IVDS SYSTEM: CHANNEL SIMULATION AND REPEATER UNIT DESIGN

Steven Craig Franks

Thesis submitted to the Faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

in

Electrical Engineering

March, 1996

Blacksburg, Virginia

Keywords: IVDS, Channel Simulator, Message Processing

IVDS SYSTEM: CHANNEL SIMULATION AND REPEATER UNIT DESIGN

Steven Craig Franks

(ABSTRACT)

In this thesis, an Interactive Video Data Service (IVDS) is developed. This service provides a mechanism for television viewers to interact with the program they are watching. Possible interactions include purchasing products from home shopping programs and requesting information from advertisers. Within the project, two areas were focused upon: channel simulation and the Repeater Unit. Additionally, the overall system was discussed along with its background.

The purpose of channel simulation was to demonstrate the viability of the unique communication channel model proposed for the IVDS system. This channel was implemented using uni-directional transmissions, without acknowledgments.

The Repeater Unit was designed to be a message processing system, intended to relay messages from system users to the home office. The design entailed both hardware and software. The hardware requirements were for a high level design, while the software required not only design, but implementation.

Acknowledgments

- Fernando Morales for his sponsorship of this project.
- Grayson Electronics and particularly Bob Brickhouse for support with the WMI.
- Dr. Nat Davis for chairing the committee and assistance throughout the project.
- Matt Kurtin for long hours spent on the simulator; which we contributed equally to both in thought and effort.
- Boris Davidson for the channel model that makes it all work.
- Steve E. Wells and John B. Conrad of Intel Corporation for being more than understanding.
- Lia Franks for early morning editing sessions.

Table of Contents

Table of Figures and Tables

1. Introduction

1.1 Thesis Statement

The goals of this thesis are to design the hardware and software for an IVDS Repeater Unit and to verify the correctness of the Channel Model through simulation.

1.2 Background

In the spring of 1994 Fernando Morales approached the Center for Wireless Communications at Virginia Tech about researching and designing an Interactive Video Data Service system. Early in 1995, the two parties signed a contract for the center to work on the project. At that point there were several faculty and staff members from different areas of Electrical and Computer Engineering involved in the project. They in turn hired a group of Graduate Research Assistants to help carry out the research and design. This group of individuals was divided up into several areas of expertise. The initial group organization is shown in Figure 6-1.

The Controls Group was headed by Dr. N. J. Davis who hired Steven Franks and Matt Kurtin to work with him. Their initial task involved writing a channel simulator. Later, the group's main responsibility was to design the digital modules of the IVDS system which coordinated and controlled the other system modules. For the next phase of the project, they worked on different aspects of the control sections of IVDS: Franks was responsible for the Repeater Unit, while Kurtin held responsibility for the User Unit.

1.3 Layout of Thesis

Following this introductory section is the Background section which gives an overview of the IVDS system, including discussing the significance of the channel and Repeater Unit to this IVDS system. The Channel Simulator section provides an overview of the channel model and describes the details of the Channel Simulator implementation. The Repeater Unit section discusses all aspects of the Repeater Unit design including hardware and software. The Summary and Conclusions section contains thoughts on future work and possible improvements. The Acknowledgments section is followed by References. The thesis ends with two appendices: one on acronyms (designated in the text by Arial Narrow) used in this writing; the other contains a project organization chart.

2. Background

2.1 Introduction

The purpose of this chapter is to familiarize the reader with the IVDS project. Included is discussion concerning the goals and requirements of IVDS. This chapter also discusses the technical aspects of the project from a high level.

2.2 General Overview

"Interactive Video Data Service (IVDS) is a system that will provide a wireless return path for a user. The primary use of this return path will be to allow a user to interact with television programming. This programming may or may not be real time."
[Far95]

The IVDS system is based on market analysis by a corporate sponsor. The Center for Wireless Telecommunications (CWT) at Virginia Tech is handling research and design of the system for the sponsor. The system incorporates a communication method that has not previously been used commercially. The IVDS concept is sufficiently new to merit research interest.

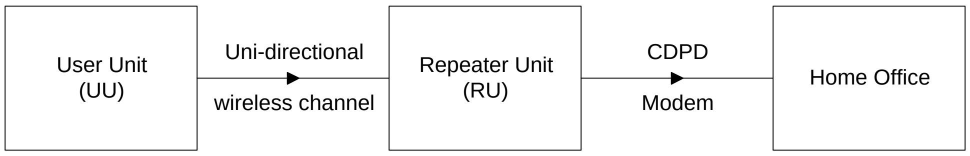

Figure 2-1: IVDS System Block Diagram

The IVDS system's primary application is to provide a method for a person to interact with conventional television broadcasts, such as advertisements or commercials. While watching television, the user is able to respond to specific events. Ordering merchandise from a television shopping network is a possible example. The system has provisions to allow users to make decisions based upon the broadcast and signal the result of their decisions. The user signals their decision by pressing a button on the IVDS User Unit (UU) which then identifies what the user is watching and sends a message to the IVDS Repeater Unit (RU), which is in the user's neighborhood. Then the RU sends the message on to the IVDS home office for processing. This path is shown as a block diagram in Figure 2-1. The advertiser is then contacted by the home office and informed of the user's interest in their product. The advertiser is charged a fee for this information. It is through fees from advertisers that the IVDS system generates revenue.

2.3 Basic Design Requirements

In this section only design requirements germane to the Channel Model and Repeater Unit are discussed. The most challenging requirement set by the project sponsor is that the UU be a completely self contained unit capable of sending messages to a remote repeater without any additional hardware in the users possession. Therefore, there may be nothing plugged into the wall, cable system, or telephone associated with the IVDS system. A high level of message reliability is also required. An other driving factor is minimizing cost of manufacture for the UU. The cost of the RU is not a significant concern due to the relatively few repeaters required compared to the number or User Units. These requirements are primarily the result of IVDS' positioning as a commercial product all stem from the sponsor.

2.4 Technical Overview

There are several major steps in the process of a user making a selection. First, broadcasts using the IVDS system require an embedded identification code (IDC) in the audio track. When a user makes their selection the UU recognizes the presence of the IDC. A message is formed that contains the IDC and a user unit identification number and transmitted wirelessly. The Repeater Unit then receives the message and tests it for errors. Error free messages are sent via a CDPD modem to the home office where they are stored. Each of these steps are discussed below.

The IDC is encoded, using digital signal processing techniques, into the audio track of the television broadcast. The encoding method, developed by the audio team of the IVDS project, is designed to be inaudible to humans when embedded in the regular audio track. The purpose of the IDC is to uniquely identify broadcasts or segments of a broadcast. The uniqueness of the IDC enables the IVDS system to record the users' selection.

The UU contains a mechanism to detect and decode the IDC. Once the IDC is extracted, the UU formulates a message, including such data as message priority, User Unit identification number (UUID), IDC, and a calculated CRC. At this point, the system is ready to send the first transmission, or original message using a direct sequence spread spectrum radio frequency transmission at 900 MHz. This transmission is uni-directional; the RU does not respond to the UU with acknowledgments. In fact, the RU cannot send a response, since it does not have a spread spectrum transmitter. Likewise, the UU does not have a spread spectrum receiver. The use of a single transmitter/receiver pair, as opposed to conventional systems which use two pairs, allows the IVDS system to reduce costs. Unfortunately, this design prevents the use of standard bi-directional communication protocols.

This acknowledgmentless transmission method creates a reliability problem, which is discussed in detail below. To deal with the reliability issue, the UU retransmits the identical message multiple times. This whole system is asynchronous; there is no way of knowing when a user will make a selection. There is also no way of keeping the transmissions of multiple users from interfering with each other. This is due to the stochastic nature of user inputs and the limited frequency bandwidth available to the system.

The RU is located on a telephone pole, or a similar location, near the user's home and receives messages from all IVDS users in its neighborhood. When it receives a transmission from one of the users, it tests the CRC value to detect transmission errors. Corrupted messages are discarded. Correct messages are stored after protocol bits are stripped away. Messages are then checked against a history list to see if they are duplicates of previously transmitted messages. Notice that if the channel is clear, most messages will be duplicates due to the retransmissions.

After a correct message is received, a single copy of it is sent to the home office by a CDPD modem. The message priority assigned by the UU is used to determine when the message is sent. As system load rises, low priority messages are delayed in order to send high priority messages. This functionality enables alternate uses of the system, such as fire alarms, to receive preferential service by the RU. These messages are of the highest priority and therefore guaranteed by the system to be processed in an expeditious manner.

2.5 Summary

The IVDS project involved many people from several disciplines which created a wireless return path for consumer use related primarily to television. The return path proceeds from a User Unit (UU) to a Repeater Unit (RU) and then on to the home office. The research interest in this project revolves around the uni-directional channel between the UU and RU. Designing a reliable uni-directional channel and the systems that will implement it, brings new challenges to field of wireless communications.

3. Channel Simulator

3.1 Introduction

The sponsor originally requested the uni-directional system to take advantage of a patent he held as well as a cut back on implementation costs. In response to this request, Boris Davidson proposed the channel model which is used by the IVDS system. However, this channel model had never been used before, and it was not known how it would perform or if it would be reliable. To answer these questions, Davidson derived a closed form analytical model, while Franks and Kurtin jointly developed a simulator for the channel. If the two independent methods could predict the same performance and reliability, they would validate each other's correctness.

3.2 Channel Model

The channel model was designed around fulfilling the requirement of constructing an inexpensive system. A major step toward that goal was achieved by creating a unidirectional channel. This required only one receiver and one transmitter: half the communications hardware normally required. While this strategy reduced the hardware required by half, it also removed the ability to use acknowledgments in communication. Without acknowledgments a highly reliable channel is required to ensure successful message transmission, which is one of the design requirements. A consumer system must maintain high levels of reliability or be faced with a loss of customer satisfaction and, ultimately, loss of business. For this system, it was determined that message loss should be less than 1 of 10^5 messages. Without acknowledgments, or any other way to establish successful message receipt, the channel relied on statistical probability. The IVDS team often referred to this method as "send and pray."

In order to create a reliable uni-directional channel, Davidson proposed sending multiple retransmissions randomly within a window of time. A complete description of the method is available in “A Novel Retransmission Technique for One-Way Packet Communication Channels.” [DaB 96]. In summary, it behaved in the following manner: a retransmission interval was assigned within which all transmissions must be sent. That time was divided into equal subintervals -- one for each retransmission. Within the constraint of its subinterval, each retransmission was sent at a random, uniformly distributed time. With this system, many messages could originate at the same instant and still have a high probability that the randomly dispersed retransmissions would not all interfere with each other. Of course, it was only necessary that one of the many transmissions of a message be received without interference for the message to be received correctly.

The channel model proposed for the system was new, hence load verse performance statistics were unknown. In order to determine if the channel was commercially viable, it was necessary to analyze the capacity of the channel. To accomplish this, two parallel tasks were undertaken. Davidson created an analytical model of the channel using a closed form solution [DaB 96]. Franks and Kurtin jointly wrote a C program to simulate the channel's performance. The belief was that if both methods yielded the same results then there is sufficient reason to believe the channel could work in a real world system.

3.3 Simulator

The simulator was designed to analyze the probability of transmission success versus system load. To accomplish this, message generation for system load was simulated using a Poisson process, which is known to be a good model for communication systems [Sch87]. The Poisson process was chosen for another reason: it was possible to create a closed form solution with a Poisson process -- which was required for the analytical model. After the original message was "transmitted", retransmissions were generated based on a uniform distribution within subintervals. Once all of the transmissions were generated, the simulator performed its real function: analyzing channel traffic to determine which messages experienced interference and the results of that interference. From there, the simulator reported statistics on channel performance. These statistics were then compared to the analytical results; a successful match would validate both methods.

3.3.1 Design

The simulator design naturally fell into three distinct phases: message generation, retransmission generation, and channel analysis. Channel analysis was the process of determining when collisions or errors occurred. Each of the three phases, plus the definitions of collisions and errors, are discussed in the following sections.

3.3.1.1 Collisions and Errors

Collisions were defined as occurring when any part of separate transmissions overlapped in time. Errors were defined as occurring any time all transmissions of a given message suffered from collisions. For the purpose of the simulation, collisions and the resulting errors were the only problems considered for transmissions. Such real system considerations as multi-path interference, signal to noise ratio, etc. were not factored into analysis at this stage.

3.3.1.2 Message Generation

Message generation was based on the assumption that messages were generated randomly throughout time. The Poisson process was used to generate these random messages. A notable property of the Poisson process is the exponentially distributed interarrival times. [HoT 93] Hence the interarrival times for a Poisson processes were calculated by:

```
InterArrivalTime = - AveInterArrivalTime * log(dRand(1));
```

where dRand produced a uniformly distributed random value in the range of [0,1). The dRand function is discussed in Section 3.3.3, Limitations. The use of exponentially distributed interarrival times led to another convenient property -- messages were generated sequentially in time. Next, messages were tagged with a unique message number. The paired data -- message number and origination time -- was stored in a linked list concluding the message generation phase.   Later, in channel analysis, it was algorithmically important that message numbers increased as message origination time increased.

3.3.1.3 Retransmission Generation

Multiple retransmissions were generated for each original message, restricted to occur within its retransmission interval. Each retransmission was placed randomly within its subinterval, enforcing the boundaries strictly to ensure that the retransmission finished by the end of its subinterval. All of these retransmissions were time stamped and stored in a linked list along with their message number, which they shared with their original message. The entire list was then sorted back into time order to ease the process of analysis.

3.3.1.4 Channel Analysis

Channel analysis was performed by scanning through the linked list detecting collisions and errors. Collisions were simple to detect. Since all transmissions were sorted after retransmission generation, only the transmissions immediately before and after the transmission in question needed to be tested for collisions. Due to the time ordering, it was not possible to collide with the transmission after the next, unless there was also a collision with the next transmission. Therefore, collision testing was accomplished in a single list traversal checking each transmission for overlapping in time with either adjacent transmission. If it did, then that transmission suffered a collision and the collision counter was incremented. While it is true that the transmission may have collided with other transmissions as well, for the purpose of this analysis the only point of significance is that it did collide with something. When the list traversal finished, the collision counter held the total number of messages experiencing collisions.

Errors posed a larger problem from a computational standpoint. Recall that a message was received in error if all of its transmissions (original plus retransmissions) experienced a collision. Therefore, while traversing the list, a message was a possible error until a transmission of that message was found that did not experience a collision. Ideally, it would have been possible to keep track of the number of collisions each message experienced in a single list traversal. Unfortunately, the data structure to accomplish that would have been either very inefficient or prohibitively large. The flow chart in Figure 3-1 outlines an algorithm that did not have significant memory requirements. That method was simple enough to implement. Unfortunately, under heavy channel traffic there was a high degree of message interleaving, which resulted in the number of list traversals approaching the number of messages in the list. This was almost an order N squared operation. Considering that the final data point in Figure 3-4 required nearly half a million messages, $O(N^2)$ operations were unacceptable.

The implemented solution was a hybrid of the above algorithm and the ideal single list traversal. The hybrid traversed the list as before, but worked on many messages at the same time, instead of just one. Recall that in the ideal case, all messages' collisions were tracked in a single pass. The hybrid tracked some messages in each traversal, and then took multiple passes. An array was used to keep track of the collision information for the messages being tracked on that list traversal. Hence, the number of traversals required was divided by the size of the array, S. This led to an operation speed-up equal to the array size. Therefore, the system was then $O(N^2/S)$. Returning to the ideal case, S equaled N resulting in O(N). However, system resources limited S to the thousands range.

3.3.2 Validation

The simulator underwent a significant testing period to verify its correctness. The first tests applied were manual analysis of the simulator's generated output. Small

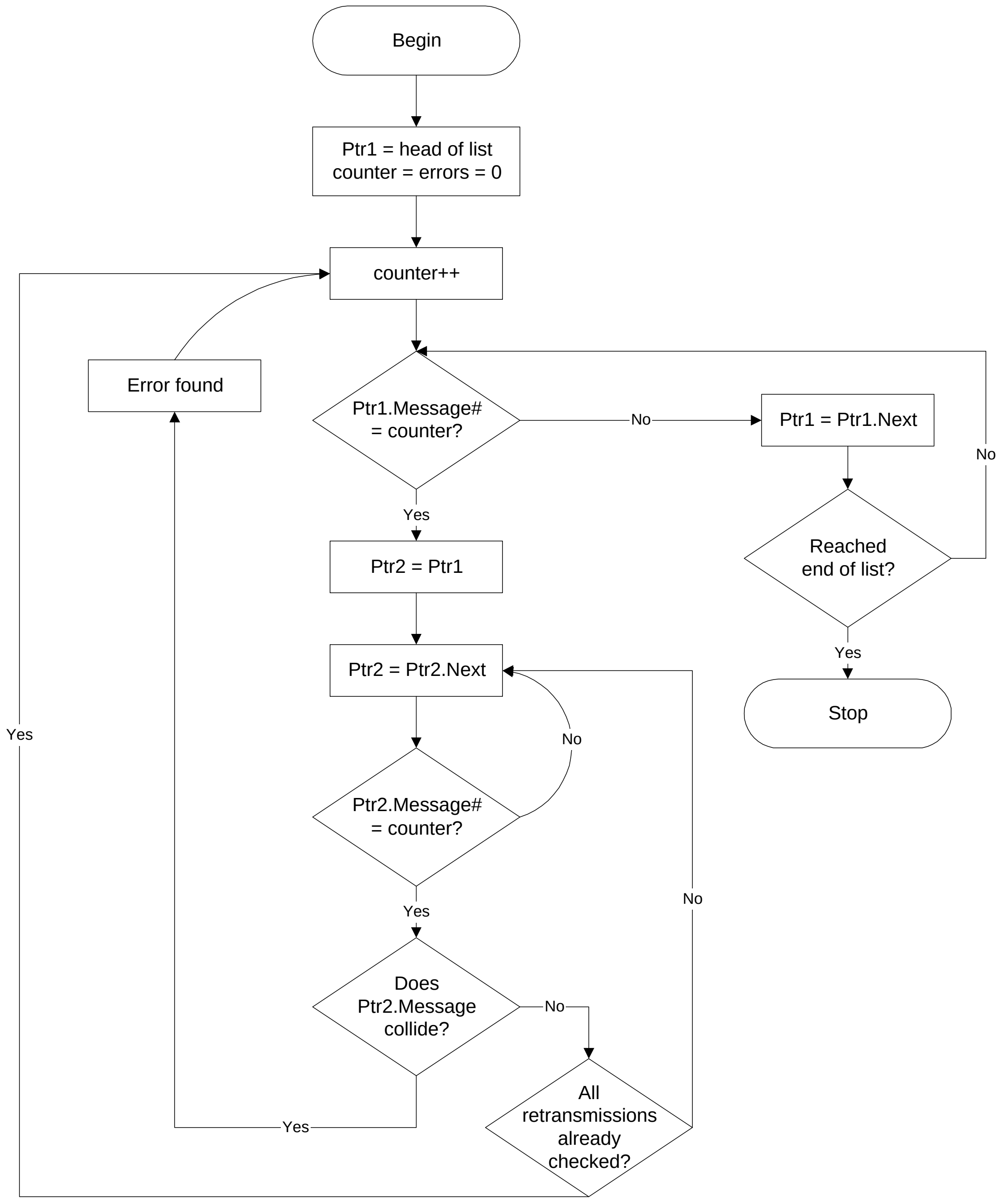

Figure 3-1: Simple Error Detection Algorithm

cases were checked for collisions and errors by hand. The problem of testing larger cases was tackled by developing a spread sheet that handled cases of a few thousand messages. The spread sheet was also used to run statistical analysis on the data to ensure the Poisson distribution was being followed. A second spread sheet was employed to test the randomness of the numbers, which were suitably random after the modifications mentioned in Section 3.3.3, Limitations, below were employed. It was believed by the designers that the simulator produced accurate results within the requirements. The final validation test was comparing the simulator results with the analytical results. The results did not match at first. After an error was discovered and corrected in the analytical model, results matched within the inherent deviation expected of a random process. Graphs displaying the near agreement are included under Section 3.4, Results.

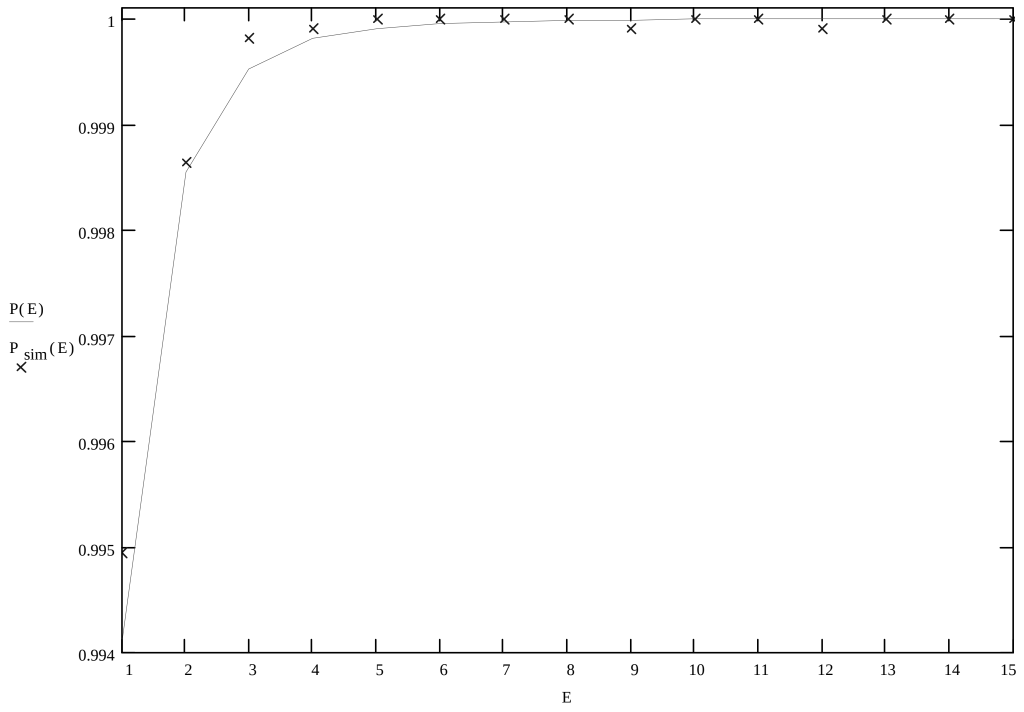

Figure 3-2: Average arrival rate of 3 messages/sec.

3.3.3 Limitations

The simulator suffered from the same general problems that most simulations incur. Since random events were being studied, many events had to be modeled to ensure statistical reliability. Unfortunately, long simulations required both large amounts of system memory and considerable time. A simulation similar to Figure 3-4 would run for an entire night. Also digital computers are only able to generate pseudo-random numbers, and if care was not taken they were not sufficiently random. Further, the standard C rand function only returns a 16 bit value, while the simulator required a 45+ bit value. The long values were achieved by concatenating three 16-bit values together. The problem of insufficient randomness was solved by creating a table of values on startup and then randomly picking a value from the table whenever a random number was needed. Once a value from the table was used it was replaced by a new value. This method produced sufficiently random results for the purposes of the simulation. Calculating six random numbers for every one actually used in the

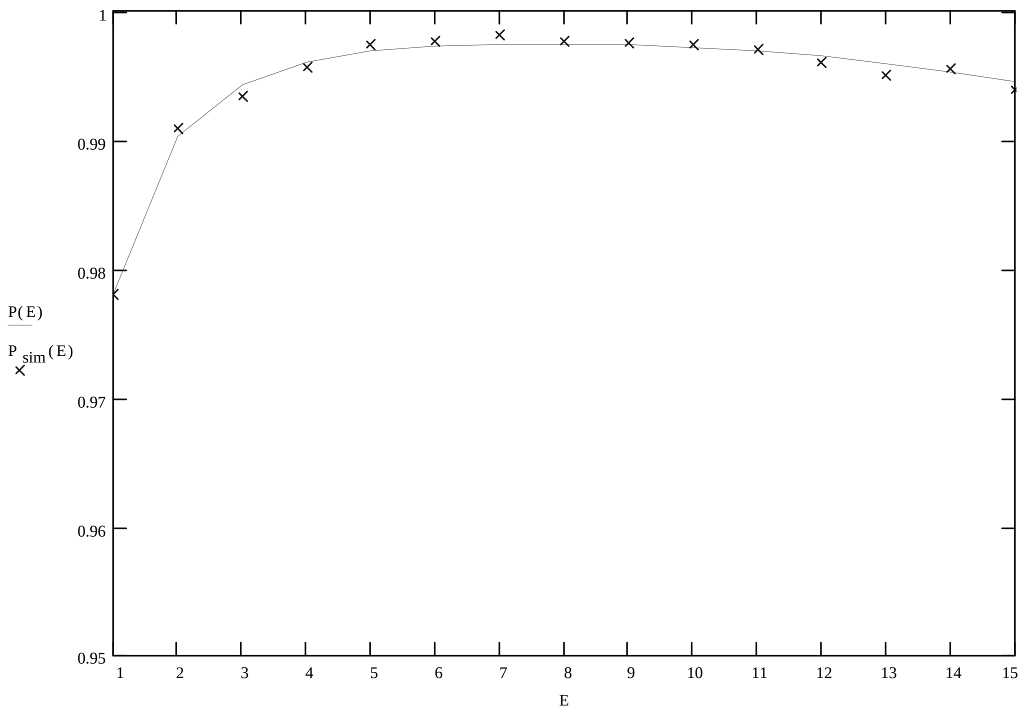

Figure 3-3: Average arrival rate of 6 message/sec.

simulator did slow things down a bit, but message generation and retransmission generation took relatively little time compared to channel analysis.

3.4 Results

This section includes results from the analytical model and from the simulator, demonstrating the near agreement of the two systems. Minor variations in the results do exist; however, they can be attributed to numerical uncertainty inherent in random processes. The channel parameters used to generate the graphs were as follows: message length of 6.67 ms, retransmission interval of 60 seconds, and simulation over the period of an hour. The longer the simulation period, the more accurate the results, and the more resource intensive the task. Simulating for one hour created a system that removed most of the numerical uncertainty while still being executable with the available computing resources. The one hour simulation generated

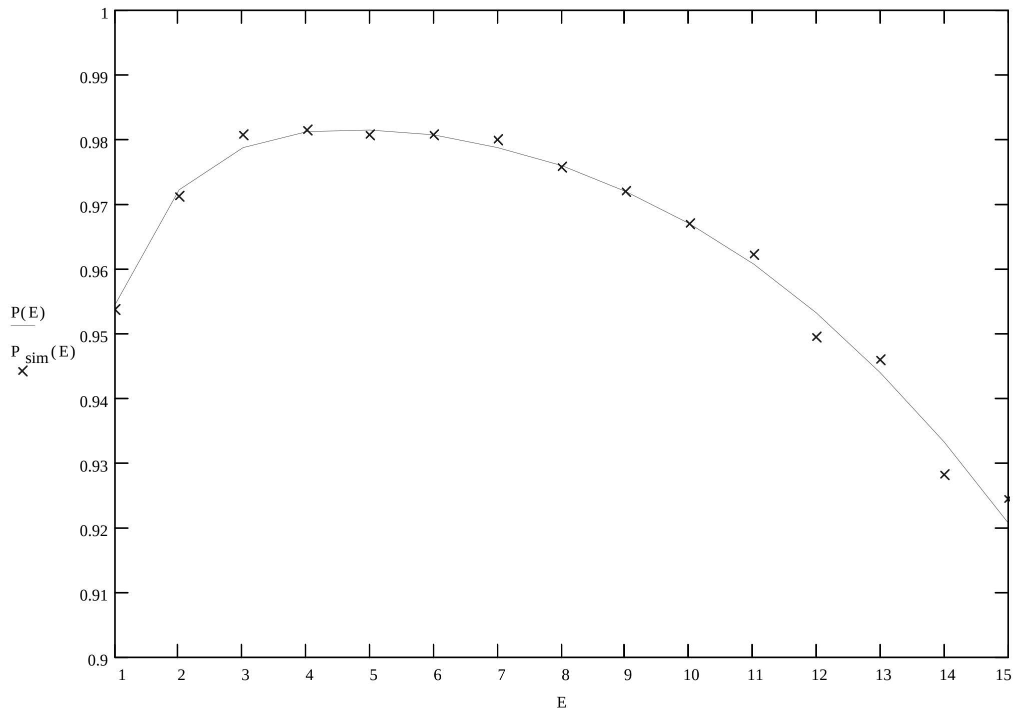

Figure 3-4: Average arrival rate of 9 messages/sec.

approximately 10,800 messages for analysis in the smallest case of an average arrival rate of 3 messages per second. That volume of messages caused the impact of any single message to represent only 0.009% of the overall results. For the larger cases with average arrival rates of 6 and 9 messages per second, the single message impact decreased even further.

The three graphs (Figure 3-2, Figure 3-3, and Figure 3-4) show the probability of successful message transmission, P(E), versus the number of retransmissions, E, for average arrival rates of 3, 6, and 9 messages per second from both simulation and the analytical model. The simulation data is marked with points while the analytical model results are denoted by a line. Notice that a different vertical scale is used in each of the figures to enhance clarity in the regions of interest.

Observe the slight drops in reliability at 9 and 12 retransmissions. These are attributed to the aforementioned numerical uncertainty, since those drops are the result of a single message experiencing an error.

An important behavior demonstrated by the graphs is that increasing the number of retransmissions beyond a certain point actually decreases the system reliability. This behavior is due to channel saturation and is particularly evident on Figure 3-4.

3.5 Conclusions

The simulator was successful in its goal of verifying the results with the analytical model. The existence of two independent methods of establishing channel performance proved to be a suitably convincing argument for the viability of the channel. The results clearly show that the goal of an error rate of less than 1 of 10^5 messages was achievable, thereby removing much of the “pray” aspect of sending messages. This early success on the project created a confidence in the technology which resulted in the continuation of the project under the industry sponsor.

4. Repeater Unit

4.1 Introduction

The Repeater Unit (RU) is, in essence, a message handler. Its purpose is to receive messages from the User Units (UU) and process them. Messages are stored, discarded, or forwarded to the home office as necessary. Custom hardware was designed for this purpose and a company, Grayson Electronics, was contracted to build the hardware.

The sections below describe the details of the Repeater Unit, beginning with a design overview. Next is a description of the hardware design; both the original and the final design. The software design section includes a detailed description of the main processing loop. The final section of the chapter are conclusions.

4.2 Design Overview

The RU functions mainly as a message handler. As a message handler, its primary design concern is the message flow path. This path winds its way though most of the RU's components, both hardware and software. Each component will be described in detail in following sections. Figure 4-1 is the system block diagram.

The message flow path begins with the reception of a message by the RF receiver. As messages are received, they are passed to the decoder board. Note in Figure 4-1 the receiver/decoder pair is shown as the IVDS Receiver. The decoder board contains the spread spectrum decoder and an 8051-series microcontroller. The microcontroller is used to initialize the receiver and spread spectrum components, and temporarily store messages as they are received. The decoder is also responsible for sending a "Data Valid" signal to the High Level Control Module (HLCM) whenever a message is received. The HLCM then reads the message from the 8051 across a serial link.

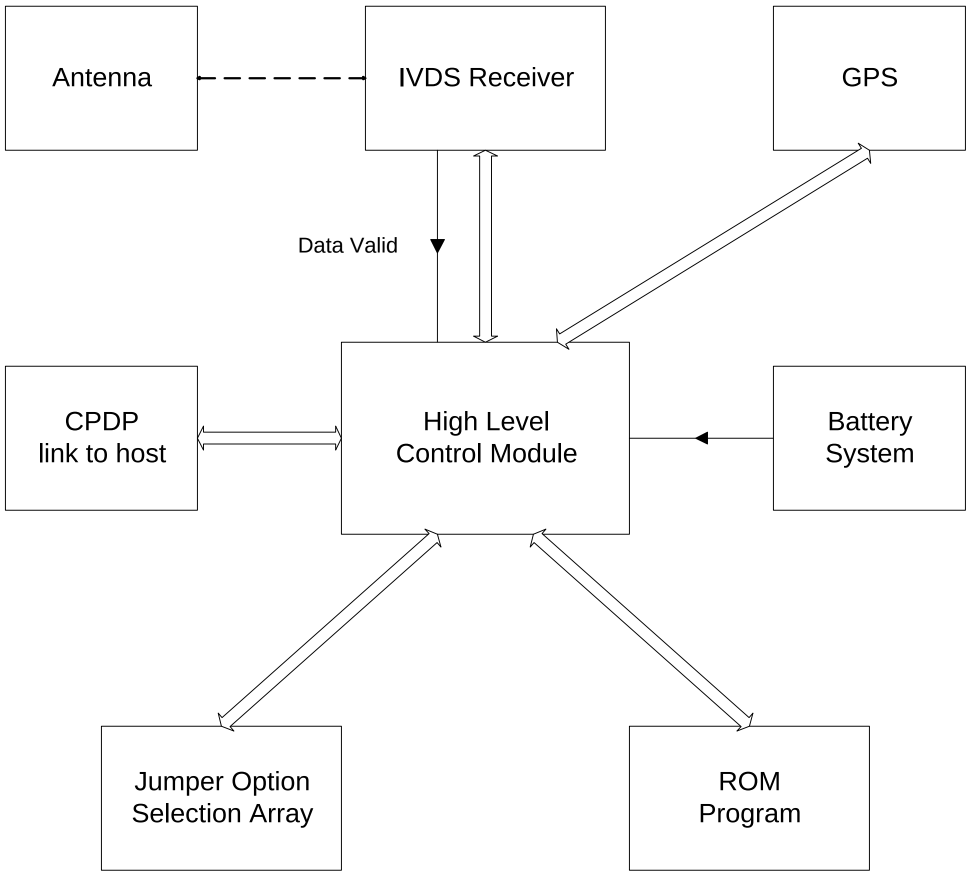

Figure 4-1: Original RU Block Diagram

The HLCM is controlled by the RU software and all messages read from the 8051 are placed in the Inbound Queue. Messages are moved from the Inbound Queue to various Priority and History Queues depending on the message's priority. From the Priority Queues, messages move to the Outbound Message Queue, at which point the RU software is finished with the messages. At periodic intervals, the Outbound Queue is emptied by an interrupt service routine which streams the messages through a serial port to the CDPD modem. The modem transmits the data to the home office.

4.3 Hardware Design

When the project began, RU hardware design was assigned to the Control Group of the IVDS project and from there to this author. The requirement was for a block level design including all major system components, a parts list, and memory map to be developed and turned over to Grayson Electronics for actual fabrication. The system block diagram is included as Figure 4-1, in which the thick lines in the diagram represent data flow, either through serial lines or data busses, while the thin lines represent signals.

Following is a description of the parts of the block diagram and how they interact. The IVDS receiver is the radio/decoder pair which will receive messages from the UU and is attached to the antenna system. The receiver is responsible for sending a Data Valid signal to the HLCM any time it receives a message. The message will then be read from the receiver by the HLCM across a serial line.

The Global Positioning System (GPS) is also attached by a serial line. The GPS was requested by the sponsor to enable tracking of system movement. When this system was designed, the requirement was for the system to be powered by batteries. The battery system was required to include circuitry to send a signal to the HLCM if battery power began to ebb.

The HLCM contains several components which are detailed in Table 4-1. The HLCM's purpose is to control activity within the repeater as a whole, and is the focus of this discussion. The CDPD modem is intended to be a standard modem using the AT command set and accessed by a serial link. The jumper option selection array is intended to allow certain system parameters, such as the frequency of sending system status messages to the home office, to be reconfigured after the system code is burned into PROM.

4.3.1 High Level Control Module

The HLCM components are listed in Table 4-1 and do not include the other sections of the RU, such as the antenna or battery. The memory map, also shown in Table 4-1, was laid out with an eye towards expansion. For instance, the system data space was not expected to exceed 2 MB, but space was left for eight just in case. The original design called for 2 MB of DRAM. The majority of that space being reserved for unprocessed message storage. A 256K PROM was expected to be sufficiently large to store the system code and still have room for later expansion. The clock/calendar was to maintain system time since several system events are time based. Two serial ports are absolutely necessary for the system: one for the receiver and one the CDPD modem. At this stage of the IVDS system, there were plans to implement only one channel, and hence there would be only a single receiver/decoder pair. An additional two serial ports were intended for the GPS and for debugging purposes while in the development phase. The Dual Universal Asynchronous Receiver Transmitter

Table 4-1: HLCM system components

Part	Description	Address
DRAM	Data space memory	00 0000h
Boot PROM	Code space memory	80 0000h
MK48T02	Clock/Calendar	A0 0000h
MC68681 (2)	DUART serial I/O controller (use 3.6864 MHz crystal)	B0 0000h B1 0000h
MC68230	Parallel interface/Timer	C0 0000h
Status LED	(write-only)	D0 0000h
Jumper Array	8 jumpers (read-only)	D1 0000h
MC68EC000	Low power microprocessor	N/A
Parallel Port	(for expansion)	N/A
Serial Port (4)	(for decoder and CDPD interface; plus expansion)	N/A

(DUART) chips require a 3.6864 MHz crystal to allow them to implement the standard baud rates - 9600, 19200, etc. The timer was required for timing short duration system events. Since the same chip also implements a parallel interface, a parallel port is was added for future expansion. The MC68EC000 is a core compatible low power version of the MC68000, a reasonably powerful full feature microprocessor.

4.3.2 Wireless Measurement Instrument

When the original RU block diagram was shown to the President of Grayson Electronics, he noticed the close similarity it bore to a system they had already built. That system, the Wireless Measurement Instrument (WMI), turned out to be a useable platform for the RU. It was adopted for use in the IVDS system. Figure 4-2 is a block

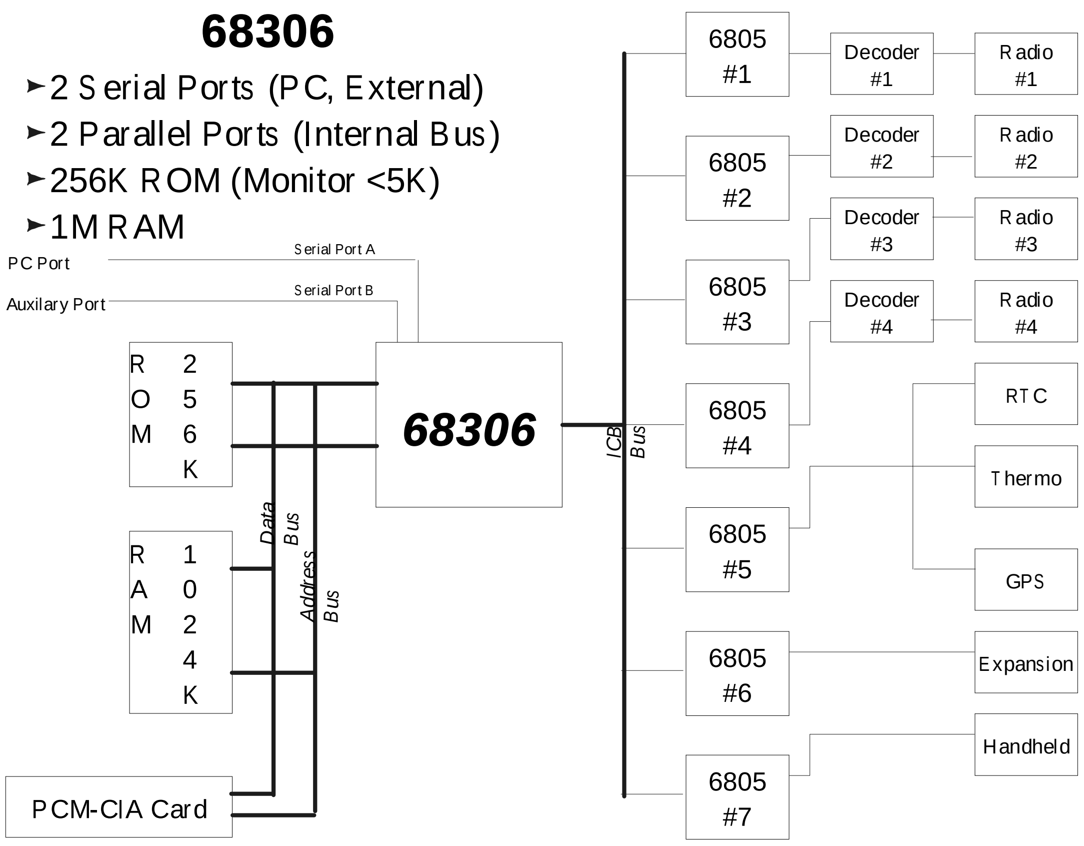

Figure 4-2: WMI Block Diagram (Courtesy Grayson Electronics)

diagram of the WMI.

Briefly, the WMI has the same goal as the RU. It receives data from radio/decoder pairs, processes it, and sends it on. The largest divergence in hardware design is the introduction of the 6805s. Their purpose in the system is to act as buffers between the 68306 and the decoders. This allows the decoders to communicate using serial data to the 6805s while the 68306 can listen on a data bus.

The WMI, already a commercial product itself, was complete with a functional software kernel. This kernel handled the interface between all applications software and the WMI hardware. In the context of the WMI the RU software is considered an application. The kernel proved itself to be a solid foundation for the RU software. An interesting point is that the system design underwent little change from before and after the introduction of the WMI, since the WMI was basically designed in the same fashion as the RU was envisioned; it was just more elaborate, being a full blown commercial product. From the RU software's stand point the largest difference was the kernel -- it provided a software layer between the hardware platform and the RU software, considerably reducing the coding effort. Without the kernel, all hardware interface routines would have been written into the RU software itself.

To highlight the changes introduced with the WMI, consider the message path. In the original design, messages traveled straight from the decoder to the Inbound Queue controlled by the 68000. With the WMI the 8051 forwards each message to a 68HC05 microcontroller that acts as a buffer. From the 6805, the message is sent to the 68306 microprocessor where the kernel resides. The kernel decoder buffer receives the message and stores it until a request is received from the RU software. At some later point, the RUS polls the decoder buffer transferring messages into the Inbound Message Queue.

To emphasize the similarities between the two designs, consider the 68306 found in the WMI. It is based on the MC68000 core and is fully compatible with the MC68EC000; additionally is has an on board DRAM controller. Lastly, it also contains a 68681 DUART with a 3.6864 MHz crystal!

4.3.3 Summary

The original design of the HLCM would never be implemented. The design that Grayson Electronics previously built implemented all of the desired functionality of the HLCM, with some extra features. It should be noted that while the two systems contained many similarities, the only instruction given to the author was to use Motorola parts. There was no previous knowledge of the WMI before or during the design of the HLCM. Yet, the WMI fit the design requirements almost perfectly, which lead directly to a reduced development time for the RU.

4.4 Software Design

The Repeater Unit Software (RUS) was written in ANSI C and designed to handle the demands of message processing in an efficient and flexible manner. In an effort to achieve these goals the system uses abstract data types for the major structures, and the bulk of the code is highly compartmentalized -- most of the system is encapsulated with most data and functions accessed from single sources. For instance, system time is accessible through the user function now() which returns a value of type tsystime which is defined as an unsigned long int. So the current time must just be a single variable held somewhere in the system and could be accessed directly. Such direct access is not done specifically to reduce errors while coding. Also, the use of such techniques simplifies code maintenance.

4.4.1 Memory Management

The system must be capable of maintaining a large amount of changing data. For instance, if the channel load increases to high traffic levels temporarily, the system cannot send out messages as quickly as they will be received; this could potentially cause a backlog of several thousand messages which will have to be stored somewhere. The most common method of dealing with such dynamic memory requirements is to allocate memory from the system heap. Unfortunately, the way C implements this allocation is a little slow and, more importantly, incurs unacceptable storage overhead. Instead the RUS reserves a large physical block of memory in the system RAM. This memory is then chopped up into nodes the size of the message structure with a few bytes of overhead.

The process of setting aside physical memory space raises a few issues. What if that space needs to be moved or resized? It would be potentially confusing, and hence bug inducing, to do that if it required changes in the software in addition to changes in the memory map. To avoid those changes, one of the simplest assembly programs possible was written:

```
_qmem_begin
        DS.B $20000
_qmem_end
```

Where the first and last lines are externally defined labels and the middle line defines space for 20,000 hexadecimal byte locations or 128 kilobytes (this size may need to be increased in the future). This code section is useful because it allows the data space to be referenced in the C sections of the RUS without any literals as follows:

```
maxNodes = ( (int)&qmem_end - (int)&qmem_begin ) / sizeof(MessageNode);
node = (MessageNode*)&qmem_begin;
for ( j = 0; j < maxNodes; j++ )
        node[j].Next = &( node[j + 1] );
node[maxNodes - 1].Next = NULL;
FreeList = &( node[0] );
```

From here the entire memory space has been segmented into message nodes with each node pointing to the node that follows it and placed on a list of available nodes. It is quick, space efficient, and most importantly the nodes can now all be manipulated just as if they had been allocated directly from the heap.

4.4.2 Priorities

There are eight priority levels used in the IVDS system numbered from zero to seven. Seven is the highest priority and denotes either a system status message or an urgent user message, such as a fire alarm. Priority seven messages are always sent immediately after being generated or received. The other priorities are considered to be non-urgent and are sent at regular intervals with preference going to the higher priorities.

4.4.3 Queue Structure

The RUS employed 18 distinct queues, which were divided into four groups. In order of occurrence the queue groups were: Inbound, History, Priority, and Outbound. Both the History and Priority queue groups contain a queue for each of the eight priority levels. History queues were used to keep track of messages that had already fed through the system. By storing old messages in the History queues the RUS was able to identify duplicate messages. The Priority queues were used as temporary storage. The Inbound and Outbound queues held messages on their way into and out of the system respectively.

4.4.4 Power Up

The kernel calls the RUS after initializing itself and most of the hardware. The RUS handles its own initialization, which consists primarily of powering on the decoders and initializing the message space and queue structures. Once all of the initializations are done the system begins its main processing loop. The first pass through the loop sends a status message to the home office alerting that the RU is powered up and ready to receive user messages.

4.4.5 Message Processing

The main processing loop is normally an infinite loop, however exceptions will cause it to exit back to the kernel. The loop calls all of the functions necessary for repeater message handling. Those functions are: process_icb, proc_timed_events, proc_inbound, proc_inqueue, proc_outqueue, and proc_outbound. Process_icb is a kernel function that keeps all of the hardware running smoothly by polling all external devices and placing received data into buffers as it becomes available. Process_icb also performs tasks such as polling the system clock, checking system temperature, and running the system fans. The remaining function called in the main loop is proc_timer. This simple function keeps the RUS system time updated. Figure 4-3 contains a flow chart of the main loop.

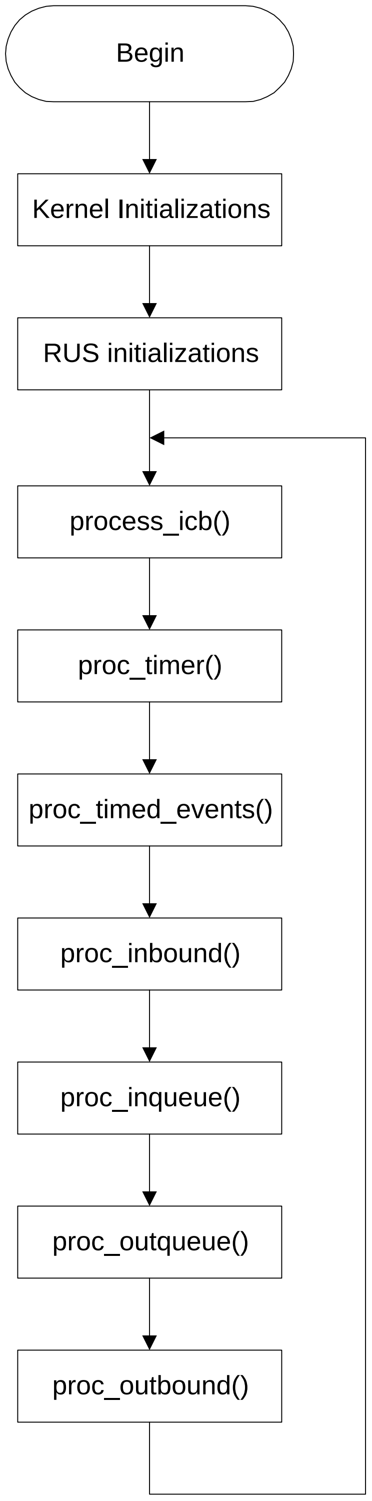

Figure 4-3: Main Processing Loop

4.4.5.1 proc_timed_events

This function controls three of the RUS events which are based on time, and ensures that they occur at regular intervals. The first is to signal the system to send pending messages anytime more than a defined number of seconds, originally 300, have passed since the last batch of messages was transmitted. Similarly, status messages are generated after a sufficiently long period of time has elapsed since the last status message; this value was originally 600 seconds. Thirdly, the history queues are serviced to dequeue old messages that are beyond their time-out period. Dequeued messages are returned to the memory manager, freeing their data space to be reused later.

To accomplish the third task, when messages are first received by the RUS they are time stamped with when they will pass their time-out period or expire. This time is determined by the time they are received plus the channel's retransmission time parameter. Recall that the channel model requires that all retransmissions much occur within a retransmission time. So, if the RUS waits until that amount of time has passed from the time a transmission was received, it knows that they will not be any more retransmissions of that message. There is a caveat: messages pass through several hardware and software layers within the RU before they reach the RUS. Neither the kernel nor the decoders currently have any provision to determine the amount of time spent in the system before a message reaches the RUS. To deal with this the channel's retransmission time parameter must be padded with the maximum time a message could spend propagating from the radio to the RUS. This maximum time is not particularly deterministic, but will be certainly be less than two seconds.

4.4.5.2 generate_status_message

This function generates a status message with a priority of seven. The information included in the message is: the GPS location, the time of day, and RUS statistics. The statistics track events such as CRC failures, partial messages received, the number of messages waiting to be processed and sent, and the number of duplicate messages received. With the number of duplicates and the number of messages received a good estimate of channel performance can be calculated. If the channel was running without collisions the ratio of duplicates to total transmissions would be retransmissions divided by retransmissions plus one. This routine also sets the time for the next status message.

4.4.5.3 proc_inbound

This function polls the kernel for available packets in the kernel decoder buffers. If a packet is available, it is read using a kernel function call. A strict protocol exists for passing packets around within the WMI which requires specific formatting and limits the maximum size of data packets. The maximum packet size used by kernel routines is smaller than the IVDS message size, which requires all messages to be spilt into two packets. To deal with these two part messages, additional protocol restrictions are placed on message packets; they must contain an additional byte indicating whether they are the beginning or end of a message.

Proc_inbound assembles these half messages into whole messages, discarding any stray packets (of which there should not be any if the decoder is working properly). Messages are then placed into message nodes, at the same time decoding the message priority and CRC value. With the message properly reassembled and formatted the CRC value is recomputed and compared to the value stored in the message. Non-matching CRC values indicate a transmission error. If an error is detected, the message is discarded. If the message is error free it is time stamped with its expiration time, as discussed in proc_timed_events. Lastly, the message is enqueued onto the inbound message queue.

4.4.5.4 proc_inqueue

This function removes messages from the inbound queue and tests them against the appropriate history queue to see if it is a duplicate message. Duplicates are discarded. New messages are then placed onto the appropriate history queue and priority queue based on the message’s priority level. For example, a priority 4 message would be copied to both history queue 4 and priority queue 4. If a message with a priority of 7 is processed, the system is signaled to send pending messages.

4.4.5.5 proc_outqueue

This function monitors the priority queues checking the number of messages that are ready to be scheduled and sent. If the number of messages exceeds a specific threshold, the system is placed into the sending state, this also happens anytime a priority 7 message is present. When the sending state is entered, a counter is set for the number of messages to be sent in this batch.

Once in the sending state messages are moved from the priority queues to the outbound queue according to a scheduling algorithm. The scheduling is done based on priority, where all priority 7 (the highest level) messages are always sent to the outbound queue first. The schedule for the other priorities is based on a $2^{(\text{priority})}$ scheme, e.g., it takes four priority 5 messages for every two priority 4 and one priority 3 message, and places them on the outbound queue. The scheduler contains additional intelligence to ensure that it will always schedule a message when asked; if a message of the requested priority is not available it will search down through the priority queues, in cyclic fashion, until it finds an available message. For example if it is trying to schedule a priority 5 message it will check for level 5 first, then 4, 3, 2, 1, 0, and cycle around to 6. This is done to avoid any priority level, particularly the low levels, from backlogging waiting for their time slot to appear.

4.4.5.6 proc_outbound

This function streams messages from the outbound queue to the kernel serial port buffer. This occurs any time the message counter is set in proc_outqueue, signaling that messages are ready to go. Messages are sent out in groups, the size of which is determined by the counter. The purpose of sending messages in batches is to reduce the frequency of connecting the CDPD link.

The serial port buffer in the kernel is not large enough to hold large batches of messages. Therefore care must be taken when filling the buffer to not overflow it. This is accomplished by always testing the buffer for sufficient free space before writing messages to the buffer. On the other side of the buffer, draining, or sending to the CDPD device, is done by an interrupt service routine in the kernel. This ensures that once data is made available it streams out steadily to the modem.

4.5 Conclusions

The high level hardware design was successful. The introduction of the WMI and the marked similarities in function between it the original hardware design had a two-fold impact. First, it validated the concept of the design by demonstrating that a similar design had been successfully built as a commercial product. Secondly, by using an already existing commercial product in the WMI, system development was more rapid.

The software section of the system was integrated into the WMI platform including interacting with the kernel. The system design was stream lined enough to handle the bandwidth requirement of the system. A prototype of the Repeater Unit was demonstrated to the sponsor in December of 1995. The demonstration went well, and the sponsor was pleased.

5. Summary and Conclusions

The IVDS project is a large project which involved several groups and many individuals. The process of defining the goals, processes, and methods was interesting to watch as it evolved. The author participated in molding that evolution, albeit primarily related to the simulator and the Repeater Unit.

The simulator proved to be an engaging task is several ways. It defined how the IVDS system was going to work using a uni-directional channel. In so doing, it helped to resolve the channel operation and -- together with the analytical model -- show that the channel, and hence the project, was viable. Also, from a programming standpoint the simulator posed several challenges. Deciding how to handle the channel analysis and what algorithms to use required significant thought. Further, once the simulator worked it was too slow to use on large models. Improving the algorithms to increase the speed to the point a person could bear to use it posed a further challenge. The simulator also proved to be a useful tool in determining optimal parameters for the channel. In sum, the simulator was a success.

The Repeater Unit was the first significant project this author took from a concept to a working prototype. It was a process filled with compromises based on cost, functionality, and, in the end, time. The Repeater Unit's main function as a message processor steered the entire development process. The fortuitous introduction of the WMI sped hardware development time, and later, software development, due to the earlier availability of a hardware platform. The RU reached a sufficient level of maturity to demonstrate to the sponsor. That demonstration displayed the Repeater Unit's ability to accept incoming messages under traffic load and handle them appropriately. The demonstration was a success and the sponsor indicated his approval.

5.1 Future Work and Possible Improvements

The Repeater Unit contains several areas which could be expanded upon. Fortunately, given that the WMI is a stable hardware platform, all of the improvements are software based. This author, after making significant contributions to the project, handed the work off to the next student. Due to the continuing nature of the project, not everything was completed before the transfer occurred. Most obviously, the GPS and CDPD modem along with the battery system had yet to be implemented. Additionally, the RUS contained the framework to not only schedule messages, but also to schedule processor time allotment between the various main loop processing functions; which could prove to be an interesting task. In the case of the simulator, while areas of possible improvement exist, the simulator's purpose has been fulfilled.

6. References

R. Brickhouse, “Wireless Measurement Instrument: WMI Kernel Software”, Grayson Electronics, 1995

B. Davidson, C. Bostian, S. Franks, M. Kurtin, N. Davis, “A Novel Retransmission Technique for One-Way Packet Communication Channels”

W. Farley, “Interactive Video Data Service Project Specification”, Virginia Tech, 1995

R. V. Hogg and E. A. Tanis, *Probability and Statistical Inference*. New York: Macmillan, 1993

M. Schwartz, *Telecommunication Networks: Protocols, Modeling and Analysis*. Reading, Massachusetts: Addison-Wesley, 1987.

Appendix A - Acronyms

Note: All acronyms are presented in the “Arial Narrow” font for simple recognition.

ANSI - American National Standards Institute

AT - The standard Hayes compatible modem command set.

CDPD - Cellular Digital Packet Data

CRC - Cyclic Redundancy Code

CWT - The Center for Wireless Telecommunications

DRAM - Dynamic Random Access Memory

DUART - Dual Universal Asynchronous Receiver Transmitter

GPS - Global Positioning System

HLCM - High Level Control Module

I/O - Input/Output

ICB - Interface Control Board, part of the WMI

IDC - Identification Code

IRS - Interactive Return Service

IVDS - Interactive Video Data Service

LED - Light Emitting Diode

MB - Megabyte

MHz - Megahertz

PROM - Programmable Read-Only Memory

RF - Radio Frequency

RU - Repeater Unit

RUS - Repeater Unit Software

UU - User Unit

UUID - User Unit Identification Number

WMI - Wireless Measurement Instrument, a product of Grayson Electronics.

Appendix B - Organization Chart

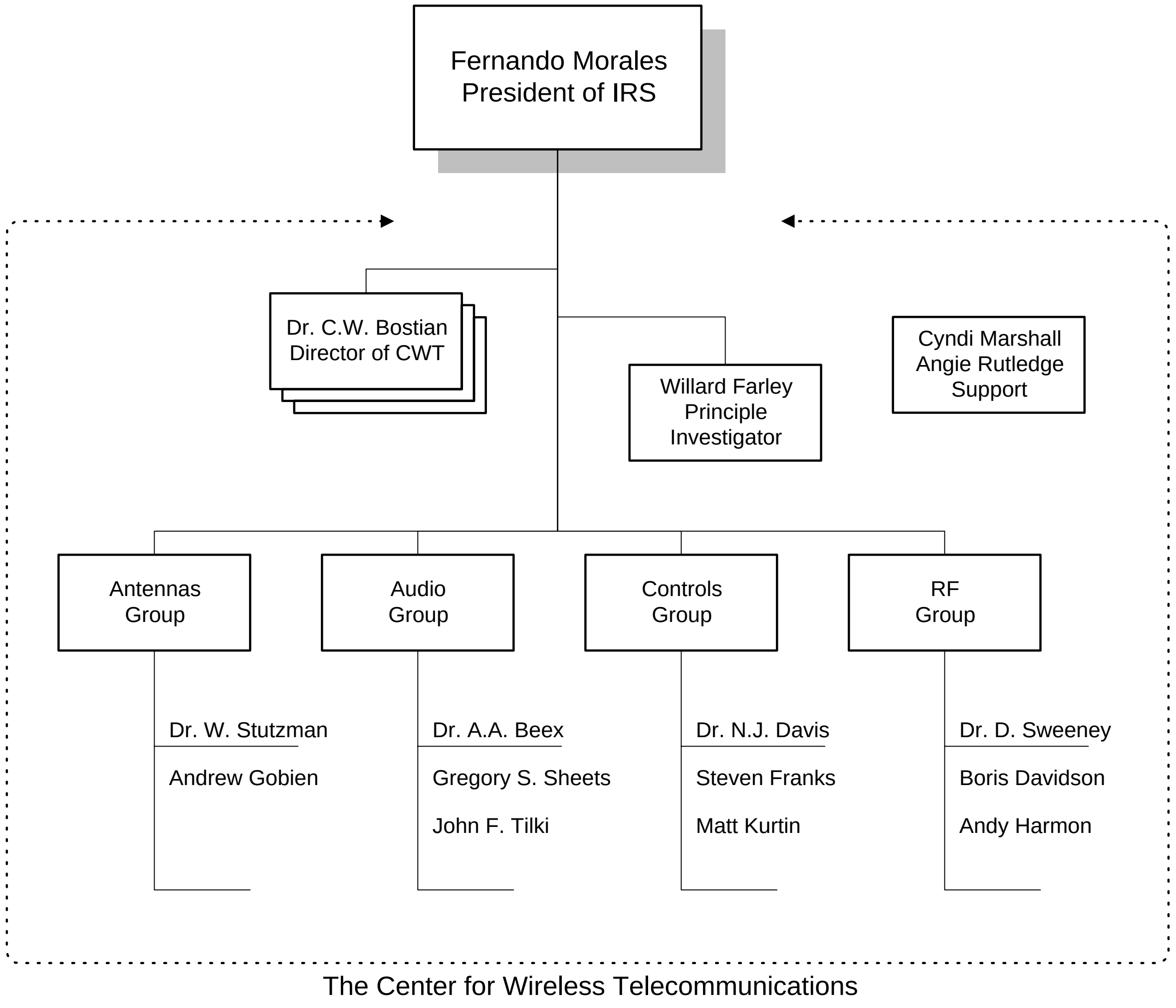

Figure 6-1: Initial Organization Chart for the IVDS Project

Vita

Steven Franks was born and raised in New Jersey. He attended Virginia Polytechnic Institute and State University for both his graduate and undergraduate degrees. Following his formal education he began his professional career with Intel Corporation working as a validation engineer on the Pentium™ Pro and Pentium™ II Processor lines.