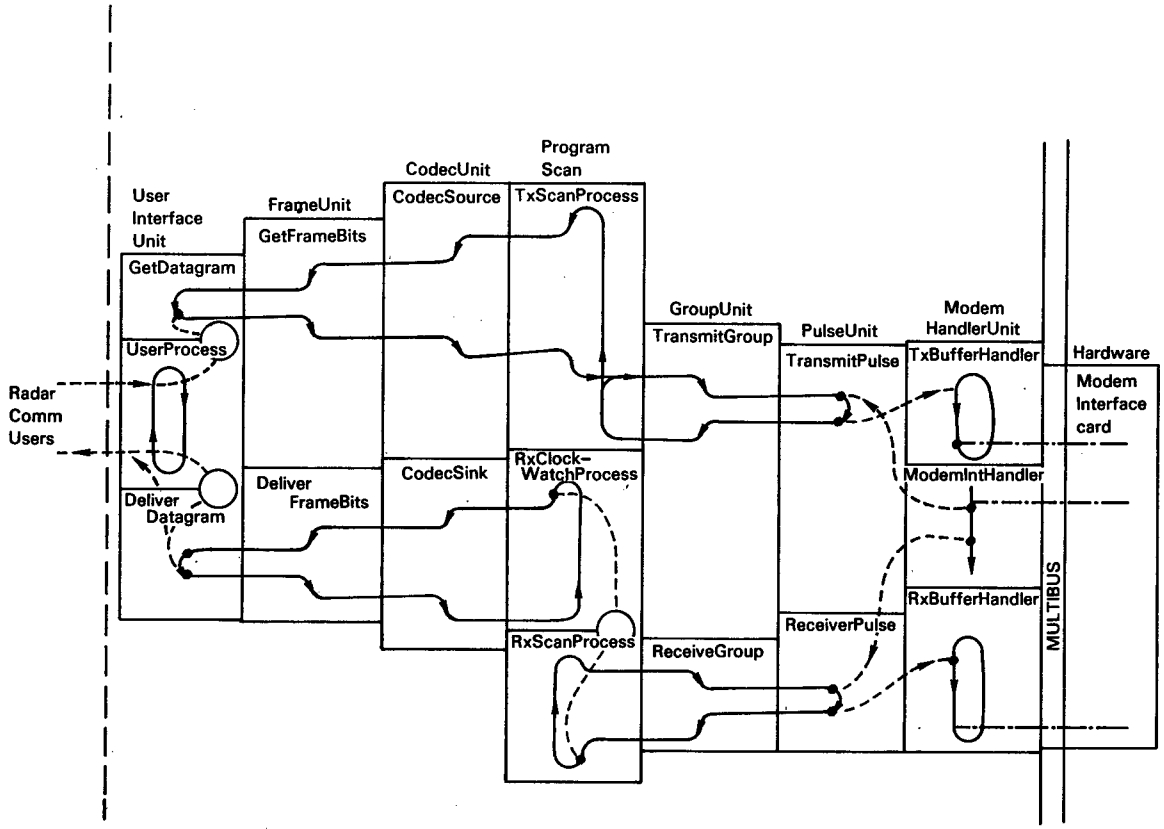


Fig. 6 — Unit and process structure of the communication link program

routines call other routines internal to the same unit in order to perform their functions. These internal routines are not given here. Likewise, the various utility units whose routines are called by those of Fig. 6 are not included.

Routines whose names end in the word "process" are run simultaneously as separate processes (also the two "BufferHandler" routines in ModemHandlerUnit). Each contains an infinite loop, so none of the processes terminates. The threads of control of the processes are represented in Fig. 6 by the solid lines with arrows that course through the various routines called. For example, RxClockWatchProcess contains an infinite loop, and inside that loop it calls the routine CodecSink in the CodecUnit. CodecSink in turn calls DeliverFrameBits in the FrameUnit, and DeliverFrameBits calls DeliverDatagram in the UserInterfaceUnit.

A circle lying on the boundary between two routines running in two different processes represents a shared database that is accessed independently from the two processes. For example, in the UserInterfaceUnit, DeliverDatagram shares a database with UserProcess. Dashed lines connect the databases with the process' threads of control at the approximate point of database access. Each database is locked in such a way that it can be accessed by only one process at a time. If a process needs access to a database that is in use, it halts execution until the database becomes available.

The dashed lines with arrows in Fig. 6 are exchanges where messages can be queued for another process to read. For example, routine TransmitPulse in PulseUnit sends messages via an exchange to TxBufferHandler in ModemHandlerUnit. The arrows show the direction of message passing. A process needing to receive a message from an empty exchange (no messages queued) automatically halts execution until a message is sent to that exchange by another process. In this way, exchanges are used to synchronize actions performed by different processes.

Interactions with multibus hardware are indicated in Fig. 6 by lines of alternating dots and dashes. These interactions were intentionally confined to a single unit, ModemHandlerUnit, to simplify program maintenance as the hardware design evolved.

The one thread of control shown in Fig. 6 that is not a loop is that of the routine ModemIntHandler in ModemHandlerUnit; this is the interrupt handler. It is triggered into execution by the operating system when an interrupt appears on the multibus from the modem interface card.

The functions of each of the processes in the communication-link program will now be discussed briefly. Very little will be said here about the actual processing of the data by the various routines; that information is described elsewhere by Coleman [5]. The discussion here will center on the dataflow between the various parts of the program. The name of the top-level procedure in which the process runs will be used as the process name.

The UserProcess in the UserInterfaceUnit receives, processes, and responds to requests from the applications using the communication link. Outgoing datagrams are queued in a database where the GetDatagram routine can later remove them. The database shared with the DeliverDatagram routine contains two queues. One is a queue of incoming datagrams. Requests from the application software for incoming datagrams provide buffers into which incoming datagrams can be copied. When such a request arrives, it is filled, if possible, from a datagram in the queue. If there is no suitable datagram in the queue, the request itself is queued in a second, request queue. Incoming datagrams, provided by the routine DeliverDatagram, are put into the datagram queue only if no match with a queued request is possible. This dual queue arrangement allows matching of incoming datagrams with requests from application software regardless of whether the datagram or request arrives first. A refinement that could be easily added is automatic expiration of old datagrams in the datagram queue.

The TxScanProcess in program scan does most of the work of preparing outgoing data for transmission. It calls CodecSource to obtain a scan's worth of data for transmission. It adds overhead information (see Ref. 5), waits for the radar antenna to become positioned to an azimuth somewhat before the azimuth of the receive site, then calls TransmitGroup on each group of data (corresponding to a radar pulse group) in the scan. After a suitable interval, the sequence is started again.

The data passed from CodecSource to TxScanProcess was ultimately obtained (except, of course, for overhead bits) from GetFrameBits. GetFrameBits, in turn, obtained its raw data for processing from GetDatagram, which has access to the datagram queue where UserProcess stores outgoing datagrams.

The data groups passed from TxScanProcess to TransmitGroup are passed on to TransmitPulse as data pulses, where they are prepared for DMA to the hardware. Empty DMA buffers are obtained by TransmitPulse by waiting at an exchange where the buffer's addresses are provided in messages from the interrupt handler, ModemIntHandler. After preparing the buffer with the data, TransmitPulse sends it as a message to another exchange, where it will be picked up by TxBufferHandler at the earliest opportunity.

The TxBufferHandler monitors the state of the transmit portion of the modem interface card, and, when the interface has room in its on-card command memory for more transmit commands, TxBufferHandler waits at the exchange where outgoing DMA buffers were placed by TransmitPulse. On obtaining a buffer at this exchange, a suitable command is sent to the hardware to cause the contained data to be transmitted.

The RxBufferHandler obtains empty DMA buffers as messages from ReceivePulse and, keeping track of the state of the receive portion of the modem interface card, keeps the on-card memory of

addresses of receive buffers as full as possible. This way, a burst of incoming data is likely to have DMA buffers ready.

When a receive DMA buffer is filled with data by the card, or when a transmit DMA buffer has been emptied by the card, a multibus interrupt is used to trigger the interrupt handler, ModemIntHandler, into execution. ModemIntHandler interrogates the card to find out how many transmit and receive DMA buffers the card is finished with. The appropriate buffers are then returned to TransmitPulse and ReceivePulse via exchanges.

RxScanProcess deals with incoming data groups. It calls ReceiveGroup to obtain each group, and, if the group obtained is marked as destined for this site, it then stores that group in a database shared with RxClockWatchProcess. ReceiveGroup calls ReceivePulse to obtain data pulses, and ReceivePulse obtains data by waiting at an exchange for ModemIntHandler to send it an incoming DMA buffer for processing.

RxClockWatchProcess simply looks into the data-group database approximately once per second until it finds that an entire scan's worth of data has arrived. It then removes those groups from the database and passes them to CodecSink for disposal. CodecSink processes the data and passes it to DeliverFrameBits, where it is divided into datagrams and passed to DeliverDatagram for disposal as discussed above.

There is one process in the communication-link program that is not shown in Fig. 6. That is the AzimuthTrackerProcess, a process that repeatedly samples the azimuth of the radar antenna, available across the multibus from the modem interface card, and uses a tracking filter to provide estimates of antenna position to those routines that need it.

The seven processes of the communication-link program each have assigned priorities that determine which will actually execute on the processor when more than one is ready to run; that is, not blocked waiting for a message at an exchange or waiting for a database to become available. The use of multiple prioritized processes to handle the asynchronous nature of the various tasks involved allowed the various portions of the program to be programmed more or less independently without need of a master "executive" or scheduling program. Structuring the routines into units allowed complementary tasks to be kept together in the source code, even when multiple processes are involved. For example, error coding is done in routine CodecSource in CodecUnit. The corresponding decoding operation is done in routine CodecSink in the same unit. Even though the two routines are executed by different processes, keeping them physically together in the source code by putting them in the same unit makes it easier for the reader of the program to verify that the decoding operation at the receive site is indeed the inverse of the coding operation at the transmit site. In addition, complementary transmit and receive routines in the same unit often share common constant definitions, data type definitions, and utility routines, all of which are included in the unit along with the major routines.

The Demonstration Program

Several application programs were written to use the communication link; they are the demonstration program and several test programs. Of these, only the demonstration program is described here.

The demonstration program interacts with the user at the keyboard and video display, and deals with requests and responses to and from the communication-link program. It allows the user to compose textual datagrams for transmission in one window on the video screen, while automatically displaying incoming datagrams in a second window as they arrive. A third screen window is provided in which the user can interact with the demonstration system by typing in commands. This command window is also used for status messages from the demonstration system; Fig. 7 shows the three windows during a

```
dest 1
Destination set to site 1
az 2020
Azimuth for site 1 set to 2020
Transmit message 1 accepted.
Transmit message 2 accepted.
Transmit message 3 accepted.
```

```
This is a test message.1
Now another
message.2
The system is in loopback test mode where outgoing messages are
turned around at the modem interface and fed back into the system.3
-
```

```
This is a test message.1
Now another
message.2
The system is in loopback test mode where outgoing messages are
turned around at the modem interface and fed back into the system.3
```

Fig. 7 — The computer screen during a test session

typical test. Capabilities provided by commands include destination site selection for outgoing datagrams and azimuth setting for receive sites.

Commands are also provided to implement a rudimentary file transfer capability that can copy a disk file from one site to another at a low priority while interactive message traffic continues (relatively) unaffected. However, the file transfer capability is included only to demonstrate the necessity of higher level protocols (than those implemented here) for real applications. File transfer in this system generally fails because no protocol is provided here for flow control, that is, feedback from receive site to transmit site to limit the rate at which data is transmitted to the rate at which it can be processed at the receiver.

The demonstration program comprises nearly 1300 lines of Pascal in four units running as six processes. Another 2100 lines of Pascal are devoted to test software.

SUMMARY

NRL's Radar Division investigated techniques for radar communications, and constructed a simple radar-communication demonstration system to illustrate a concept suitable both for new radar designs and for retrofitting onto existing Navy radars. The demonstration system illustrates ideas on which a radar-communication system could be built. By using this concept, data are sent through the radar transmitter and antenna in place of selected radar pulses as the antenna beam passes over the receive site. By using high data rates during the short intervals in which data are actually being transmitted, data may be transmitted at an average data rate of several thousand bits per second. Because the radar function is preempted for communication only a small part of the time, degradation of radar performance is minimal.

NRL's radar communication demonstration system implements a two-way link between two sites at CBD. The system at one site was built around an experimental L-band surveillance radar. For the demonstration, messages can be typed into a small computer at either site for transmission to the other as the antenna position allows. Messages are transferred using a layered set of protocols that provide frame demarcation, forward error correction, and structuring and labeling of the data pulses. In addition to preparing and formatting messages, the computer monitors various transmitter functions and the radar's antenna position. With this information, it controls the switching between radar and communication functions.

The successful demonstration of a radar communication system has shown that a surveillance radar can be used with a minimum of difficulty to transmit data with only a small degradation in radar performance.

REFERENCES

1. B. H. Cantrell, J. O. Coleman, and G. V. Trunk, "Radar Communications," NRL Report 8515, August 1981.

2. A. S. Tanenbaum, "Network Protocols," *ACM Computing Surveys* **13**(4), 453-489 (1981).

3. P. E. Green, Jr., "An Introduction to Network Architectures and Protocols," *IEEE Trans. Communications* **COM-28**(4), 413-424 (1980).

4. H. Zimmermann, "OSI Reference Model — The ISO Model of Architecture for Open Systems Interconnection," *IEEE Trans. Communications* **COM-28** (4), 435-432 (1980).

5. J. O. Coleman, "Architecture for a Demonstration Radar-Communication Link," NRL Report 8829 (in publication).

6. Jon Postel, "Internet Protocol, DARPA Internet Program Protocol Specification," RFC-791, Network Information Center, SRI International, Menlo Park, CA 94025 (September 1981).

7. Jon Postel, "Transmission Control Protocol," RFC-793, Network Information Center, SRI International, Menlo Park, CA 94025 (September 1981).

8. Jon Postel, "User Datagram Protocol," RFC-768, Network Information Center, SRI International, Menlo Park, CA 94025 (August 20, 1980).

9. J. O. Coleman, "Optimum Synchronization Codes to Follow an Alternating Mark/Space Prefix," NRL Report 8494, July 1981.