



IMPLEMENTING INSTITUTE OF ELECTRICAL AND ELECTRONICS
ENGINEERS (IEEE) 802.11 STANDARD MEDIUM ACCESS CONTROL
PROTOCOL FOR WIRELESS LOCAL AREA NETWORKS (LANS) ON A
LABORATORY HARDWARE PROTOTYPE

THESIS

Joshua D. Green
Captain, USAF

AFIT/GE/ENG/04-11

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the United States Government

IMPLEMENTING INSTITUTE OF ELECTRICAL AND ELECTRONICS
ENGINEERS (IEEE) 802.11 STANDARD MEDIUM ACCESS CONTROL
PROTOCOL FOR WIRELESS LOCAL AREA NETWORKS (LANS) ON A
LABORATORY HARDWARE PROTOTYPE

THESIS

Presented to the Faculty
Department of Electrical and Computer Engineering
Graduate School of Engineering and Management
Air Force Institute of Technology
Air University
Air Education and Training Command
In Partial Fulfillment of the Requirements for the
Degree of Master of Science in Electrical Engineering

Joshua D. Green, B.S.E.E.
Captain, USAF

June 2004

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

IMPLEMENTING INSTITUTE OF ELECTRICAL AND ELECTRONICS
ENGINEERS (IEEE) 802.11 STANDARD MEDIUM ACCESS CONTROL
PROTOCOL FOR WIRELESS LOCAL AREA NETWORKS (LANS) ON A
LABORATORY HARDWARE PROTOTYPE

Joshua D. Green, B.S.E.E.
Captain, USAF

Approved:

___//signed///_____

Dr. Rusty O. Baldwin
Thesis Advisor

___ 1 July 2004 ___

Date

___//signed///_____

Dr. Michael A. Temple
Committee Member

___ 1 July 2004 ___

Date

___//signed///_____

Dr. Richard A. Raines
Committee Member

___ 1 July 2004 ___

Date

Acknowledgements

I would like to thank my wife for all her love and support through the rigorous demands placed on us at AFIT. Without her encouragement and care, this thesis would not be possible.

I would also like to recognize my family for their strength and appreciation of my challenges here.

Finally, I would like to thank my advisor, Maj Rusty Baldwin (ret.) for his extreme patience with me as I slowly but surely completed this research.

JOSHUA D. GREEN, Capt, USAF

Table of Contents

	Page
Acknowledgements	v
Table of Contents	vi
List of Figures	x
List of Tables	xii
Abstract	xiv
1. Research Introduction	1
1.1. Introduction.....	1
1.2. Research Goal	2
1.3. Document Overview	2
2. Background and Literature Survey	5
2.1. Introduction.....	5
2.2. Open Systems Interconnection (OSI) Reference Model.....	5
2.3. Other Wireless Medium Access Control (MAC) Protocols	8
2.3.1. ALOHA.....	8
2.3.2. Carrier Sense Multiple Access (CSMA).....	11
2.3.3. IEEE 802.11 Wireless LAN.....	13
2.3.3.1. IEEE 802.11 Performance	16
2.4. Current Research Efforts	26
2.4.1. Modifying Channel Free Wait Times (DIFS).....	26
2.4.2. Black-Burst (BB) Contention Mechanism.....	27
2.4.3. Real-Time Medium Access Control (RT-MAC)	28
2.5. Summary	29

3. Objectives and Methodology	31
3.1. Introduction.....	31
3.2. Research Goals	31
3.3. Approach.....	31
3.4. System Boundaries	32
3.5. System Services	32
3.6. Performance Metrics.....	32
3.7. Parameters.....	33
3.8. System Factors	35
3.9. Evaluation Techniques.....	36
3.10. Workload	36
3.11. Experimental Design.....	36
3.12. Analyze and Interpret Results.....	37
3.13. Summary	37
4. Experiments, Data, and Results	39
4.1. Introduction.....	39
4.2. The XInC test set	39
4.3. Experimental Setup.....	42
4.4. Experimental Results	45
4.4.1. Telemetry Results	46
4.4.2. Avionics Results	53
4.4.2.2. Mean Delay	58
4.5. Summary	60

5. Evaluation and Results.....	61
5.1. Introduction.....	61
5.2. Research Impact.....	61
5.3. Review and Conclusions.....	61
5.4. Outlines of Future Work.....	62
5.5. Summary.....	63
Appendix A - Experimental Data Tables.....	A-1
Appendix B - MatLab® Code.....	B-1
Appendix C - XInC Assembly Code	C-1
C.1. Introduction.....	C-1
C.2. WiFi.main.....	C-1
C.3. Constants.h.....	C-4
C.4. Delay.asm.....	C-5
C.5. FiftyMegaHertz.asm.....	C-7
C.6. Frame_Format.asm.....	C-8
C.7. Init.asm.....	C-34
C.8. LEDs.asm.....	C-36
C.9. Long_Data.asm	C-39
C.10. Math.asm.....	C-42
C.11. RFWaves.asm.....	C-44
C.12. RFWaves_data.asm.....	C-62
C.13. Short_Data.asm	C-63
C.14. Thread0.asm	C-65
C.15. Thread1.asm	C-76

C.16. Thread2	C-77
C.17. Thread3.asm	C-83
C.18. Thread4.asm	C-85
C.19. Thread5.asm	C-87
C.20. Thread6.asm	C-90
C.21. Thread7.asm	C-97
C.22. XInC.c	C-100
C.23. XPD_Echo.asm	C-103
C.24. XPD_Echo_Data.asm.....	C-111
C.25. XPD_Serial.asm	C-112
Appendix D - Experimental Data	D-1
Bibliography	Bib-1
Vita.....	Vita-1

List of Figures

Figure	Page
Figure 1. The OSI Model	6
Figure 2. Performance of ALOHA Protocols [Bal99]	10
Figure 3. Throughput of Non-Persistent CSMA [Bal99]	12
Figure 4. IEEE 802.11 Basic Access Method [IEEE99]	14
Figure 5. No-ACK CDMA/CA [KL99]	16
Figure 6. Performance of IEEE 802.11 verses ALOHA and CSMA	25
Figure 7. System Boundaries	32
Figure 8. Eleven Engineering Inc XInC 2.4 GHz RF 3.0 Mbps DSSS Development Kit	40
Figure 9. XInC Development Board with a RF Waves 1 Mbps/3 Mbps adapter card	40
Figure 10. RF Waves 2.4 GHz, 3 Mbps DSSS RF Module	41
Figure 11. Telemetry Goodput ($M = 2$)	48
Figure 12. Telemetry Goodput ($M = 3$)	48
Figure 13. Telemetry Goodput ($M = 4$)	49
Figure 14. Number of Packets Presented to the Queue verses the Transmission Attempts compared to the Normalized Telemetry Goodput ($M = 2$)	49
Figure 15. Number of Packets Presented to the Queue verses the Transmission Attempts compared to the Normalized Telemetry Goodput ($M = 3$)	50
Figure 16. Number of Packets Presented to the Queue verses the Transmission Attempts compared to the Normalized Telemetry Goodput ($M = 4$)	50

Figure 17. Telemetry Mean Delay ($M = 2$).....	51
Figure 18. Telemetry Mean Delay ($M = 3$).....	52
Figure 19. Telemetry Mean Delay ($M = 4$).....	52
Figure 20. Avionics Goodput ($M = 2$)	55
Figure 21. Avionics Goodput ($M = 3$)	55
Figure 22. Avionics Goodput ($M = 4$)	56
Figure 23. Number of Packets Presented to the Queue verses the Transmission Attempts compared to the Normalized Avionics Goodput ($M = 2$).....	56
Figure 24. Number of Packets Presented to the Queue verses the Transmission Attempts compared to the Normalized Avionics Goodput ($M = 3$).....	57
Figure 25. Number of Packets Presented to the Queue verses the Transmission Attempts compared to the Normalized Avionics Goodput ($M = 4$).....	57
Figure 26. Avionics Mean Delay ($M = 2$).....	58
Figure 27. Avionics Mean Delay ($M = 3$).....	59
Figure 28. Avionics Mean Delay ($M = 4$).....	59

List of Tables

Table	Page
Table 1. Some IEEE 802.11 Standards	13
Table 2. MAC Protocol Parameters	34
Table 3. Example of Telemetry Throughput Data	44
Table 4. Telemetry Goodput ($M = 2$) [Figure 11]	1
Table 5. Telemetry Goodput ($M = 3$) [Figure 12]	1
Table 6. Telemetry Goodput ($M = 4$) [Figure 13]	2
Table 7. Number of Packets Presented to the Queue verses the Transmission Attempts compared to the Normalized Telemetry Goodput ($M = 2$) [Figure 14]	2
Table 8. Number of Packets Presented to the Queue verses the Transmission Attempts compared to the Normalized Telemetry Goodput ($M = 3$) [Figure 15]	2
Table 9. Number of Packets Presented to the Queue verses the Transmission Attempts compared to the Normalized Telemetry Goodput ($M = 4$) [Figure 16]	3
Table 10. Telemetry Mean Delay ($M = 2$) [Figure 17]	3
Table 11. Telemetry Mean Delay ($M = 3$) [Figure 18]	3
Table 12. Telemetry Mean Delay ($M = 4$) [Figure 19]	4
Table 13. Avionics Goodput ($M = 2$) [Figure 20]	4
Table 14. Avionics Goodput ($M = 3$) [Figure 21]	4
Table 15. Avionics Goodput ($M = 4$) [Figure 22]	5

Table 16.	Number of Packets Presented to the Queue verses the Transmission Attempts compared to the Normalized Avionics Goodput ($M = 2$) [Figure 23]	5
Table 17.	Number of Packets Presented to the Queue verses the Transmission Attempts compared to the Normalized Avionics Goodput ($M = 3$) [Figure 24]	5
Table 18.	Number of Packets Presented to the Queue verses the Transmission Attempts compared to the Normalized Avionics Goodput ($M = 4$) [Figure 25]	6
Table 19.	Avionics Mean Delay ($M = 2$) [Figure 26]	6
Table 20.	Avionics Mean Delay ($M = 3$) [Figure 27]	6
Table 21.	Avionics Mean Delay ($M = 4$) [Figure 28]	7

Abstract

Wireless Local Area Networks (LANs) are extremely convenient, flexible, and easy to deploy. All LANs in which multiple hosts must access the same medium use a Medium Access Control (MAC) protocol to coordinate channel access. The MAC is part of the Data Link Layer of the Open Systems Interconnection (OSI) Reference Model. One MAC protocol in extensive use today is the Institute of Electrical and Electronics Engineers (IEEE) 802.11 standard.

Since IEEE 802.11 devices are so prevalent in today's world, many researcher are exploring modifications and enhancements to the protocol. There are several well developed analytical and simulation models for IEEE 802.11 available to researchers, yet one significant obstacle remains: the lack of a means to obtain experimental data based on proposed protocol changes. Without real world experimental data, researchers lack the ability to test out their proposals in a real world environment.

To fill this need, this thesis created a hardware prototype from which researchers can obtain experimental data about IEEE 802.11. This hardware prototype can now be used by researchers to gain real world data on their proposed modifications to IEEE 802.11.

This page intentionally left blank

*Implementing Institute of Electrical and Electronics Engineers (IEEE)
802.11 Standard Medium Access Control Protocol for Wireless Local Area
Networks (LANs) on a Laboratory Hardware Prototype*

1. Research Introduction

1.1. Introduction

Wireless Local Area Networks (LANs) are extremely convenient, flexible, and easy to deploy. Existing Wireless LANs are designed primarily to handle bursts of traffic in an efficient manner. They are outstanding for the error free transfer of large amounts of data [LARO02].

All LANs in which multiple hosts must access the same medium use a Medium Access Control (MAC) protocol to coordinate channel access. The MAC is part of the Data Link Layer of the Open Systems Interconnection (OSI) Reference Model. One MAC protocol in extensive use today is the Institute of Electrical and Electronics Engineers (IEEE) 802.11 standard.

The IEEE 802.11 standard was first published in 1997. Since then, several simulation (i.e. [Bal99]) and analytical models (i.e. [ZiA02]) have explored IEEE 802.11's performance characteristics and have sought to improve the protocol for either general or specific purposes. However, a significant obstacle encountered by researchers in this area is the lack of a means to obtain experimental data based on proposed protocol changes. Devices using IEEE 802.11 standard are abundantly distributed throughout the world, but there are few if any manufacturers who will sell their IEEE

802.11 source code or development kits; this is for economic reasons: writing source code and developing IEEE 802.11 hardware requires a significant investment. Any company who has dedicated resources and capital into developing IEEE 802.11 devices will not want to part with that knowledge without suitable compensation (usually several hundred thousands, if not millions of dollars). This kind of capital is well outside the reach of most organizations that perform research on the MAC.

1.2. Research Goal

The goal of this research is straightforward: to create a hardware prototype and provide experimental data about IEEE 802.11 to researchers. This hardware prototype can then be used to validate proposed modifications to IEEE 802.11.

1.3. Document Overview

This chapter gives a brief overview of the problem addressed and the research goals. Chapter 2 presents an overview of wireless LANs by first describing the Open Systems Interconnection (OSI) seven-layer network model. Next, the chapter describes several Wireless Medium Access Control (MAC) protocols, especially ALOHA, Carrier Sense Multiple Access (CSMA), and IEEE 802.11 itself. The chapter concludes with a brief description of some relevant research in wireless networks. Chapter 3 presents the methodology used to meet the research objectives. Chapter 4 discusses the research results, comparing experimental data to an analytical model. Chapter 5 contains the conclusion and recommendations for future research. Appendix A includes the experimental data tables used in the figures in this document. Appendix B contains the

MatLab® code used to create this document's figures. Appendix C holds the XInC assembly code for the experiment.

This page intentionally left blank

2. Background and Literature Survey

2.1. Introduction

This chapter includes background information helpful in establishing the foundation for the research. Section 2.1 presents an overview of the Open Systems Interconnection (OSI) model, its purpose, and the parts of the model of interest to this research. In wireless networks, the Medium Access Control (MAC) layer is the prime focus of interest and thus several MAC protocols are presented in Section 2.2. Protocols such as ALOHA and Carrier Sense Multiple Access (CSMA) are compared. Section 2.2.3 describes the IEEE 802.11 standard [IEEE99] and gives an extensive explanation of the analytical model used in this research. Finally, an overview of related research efforts is given in Section 2.3.

2.2. Open Systems Interconnection (OSI) Reference Model

Forming a network of systems can be a very complicated task. To make this task more manageable, the OSI Reference Model partitioned the functions of a network into broad areas. The model defines seven different layers or functions that are typically performed during communication between two network nodes (Figure 1). The rest of this section briefly discusses each layer, starting at the lowest layer (Layer 1 or the Physical Layer) and working up to the top layer (Layer 7 or the Application Layer). Layer 2, the Data Link Layer (DLL), is the focus of this effort and thus will be discussed in more detail in Section 2.2.

Layer 1, the Physical Layer, receives binary data from the Data Link Layer (DLL), converts the bits into symbols, and transmits them over a physical medium such

as a wire or a fiber optic cable. The Physical Layer's task is to ensure individual symbols are received error free, in the proper order, and are converted to the appropriate bit stream for submission to the DLL.

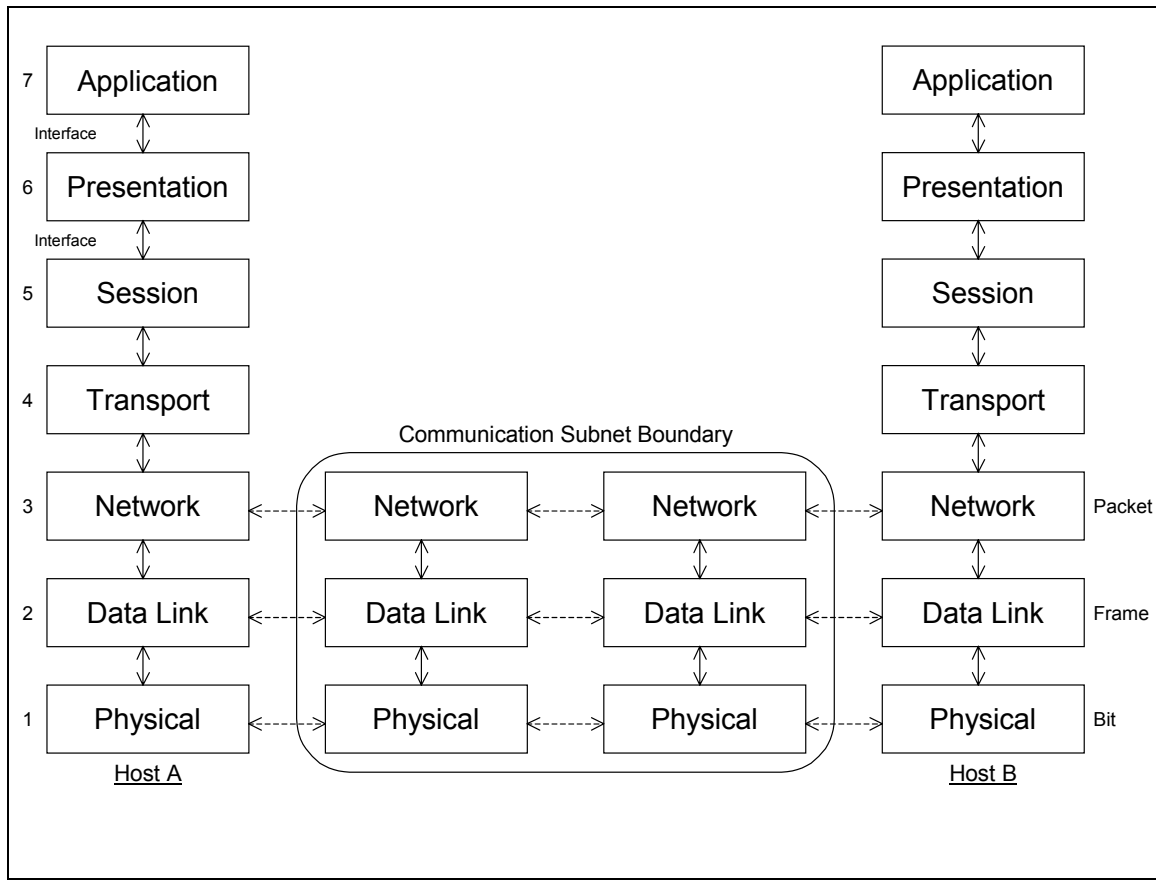


Figure 1. The OSI Model

Logically above the Physical Layer is the DLL. The DLL takes bits from the Physical Layer, assembles them into data frames, detects any errors, corrects them if possible, and requests a retransmission if necessary. Once the DLL has an error free frame, it passes the information up to the next level, the Network Layer. However, if multiple hosts share the same medium, a sub-layer of the DLL called the Medium Access Control (MAC) manages access to the channel. For instance, in wireless LANs the IEEE

802.11 standard defines the MAC sub-layer. The MAC sub-layer is discussed in more detail in Section 2.2.

The Network Layer determines how packets are routed from one network to another. In situations where all the hosts can hear each other, as on a LAN, this is an extremely simple process. In a large network, such as a Wide Area Network (WAN), the layer determines where a packet must be sent in order to arrive at its final destination. In these situations, the network layer may use flow control to avoid network congestion. In a LAN, the DLL Layer handles the flow control.

Above the Network Layer is the Transport Layer which establishes and terminates reliable source-to-destination or end-to-end connections. The Transport Layer differs from layers 1 to 3 because it communicates between different host's processes rather than between the hosts themselves.

Like the Transport Layer, the Session Layer creates end-to-end connections between processes, but the Session Layer also provides some advanced services. For instance, the Session Layer determines if two processes will communicate in simple or full duplex.

The Presentation Layer's responsibility is the data's syntax and semantics. Some examples are encryption/decryption, compression/decompression, and code conversion.

The top layer of the model is the Application Layer. It provides a user interface for all applications written to run over the network. In contrast to the other layers, the

Application Layer handles tasks that are written for a specific application, while the other layers handle services common to all applications.

To understand the workings of any network protocol, it is important to understand the OSI model. However, in real applications the model is never implemented in the form described. Instead, standard bodies like the IEEE developed their own protocols that often do not match the OSI model. For instance, many commercially available network devices use a Physical and Data Link Layer defined by the IEEE 802 family of standards, of which 802.11 (the wireless LAN standard) is a part.

2.3. Other Wireless Medium Access Control (MAC) Protocols

Wireless LANs have been around since the early 1970s. Briefly described in Section 2.1, MAC protocols are part of the DLL in the OSI Model. MAC protocols are necessary whenever the Medium is shared between multiple hosts. This section describes three MAC protocols used in wireless networks.

2.3.1. ALOHA

ALOHA, developed in the 1970s at the University of Hawaii, is a simple and elegant way to allow multiple host access to the same channel [BG92]. Pure ALOHA is a contention-based protocol, meaning all the hosts must compete for the shared medium at the same time.

The system is rather simple: to transmit a frame of data, a sending host transmits the data immediately, whenever its data is ready to send. When the receiving host receives a good frame, it sends back an acknowledgment (ACK) to the sending host. If

the sending host does not receive an ACK for the frame it sent, it assumes the frame is lost due to a collision with another transmitting host. The sending host will wait a random amount of time and retransmit the same frame. The random amount of time is important. Otherwise, two sending hosts could continue to transmit the frames at the same time, causing repeated collisions and filling up the channel.

The throughput of Pure ALOHA is $S = Ge^{-2G}$ [Abr77], where S is defined as the normalized channel throughput and G is the normalized channel traffic in frames. Given this equation, Pure ALOHA attains a maximum throughput of $S = 1/2e = 0.184$ when $G = 0.5$. This means that Pure ALOHA is a rather inefficient protocol, for it only uses 18.4% of its channel at its maximum throughput. However inefficient, Pure ALOHA is a very simple protocol and thus is very straightforward to implement.

In terms of channel utilization, an improvement over Pure ALOHA is Slotted ALOHA. Slotted ALOHA divides access to the channel into discrete intervals, with each interval corresponding to one frame. This enhances the throughput equation to $S = Ge^{-G}$, and produces a new throughput value of $S = 1/e = 0.368$ when $G = 1$ [Abr77]. Slotted ALOHA is more complex than Pure ALOHA, but the added complexity gives a substantial gain in channel throughput.

Another variant of the ALOHA protocol is Reservation-ALOHA (R-ALOHA) and is described in [CN95]. R-ALOHA works by first synchronizing the channel just like Slotted ALOHA. At the beginning of a time slot, rather than broadcasting its information, R-ALOHA instead broadcasts a short reservation-request (which is itself vulnerable to collisions). If the reservation-request is accepted, the host receives

exclusive access to the channel for a given period of time. This means R-ALOHA is not a connection based protocol and thus differs from other ALOHA protocols.

The amount of time a host is given sole access to the channel is defined as v^{-1} where v is the ratio of reservations request duration to length of the frame. When $v = 0.05$ and $G = 20$, $S = 0.88$, R-ALOHA reaches 88% utilization and clearly surpasses all other ALOHA protocols. However, the results come with the cost of increased complexity.

Figure 2 gives a comparison between the differing variations of ALOHA. R-ALOHA and Slotted ALOHA outperform Pure ALOHA in all cases. R-ALOHA and Slotted ALOHA perform almost the same until $G = 0.2$. Above this load, R-ALOHA performs better.

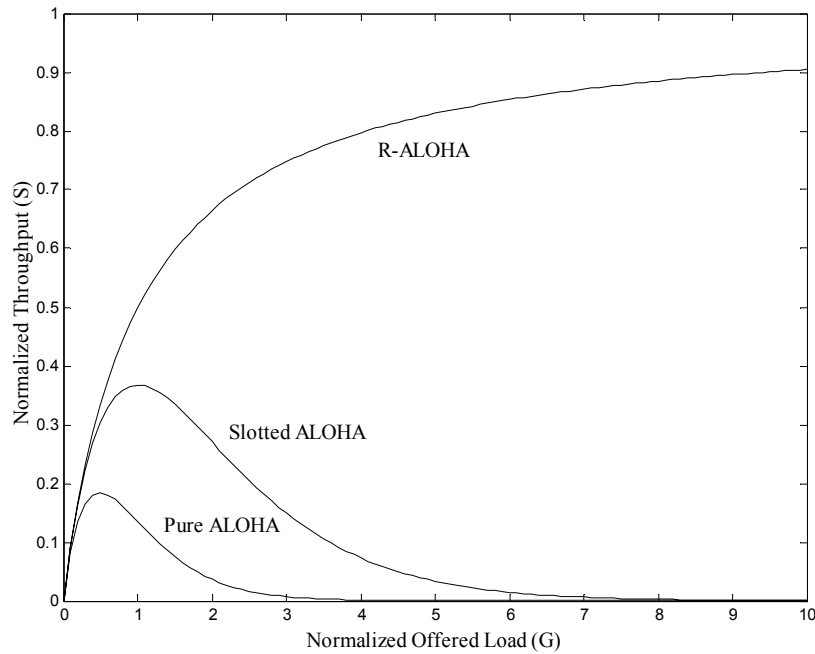


Figure 2. Performance of ALOHA Protocols [Bal99]

2.3.2. *Carrier Sense Multiple Access (CSMA)*

One of ALOHA's key features is hosts broadcast at will without regard to what other hosts are doing. Thus, collisions are inevitable. To reduce the likelihood of a collision a host can first monitor the channel, and if another host is transmitting, defer the transmission. Protocols that follow this procedure are called Carrier Sense Multiple Access (CSMA).

There are several variants of CSMA, such as non-persistent CSMA, 1-persistent CSMA, and p -persistent CSMA. Each version of CSMA prepares to send a frame in the same way. They all use a slotted channel and they all listen to the channel before transmitting to determine if the channel is clear. What distinguishes each version is how it responds to a busy medium. Non-persistent CSMA responds by rescheduling a frame for later transmission, while p -persistent CSMA reschedules a frame for retransmission with probability p (upon the medium becoming idle). Finally, 1-persistent CSMA transmits a frame when the medium becoming idle with certainty [Bal99].

The performance of CSMA is closely tied to delays caused by propagation and signal detection. Propagation and detection delay as a ratio of the frame size is [BG92]

$$\beta = \frac{\tau C}{L} \quad (2.1)$$

where τ is the total delay in seconds, C is the channel bit rate, and L is the expected number of bits in a given frame. As β increases, performance decreases because a host attempting to sense a signal must wait longer before transmitting. Thus, the key

parameters affecting the performance of CSMA are the channel bit rate, C , and the bits per frame, L .

To further illustrate the effect of propagation and detection delay on system throughput, consider the throughput of non-persistent CSMA [KT75]

$$S = \frac{Ge^{-\beta G}}{G(1 + 2\beta) + e^{-\beta G}} \quad (2.2)$$

where β is the propagation and detection delay, and G is the normalized offered load. Figure 3 shows the results using various β . Note that as β gets larger, throughput drops significantly.

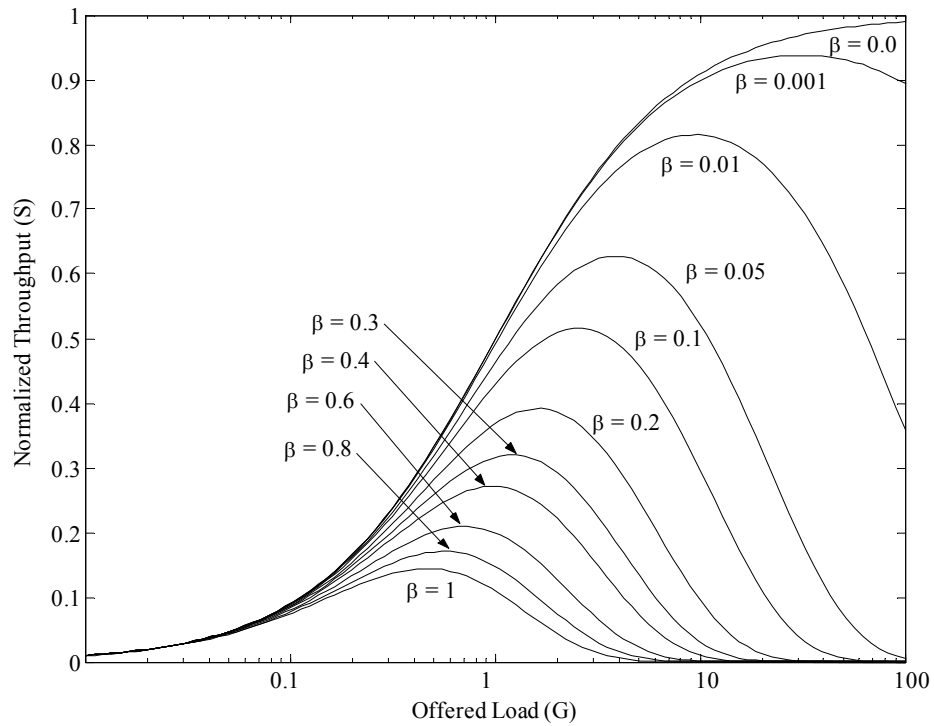


Figure 3. Throughput of Non-Persistent CSMA [Bal99]

2.3.3. IEEE 802.11 Wireless LAN

IEEE 802.11 [IEEE99] defines both MAC and Physical Layer (PHY) specifications for Wireless LANs (WLANs). IEEE 802.11 has many different varieties. Some are listed below in Table 1.

Table 1. Some IEEE 802.11 Standards

Standard	Operating Frequency	Maximum Throughput
802.11	Infrared - 850 nm to 950 nm Radio Frequency - 2.4 GHz	1-2 Mbps
802.11a	5 GHz	54 Mbps
802.11b	2.4 GHz	11 Mbps
802.11g	2.4 GHz	54 Mbps

The IEEE 802.11 MAC protocol uses two different mechanisms to gain access to the medium: the Distributed Coordination Function (DCF) and the Point Coordination Function (PCF). PCF is a contention-free scheme under the control of a single Point Coordinator (PC), and provides collision free and time-sensitive services. DCF provides access to the medium via a Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) protocol. It should be noted that PCF ultimately uses DCF to access to the medium. The PC in the PCF scheme ensures only one host accesses the medium at a time.

DCF controls access to the medium by two different methods. The default is called the basic access method and uses a two-way handshaking technique. This technique is distinguished by a receiver sending a positive acknowledgment (ACK) upon successfully receiving a frame to the sending node.

The second method is a four-way handshake using a request-to-send/clear to send (RTS/CTS) process. A transmitting node must first “reserve” the channel by transmitting to the receiving node a RTS frame. The receiving host acknowledges the RTS by sending back a CTS, after which normal data transfer and ACK responses occur. The RTS/CTS process has an advantage over the basic access method because collisions can only occur during the transmission of the RTS frame, and these collisions can be easily detected by the lack of a CTS response. This process can increase system performance by reducing the duration of a collision for long packets, although it also adds significant overhead [Bia00].

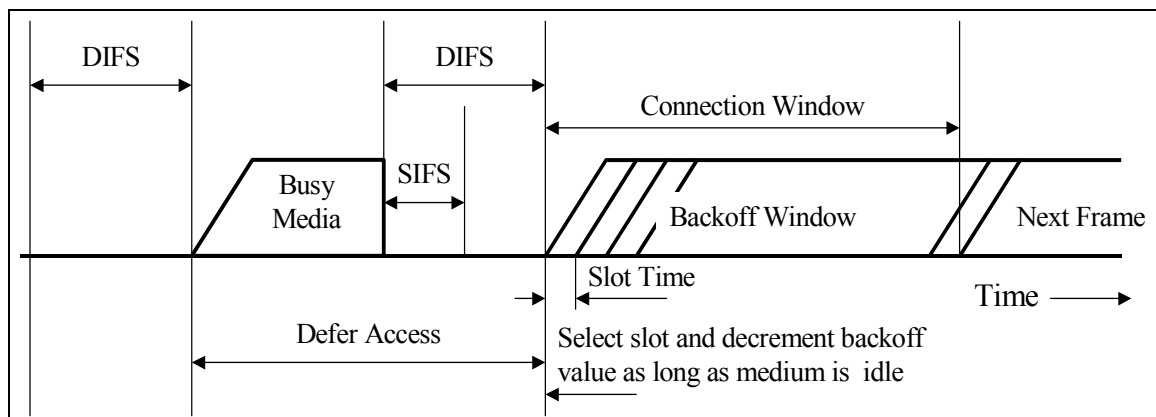


Figure 4. IEEE 802.11 Basic Access Method [IEEE99]

Whether a host uses the basic access method or the RTS/CTS process, the inner-workings of the CSMA/CA protocol operate in the same way and are shown in Figure 4. First, the channel is slotted, represented by the “Slot Time,” and a node transmits only at the beginning of a given time slot. When a host wants to transmit a new frame, it checks the channel for any activity. If the channel is idle for a period of time known as the Distributed Inter-frame Space (DIFS), the host transmits. If the host senses the channel is

busy during the DIFS, it continues to monitor the channel until the channel is once again idle for a DIFS period. At this point the host waits for a random amount of time, known as a backoff interval, before transmitting. This reduces the chance of a collision with another transmitting host.

The backoff interval is measured in slots equal to the slot time and is based on a randomly chosen discrete integer called the backoff value (BV). The BV is in the range of $[1, w - 1]$ where w is the width of the Contention Window (Figure 4). The Contention Window is determined by the number of failed attempted transmissions. At the first transmission attempt, $w = CW_{\min}$ or the Minimum Contention Window. After each unsuccessful attempt, w is doubled until it reaches a set maximum value, $CW_{\max}$. Both $CW_{\min}$ and $CW_{\max}$ are fixed integers and specific to the Physical Layer in use.

For every idle time slot, the value of BV is decremented by one. If the channel is sensed to be busy, the counter is not decremented again until the channel is idle for a DIFS period. Once $BV = 0$, the host transmits.

When a node successfully receives a frame, it responds with an ACK frame. The ACK is transmitted after a delay equal to a Short Inter-frame Space (SIFS), which is less than a DIFS (see Figure 4). When the transmitting node receives an ACK, it knows its frame was successfully received. If the transmitting node does not receive an ACK after a predefined amount of time, known as the ACK timeout period, it assumes its frame was lost and retransmits the frame.

2.3.3.1. IEEE 802.11 Performance

The theoretical performance of IEEE 802.11 is described in detail in [KL99] and is refined in [ZiA02]. It starts by assuming that the systems states alternate between two periods: 1) idle periods (I), when no station is transmitting, and 2) busy periods (B), when at least one station is transmitting. U is defined as the time spent doing useful transmissions during a Busy Period. If $\bar{X}$ (or $E[X]$) denotes the expected value of the random variable X , then it follows that the normalized throughput of IEEE 802.11 is

$$S = \frac{\bar{U}}{\bar{B} + \bar{I}} \quad (2.3)$$

where $\bar{I}$ is the expected value of the idle time when a host has nothing to transmit, $\bar{B}$ is the expected value of the time at least one host transmits a frame, and $\bar{U}$ is the expected value of the time spent in useful transmission [ZiA02].

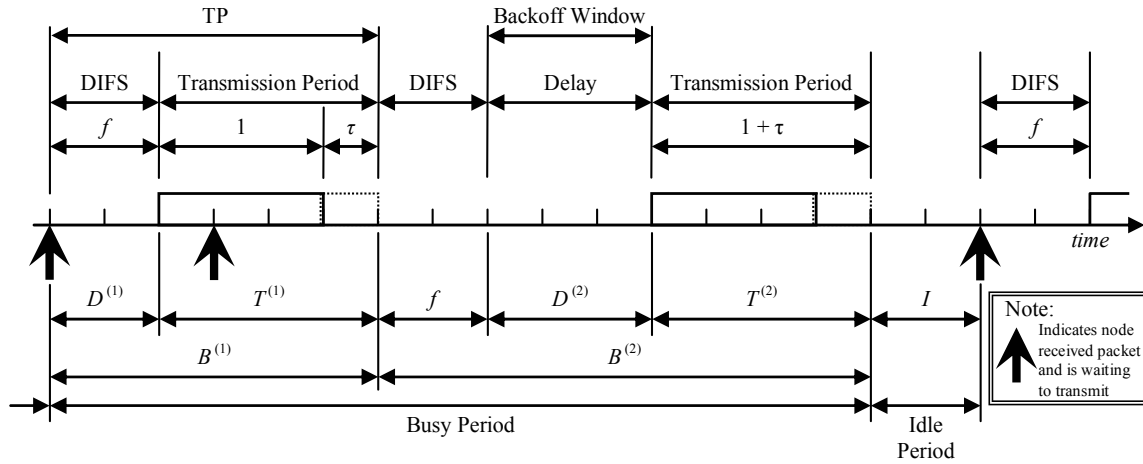


Figure 5. No-ACK CDMA/CA [KL99]

Although IEEE 802.11 uses either the basic access method CSMA/CA or RTS/CTS CSMA/CA, [ZiA02] considers a third model for this analysis, a No-ACK

CSMA/CA. In No-ACK CSMA/CA, a node transmits its packets and does not wait for an ACK. The No-ACK model is presented in Figure 5. When not transmitting, a node is in an Idle Period, I . When a node wants to transmit, it moved from an Idle Period to a Busy Period, B .

The Busy Period can react in one of two ways. First, if during the Busy Period's DIFS the node determines that the medium is idle the node will transmit immediately following the DIFS period. This occurs if a station transmits right after an Idle Period. In this way, No-ACK CSMA/CA works just like a 1-persistent CSMA protocol. However, if the node does detect the medium is busy (because another node is transmitting), it will invoke a backoff mechanism, making the No-ACK CSMA/CA response like p -persistent CSMA. [CCG00] demonstrated that in IEEE 802.11, $p = 2/(E[CW] - 1)$ where $E[CW]$ is the average connection window. (Note: Finding $E[CW]$ for a given number of nodes can be quite involved. However, Table IV of [CCG00] lists several of them and those values are what are used in this research.)

Figure 5 employs some methodology used in [KL85] and [KL99]. First, the busy periods of No-ACK CSMA/CA are divided up into several sub-busy periods, where j is the number of sub-busy periods in a Busy Period and is denoted by $B^{(j)}$. For $j = 1$ (the first sub-busy period), $B^{(1)} = D^{(1)} + T^{(1)}$, with $D^{(1)} =$ the DIFS period, f , and $T^{(1)} = 1 + \tau$, where 1 represents the normalized frame length and τ is the propagation delay normalized to the time it takes to transmit one frame. For $j \geq 2$ (the second or higher sub-busy period), $B^{(j)} = f + D^{(j)} + T^{(j)}$ where f is the DIFS period, $D^{(j)}$ is the delay caused by the backoff window (to be explained later), and $T^{(j)} = 1 + \tau$. (Note: $T^{(j)}$ is the same

regardless if the transmission was successful or not.) Busy Periods continue as long as there 1) is at least one station waiting to transmit during a transmission period or 2) if a station transmits during a DIFS period. To this end, the transmission period, TP, is defined as $TP = 1 + \tau + f$.

The expected value of the Idle Period, $\bar{I}$, is assumed independent and geometrically distributed, and thus it follows that

$$\bar{I} = \frac{a}{1 - (1 - g)^M} \quad (2.4)$$

where a is the backoff slot time normalized to the time it takes to transmit one frame, g is the probability a host generates a frame during a time slot, and M is the number of hosts in the network. The probability a host generates a frame is $g = aG/M$ where G is the normalized offered load [KT85].

The expected value of the Busy Period, $\bar{B}$, uses the delay, $D^{(j)}$. As mentioned above, for $j = 1$, $D^{(1)}$ = the DIFS period, f . However, for $j \geq 2$, $D^{(j)}$ is a stochastic random variable and its expectation is defined

$$\begin{aligned} \overline{D^{(j)}} = & \frac{a}{1 - (1 - g)^{(TP/a)M}} \left(\sum_{k=1}^{\infty} \left\{ (1 - p)^k \right. \right. \\ & \left. \left. - (1 - g)^{TP/a} \left[(1 - p)^k - (1 - g)^k \right] \right\}^M \right. \\ & \left. - (1 - g)^{(TP/a)M} \sum_{k=1}^{\infty} (1 - g)^{kM} \right) \end{aligned} \quad (2.5)$$

where k is the number of backoff slots left in the Backoff Window. Note that when $k = 0$, a node transmits [ZiA02].

If J is the number of sub-busy periods in a Busy Period, then the Busy Period, B , is given by $B = \sum_{j=1}^J B^{(j)}$, and from this the sum of the expectation of the Busy Period is [KL99]

$$\begin{aligned} \bar{B} = & f \left[1 - (1-g)^M \right] + 1 + \tau + \frac{1}{(1-g)^{(TP/a)M}} \left\{ (f+1+\tau) \left[1 - (1-g)^{(TP/a)M} \right] \right. \\ & + a \sum_{k=1}^{\infty} \left\{ (1-p)^k - (1-g)^{(TP/a)} \left[(1-p)^k - (1-g)^k \right] \right\}^M \\ & \left. - a (1-g)^{(TP/a)M} \sum_{k=1}^{\infty} (1-g)^{kM} \right\}. \end{aligned} \quad (2.6)$$

The next calculation is the expected value of the Useful Transmission Period, $\bar{U}$. Begin by calculating the expected value of the first sub-busy period, $\bar{U}^{(1)}$. This is done by considering that a transmission is only successful (and thereby useful) at $j = 1$ when there is only one packet arrival in the last slot of the Idle Period. Thus [ZiA02]

$$\bar{U}^{(1)} = \frac{1}{1 - (1-g)^M} M g (1-g)^{(M-1)}. \quad (2.7)$$

To calculate $\bar{U}^{(j)}$ when $j \geq 2$, first let $P_n(X)$ be the probability that n packets arrive among M nodes during X time slots. $P_n(X)$ is expressed as

$$P_n(X) = \sum_{n=1}^M \left\{ \frac{\binom{M}{n} \left[1 - (1-g)^{X/a} \right]^n (1-g)^{X(M-n)/a}}{1 - (1-g)^{X \cdot M/a}} \right\}. \quad (2.8)$$

Also, consider $N_0^{(j)}$ to be the number of packets accumulated at the end of a transmission period. Given this, the distribution of $N_0^{(j)}$ is $\text{Prob}[N_0^{(j)} = n] = P_n(TP)$ for $j \geq 2$ [KL99].

For $j \geq 2$, a node successfully transmits only when one node in a network transmits and there are no collisions. Put another way, a Useful Transmission Period occurs only when $N_0^{(j)} = n$ and $D^{(j)} \geq k \cdot a$. This could occur in two cases. First, if $k = 0$ (at least one node has its backoff counter at zero) the transmission is successful only when one station of the n nodes with packets waiting to transmit does so. Second, if $k \geq 1$ the transmission is successful when: 1) one station of the n nodes with packets waiting to transmit does so or 2) only one among the remaining stations with no packets waiting to transmit (which is $M - n$) is given a packet to transmit. Given this, the expected value of $U^{(j)}$ given $N_0^{(j)} = n$ and $D^{(j)} \geq k \cdot a$ is

$$E[U^{(j)} | N_0^{(j)} = n, D^{(j)} \geq k \cdot a] = \begin{cases} np(1-p)^{n-1} & k = 0 \\ np(1-p)^{n-1} + (M-n)g(1-g)^{M-n-1} & \\ -n(M-n)pg(1-p)^{n-1}(1-g)^{M-n-1} & k \geq 1. \end{cases} \quad (2.9)$$

If J is the number of sub-busy periods in a Useful Transmission Period, than the Useful Transmission Period, U , is given by $U = \sum_{j=1}^J U^{(j)}$. By using the theorem of total probability on (2.9) and from this summing all $\overline{U^{(j)}}$, the expectation of the Useful Transmission Period is

$$\begin{aligned}
\bar{U} = & \frac{1}{1-(1-g)^M} Mg(1-g)^{(M-1)} \\
& + \left[\frac{1}{(1-g)^{(TP/a)M}} - 1 \right] \sum_{n=1}^M \left\{ np(1-p)^{n-1} + \left[np(1-p)^{n-1} \right. \right. \\
& \left. \left. + (M-n)g(1-g)^{M-n-1} - n(M-n)pg(1-p)^{n-1}(1-g)^{M-n-1} \right] \right. \\
& \left. \cdot \frac{(1-p)^n(1-g)^{M-n}}{1-(1-p)^n(1-g)^{M-n}} \right\} \cdot \left\{ \frac{\binom{M}{n} [1-(1-g)^{TP/a}]^n (1-g)^{(TP/a)(M-n)}}{1-(1-g)^{(TP/a)M}} \right\}.
\end{aligned} \tag{2.10}$$

Substituting Equation (2.4), (2.6), and (2.10) into Equation (2.3) will give the channel throughput for No-ACK CSMA/CA [ZiA02].

Calculating the throughput for IEEE 802.11 basic access method follows the same analysis. The difference between No-ACK CSMA/CA and IEEE 802.11 lies in the time lengths of successful and non-successful transmission periods. For No-ACK CSMA/CA, the time lengths of both successful and non-successful transmission periods are the same. For the IEEE 802.11, the time lengths are different.

IEEE 802.11 basic access method throughput analysis starts with defining the successful transmission period, TP_S , and the non-successful transmission period, TP_F , as

$$\begin{aligned}
TP_S &= 1 + \beta + \delta + 2\tau + f, \text{ and} \\
TP_F &= 1 + \tau + f
\end{aligned} \tag{2.11}$$

where β is the normalized length of the SIFS, δ is the normalized length of an ACK frame, τ is the normalized length of a frame's propagation delay, and f is the normalized length of a DIFS.

It is assumed that the j th transmission of the Busy Period, B , is X time slots in length. Therefore, the length of the next sub-busy period or the $(j+1)$ th slot is dependant on the success or failure of the transmission immediately before it (the j th transmission). This makes the length of the remaining Busy Periods a function of X . Let $B(X)$ be the mean of the Busy Period after a frame buildup time of X slots and let $U(X)$ be the Useful Transmission Period during the same Busy Period. $B(X)$ and $U(X)$ can now be found using [ZiA02]

$$\begin{aligned}
B(X) &= d(X) \\
&+ \left\{ TP_S + \left[1 - (1-g)^{(TP_S/a)} \right] B(TP_S) \right\} u(X) \\
&+ \left\{ TP_F + \left[1 - (1-g)^{(TP_F/a)} \right] B(TP_F) \right\} [1-u(X)]
\end{aligned} \tag{2.12}$$

$$\begin{aligned}
U(X) &= \left\{ 1 + \left[1 - (1-g)^{TP_S/a} \right] \cdot U(TP_S) \right\} \cdot u(X) \\
&+ \left\{ \left[1 - (1-g)^{TP_F/a} \right] \cdot U(TP_F/1) \right\} \cdot [1-u(X)]
\end{aligned} \tag{2.13}$$

where $d(X)$ is [KL99]

for $X = 1$

$$d(1) = f \left[1 - (1-g)^M \right]$$

for $X \neq 1$

$$\begin{aligned}
d(X) &= \frac{a}{1 - (1-g)^{(X/a)M}} \left(\sum_{k=1}^{\infty} \left\{ (1-p)^k \right. \right. \\
&\quad \left. \left. - (1-g)^{X/a} \left[(1-p)^k - (1-g)^k \right] \right\}^M \right. \\
&\quad \left. - (1-g)^{(X/a)M} \sum_{k=1}^{\infty} (1-g)^k \right)
\end{aligned} \tag{2.14}$$

and $u(X)$ is[ZiA02]

for $X = 1$

$$u(1) = \frac{1}{1 - (1 - g)^M} M g (1 - g)^{M-1}$$

for $X \neq 1$

$$u(X) = \sum_{n=1}^M \left\{ np(1-p)^{n-1} + \left[np(1-p)^{n-1} + (M-n)g(1-g)^{M-n-1} - n(M-n)pg(1-p)^{n-1}(1-g)^{M-n-1} \right] \frac{(1-p)^n(1-g)^{M-n}}{1 - (1-p)^n(1-g)^{M-n}} \right\} \quad (2.15)$$

$$\cdot \left\{ \frac{\binom{M}{n} \left[1 - (1-g)^{X/a} \right]^n (1-g)^{(X/a)(M-n)}}{1 - (1-g)^{(X/a)M}} \right\}.$$

The number of packet arrivals during the last slot of the Idle Period determines the lengths of the Busy and Useful Time Periods. Thus, for $j \geq 1$ the expected value of the Busy Period is $\bar{B} = B(1)$ and the expected value of the time spent in useful transmission is $\bar{U} = U(1)$. The expected value of the Idle Period, $\bar{I}$, remains the same from the No-ACK CSMA/CA analysis. Placing these values into the original throughput equation (2.3) it follows that

$$S = \frac{U(1)}{\left(B(1) + \frac{a}{1 - (1-g)^M} \right)} \quad (2.16)$$

To find the system throughput, S , it is necessary to find $B(TP_S)$, $B(TP_F)$, $U(TP_S)$ and $U(TP_F)$. To solve $B(TP_S)$ and $B(TP_F)$ take Equation (2.12) and set $X = TP_S$ and $X = TP_F$. This produces two equations with two unknowns, such that

$$\begin{aligned}
& B(TP_S) \cdot \left\{ u(TP_S) \cdot \left[1 - (1-g)^{TP_S/a} \right] - 1 \right\} \\
& + B(TP_F) \cdot \left\{ \left[1 - u(TP_S) \right] \cdot \left[1 - (1-g)^{TP_F/a} \right] \right\} \\
& = TP_F \cdot \left[u(TP_S) - 1 \right] - TP_S \cdot u(TP_S) - d(TP_S)
\end{aligned}$$

(2.17)

and

$$\begin{aligned}
& B(TP_S) \cdot \left\{ u(TP_F) \cdot \left[1 - (1-g)^{TP_S/a} \right] \right\} \\
& + B(TP_F) \cdot \left\{ \left[1 - u(TP_F) \right] \cdot \left[1 - (1-g)^{TP_F/a} \right] - 1 \right\} \\
& = TP_F \cdot \left[u(TP_F) - 1 \right] - d(TP_F) - TP_S \cdot u(TP_F).
\end{aligned}$$

$B(TP_S)$ and $B(TP_F)$ can now be found via a linear algebra inverse matrix operation.

$U(TP_S)$ and $U(TP_F)$ are found in the same manner as $B(TP_S)$ and $B(TP_F)$, producing

$$\begin{aligned}
& U(TP_S) \cdot \left\{ u(TP_S) \cdot \left[1 - (1-g)^{TP_S/a} \right] - 1 \right\} \\
& + U(TP_F) \cdot \left\{ \left[1 - u(TP_S) \right] \cdot \left[1 - (1-g)^{TP_F/a} \right] \right\} \\
& = -u(TP_S)
\end{aligned}$$

(2.18)

and

$$\begin{aligned}
& U(TP_S) \cdot \left\{ u(TP_F) \cdot \left[1 - (1-g)^{TP_S/a} \right] \right\} \\
& + U(TP_F) \cdot \left\{ \left[1 - u(TP_F) \right] \cdot \left[1 - (1-g)^{TP_F/a} \right] - 1 \right\} \\
& = -u(TP_F)
\end{aligned}$$

$U(TP_S)$ and $U(TP_F)$ can now be found via a linear algebra inverse matrix operation.

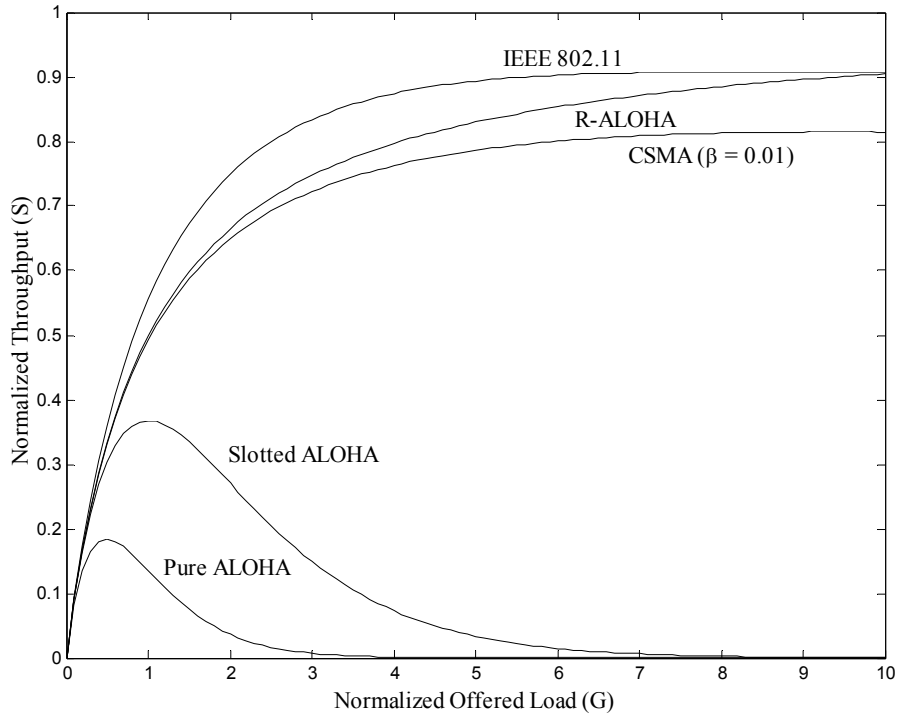


Figure 6. Performance of IEEE 802.11 verses ALOHA and CSMA

Figure 6 shows the normalized throughput of various types of ALOHA, CSMA, and IEEE 802.11 protocols. The graph of CSMA is shown with $\beta = 0.01$. IEEE 802.11 is shown with the propagation delay = 1 μ s, the Slot Time = 20 μ s, the SIFS period = 10 μ s, the DIFS period = 50 μ s, the Frame Size = 18,848 bits (maximum size of IEEE 802.11 frame), the ACK Frame Size = 240 bits, the Channel Capacity = 1 Mbps, $p = 0.05$, and the number of stations (M) = 4.

Figure 6 shows that IEEE 802.11 outperforms the other MAC protocols, producing a maximum normalized throughput of 90%. However, IEEE 802.11 is a relatively complicated protocol and is far more difficult to implement than ALOHA or CSMA. Thus, the performance gains in IEEE 802.11 are made by sacrificing simplicity.

2.4. Current Research Efforts

Many current research efforts involving MAC protocols focus on time-sensitive data, meaning the data must reach its intended destination before a certain deadline or the data is no longer useful. Time sensitive systems, or real-time systems, fall into two types: hard and soft. If missing a deadline causes a catastrophic failure in the system, the system is known as a hard real-time system. An example of a hard real-time system is an automated vehicle guidance system. Systems that can tolerate some delay beyond a scheduled delivery time are known as soft real-time systems, of which digital voice traffic is a good example. Because of their sensitivity to delays, real-time data is not normally transmitted over a wired (not to mention wireless) shared network.

To meet the needs of real-time systems on a wireless network, one approach modifies the DIFS to give an advantage to real-time traffic over nonreal-time traffic. Another gives an advantage to real-time hosts by transmitting pulses of energy before sending packets. A third approach is the protocol Real-Time MAC (RT-MAC).

2.4.1. Modifying Channel Free Wait Times (DIFS)

One MAC layer protocol used for hard real-time traffic is called Elimination by Sieving-Distributed Coordination Function (ES-DCF) [PDO02]. This protocol uses a dynamic distributed sieve-like mechanism in the collision avoidance phase of the channel access cycle for each real-time node. The MAC layer protocol used by non-real-time nodes the almost same as IEEE 802.11's DCF. However, each frame is given a grade based on how close the frame is to its deadline. The closer to its deadline, the lower the grade. The lower the grade, the smaller the channel free wait time (DIFS). For this

protocol to work, nonreal-time nodes must use a considerably larger DIFS value than any of the real-time nodes. Since the large DIFS value for the non-real time nodes is often greater than that specified in IEEE 802.11, this protocol cannot operate within an existing IEEE 802.11 network.

A similar method is called forward backoff scheme [LL03]. Depending on the network traffic load, the forward backoff scheme automatically adjusts the contention window boundary between real-time traffic and non-real-time traffic. By using such a scheme, real-time traffic always has a smaller backoff time than non-real-time traffic and real-time data is delivered before non-real-time data. Additionally, call admission control (CAC) is used which provides a Quality of Service (QoS) for soft-real-time data. The CAC determines whether a requesting connection can be accepted based on the connection bandwidth, the bandwidth currently in use, and the capacity of the network. By keeping less important frames off the medium, the CAC avoids unnecessary collisions caused by low priority data and traffic overload can be avoided. The protocol has the advantage of being compatible with IEEE 802.11, but it is only suitable for soft-real-time systems.

2.4.2. Black-Burst (BB) Contention Mechanism

Another method proposed for real-time traffic delivery is the Black-Burst contention mechanism [SK99]. With this scheme, real-time nodes contend for access to the channel with pulses of energy (so called BB's), the durations of which are a function of the frame's deadline. The closer a host's frame is to its deadline, the longer the BB is. This way all hosts can negotiate which has the highest priority transmission, after which

that host gains exclusive access to the channel. Real-time packets are not subject to collisions and have priority access over non-real-time data packets. The performance of the network approaches that attained under ideal time division multiplexing (TDM) via a distributed algorithm that groups real-time packet transmissions into chains [SK99]. However, sending BBs for each real-time packet wastes bandwidth.

2.4.3. Real-Time Medium Access Control (RT-MAC)

RT-MAC [Bal99] uses two additional pieces of information not used in IEEE 802.11: a transmission deadline (TD), which the sending node uses to determine if a piece of data has passed its deadline, and the transmitting node's next backoff value (BV). RT-MAC uses a Transmission Control Algorithm to manage the TD and an Enhanced Collision Avoidance (ECA) Algorithm to control BVs.

The Transmission Control Algorithm places a TD on any frame with real-time data i.e., the time by which a transmission must begin. The TD is only required until the frame is successfully transmitted or discarded, and thus does not need to be part of the frame itself. This maintains compatibility with existing IEEE 802.11 networks. If the TD expires, the Transmission Control Algorithm discards the frame and it is not transmitted.

The ECA algorithm has two parts. First, instead of utilizing a fixed initial value for the $CW_{\min}$ the algorithm uses

$$CW_{\min} = 2 + \left\lceil \frac{6}{\sqrt{C}} \right\rceil \hat{N} \quad (2.19)$$

where $\hat{N}$ is an estimate of the number of hosts in the network and C is the channel data rate in Mbps. For a detailed explanation of the equation, see [BFO96] and [Bal99]. The ratio has the effect of making the number of collisions suffered on a network less dependent on the number of host. Although this will lower the number of collisions, it will not eliminate them. To counter this, the second component of ECA is employed. In ECA, all hosts advertise their *next* BV as well as tracking other host's BVs. If a host has the same BV as another host, it will select another BV using a smaller contention window range than the first BV selected, further reducing collisions and thus delays in the system. RT-MAC is compatible with IEEE 802.11 and can work with both soft and hard real-time systems.

2.5. Summary

This chapter discusses wireless LANs. Section 2.1 presented an overview of the Open Systems Interconnection (OSI) model. The DLL layer of the OSI model is be the focus of this research. Notable MAC protocols were presented in Section 2.2, including ALOHA, CSMA, and IEEE 802.11. Each was briefly described and compared, along with a brief tutorial of IEEE 802.11. Finally, related research into real-time wireless networks was presented in Section 2.4. The focus on research thus far has been on 1) modifying the DIFS to give an advantage for real-time traffic over nonreal-time traffic, 2) giving an advantage to real-time hosts by jamming the channel with pulses of energy before sending their packets, and 3) RT-MAC with its transmission deadline and its Enhanced Collision Avoidance (ECA) Algorithm.

This page intentionally left blank

3. Objectives and Methodology

3.1. Introduction

This chapter discusses the problem definition, specific research objectives, and a solution methodology. First, the problem definition is discussed including the reason for this research. Second, the objectives are presented followed by discussion of the hardware used. Finally, a solution methodology is presented in detail to include the system boundaries and parameters, evaluation technique, and experiment design and validation.

3.2. Research Goals

The goal of this research is to develop a hardware test bed for IEEE 802.11. Extensive work has been done on modeling, simulating, and suggesting improvements to the 802.11 MAC layer protocol. The purpose of this thesis was to create a laboratory prototype on which these improvements can be tested and verified.

3.3. Approach

To meet the goal, four hardware test beds are set up as IEEE 802.11 nodes. The test beds are all XInC Professional Development Kits produced by Eleven Engineering Incorporated [EE04]. They have an interface board and an RF unit. The boards have a proprietary processor programmed in assembly language and support eight hardware threads. Each thread behaves as an independent processor with its own access to main memory and the peripheral bus. Each thread runs at 6.25 MHz. The RF unit can support up to a 3 Mbps transmission rate.

3.4. System Boundaries

The system under test (SUT) is the MAC protocol itself (see Figure 7). The specific component under test is the IEEE 802.11 Distributed Coordination Function (DCF). The Basic Access Method of the Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) is implemented. The transmission rate is set at 1 Mbps.

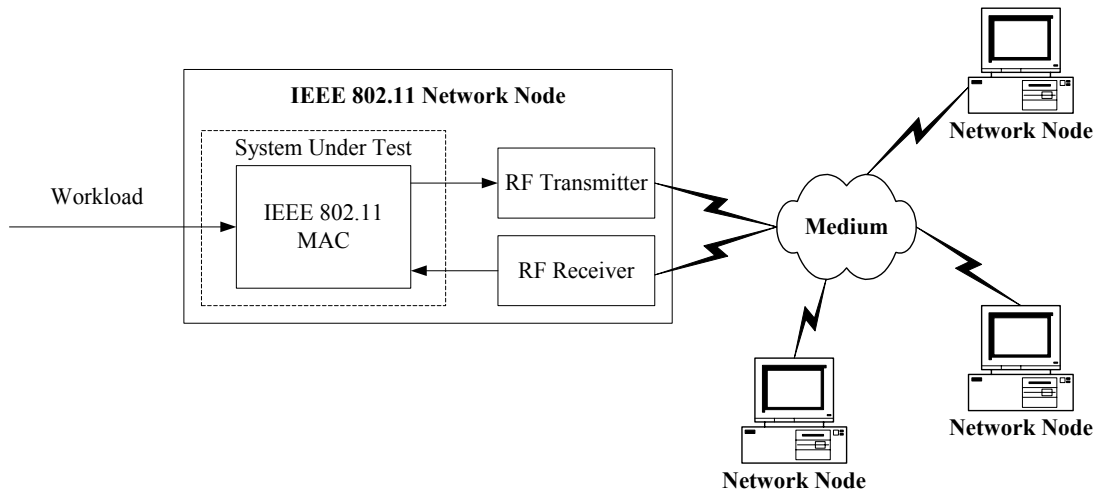


Figure 7. System Boundaries

3.5. System Services

The system provides only one service: Delivery of data by transmitting frames of binary information. The service was designed to guarantee delivery of a data frame, and had two possible outcomes: successful delivery of frame (success) or no delivery (failure).

3.6. Performance Metrics

The performance metrics are throughput and mean delay. Throughput is defined as the size of the data (in bits) sent divided by the amount of time needed to successfully

receive it. Throughput is one of the key measurements for any network, for it provides insight into the network's capacity and provides a basis for comparing protocols.

Mean delay is calculated as the arithmetic mean of the time difference from frame creation to successful reception of the last bit of an ACK from the receiving node. Mean delay is an important metric to collect, for it allows the hardware set to be compared to other IEEE 802.11 models.

3.7. Parameters

The system parameters for this experiment are as follows:

- Number of Stations – The number of stations can greatly affect the performance of a network. There are only four boards available for this experiment, and thus the total number of stations required for this research is between two and four.
- Physical Layer Transmission Speed – The XInC test beds are capable of transmitting up to 3 Mbps. For this research transmission speed was restricted to 1 Mbps, which follows the IEEE 802.11 standard.
- Capture – Capture is a technique where a station can retrieve a single transmission from many that are simultaneously transmitted. The test boards are not capable of performing capture, thus for this research capture is not used.

- Network Topology – The network topology for this experiment is a shared common bus.
- MAC Protocol – The MAC protocol for this research is the Distributed Coordination Function (DCF) of IEEE 802.11.
- Packet Queue Size – The hardware has a limited physical memory space. For this reason, the packet queue is restricted to 256 MAC frames (regardless of packet size). All packets presented to the MAC layer while the packet queue is full are discarded.
- MAC Protocol Parameters – Listed below in Table 2. All are taken from [IEEE99] except the PHY header length, which is a physical property of the test set boards.

Table 2. MAC Protocol Parameters

Mac Parameter	Value
Minimum Width of Contention Window (CW _{min})	31
Maximum Width of Contention Window (CW _{max})	1023
Slot Time	20 μ s
Short Inter-frame Spacing (SIFS)	10 μ s
Distributed IFS (DIFS)	50 μ s
Extended IFS (EIFS)	1068 μ s
ACK length	14 bytes
PHY header length (Preamble + Postamble)	16 bytes
ACK timeout	212 μ s

The workload parameters for this experiment are as follows:

- Traffic Model - The type and format of traffic used by the system has a great bearing on system performance. For this research, two forms of traffic models are investigated for the following applications: telemetry and avionics. The telemetry application is modeled after the MIL-STD-1553B data bus. The avionics traffic model is representative of the Boeing 777 data bus. Both traffic characteristics are described in detail in Section 3.10.
- Normalized Offered Work Load - This parameter is defined as the amount of traffic all stations produce divided by the maximum traffic the network can support.

3.8. System Factors

The factors and corresponding values for this experiment are:

- Numbers of Stations – (2, 3 and 4) – Wireless networks are ad hoc in their implementation, meaning the number of stations can vary greatly from one implementation to the next. For this reason, the number of stations is varied from two to four.
- Normalized Offered Work Load – (0.2, 0.33, 0.5, 0.66, 0.8, and 1.0) – The Normalized Offered Work Load is intended to offer the network a series of loads representing light, medium, and high loads.

3.9. Evaluation Techniques

The experiment is conducted using direct system measurement. This technique is selected since the research goal is to validate analytic and simulation results on a laboratory prototype test set. Direct measurement of prototype results provides an immediate means of accomplishing the research goal. Results of the measurement technique are validated using an analytical model for IEEE 802.11 [ZiA02].

3.10. Workload

The workload is intended to emulate one of two applications: a telemetry model based on the MIL-STD-1553B data bus or avionics traffic model based on the avionics Boeing 777 bus. These applications represent a small and large packet size to bring different loads on the channel. The telemetry traffic model has a fixed frame size of 84 bytes, while the avionics traffic model uses a fixed frame size of 776 bytes. Due to limitations of the boards, frame arrival rate for both models follows a uniform distribution.

3.11. Experimental Design

The experimental design for this research is a full factorial design with replications. The full factorial design allows examination of every possible combination of configuration and workloads. Replication allowed for estimation of experimental errors and establishment of a suitable confidence interval, and thus each combination of factors was replicated five times. The number of factors, levels, and repetitions results in

$$\begin{aligned}
& (\text{Total Number of Experiments}) = (\text{Number of Stations}) \times (\text{Normalized Offered Workload}) \\
& \quad \times (\text{Traffic Model}) \times (\text{Number of Replications}) \\
& = (3) \times (6) \times (2) \times (5) \\
& = 180 \text{ Experiments.}
\end{aligned}$$

3.12. Analyze and Interpret Results

The effects of selected factors are quantified to determine if the hardware setup is statistically different from an analytical model of IEEE 802.11. The observations for the experimental and analytical IEEE 802.11 throughput are paired observations, and thus the analysis is straightforward. The recorded metrics from the hardware setup are treated as one observation, from which a confidence interval is computed. A visual statistical test is used to determine if the experimental data matches the analytical data.

For the IEEE 802.11 Mean Delay data, all the analytical models found required saturation of the channel for them to work. Only a few data points on the experimental data are in saturation, and thus a statistically different between the analytical and experimental data could not be determined. Instead, the experimental data is just presented.

3.13. Summary

This chapter defines the testing methodology of an implementation of the IEEE 802.11 protocol on a hardware device. The chapter describes the system boundaries, which are defined as the MAC layer itself. The system's service is the delivery of data by transmission of frames of binary information. The chapter identifies the performance metric as throughput and the system parameters and draws from them three factors:

Number of stations, traffic model, and the normalized offered workload. Direct system measurement is the evaluation technique. The workload used is a telemetry model based on MIL-STD-1553B data bus and an avionics bus of a Boeing 777. Finally, the chapter concludes by describing the full factorial experimental design and explains the evaluation technique to determine if the hardware setup produces results like those found in an analytical IEEE 802.11 model.

4. Experiments, Data, and Results

4.1. Introduction

This chapter introduces the experimental design and results obtained during the test runs. First, a description is given of the design. Discussed next is the experimental setup. Finally, results are discussed, comparing the experimental and analytical results with an explanation given to any discrepancies.

4.2. The XInC test set

The XInC test set is manufactured by Eleven Engineering Inc [EE04] (Figure 8). The test set consists of a test board with a XInC (pronounced “zinc”) RISC-based processor connected to an RF Waves 1 Mbps/3 Mbps adapter card (Figure 9), a RF Waves 2.4 GHz 3 Mbps DSSS RF Module (Figure 10), and an assembly code compiler for the XInC machine language. The 16-bit XInC processor supports eight hardware threads, acting as eight independent processors, each with access to main memory and the peripheral bus. The threads share hardware resources with the exception of each thread’s dedicated register set. The board’s system clock runs at 50 MHz, and all hardware thread execute at 1/8 of the system clock (or at 6.25 MHz). The RF Waves 2.4 GHz RF Module operates with direct sequence spread spectrum (DSSS) encoding in the 2.4 GHz band, which is designated for Industrial, Scientific, and Medical (ISM) application. The XInC machine language consists of 18 instructions with six address modes and supports 26 instruction/address-mode combinations.

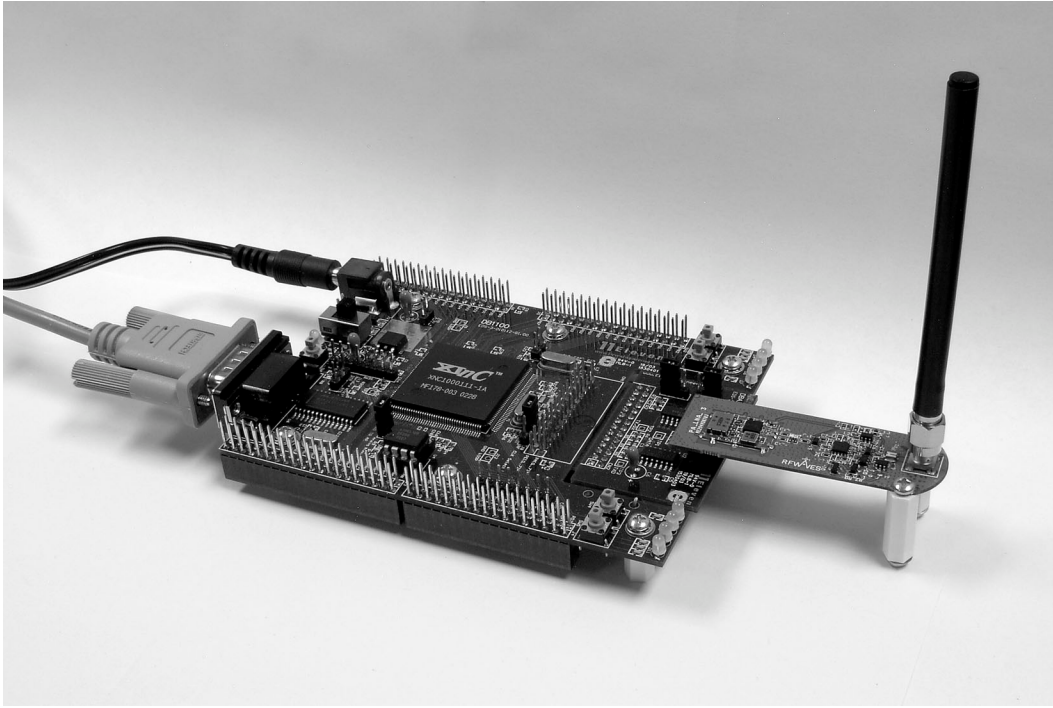


Figure 8. Eleven Engineering Inc XInC 2.4 GHz RF 3.0 Mbps DSSS Development Kit

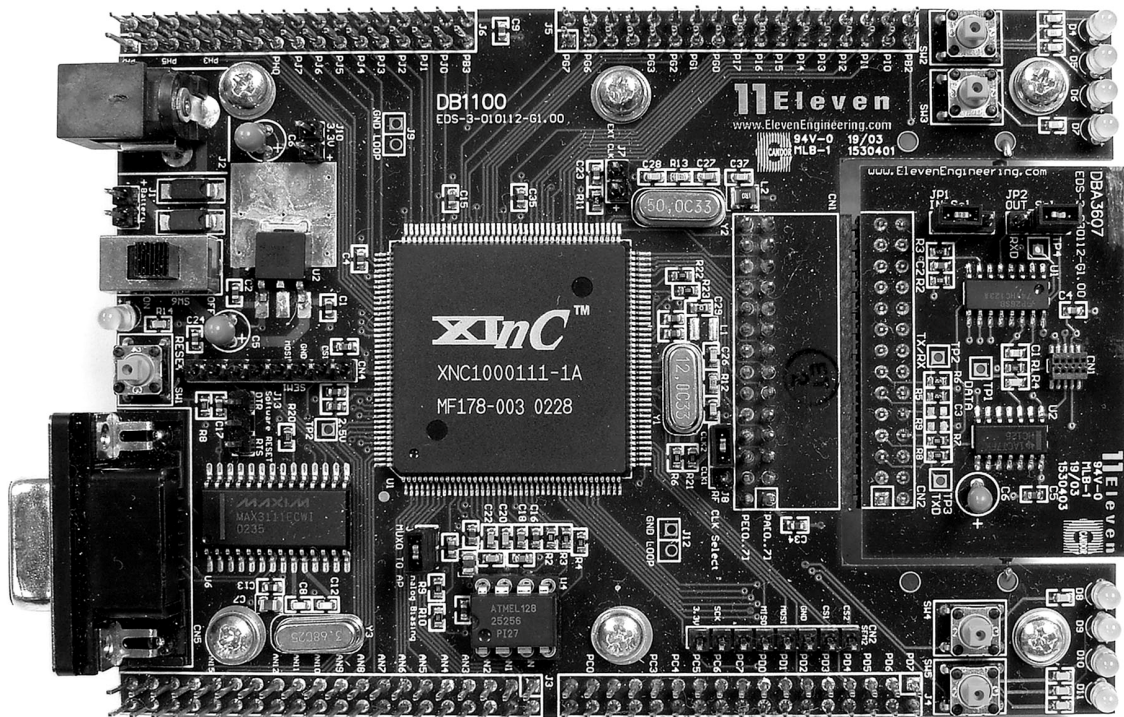


Figure 9. XInC Development Board with a RF Waves 1 Mbps/3 Mbps adapter card

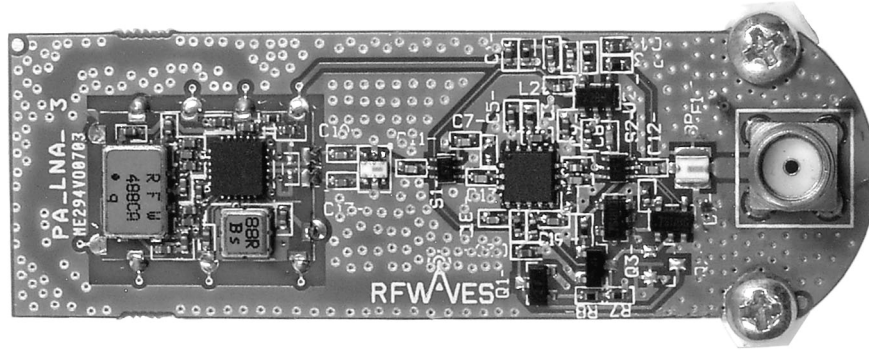


Figure 10. RF Waves 2.4 GHz, 3 Mbps DSSS RF Module

For this research, each of the eight hardware threads is programmed to carry out a specific set of tasks:

- Thread 0 – The main thread running the IEEE 802.11 protocol. It also handled all packet transmission.
- Thread 1 – Polling thread. This thread runs a clock that tells Thread 0 when it can transmit a packet. Thread 1 creates the slotted the channel in accordance with IEEE 802.11.
- Thread 2 – Receiver thread. Receives all packets transmitted on the medium, determines if the packets are for the node, in the proper order, and without errors. It also communicates to Thread 0 whenever the medium is sensed busy.
- Thread 3 – Random Number Generator. Using a 16-bit linear shift register, this thread produces uniform random numbers. The random

numbers are used by Thread 3 to calculate backoff values for the IEEE 802.11 protocol in Thread 0.

- Thread 4 – Timing Thread. The thread runs a clock storing the time in seconds, milliseconds, and microseconds. This is necessary because the running clock on the boards roles over after only 1.31 ms.
- Thread 5 – Packet Generation. Offers packets to the MAC layer’s queue. It takes a random number generated by Thread 3 and uses it in conjunction this clock from Thread 4 to randomly offer packets to the queue. The thread also randomly chooses the destination address of the packet it loads into the queue. If the queue is determined full, it discards the packet.
- Thread 6 – Testing and Recording. Starts and stops the testing for each trial. The thread also records all the information gathered from each trail.
- Thread 7 – Print to Screen. This thread takes the data recorded by Thread 6 and displays it on computer attached to the boards. The data is then manually copied and saved to disk.

The final code has over 10,500 lines of code and took nine months to complete. It is shown in Appendix C.

4.3. Experimental Setup

For the experiment, the boards are programmed to transmit packets at a specified rate to provide a desired load on the channel. For all experimental runs, the boards were

turned on and started transmitting. They transmitted for at least 10 seconds to allow the system to stabilize, and then data was collected for 60 seconds from each board. The data collected included:

- Time of Test (t_n) – The total time of data collection.
- Packets Presented to the Queue (P_n) – The total number of packets presented to the MAC layer queue by the Network layer. The queue could hold 256 packets (regardless of the size of the packet). If a packet is presented to the MAC layer and the queue is full, the packet is discarded.
- Transmission Attempts (TA_n) – The number of initial packet transmissions attempted. Note that this was only recording the first attempt to send a packet and does NOT represent any re-transmissions.
- ACKs received (A_n) – The total number of ACKs received acknowledging a transmission from the node. These represent the total number of successfully transmitted packets.
- Transmissions Failed (TF_n) – The number of times a transmission was repeated four times (the initial transmission followed by three retransmissions). After this, the MAC discarded the packet.
- Mean Delay ($D_n^{(s)}, D_n^{(ms)}, D_n^{(\mu s)}$) – The total amount of time packets are waiting before delivery. It represents the difference from the time a packet is placed in the queue till the time an ACK is successfully received

by the transmitting node. The results are given in seconds ($D_n^{(s)}$), milliseconds ($D_n^{(ms)}$), and microseconds ($D_n^{(\mu s)}$).

An example of the data collected is shown in Table 3.

Table 3. Example of Telemetry Throughput Data

Station Number (n)	Time of Test (t_n)	Packets Presented to Queue (P_n)	TX Attempts (TA_n)	ACKs Received (A_n)	TX Failures (TF_n)	--Mean Delay--		
						sec ($D_n^{(s)}$)	ms ($D_n^{(ms)}$)	μs ($D_n^{(\mu s)}$)
1	60	10491	9047	8967	80	7044	2258	63084
2	60	10760	10715	10672	43	8357	51967	3912
3	60	10466	10362	10308	54	6617	8352	53404
4	60	10454	9632	9572	60	6863	7656	11712
								13211
Totals	240	42171	39756	39519	237	28881	70233	2
Number of Stations (M) = 4, Size of Data in Packet (d) = 672 bits (84 bytes),								
Channel Capacity (C) = 1 Mbps, Normalized Offered Load (G) = 1.17								

Using the collected data, the offered load is

$$G = M \frac{P \cdot (d + 352)}{C \cdot T} \quad (4.1)$$

where M is the total number of stations, P is the total number of packets placed in the

queue for all stations $\left(P = \sum_{n=1}^M P_n \right)$, d is the size in bits of the data placed in a MAC frame

in bits, plus 352 bits for the physical and MAC headers, C is the channel capacity in bits

per second (bps), and T is the total time of the test for all stations in seconds $\left(T = \sum_{n=1}^M t_n \right)$.

To calculate a trial's normalized throughput, S , each individual station's normalized throughput

$$S_n = \frac{d \cdot A_n}{t_n \cdot C} \quad (4.2)$$

is calculated where A_n is the number of ACKs received by an individual station, t_n is the time of an individual station's test. The total normalized throughput is $S = \sum_{n=1}^M S_n$.

The Mean Delay in seconds is

$$MD = \frac{(D^{(s)} + D^{(ms)} \cdot 10^{-3} + D^{(\mu s)} \cdot 10^{-6})}{(TA - TF)} \quad (4.3)$$

where $D^{(s)}$ is the system's total delay in seconds $\left(D^{(s)} = \sum_{n=1}^M D_n^{(s)}\right)$, $D^{(ms)}$ is the system's total delay in milliseconds $\left(D^{(ms)} = \sum_{n=1}^M D_n^{(ms)}\right)$, $D^{(\mu s)}$ is the system's total delay in microseconds $\left(D^{(\mu s)} = \sum_{n=1}^M D_n^{(\mu s)}\right)$, TA is the total number of transmission attempts $\left(TA = \sum_{n=1}^M TA_n\right)$, and TF is the total number of failed transmission attempts $\left(TF = \sum_{n=1}^M TF_n\right)$.

4.4. Experimental Results

This part of the chapter presents the experimental results. Appendix A contains the data (including confidence intervals) from which the figures in this chapter were made. Appendix C contains the MATLAB® code used to generate the figures.

4.4.1. Telemetry Results

The Telemetry traffic model uses a fixed frame size of 84 bytes. The packet arrivals to the MAC layer are based on a uniform random distribution. The short packet size induced a high network overhead as well as an increased number of transmissions when compared to a larger packet size.

4.4.1.1. Normalized Goodput

It should be noted that the experimental normalized offered load, G , was based on the load offered to the MAC layer. However, the analytical model used a G based on the load offered to the channel. Thus, the G of the experimental data was modified to correspond to the G used for the analytical data.

Figures 11, 12, and 13 show the experiment's results. The analytical data shown was calculated via the IEEE 802.11 throughput equation from [ZiA02] and detailed in Section 2.3.3.1. The experimental and analytical data are shown, with the whisker lines both above and below the experimental data points representing a confidence interval of 90%. The x-axis shows the normalized goodput or the useful system throughput. The y-axis shows the normalized offered load on the channel.

For all the telemetry experiments, the experimental and analytical data follow the same trends. The two data sets only start to diverge from one another when $G > 1$. This makes sense, for it is not possible for the experimental model to increase its throughput once the channel use reaches 100% (or $G = 1$). At $G = 1$, the MAC layer is transmitting packets at its maximum rate. For $G > 1$, the MAC is receiving packets at a faster rate than it can transmit them over the channel. Thus, the packets begin to fill up the MAC

layer's queue, which has a limited size of only 256 packets. With the rate of the number of packets presented to the queue larger than the rate at which the MAC layer can remove them from the queue (via successful transmission), the queue becomes full. Once the queue is full, the MAC layer discards all packets presented to it until a packet is transmitted and a slot opens up in the queue.

This effect can be seen in Figures 14, 15, and 16. The bar graphs in these figures show in the number of packets presented to the queue verses the number of attempted transmissions for a given offered load. A line showing the normalized throughput is also in the figure to show how the effects relate to one another. Although the offered load to the MAC layer increases, when $G \geq 1$ the offered load to the channel remains the same and thus the normalized throughput stays constant at around 0.5.

It is interesting to note that the queues become full at $G \approx 0.75$. This is due to the small packet size with its high overhead. For the Telemetry traffic model, even under ideal conditions the time spent transmitting data constitutes only 50% of a successful transmission (the rest of the time is taken up with the DIFS, the SIFS, the ACK, etc.). This percentage goes down significantly under heavy load conditions (due to an increase in a transmission's backoff value, collisions, number of retransmissions, etc.). Because of the overhead, the offered load placed on the MAC layer will be less than the offered load the MAC layer places on the channel. This means the MAC layer's queue will fill up before $G = 1$ because the rate that packets are presented to the queue is higher than the rate that the MAC layer can remove the packets from the queue via a successful

transmission. The experimental data backs up this statement, as the number of packets presented to the queue exceeds the number of transmission attempts at $G \approx 0.75$.

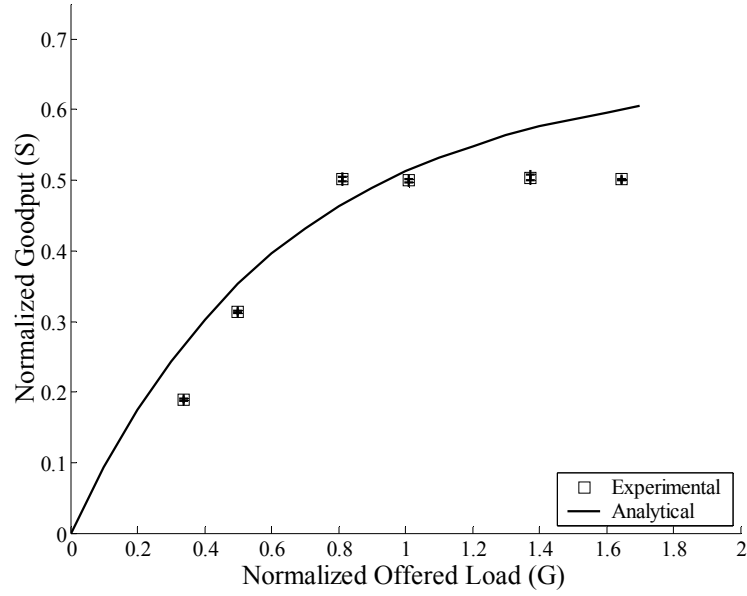


Figure 11. Telemetry Goodput ($M = 2$)

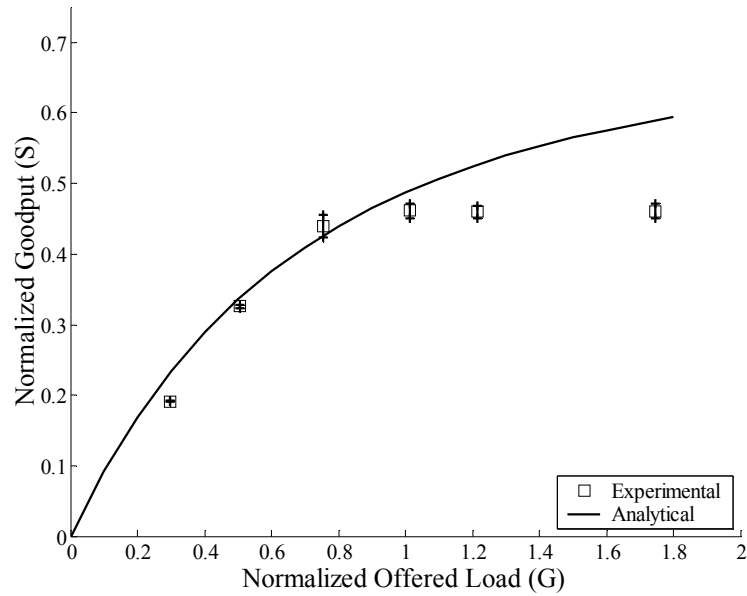


Figure 12. Telemetry Goodput ($M = 3$)

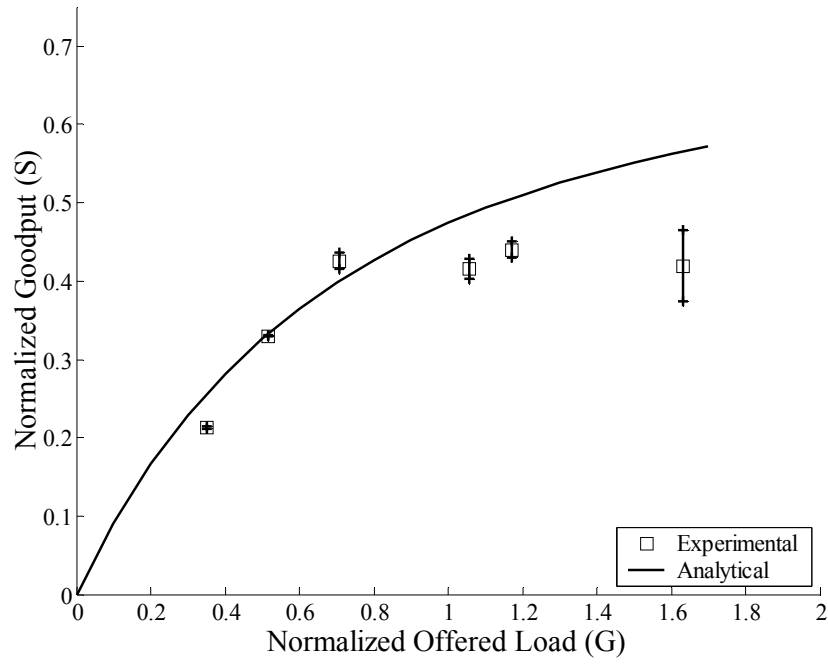


Figure 13. Telemetry Goodput ($M = 4$)

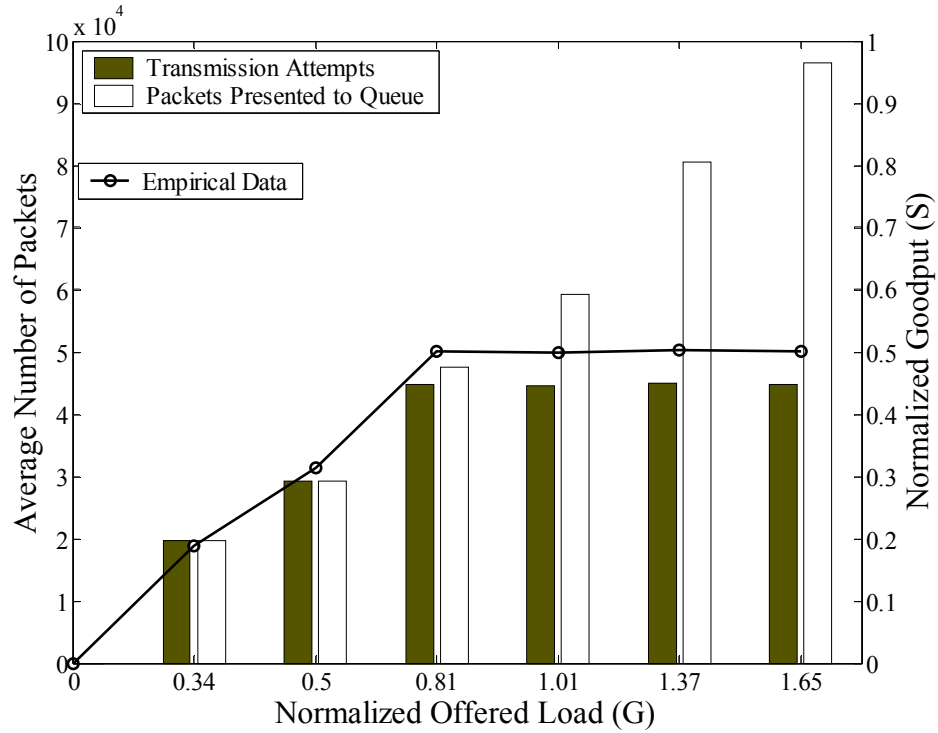


Figure 14. Number of Packets Presented to the Queue versus the Transmission Attempts compared to the Normalized Telemetry Goodput ($M = 2$)

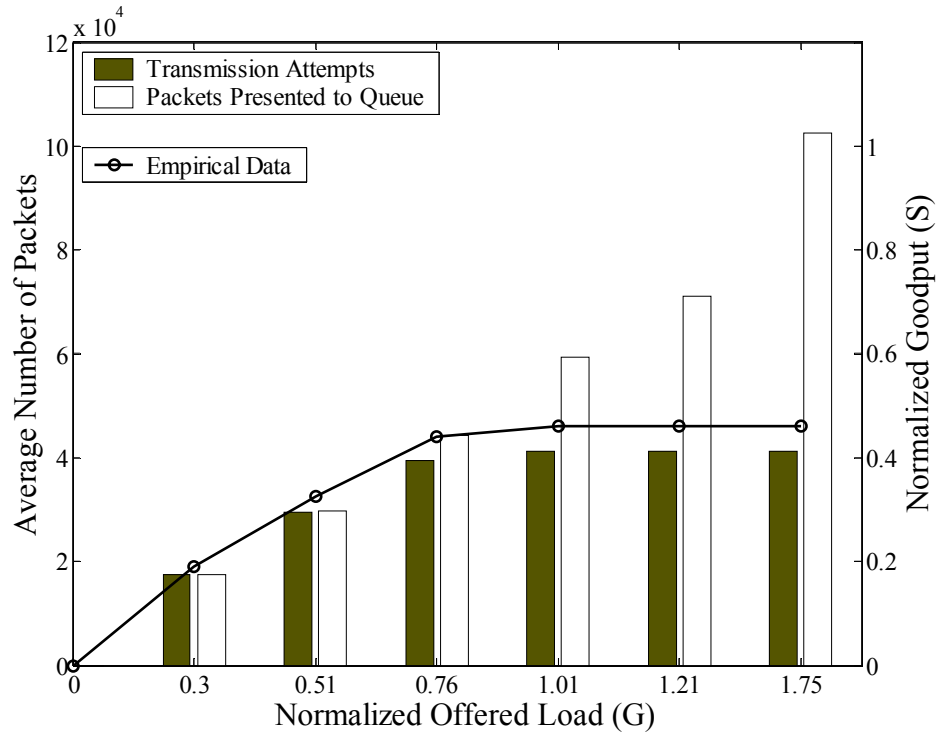


Figure 15. Number of Packets Presented to the Queue versus the Transmission Attempts compared to the Normalized Telemetry Goodput ($M = 3$)

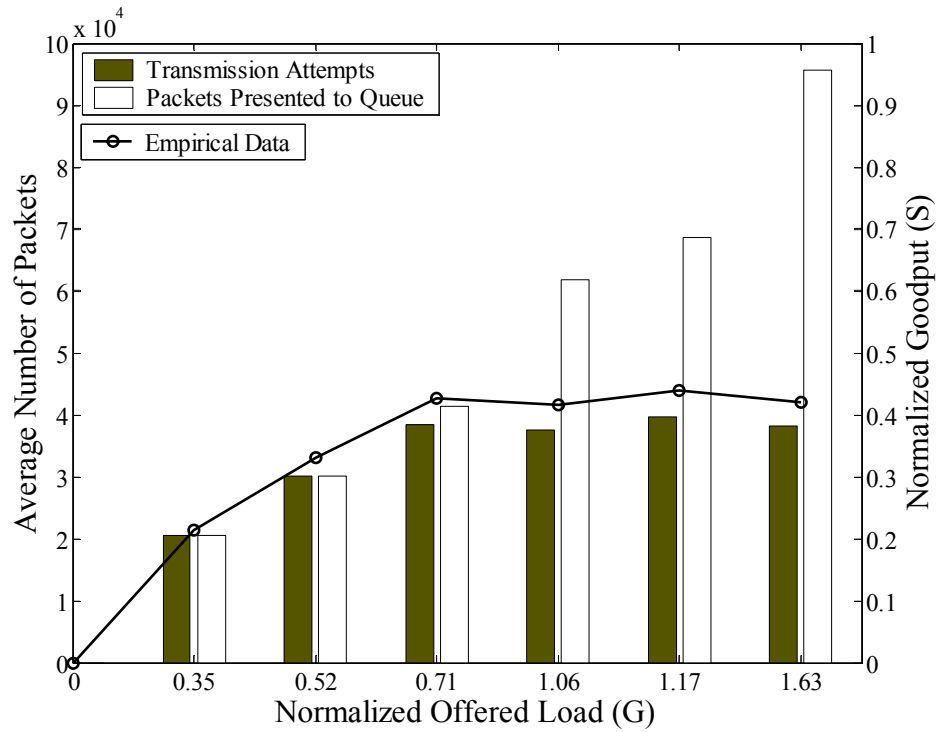


Figure 16. Number of Packets Presented to the Queue versus the Transmission Attempts compared to the Normalized Telemetry Goodput ($M = 4$)

4.4.1.2. Mean Delay

Mean delay is calculated as the arithmetic mean of the time difference from packet creation to successful reception of an ACK from the receiving node. The delay of discarded packets does not contribute to mean delay since, in effect, their delay is infinite. The results are displayed in Figures 17, 18, and 19. The experimental data is shown with the whiskers both above and below the experimental data points representing a confidence interval of 90%. The experimental results cannot be compared to an analytical model, for the analytical model depends on saturation.

For all the results, as the G increase, Mean Delay rises quickly, peaking at $G \approx 0.8$ and then dropping off slightly. This is caused by the fact that at $G \approx 0.8$ more packets are lost due to collisions (and thus forcing a longer delay) than by the use of a backoff mechanism. For instance, with $M = 2$ the average retransmission rate (and thus indicates lost packets) was twice as high for $G = 0.8$ than when $G = 1$.

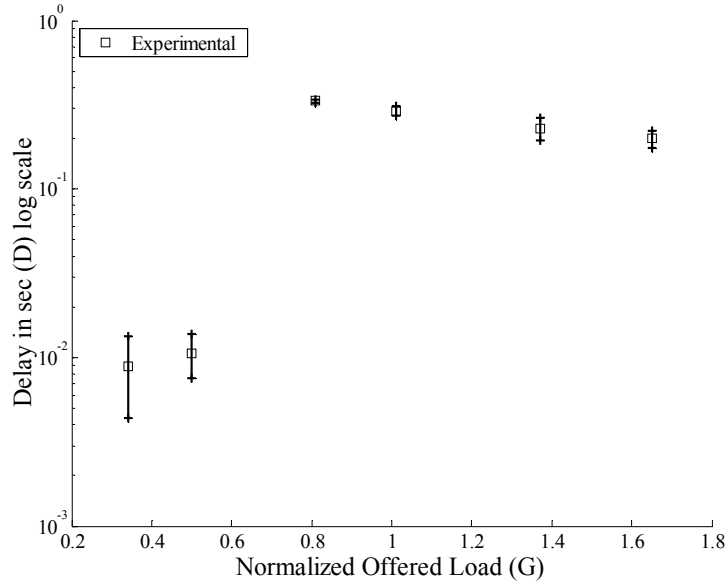


Figure 17. Telemetry Mean Delay ($M = 2$)

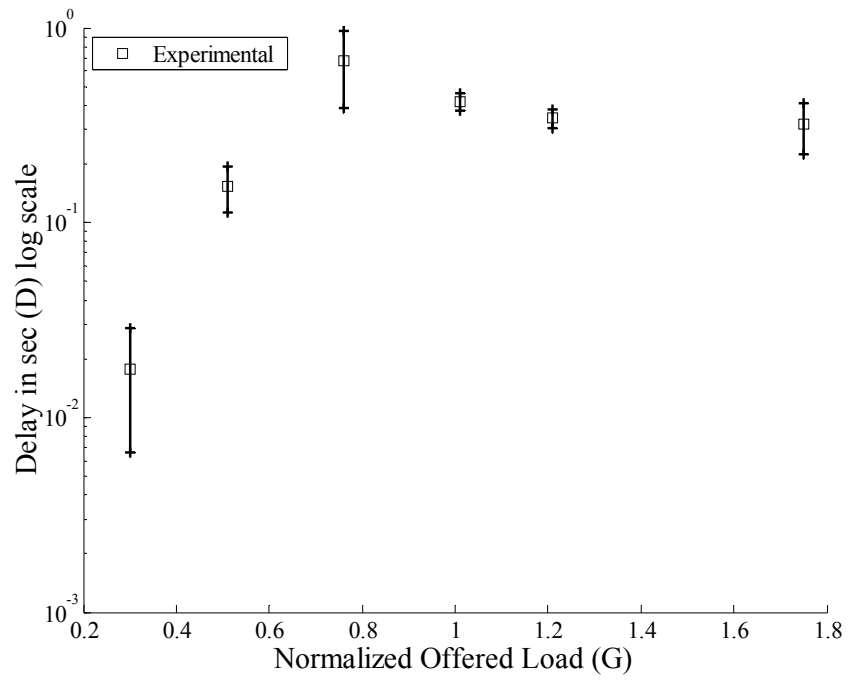


Figure 18. Telemetry Mean Delay ($M=3$)

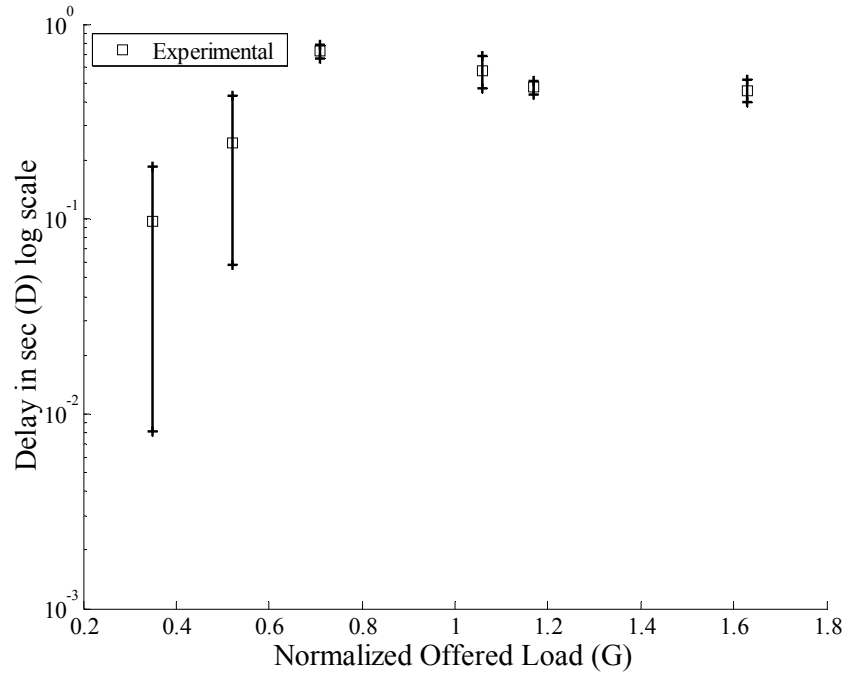


Figure 19. Telemetry Mean Delay ($M=4$)

4.4.2. Avionics Results

The Avionics traffic model uses a fixed frame size of 776 bytes. The packets arrived at the MAC layer in a uniform random distribution. The packet size is moderate, and thus will not induce a high overhead on the network nor as many transmissions as compared to the Telemetry model.

4.4.2.1. Normalized Throughput

It should be noted that the normalized offered load, G , was based on the load offered to the MAC layer. However, the analytical model used a G based on the load offered to the actual channel. Thus, the G of the experimental data was modified to correspond to the G used for the analytical data.

Figures 20, 21, and 22 show the results of the experiments. The analytical data shown was calculated via the IEEE 802.11 throughput equation from [ZiA02] and detailed in Section 2.3.3.1. The experimental and analytical data are shown, with the whisker lines both above and below the experimental data point representing a confidence interval of 90%. The x-axis shows the normalized goodput or the useful system throughput. The y-axis shows the offered load on the channel.

For all the avionics experiments, the experimental and analytical data follow each other rather closely. The two data sets only start to diverge from one another when $G > 1$. This makes sense, for it is not possible for the experimental model to increase its throughput once the channel use reaches 100% (or $G = 1$). At $G = 1$, the MAC layer is transmitting packets at its maximum rate. For $G > 1$, MAC is receiving packets at a faster rate than it can transmit them over the channel. Thus, the packets begin to fill up

the MAC layer's queue, which has a limited size of only 256 packets. With the rate of the number of packets presented to the queue larger than the rate the MAC layer can remove them from the queue (via successful transmission), the queue becomes full. Once the queue is full, the MAC layer will discard any packets presented to it until a packet is transmitted and a slot opens up in the queue.

This effect can be seen in Figures 23, 24, and 25. These figures show in the bar graph the number of packets presented to the queue versus the number of attempted transmissions for a given offered load. A line showing the normalized throughput is also in the figure to show how the effects relate to one another. Although the offered load to the MAC layer increases, when $G \geq 1$ the offered load to the channel remains the same and thus the normalized throughput stays constant at around 0.5.

It is interesting to note that the queues become full at $G \approx 0.6$. For the Avionics traffic model, even under ideal conditions the time spent transmitting data constitutes only 90% of a successful transmission (the rest of the time is taken up with the DIFS, the SIFS, the ACK, etc.). However, this percentage goes down significantly under heavy load conditions (due to an increase in a transmission's backoff value, collisions, number of retransmissions, etc.). Because of this, the offered load placed on the MAC layer will be less than the offered load the MAC layer places on the channel. This means the MAC layer's queue will fill up before $G = 1$ because the rate that packets are presented to the queue is higher than the rate that the MAC layer can remove the packets from the queue via a successful transmission. The experimental data backs up this statement, as the

number of packets presented to the queue exceeds the number of transmission attempts at $G \approx 0.7$ (when $M = 3$ or 4).

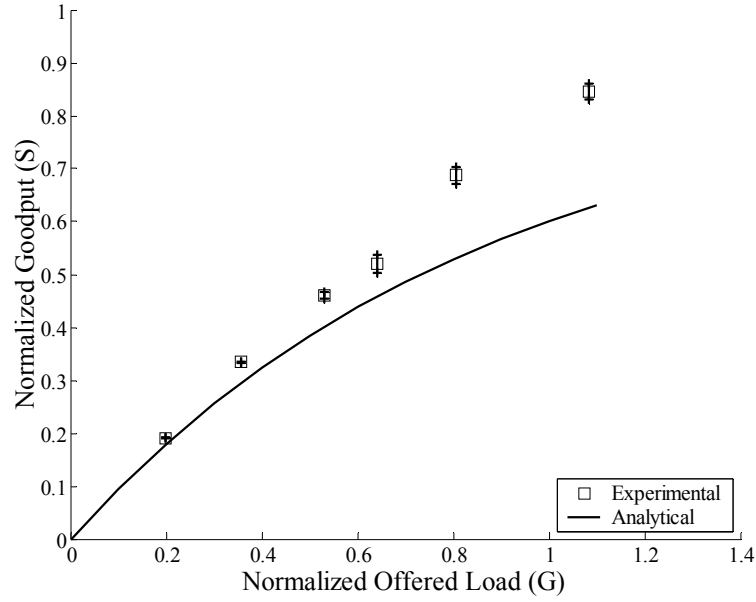


Figure 20. Avionics Goodput ($M = 2$)

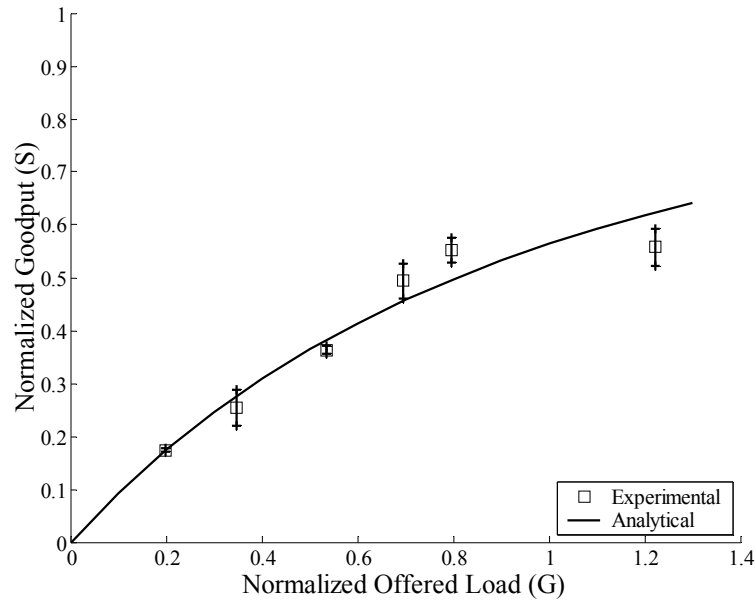


Figure 21. Avionics Goodput ($M = 3$)

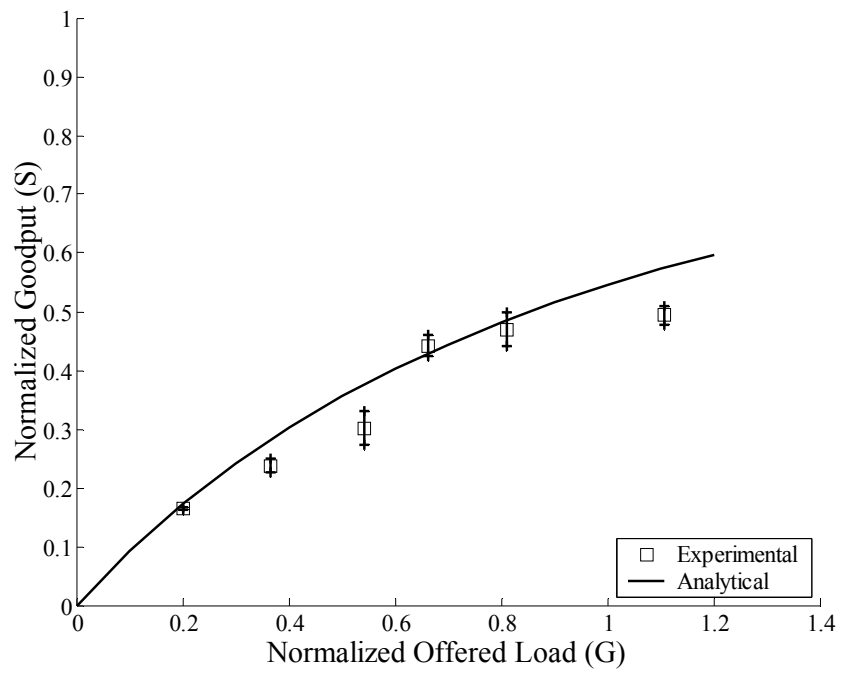


Figure 22. Avionics Goodput ($M = 4$)

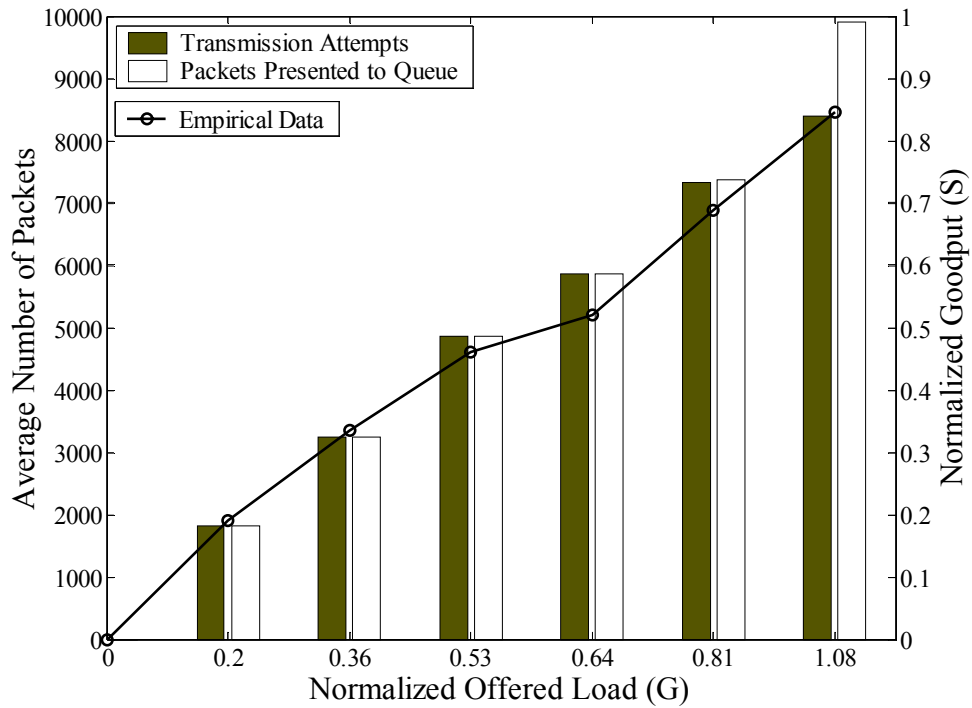


Figure 23. Number of Packets Presented to the Queue versus the Transmission Attempts compared to the Normalized Avionics Goodput ($M = 2$)

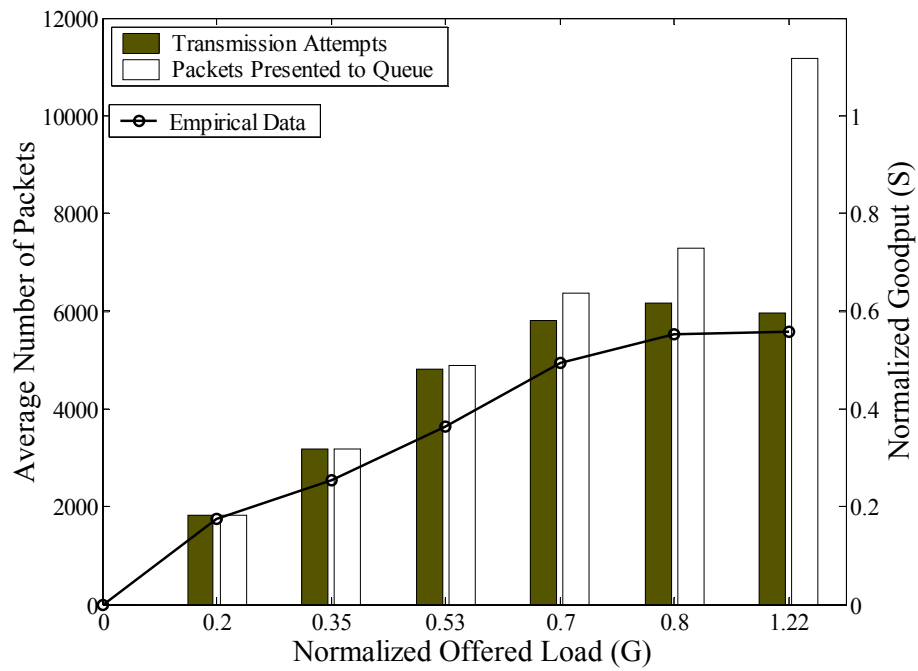


Figure 24. Number of Packets Presented to the Queue versus the Transmission Attempts compared to the Normalized Avionics Goodput ($M = 3$)

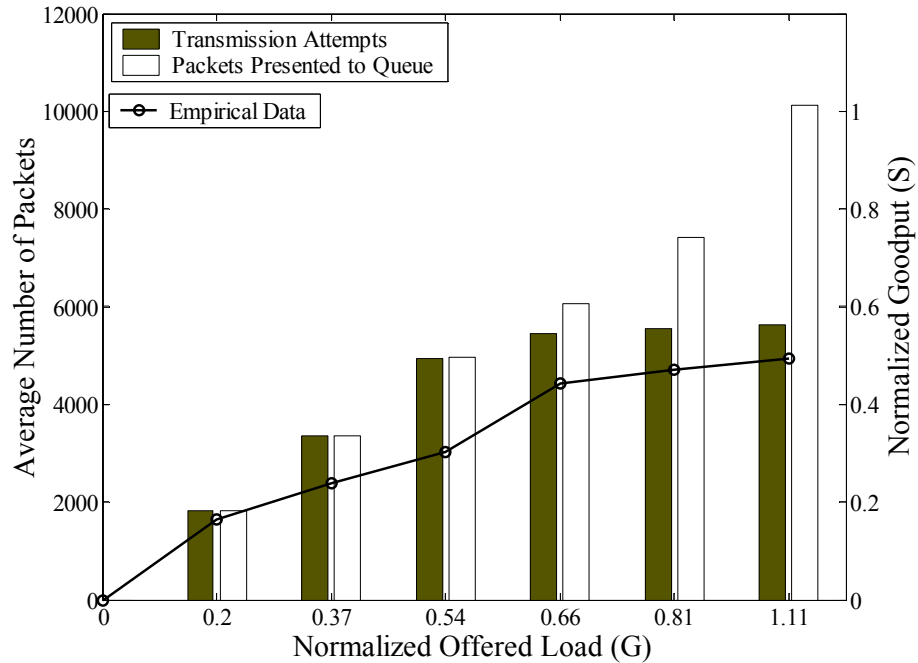


Figure 25. Number of Packets Presented to the Queue versus the Transmission Attempts compared to the Normalized Avionics Goodput ($M = 4$)

4.4.2.2. Mean Delay

Mean delay is calculated as the arithmetic mean of the time difference from packet creation to successful reception of an ACK from the receiving node. Delay that discarded packets suffer do not contribute to mean delay since, in effect, their delay is infinite. The results are displayed in Figures 26, 27, and 28. The experimental data is shown with the whiskers both above and below the experimental data points representing a confidence interval of 90%. The experimental results cannot be compared to an analytical model, for the analytical model depends on saturation.

For all the results, as the G increase, Mean Delay rises quickly. It tends to stabilize at $G \approx 0.8$ do to buffer overflow (Figure 24 and 25). The one exception to this is when $M = 2$. In these trials, there was no buffer overflow until $G = 1.08$ (see Figure 20) and the mean delay continued to increase until then.

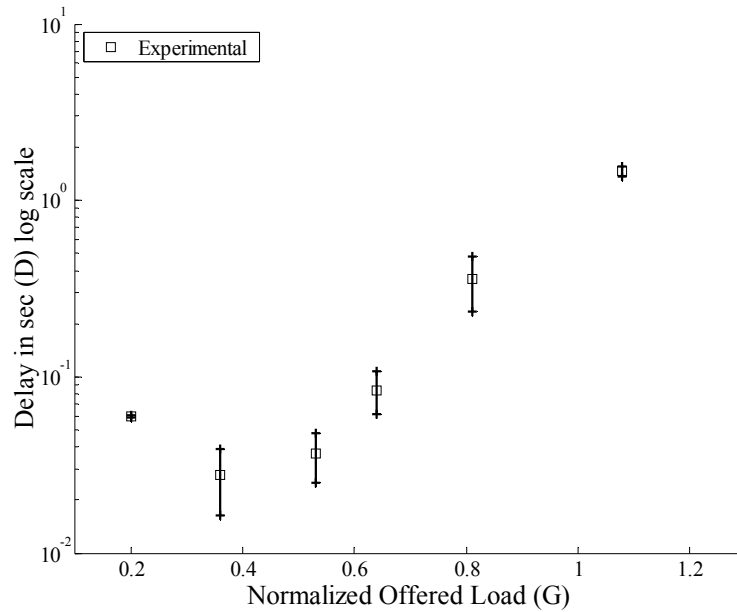


Figure 26. Avionics Mean Delay ($M = 2$)

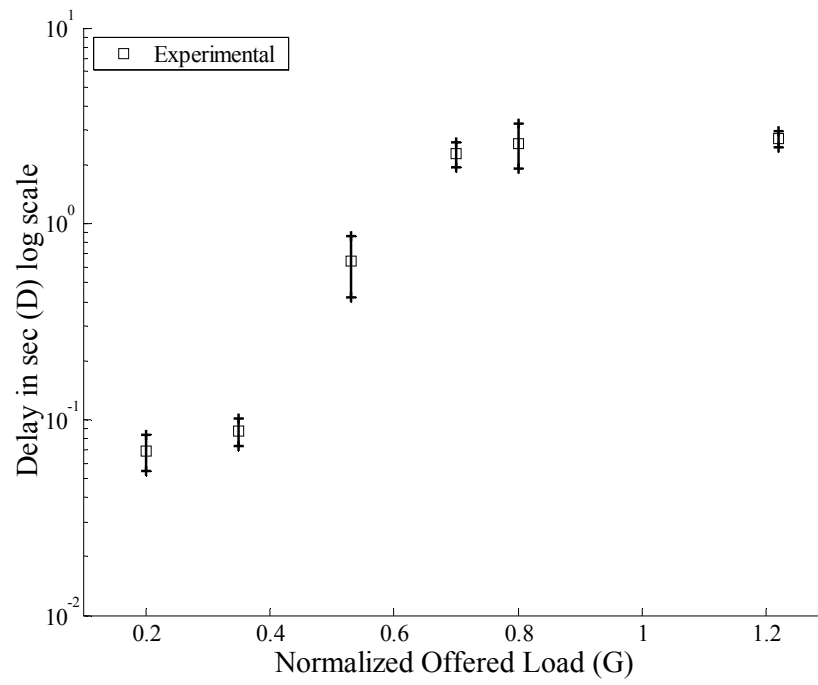


Figure 27. Avionics Mean Delay ($M=3$)

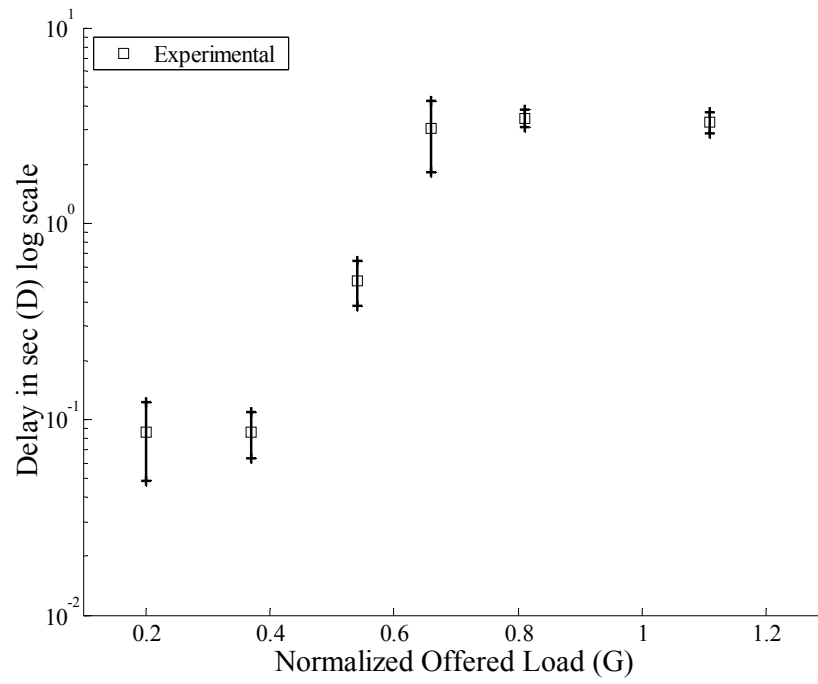


Figure 28. Avionics Mean Delay ($M=4$)

4.5. Summary

This chapter presented the experimental design and results obtained during the experimental test runs. It was shown that the experimental throughput data from both the telemetry and the avionics models follows the same trends analytical data. A discrepancy between the analytical and experimental data was caused by the hardware set saturating the medium. Since the throughput analytical data follows the same trends as the experimental data, this validates that the hardware setup produced for this research does mimic the Basic Access Method for the IEEE 802.11 Distributive Coordination Function (DCF). Data on the experimental mean delay was presented, but could not be compared to an analytical model.

5. Evaluation and Results

5.1. Introduction

This chapter reviews and summarizes the research and its objectives. First, the potential use of the research is discussed. Next, the objectives and the experiment are reviewed along with conclusions drawn. Last, potential follow on areas of study are outlined.

5.2. Research Impact

Wireless LANs based on IEEE 802.11 have been studied for years. There have been several simulation programs and analytical models developed to evaluate them, and researchers use them to improve IEEE 802.11. However, despite the abundance of commercially available IEEE 802.11 devices, researchers have encountered a great deal of difficulty obtaining a way to evaluate their proposed modifications or improvements experimentally. Vendors of Wireless LAN equipment do not sell, or sell at a very high price, their hardware and software development kits for IEEE 802.11. For this reason, this research has created a means via a hardware prototype that researchers could gain experimental data on IEEE 802.11.

5.3. Review and Conclusions

The experiment was run on a XInC development set produced by Eleven Engineering. Two traffic models were used: one for telemetry and one for avionics. After extensive testing, it was found that the experimental data from the boards produced normalized throughput levels that followed the same trends as an IEEE 802.11 analytical

model. Thus, the research has successfully shown that the hardware implementation can be used by researchers to gain experimental data about IEEE 802.11.

5.4. Outlines of Future Work

This thesis completed the initial groundwork in producing a hardware prototyping device for IEEE 802.11. It has set the stage for continued development and this thesis created the beginning components to make a functional IEEE 802.11 prototyping tool. Some future areas of research include:

- Develop a true IEEE 802.11 Clear Channel Assessment (CCA) procedure
 - The procedure used in this thesis to determine if the channel was idle is not one recommended for use by IEEE 802.11. One area of study, which could improve the prototype, is to develop the code that uses an IEEE 802.11 CCA procedure.
- Move packet production off the boards – This thesis used the boards themselves to produce packets to place in the queue for the MAC layer. However, the boards were very limited in this way. For instance, the boards were only capable of generating packets using a uniform random variable. The boards also had a very limited queue size of 256 packets. Future studies could move the packet creation and queuing off the boards and onto a PC, allowing for much more extensive experiments.

- Variable packet size – The current prototype can only be used for packets of fixed sizes. Developing code that can process packets of varying sizes would be very useful.
- Additional stations – This thesis was limited because only four boards were available. Another area of interest is to see if the protocol written for the board's works when 5, 10, or 20 boards are present.
- Testing other protocols – There are many other types of MAC protocols other than IEEE 802.11 (like RT-MAC) that could be tested out on the boards.

5.5. Summary

This thesis implemented in a hardware test bed for the IEEE 802.11 protocol on a Wireless Local Area Network (WLAN). Modeling and simulation work on the IEEE 802.11 protocol is extensive, and this thesis allows researchers the ability to validate analytic and simulation results on a laboratory prototype.

This page intentionally left blank

Appendix A - Experimental Data Tables

This appendix contains the data gathered during the hardware test set trials that was used to produce the graphs in Section 4. For all trials, the number of replications was $n = 5$ and the confidence interval was $\alpha = 0.10$. The mean values for the number of replications are given along with the high and low associated confidence interval. The figure reference in the table caption in brackets (i.e., [Figure 11] refers to the figure which used the data in the table).

Table 4. Telemetry Goodput ($M = 2$) [Figure 11]

Normalized Goodput (S)	Offered Load (G)					
	0.339	0.499	0.811	1.011	1.374	1.646
Upper Confidence Interval	0.190	0.316	0.505	0.503	0.508	0.502
Mean Normalized Goodput	0.189	0.314	0.502	0.500	0.504	0.501
Lower Confidence Interval	0.189	0.312	0.499	0.497	0.500	0.501

Table 5. Telemetry Goodput ($M = 3$) [Figure 12]

Normalized Goodput (S)	Offered Load (G)					
	0.299	0.506	0.755	1.013	1.213	1.748
Upper Confidence Interval	0.192	0.328	0.456	0.472	0.469	0.471
Mean Normalized Goodput	0.191	0.326	0.440	0.462	0.460	0.461
Lower Confidence Interval	0.190	0.324	0.424	0.451	0.450	0.451

Table 6. Telemetry Goodput ($M = 4$) [Figure 13]

Normalized Goodput (S)	Offered Load (G)					
	0.351	0.515	0.707	1.056	1.170	1.631
Upper Confidence Interval	0.214	0.331	0.437	0.429	0.450	0.465
Mean Normalized Goodput	0.213	0.330	0.426	0.415	0.440	0.420
Lower Confidence Interval	0.211	0.329	0.415	0.402	0.430	0.375

Table 7. Number of Packets Presented to the Queue verses the Transmission Attempts compared to the Normalized Telemetry Goodput ($M = 2$) [Figure 14]

	Offered Load (G)					
	0.340	0.500	0.810	1.010	1.370	1.650
Transmission Attempts	19736	29252	44847	44627	44993	44800
Packets Presented to Queue	19739	29253	47497	59253	80511	96433
Mean Normalized Goodput	0.189	0.314	0.502	0.500	0.504	0.501

Table 8. Number of Packets Presented to the Queue verses the Transmission Attempts compared to the Normalized Telemetry Goodput ($M = 3$) [Figure 15]

	Offered Load (G)					
	0.300	0.510	0.760	1.010	1.210	1.750
Transmission Attempts	17522	29592	39448	41336	41159	41281
Packets Presented to Queue	17519	29623	44262	59378	71101	102395
Mean Normalized Goodput	0.191	0.326	0.440	0.462	0.460	0.461

Table 9. Number of Packets Presented to the Queue verses the Transmission Attempts compared to the Normalized Telemetry Goodput ($M = 4$) [Figure 16]

	Offered Load (G)					
	0.350	0.520	0.710	1.060	1.170	1.630
Transmission Attempts	20528	30190	38406	37486	39602	38283
Packets Presented to Queue	20539	30199	41424	61883	68550	95564
Mean Normalized Goodput	0.213	0.330	0.426	0.415	0.440	0.420

Table 10. Telemetry Mean Delay ($M = 2$) [Figure 17]

	Offered Load (G)					
	0.340	0.500	0.810	1.010	1.370	1.650
Upper Confidence Interval	0.013	0.014	0.341	0.310	0.265	0.223
Mean Delay	0.009	0.011	0.333	0.291	0.229	0.199
Lower Confidence Interval	0.004	0.008	0.326	0.271	0.193	0.175

Table 11. Telemetry Mean Delay ($M = 3$) [Figure 18]

	Offered Load (G)					
	0.300	0.510	0.760	1.010	1.210	1.750
Upper Confidence Interval	0.029	0.195	0.963	0.462	0.384	0.414
Mean Delay	0.018	0.154	0.675	0.419	0.344	0.320
Lower Confidence Interval	0.007	0.113	0.386	0.376	0.304	0.226

Table 12. Telemetry Mean Delay ($M = 4$) [Figure 19]

	Offered Load (G)					
	0.350	0.520	0.710	1.060	1.170	1.630
Upper Confidence Interval	0.187	0.432	0.789	0.691	0.513	0.517
Mean Delay	0.098	0.245	0.728	0.580	0.473	0.458
Lower Confidence Interval	0.008	0.058	0.667	0.470	0.433	0.400

Table 13. Avionics Goodput ($M = 2$) [Figure 20]

Normalized Goodput (S)	Offered Load (G)					
	0.199	0.356	0.532	0.641	0.806	1.083
Upper Confidence Interval	0.192	0.335	0.468	0.537	0.703	0.862
Mean Normalized Goodput	0.191	0.334	0.461	0.520	0.688	0.846
Lower Confidence Interval	0.190	0.333	0.453	0.504	0.672	0.831

Table 14. Avionics Goodput ($M = 3$) [Figure 21]

Normalized Goodput (S)	Offered Load (G)					
	0.199	0.347	0.534	0.696	0.797	1.222
Upper Confidence Interval	0.178	0.289	0.370	0.528	0.577	0.593
Mean Normalized Goodput	0.174	0.255	0.363	0.494	0.552	0.558
Lower Confidence Interval	0.170	0.221	0.356	0.461	0.528	0.523

Table 15. Avionics Goodput ($M = 4$) [Figure 22]

Normalized Goodput (S)	Offered Load (G)					
	0.200	0.367	0.542	0.663	0.811	1.108
Upper Confidence Interval	0.166	0.249	0.330	0.461	0.498	0.510
Mean Normalized Goodput	0.164	0.238	0.302	0.442	0.470	0.494
Lower Confidence Interval	0.162	0.227	0.274	0.424	0.442	0.478

Table 16. Number of Packets Presented to the Queue verses the Transmission Attempts compared to the Normalized Avionics Goodput ($M = 2$) [Figure 23]

	Offered Load (G)					
	0.200	0.360	0.530	0.640	0.810	1.080
Transmission Attempts	1825	3254	4862	5861	7339	8386
Packets Presented to Queue	1825	3254	4863	5865	7373	9907
Mean Normalized Goodput	0.191	0.334	0.461	0.520	0.688	0.846

Table 17. Number of Packets Presented to the Queue verses the Transmission Attempts compared to the Normalized Avionics Goodput ($M = 3$) [Figure 24]

	Offered Load (G)					
	0.200	0.350	0.530	0.700	0.800	1.220
Transmission Attempts	1821	3169	4824	5798	6161	5965
Packets Presented to Queue	1821	3172	4889	6368	7287	11175
Mean Normalized Goodput	0.174	0.255	0.363	0.494	0.552	0.558

Table 18. Number of Packets Presented to the Queue verses the Transmission Attempts compared to the Normalized Avionics Goodput ($M = 4$) [Figure 25]

	Offered Load (G)					
	0.200	0.370	0.540	0.660	0.810	1.110
Transmission Attempts	1830	3357	4944	5442	5558	5631
Packets Presented to Queue	1830	3357	4960	6068	7421	10130
Mean Normalized Goodput	0.164	0.238	0.302	0.442	0.470	0.494

Table 19. Avionics Mean Delay ($M = 2$) [Figure 26]

	Offered Load (G)					
	0.200	0.360	0.530	0.640	0.810	1.080
Upper Confidence Interval	0.060	0.039	0.048	0.107	0.478	1.569
Mean Delay	0.060	0.028	0.037	0.084	0.356	1.468
Lower Confidence Interval	0.059	0.016	0.025	0.062	0.233	1.368

Table 20. Avionics Mean Delay ($M = 3$) [Figure 27]

	Offered Load (G)					
	0.200	0.350	0.530	0.700	0.800	1.220
Upper Confidence Interval	0.083	0.101	0.866	2.614	3.228	2.988
Mean Delay	0.069	0.087	0.645	2.283	2.575	2.721
Lower Confidence Interval	0.055	0.073	0.424	1.952	1.921	2.453

Table 21. Avionics Mean Delay ($M = 4$) [Figure 28]

	Offered Load (G)					
	0.200	0.370	0.540	0.660	0.810	1.110
Upper Confidence Interval	0.123	0.110	0.641	4.262	3.799	3.686
Mean Delay (D)	0.086	0.086	0.511	3.041	3.457	3.292
Lower Confidence Interval	0.049	0.063	0.380	1.821	3.115	2.898

This page intentionally left blank

Appendix B - MatLab® Code

MatLab® code use to produce the figures in Section 4. The particular figures the code produced are listed in the code's description.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% % This routine was written for MATLAB® 6.0 or higher.
% %
% % File Name: Fig_Throughput.m
% %
% % It produces a plot of the Goodput graphs for the experimental data verses an analytical data for a give
% % data size and number of stations. The *.m files was used to create Figures 11-13 and 20-22
% %
% % The two variables that must be adjusted are the DATA and M
% %
% % DATA must be set to 672 for the Telemetry Model or 6208 for the Avionics model.
% %
% % M must be set to equal the number of stations and can ONLY be set to 2, 3, or 4.
% %
% % The Fig_Throughput.m requires two other *.m files in order to work: u_of_X.m and d_of_X.m
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clf;
clear all;

set(gcf,'DefaultLineLineWidth', 1)
set(gca,'FontName', 'Times New Roman')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% % Set variables DATA and M
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
DATA = 6208 % Size of data in packet.
% Must be 672 for the Telemetry Model or 6208 for the Avionics model.
M = 4 % Number of Machines. Must be 2, 3, or 4.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

FRAME = 112+224+16+DATA; % Size of actual Data frame place in channel (in bits)
% 112 = PHY Preamble (bits)
% 224 = IEEE Frame Headers (bits)
% 16 = PHY Postamble (bits)

a = 20/FRAME; % Normalized Slot Time (slot time = 20 µs)

ACK = (112+16+7*16)/FRAME; % Normalized time of an ACK frame
% 112 = PHY Preamble (bits)
% 16 = PHY Postamble (bits)
% 7*16 = size of ACK frame (bits)

DIFS = 50/FRAME; % Normalized time of Distributed Inter-frame Space (DIFS)
% DIFS time = 50 µs

SIFS = 10/FRAME; % Normalized time of Short Inter-frame Space (SIFS)
% SIFS time = 10 µs

DELAY = 1/FRAME; % Normalized Frame Delay (which was 1 µs)

ECW = [ NaN 34.05 36.2 (36.2+40.5)/2 40.52] % Average connection window
% taken from [CCG00]

EW = (ECW(M) - 1)/2;
p = 1/(EW + 1) % probability that a station will transmit in a p-persistent CSMA protocol

% % and loads in the correct empirical data for a given DATA and M
if ((DATA==672)&(M==2))
```

```

% Telemetry - 2 Stations
Type = 'Telemetry (packet size = 84 bytes, M = ';
S_e = [ 0.190001353    0.315658319    0.505077465    0.50259405    0.507560738    0.502373136    ;
        0.189          0.31383968    0.50181152    0.49954912    0.50360352    0.50148224    ;
        0.187998647    0.312021041    0.498545575    0.49650419    0.499646302    0.500591344    ];
G_e = [ 0.33687552    0.499247787    0.810615467    1.011254613    1.374057813    1.645796693    ];

elseif ((DATA==672)&(M==3))
% Telemetry - 3 Stations
Type = 'Telemetry (packet size = 84 bytes, M = ';
S_e = [ 0.192069836    0.328226508    0.455921614    0.472068545    0.469067021    0.470762155    ;
        0.19121536    0.3259648    0.4398352    0.46173792    0.45973536    0.46112864    ;
        0.190360884    0.323703092    0.423748786    0.451407295    0.450403699    0.451495125    ];
G_e = [ 0.298990933    0.505565867    0.7554048    1.01338112    1.213463893    1.747541333    ];

elseif ((DATA==672)&(M==4))
% Telemetry - 4 Stations
Type = 'Telemetry (packet size = 84 bytes, M = ';
S_e = [ 0.214222944    0.331077246    0.436588784    0.428647663    0.450343932    0.464707528    ;
        0.21277312    0.33020288    0.42577472    0.41548192    0.4399584    0.41963712    ;
        0.211323296    0.329328514    0.414960656    0.402316177    0.429572868    0.374566712    ];
G_e = [ 0.35053568    0.515392853    0.706976427    1.05613312    1.169923413    1.630952107    ];

elseif ((DATA==6208)&(M==2))
% Avionics - 2 Stations
Type = 'Avionics (packet size = 776 bytes, M = ';
S_e = [ 0.19183209    0.335038058    0.467677047    0.536694913    0.703361237    0.861520036    ;
        0.190978773    0.33423872    0.46057152    0.52041664    0.687639467    0.84633664    ;
        0.190125457    0.333439382    0.453465993    0.504138367    0.671917696    0.831153244    ];
G_e = [ 0.1994896    0.355770667    0.531709867    0.641283733    0.806070933    1.0831872    ];

elseif ((DATA==6208)&(M==3))
% Avionics - 3 Stations
Type = 'Avionics (packet size = 776 bytes, M = ';
S_e = [ 0.177891103    0.288708426    0.370417132    0.527506043    0.576574042    0.592950635    ;
        0.174113707    0.254734933    0.36341632    0.494301653    0.552346453    0.5580992    ;
        0.170336311    0.22076144    0.356415508    0.461097264    0.528118865    0.523247765    ];
G_e = [ 0.199139733    0.346783467    0.534486933    0.6962784    0.796755733    1.2218    ];

elseif ((DATA==6208)&(M==4))
% Avionics - 4 Stations
Type = 'Avionics (packet size = 776 bytes, M = ';
S_e = [ 0.166476014    0.24895075    0.330364908    0.460749505    0.497978691    0.509573414    ;
        0.164160213    0.23820096    0.302101973    0.442175147    0.4699456    0.493929173    ;
        0.161844412    0.22745117    0.273839039    0.423600788    0.441912509    0.478284933    ];
G_e = [ 0.200058133    0.367010133    0.542271467    0.6634784    0.8113408    1.1075248    ];

else
error('DATA or M is incorrect')
end

G = 0:1:(ceil(G_e(length(G_e))*10)/10); % Normalized load

IEEE(1) = 0;

%% For loop that calculates the data points for the analytical model
for i=2:length(G)

    g = a*(G(i)/M); % Probability that a host generates a frame during a time slot

    % Equation (2.6)
    TPs = 1 + SIFS + ACK + 2*DELAY + DIFS; % The normalized time of a successful transmission
    TPf = 1 + DELAY + DIFS; % The normalized time of a failed transmission

    % Equation (2.4)
    I = a/(1-(1-g)^M); % The expected value of the idle time when a host has nothing to transmit

    % Equation (2.7)
    d1 = DIFS*(1 - (1 - g)^M);

```

```

% Equation (2.8)
u1 = (1/(1-(1-g)^M))*M*g*(1-g)^(M-1);

% Equation (2.7)
ds = d_of_X(TPs,a,p,g,M); % d(X) with X = TPs
df = d_of_X(TPf,a,p,g,M); % d(X) with X = TPf

gs = 1-(1-g)^(TPs/a); % temp variable
gf = 1-(1-g)^(TPf/a); % temp variable

% Equation (2.8)
us = u_of_X(TPs,a,p,g,M); % u(X) with X = TPs
uf = u_of_X(TPf,a,p,g,M); % u(X) with X = TPf

% Equation (2.9)
A1 = [(gs*us - 1) (gf*(1-us)) ; gs*us (gf*(1-uf)-1)];
b1 = [-(ds + TPs*us + TPf*(1-us)) ; -(df + TPs + TPf*(1-uf))];
B_TP = inv(A1)*b1;
B_TPs = B_TP(1); % B(TPs)
B_TPf = B_TP(2); % B(TPf)

% Equation (2.5)
B_1 = d1 + (TPs + gs*B_TPs)*u1 + (TPf + gf*B_TPf)*(1-u1);

% Equation (2.11)
A2 = [(gs*us-1) gf*(1-us) ; gs*uf (gf*(1-uf)-1)];
b2 = [-us ; -uf];
U_TP = inv(A2)*b2;
U_TPs = U_TP(1);
U_TPf = U_TP(2);

% Equation (2.10)
U_1 = (1 + gs*U_TPs)*u1 + (gf*U_TPf)*(1-u1);

% Equation 2.12
IEEE(i) = U_1/(B_1 + I);

end

hold on
% Plot Empirical and Analytical results
plot(G_e,S_e(2,:), 'ks','MarkerSize',10)
plot(G, IEEE,'k','LineWidth',2);

set(gca,'FontName', 'Times New Roman','FontSize',14)
legend('Experimental','Analytical',4);

% Plot Confidence Interval lines
plot([G_e(1) G_e(1)], [S_e(1,1) S_e(3,1)], 'k-+', ...
[G_e(2) G_e(2)], [S_e(1,2) S_e(3,2)], 'k-+', ...
[G_e(3) G_e(3)], [S_e(1,3) S_e(3,3)], 'k-+', ...
[G_e(4) G_e(4)], [S_e(1,4) S_e(3,4)], 'k-+', ...
[G_e(5) G_e(5)], [S_e(1,5) S_e(3,5)], 'k-+', ...
[G_e(6) G_e(6)], [S_e(1,6) S_e(3,6)], 'k-+', 'LineWidth',2);
hold off

if(DATA==672)
axis([0 2 0 .75]);
else
axis([0 1.4 0 1]);
end

% Title - Normally commented out
% title([Type, num2str(M), '']);

% Label Accesses
xlabel('Normalized Offered Load (G)', 'FontName', 'Times New Roman', ...

```

```

'FontSize',18);
ylabel('Normalized Goodput (S)','FontName', 'Times New Roman',...
'FontSize',18);

beep on;
beep;
beep off;

```

Fig_Throughput.m requires two other *.m files. The first is called “u_of_X.m”

```

function uX = u_of_X(X,a,p,g,M)
% Calculates  $u(X)$  with  $X \neq 1$  from Equation (2.15)
% Filename: u_of_X.m

uX = 0;
for n=1:M
    uX = uX + (n*p*(1-p)^(n-1)...
        + (n*p*(1-p)^(n-1) + (M-n)*g*(1-g)^(M-n-1) - n*(M-n)...
        * p*g*((1-p)^(n-1))*((1-g)^(M-n-1)))...
        * ((1-p)^n) * ((1-g)^(M-n)) ) / ( 1 - ((1-p)^n) * ((1-g)^(M-n)) ) )...
        * ( nchoosek(M,n) * ( (1-(1-g)^(X/a))^n ) * (1-g)^((X/a)*(M-n)) )...
        / ( 1 - (1-g)^(X*M/a) ) );
end

```

The second *.m file is called “d_of_X.m” and is as follows:

```

function dX = dofX(X,a,p,g,M)
% Calculates  $d(X)$  with  $X \neq 1$  from Equation (2.14)
% Filename: d_of_X.m

dXsum1 = 0;
for k=1:10^4 % Equation (2.14) has k go from 1 to  $\infty$ , but reasonably  $10^4$  is enough
    dXsum1 = dXsum1 + ((1-p)^k - ((1-g)^(X/a))*((1-p)^k-(1-g)^k))^M;
end

dXsum2 = 0;
for k=1:10^4 % Equation (2.14) has k go from 1 to  $\infty$ , but reasonably  $10^4$  is enough
    dXsum2 = dXsum2 + (1-g)^(k*M);
end

dX = (a/(1-(1-g)^(X*M/a))) * (dXsum1 - ((1-g)^(X*M/a))*dXsum2);

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% This routine was written for MATLAB® 6.0 or higher.
%%
%% File Name: Fig_Packet_Q.m
%%
%% It produces a plot of the Number of Packets Presented to the Queue verses the Transmission Attempts
%% compared to the Normalized Throughput graphs for the experimental data for a give
%% data size and number of stations. The *.m file was used to create Figures 14-16 and 23-25
%%
%% The two variables that must be adjusted are the DATA and M
%%
%% DATA must be set to 672 for the Telemetry Model or 6208 for the Avionics model.
%%
%% M must be set to equal the number of stations and can ONLY be set to 2, 3, or 4.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clf;
clear all;

set(gcf,'DefaultLineLineWidth', 1)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Set variables DATA and M
%% DATA = 6208 % Size of data in packet.
%% Must be 672 for the Telemetry Model or 6208 for the Avionics model.
M = 4 % Number of Machines. Must be 2, 3, or 4.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% Determines if DATA and M variables were inputted correctly

%% and loads in the correct empirical data for a given DATA and M

if ((DATA==672)&(M==2))
    % Telemetry - 2 Stations
    R = [ 0 19736 29252 44847 44627 44993 44800 ;
          0 19739 29253 47497 59253 80511 96433 ];
    S = [ 0.00000 0.18900 0.31384 0.50181 0.49955 0.50360 0.50148 ];
    G_e = [ 0 0.34 0.50 0.81 1.01 1.37 1.65 ];

elseif ((DATA==672)&(M==3))
    % Telemetry - 3 Stations
    R = [ 0 17522 29592 39448 41336 41159 41281 ;
          0 17519 29623 44262 59378 71101 102395 ];
    S = [ 0.00000 0.19122 0.32596 0.43984 0.46174 0.45974 0.46113 ];
    G_e = [ 0 0.30 0.51 0.76 1.01 1.21 1.75 ];

elseif ((DATA==672)&(M==4))
    % Telemetry - 4 Stations
    R = [ 0 20528 30190 38406 37486 39602 38283 ;
          0 20539 30199 41424 61883 68550 95564 ];
    S = [ 0.00000 0.21277 0.33020 0.42577 0.41548 0.43996 0.41964 ];
    G_e = [ 0 0.35 0.52 0.71 1.06 1.17 1.63 ];

elseif ((DATA==6208)&(M==2))
    % Avionics - 2 Stations
    R = [ 0 1825 3254 4862 5861 7339 8386 ;
          0 1825 3254 4863 5865 7373 9907 ];
    S = [ 0.00000 0.19098 0.33424 0.46057 0.52042 0.68764 0.84634 ];
    G_e = [ 0 0.20 0.36 0.53 0.64 0.81 1.08 ];

elseif ((DATA==6208)&(M==3))
    % Avionics - 3 Stations
    R = [ 0 1821 3169 4824 5798 6161 5965 ;
          0 1821 3172 4889 6368 7287 11175 ];
    S = [ 0.00000 0.17411 0.25473 0.36342 0.49430 0.55235 0.55810 ];
    G_e = [ 0 0.20 0.35 0.53 0.70 0.80 1.22 ];

```

```

elseif ((DATA==6208)&(M==4))
    % Avionics - 4 Stations
    R = [    0      1830   3357   4944   5442   5558   5631   ;
          0      1830   3357   4960   6068   7421   10130  ];
    S = [    0.00000  0.16416  0.23820  0.30210  0.44218  0.46995  0.49393 ];
    G_e = [    0      0.20   0.37   0.54   0.66   0.81   1.11   ];

else
    error('DATA or M is incorrect')
end

bar(R');

set(gca,'FontName','Times New Roman','FontSize',14)

legend('Transmission Attempts','Packets Presented to Queue',2);

ylabel('Average Number of Packets','FontName','Times New Roman',...
'FontSize',18);
xlabel('Normalized Offered Load (G)','FontName','Times New Roman',...
'FontSize',18);

a = axis;
axis([1 7.5 a(3) a(4)]);
set(gca,'XTickLabel',G_e)

temp = a(4)

h1 = gca;
h2 = axes('Position',get(h1,'Position'));

if(DATA==672)
    plot(S,'k-o','LineWidth',2);

    a = axis;
    axis([1 7.5 a(3) (temp/10^5)]);
else
    plot(S,'k-o','LineWidth',2);

    a = axis;
    axis([1 7.5 a(3) (temp/10^4)]);
end
set(gca,'FontName','Times New Roman','FontSize',14)

legend('Empirical Data');

ylabel('Normalized Throughput (S)','FontName','Times New Roman',...
'FontSize',12);

set(h2,'YAxisLocation','right','Color','none','XTickLabel',[])
set(h2,'XLim',get(h1,'XLim'),'Layer','top')

ylabel('Normalized Goodput (S)','FontName','Times New Roman',...
'FontSize',18);

colormap('colorcube')

beep on;
beep;
beep off;

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% This routine was written for MATLAB® 6.0 or higher.
%%
%% File Name: Fig_Mean_Delay.m
%%
%% It produces a plot of the Mean Delay of the experimental data for a give
%% data size and number of stations. The *.m files was used to create Figures 17-19 and 26-28
%%
%% The two variables that must be adjusted are the DATA and M
%%
%% DATA must be set to 672 for the Telemetry Model or 6208 for the Avionics model.
%%
%% M must be set to equal the number of stations and can ONLY be set to 2, 3, or 4.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clf;
clear all;

set(gcf,'DefaultLineLineWidth', 1)
set(gca,'FontName','Times New Roman')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Set variables DATA and M
%% DATA = 6208 % Size of data in packet.
%% Must be 672 for the Telemetry Model or 6208 for the Avionics model.
M = 4 % Number of Machines. Must be 2, 3, or 4.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% Determines if DATA and M variables were inputted correctly

%% and loads in the correct empirical data for a given DATA and M

if ((DATA==672)&(M==2))
    % Telemetry - 2 Stations
    M_D = [0.0134 0.0137 0.3411 0.31 0.2649 0.2233 ;
           0.0089 0.0106 0.3333 0.2905 0.2289 0.1992 ;
           0.0044 0.0076 0.3256 0.2711 0.1929 0.1751 ];
    G_e = [ 0.34 0.5 0.81 1.01 1.37 1.65 ];

elseif ((DATA==672)&(M==3))
    % Telemetry - 3 Stations
    M_D = [0.0287 0.1953 0.9634 0.4616 0.3835 0.4135 ;
           0.0176 0.1539 0.6747 0.4186 0.3439 0.3196 ;
           0.0066 0.1125 0.3861 0.3755 0.3043 0.2257 ];
    G_e = [ 0.3 0.51 0.76 1.01 1.21 1.75 ];

elseif ((DATA==672)&(M==4))
    % Telemetry - 4 Stations
    M_D = [0.1872 0.4323 0.7893 0.6905 0.5126 0.5173 ;
           0.0976 0.2451 0.7283 0.5801 0.4727 0.4584 ;
           0.0081 0.0579 0.6672 0.4698 0.4327 0.3995 ];
    G_e = [ 0.35 0.52 0.71 1.06 1.17 1.63 ];

elseif ((DATA==6208)&(M==2))
    % Avionics - 2 Stations
    M_D = [0.0604 0.039 0.048 0.1066 0.4781 1.5687 ;
           0.0595 0.0277 0.0365 0.0841 0.3557 1.4682 ;
           0.0586 0.0163 0.025 0.0616 0.2332 1.3677 ];
    G_e = [ 0.2 0.36 0.53 0.64 0.81 1.08 ];

elseif ((DATA==6208)&(M==3))
    % Avionics - 3 Stations
    M_D = [0.0832 0.1007 0.8662 2.6144 3.228 2.9877 ;
           0.0689 0.087 0.6448 2.2832 2.5745 2.7205 ;
           0.0547 0.0734 0.4235 1.9521 1.9211 2.4534 ];
    G_e = [ 0.2 0.35 0.53 0.7 0.8 1.22 ];

```

```

elseif ((DATA==6208)&(M==4))
    % Avionics - 4 Stations
    M_D = [ 0.1226  0.1097  0.6413  4.2615  3.7992  3.6855 ;
            0.0857  0.0863  0.5107  3.0411  3.457   3.2916 ;
            0.0488  0.063   0.3801  1.8208  3.1148  2.8978 ];
    G_e = [ 0.2    0.37   0.54   0.66   0.81   1.11   ];

else
    error('DATA or M is incorrect')
end

% Plot Experimental results

hold on
% Plot Experimental and Analytical results
plot(G_e,M_D(2,:), 'ks', 'MarkerSize', 8)

set(gca, 'FontName', 'Times New Roman', 'FontSize', 14)
legend('Experimental', 2);

% Plot Confidence Interval lines
plot([G_e(1) G_e(1)], [M_D(1,1) M_D(3,1)], 'k-+', ...
     [G_e(2) G_e(2)], [M_D(1,2) M_D(3,2)], 'k-+', ...
     [G_e(3) G_e(3)], [M_D(1,3) M_D(3,3)], 'k-+', ...
     [G_e(4) G_e(4)], [M_D(1,4) M_D(3,4)], 'k-+', ...
     [G_e(5) G_e(5)], [M_D(1,5) M_D(3,5)], 'k-+', ...
     [G_e(6) G_e(6)], [M_D(1,6) M_D(3,6)], 'k-+', 'LineWidth', 2);

hold off

a = axis;
if(DATA==672)
    axis([.2 1.8 (10^-3) 1]);
else
    axis([.1 1.3 (10^-2) 10]);
end

set(gca, 'YScale', 'log')

% Title - Normally commented out
% title(['Type, num2str(M),']);

% Label Accesses
xlabel('Normalized Offered Load (G)', 'FontName', 'Times New Roman', ...
      'FontSize', 18);
ylabel('Delay in sec (D) log scale', 'FontName', 'Times New Roman', ...
      'FontSize', 18);

```

Appendix C - XInC Assembly Code

C.1. Introduction

The following is the assembly code used for this research. The code is written and compiled in the program XInC Development Environment version 3.2.3.0. Instructions on how the basic framework of a XInC assembly program operates and what each command does can be found in XInC Development Kit's documentation from [EE04].

C.2. WiFi.main

This is the main file of the project. This file sets up the memory map of the XInC program and houses all other code modules.

```
//*****
//***** (C) 2002 by Eleven Engineering Incorporated *****
//*****
/**
/**      Tabs:  This file looks best with tab stops set every 6 spaces.
/**
//*****
//*****
/**
/** File:      WiFi.main
/** Project: IEEE 802.11 MAC emulator.  It can send to multiple (1-4) stations
/** Created: 1 June 2004 by Capt Joshua D. Green
/**
/** Description: The main file.
/**
/** Disclaimer: This code was descended from Eleven Engineering sample
/**              source code, but changes were made by Capt Joshua D. Green
/**
//*****
//*****

//=====
// Conditional Assembly Switches:
//      Uncomment the defines below for the threads you want to run.
//=====

// !!!!!Define one of two programs to run
#define THROUGHPUT

// !!! Define Station
#define STATION_1
// #define STATION_2
// #define STATION_3
// #define STATION_4

// Debugging Stuff
#define DEBUG_LEDS

#ifndef THROUGHPUT
#define      _T0_
#define      _T1_
#define      _T2_
#define      _T3_
#define      _T4_
#define      _T5_
#define      _T6_
```

```

        #define          __T7__

// !!! Define the number of stations to be used
// #define THROUGHPUT_2_STATIONS // Must use either Station #2 or #3
// #define THROUGHPUT_3_STATIONS // Must use either Station #1, #2, or #3
// #define THROUGHPUT_4_STATIONS // Must use either Station #1, #2, #3, or #4

// !!! Define if want to run mutiple tests
#define MULT_TESTS

// !!! Define Packet Data Size
// #define TELEMETRY
#define AVIONICS

#endif

// Define if NOT using a CRC function
#define NO_CALC_CRC

// #define PrintErrors
// #define PrintBBUTime

#define RFWaves1Mbps
// #define RFWaves2Mbps
// #define RFWaves3Mbps

// =====
// Code and Data Size:
// After assembly, check the values assigned to these constants in the list file.
// =====

SizeOfAppCode      = ( __AppCode_End__ - __AppCode_Start__ )
SizeOfAppData      = ( __AppData_End__ - __AppData_Start__ )
SizeOfShortData    = ( __ShortData_End__ - __ShortData_Start__ )

FreeAppCodeSpace    = ( __AppData_Start__ - __AppCode_End__ )           // If any of these three
FreeAppDataSpace    = ( kRAM_End - 127 - __AppData_End__ )           // constants are negative,
FreeShortDataSpace  = ( kRAM_End - __ShortData_End__ )                // there is an overflow.

// =====
// Header Files:
// This section includes files defining constants.
// =====

#include "XInC.h"
#include "Constants.h"

// =====
// Code Space:
// Only Code should be included in this segment.
// =====

@ = kRAM_Block0_Start
__AppCode_Start__:

// -----
// Initialization Code

#include "Init.asm"
#include "FiftyMegaHertz.asm"

// -----
// Thread Code

#ifdef __T0__
Thread0:                                // Thread 0 Code
    #include "Thread0.asm"
    bra Thread0
#endif

#ifdef __T1__
Thread1:                                // Thread 1 Code
    #include "Thread1.asm"
    bra Thread1
#endif

#ifdef __T2__
Thread2:                                // Thread 2 Code
    #include "Thread2.asm"

```

```

        bra Thread2
#endif

#ifdef __T3__
Thread3:                                     // Thread 3 Code
        #include "Thread3.asm"
        bra Thread3
#endif

#ifdef __T4__
Thread4:                                     // Thread 4 Code
        #include "Thread4.asm"
        bra Thread4
#endif

#ifdef __T5__
Thread5:                                     // Thread 5 Code
        #include "Thread5.asm"
        bra Thread5
#endif

#ifdef __T6__
Thread6:                                     // Thread 6 Code
        #include "Thread6.asm"
        bra Thread6
#endif

#ifdef __T7__
Thread7:                                     // Thread 7 Code
        #include "Thread7.asm"
        bra Thread7
#endif

//-----
// Other Source Files

//-----
// !!! Speed Selection. Either 1Mbps or 3Mbps must be defined

#include "RFWaves.asm"
#include "XPD_Echo.asm"
#include "Delay.asm"
#include "Frame_Format.asm"
#include "LEDs.asm"

__AppCode_End__:

//=====
// Data Space:
//      All Data must be in a separate 2kWord Memory Block from any Code.
//=====

@ = (@ + 0x800-1) & -0x800                  // Round up to the next 2kWord Memory Block
__AppData_Start__:

#include "Long_Data.asm"
#include "XPD_Echo_Data.asm"

__AppData_End__:

//=====
// Short Address Space:
//      Any Data placed in this space may be accessed with a single word instruction.
//=====

@ = kRAM_End - 127                          // Start of the short address space
__ShortData_Start__:

#include "RFWaves_data.asm"
#include "Short_Data.asm"

__ShortData_End__:

```

C.3. Constants.h

This file contains the user defined constants used by the program. All constants in this file are accessible to all other modules in the program. Constants are only used by the assembler program only and do not take up any memory space on the boards.

```
//*****
//***** (C) 2002 by Eleven Engineering Incorporated *****
//*****
/**
/**      Tabs: This file looks best with tab stops set every 6 spaces.
/**
//*****
//** File:          Constants.h
//**
//** Project: Two-Way Text Messaging, can send to multiple (1-4) stations
//** Created: 1 June 2004 by Capt Joshua D. Green
//**
//** Description: Contains the constants used by IEEE 802.11 hardware test set program.
//**
//** Disclaimer: This code was descended from Eleven Engineering sample
//**              source code, but changes were made by Capt Joshua D. Green
//**
//*****
//*****

#define          kStackSize          64

// Testing Parameters

#define          kNumber_of_tests    1          // Defines the number of data dumps to
screen the program will execute
#define          kTime_of_Testing_Period    60          // length of testing period IN SECONDS
#define          kDelay_Between_Tx      32          // Delay in **us** between sending packets
#define          kDelay_Between_TX_MASK 4096          // Sets the RN maximum value.
// Will mask out part of RN and
// add 1 to it
// Must be one of the following values:
// 4
// 8
// 16
// 32
// 64
// 128
// 256
// 512
// 1024
// 2048
// 4096
// 8192
// 16384
// 32768

// SEMAPHORES - used to share a resource
#define          kSPiOCS_Semaphore    0
#define          kFailed_TX_SEMAPHORE 1
#define          kData_Dump_SEMAPHORE 2
#define          kCreate_RN_BV_SEMAPHORE 3
#define          kReceived_some_text_SEMAPHORE 4
#define          kPacket_Start_Time_SEMAPHORE 5
#define          kReceived_TX_SEMAPHORE 6
#define          kReceived_TX_DONE_SEMAPHORE 7
#define          kRN_SEMAPHORE        8
#define          kPackets_in_Que_SEMAPHORE 9
#define          kGO_SEMAPHORE        10
#define          kStart_Stop_SEMAPHORE 11
#define          kTime_SEMAPHORE      12
#define          kTx_Data_Address_1_SEMAPHORE 13
#define          kACK_SEMAPHORE       14
#define          kDevLEDS_Semaphore   0x8000
```

```

// Maximum number of transmissions before IEEE 802.11 protocol gives up
#define      kMaxReTransmit      4

#define      kTransmitter_Buffer_Size      256
//!!!Note:
// kTransmitter_Buffer_Size must be set to one of the following values:
// 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, or 2048.
// If it is NOT a power of 2, the buffer size mask used to filter the
// buffer counter will be really screwed up.
// If the value is larger then 2048, then the buffer itself will be
// to big for the memory on the boards, and then bad will happen.

// IEEE 802.11 MAC parameters
// System Clock is set at 50 MHz, so one clock cycle is = 0.02 usec
// BBU Clock is set at 1 MHz Baud rate (1 Mbps throughput), so one clock cycle = 1 usec
#define      kCWmin      31      // Minimum size of contention window, in units of kSlotTime
#define      kCWmax      1023    // Maximum size of contention window, in units of kSlotTime
#define      kSIFSTime    500     // Short Interface Space Time = 10 usec = 500 SCUtime cycles
#define      kSlotTime    1000    // Slot Time = 20 usec = 1000 SCUtime cycles
#define      kDIFSTime    2500    // DCF Interframe Space Time = 50 usec or 2500 SCUtime cycles
#define      kACK_Timeout 10600    // ACK Timeout = 212 usec = 10600 SCUtime cycles

// Sets adjustment for kDIFSTime in loop in Thread0.
// The time varies depending on if DEBUG_LEDS is defined or not.
#ifdef DEBUG_LEDS
#define      kDIFSTime_Adjustment      450
#else
#define      kDIFSTime_Adjustment      250
#endif

// Sets adjustment for kSlotTime in loop in Thread0.
#define      kSIFSTime_Adjustment      486

// Use to start and stop Thread 2
#define      kStart_Thread_2      0b00000000
#define      kStop_Thread_2      0b00000100

// Define Station Numbers (ASCII Characters)
#define      Station_01      49
#define      Station_02      50
#define      Station_03      51
#define      Station_04      52

```

C.4. Delay.asm

XInC library file included with the development kit. The library file Delay.asm defines routines to delay a certain number of clock ticks.

```

//*****
//***** (C) 2002 by Eleven Engineering Incorporated *****
//*****
/**
**      Tabs: This file looks best with tab stops set every 6 spaces.
**
//*****
//**      $RCSfile: Delay.asm,v $
//**      $Revision: 1.3 $
//**      Tag $Name: $
//**      $Date: 2003/02/12 21:17:11 $
//**      $Author: eleven $
//**
//**      Project: XInC Library
//**      Description: Routines to delay a certain number of clock ticks.
//**
//**      Disclaimer: You may incorporate this sample source code into your
//**                  program(s) without restriction. This sample source code has
//**                  been provided "AS IS" and the responsibility for its

```

```

/**          operation is yours.  You are not permitted to redistribute
/**          this sample source code as "Eleven sample source code" after
/**          having made changes.  If you're going to re-distribute the
/**          source, we require that you make it clear in the source that
/**          the code was descended from Eleven sample source code, but
/**          that you've made changes.
/**
/**          *****
/**          *****
/**
/**      Delay  (r1 = delay length)
/**      DelayLong
/**      DelayReallyLong
/**
/**          *****
/**          *****

#ifndef __DELAY_UTILS__
#define __DELAY_UTILS__

//=====
// Input Params:      r1 = Delay Length
// Output Params:     None
//-----
// Description:       Delays for a given amount of time.  This function will return
//                    after (5 + 2*r1) instruction times have elapsed.
//=====
Delay:
                st      r1, sp, 0

                add     r1, r1, 0
                bc      ZS, Delay_END

Delay_loop:
                sub     r1, r1, 1
                bc      ZC, Delay_loop

Delay_END:
                ld      r1, sp, 0

                jsr     r6, r6

//=====
// Input Params:      None
// Output Params:     None
//-----
// Description:       Delays for a long time (returns after 131074 instruction
//                    times have elapsed which is roughly 1/12 of a second when
//                    using a 12MHz clock).
//=====
DelayLong:
                st      r0, sp, 0

                mov     r0, 0xFFFF

DelayLong_Loop:
                sub     r0, r0, 1
                bc      ZC, DelayLong_Loop

DelayLong_END:
                ld      r0, sp, 0

                jsr     r6, r6

//=====
// Input Params:      None
// Output Params:     None
//-----
// Description:       Delays for a really long time (returns after 524299
//                    instruction times have elapsed which is roughly 1/3 of a
//                    second when using a 12MHz clock).
//=====
DelayReallyLong:
                st      r6, sp, 0

                jsr     r6, DelayLong
                jsr     r6, DelayLong
                jsr     r6, DelayLong
                jsr     r6, DelayLong

DelayReallyLong_END:
                ld      r6, sp, 0

```

```

jsr    r6, r6

#endif

```

C.5. FiftyMegaHertz.asm

XInC library file included with the development kit. The library file FiftyMegaHertz.asm is a XInC initialization file that sets the board's system clock at 50 MHz.

```

//*****
//***** (C) 2002 by Eleven Engineering Incorporated *****
//*****
//**
//**      Tabs:  This file looks best with tab stops set every 6 spaces.
//**
//*****
//*****
//**
//** File:      FiftyMegaHertz.asm
//** Project: WUSB
//** Created: 8 Apr 2003 by Jason Hennig
//** Revised:
//**
//** Description:  XInC Initialization File
//**
//*****
//*****

FiftyMegaHertz:
    inp    r0, SCXclkCfg      // input clock config
    bic    r0, r0, 10        // select RC clock
    outp   r0, SCXclkCfg

FiftyMegaHertz_RC:
    inp    r0, SCXclkCfg
    bc     NS, FiftyMegaHertz_RC    // wait for switch

    bic    r0, r0, 9          // enable feedback resistor
    bis    r0, r0, 8          // select high frequency mode
    bis    r0, r0, 7          // select overtone mode
    bic    r0, r0, 6          // disable tri-state
    bis    r0, r0, 5          // enable

    outp   r0, SCXclkCfg

    rol    r1, r1, 0          // nop for stability
    rol    r6, r6, 0          // nop

    inp    r0, SCXclkCfg
    bis    r0, r0, 11         // select oscillator 2
    outp   r0, SCXclkCfg

    rol    r1, r1, 0          // nop for stability
    rol    r6, r6, 0          // nop

    inp    r0, SCXclkCfg
    bis    r0, r0, 10         // select oscillator
    outp   r0, SCXclkCfg

FiftyMegaHertz_X2:
    inp    r0, SCXclkCfg
    bc     NC, FiftyMegaHertz_X2    // wait for switch

    mov    r1, 0b0000001100100001 // enable output buffers
    outp   r1, SCXclkBuf

```

C.6. *Frame_Format.asm*

This library file has two routines that format IEEE 802.11 frames (data and ACK frames) and stores them to memory.

```
//*****
//***** (C) 2002 by Eleven Engineering Incorporated *****
//*****
/**
/**      Tabs:  This file looks best with tab stops set every 6 spaces.
/**
/**
//*****
//*****
/**
/** File:      Frame_Format.asm
/**
/**
/** Project: IEEE 802.11 MAC emulator.  It can send to multiple (1-4) stations
/** Created: 1 June 2004 by Capt Joshua D. Green
/**
/** Description: Formats IEEE 802.11 frames (data and ACK frames) and stores them to memory.
/**
/** Disclaimer: This code was descended from Eleven Engineering sample
/**              source code, but changes were made by Capt Joshua D. Green
/**
//*****
//*****

#ifndef __FRAME_FORMAT__
#define __FRAME_FORMAT__

// Call these functions to load into memory frames that are to be transmitted.
//
// Initialize_Data_Frame
// Initialize_ACK_Frame
//

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

//-----
// Frame Control field for an IEEE 801.11 MAC Frame.
// 2 Octets (2 bytes or 16 bits) long.
// See IEEE 802.11 Standard for details on what each bit means
// Note:  For the purpose of this code, the Retry field bit is always set LOW (0).

#define      kData_Frame_Control      32      // 0000 0000 0010 0000
#define      kACK_Frame_Control       29      // 0000 0000 0001 1101

//-----
// Duration/ID field - Set with a fixed data length of 84 bytes of data, which is 42 16-bit words

#define      kDuration_ID_field       42      // 0000 0000 0010 1010

//-----
// MAC Addresses for each station defined
// Only using last 6 bits of MAC address to identify station
// This is done to compensate for the 6/16 encoding
#define      kSA_Address_1st_16_bits  0x000C
#define      kSA_Address_2nd_16_bits  0x003A
#define      kSA_Address_STATION_01   49
#define      kSA_Address_STATION_02   50
#define      kSA_Address_STATION_03   51
#define      kSA_Address_STATION_04   52

//-----
// BSSID -- Only using last 6 bits of MAC address to identify station
// This is done to compensate for the 6/16 encoding

#define      kBSSID                    0x002C
```

```

//-----
// Randomly created data for Data Field in 802.11 Frame created with MS Excel XP's RAND feature
// Command in Excel used was =RAND()*(1-0)+0. Data represents a frame of MIL-STD-1553B data
// The frame data is 83 bytes. It is padded with one byte of zeros. This is so the actual
// data frame will neatly fit into the XInC board's transmission register, BBUTx. The
// register BBUTx is a 16-bit register and will accept only an even number of bytes.
// Thus the one byte padding was necessary. Thus, the frame data's end length is 84 bytes total.

#ifdef TELEMETRY
    #define kData_01 47
    #define kData_02 6
    #define kData_03 38
    #define kData_04 57
    #define kData_05 62
    #define kData_06 24
    #define kData_07 29
    #define kData_08 43
    #define kData_09 34
    #define kData_10 18
    #define kData_11 8
    #define kData_12 19
    #define kData_13 18
    #define kData_14 4
    #define kData_15 55
    #define kData_16 8
    #define kData_17 6
    #define kData_18 55
    #define kData_19 39
    #define kData_20 34
    #define kData_21 29
    #define kData_22 6
    #define kData_23 37
    #define kData_24 5
    #define kData_25 16
    #define kData_26 27
    #define kData_27 61
    #define kData_28 28
    #define kData_29 4
    #define kData_30 16
    #define kData_31 29
    #define kData_32 30
    #define kData_33 7
    #define kData_34 4
    #define kData_35 16
    #define kData_36 58
    #define kData_37 32
    #define kData_38 21
    #define kData_39 37
    #define kData_40 11
    #define kData_41 40
    #define kData_42 7
#endif

#ifdef AVIONICS
    #define kData_01 4
    #define kData_02 3
    #define kData_03 18
    #define kData_04 38
    #define kData_05 14
    #define kData_06 15
    #define kData_07 8
    #define kData_08 15
    #define kData_09 24
    #define kData_10 14
    #define kData_11 53
    #define kData_12 60
    #define kData_13 23
    #define kData_14 57
    #define kData_15 11
    #define kData_16 37
    #define kData_17 3
    #define kData_18 24
    #define kData_19 43
    #define kData_20 10
    #define kData_21 3
    #define kData_22 44
    #define kData_23 29
    #define kData_24 49
    #define kData_25 53
    #define kData_26 13

```

#define	kData_27	6
#define	kData_28	17
#define	kData_29	25
#define	kData_30	29
#define	kData_31	45
#define	kData_32	49
#define	kData_33	10
#define	kData_34	39
#define	kData_35	8
#define	kData_36	52
#define	kData_37	31
#define	kData_38	12
#define	kData_39	58
#define	kData_40	5
#define	kData_41	15
#define	kData_42	30
#define	kData_43	9
#define	kData_44	60
#define	kData_45	39
#define	kData_46	62
#define	kData_47	53
#define	kData_48	41
#define	kData_49	41
#define	kData_50	55
#define	kData_51	61
#define	kData_52	18
#define	kData_53	49
#define	kData_54	20
#define	kData_55	31
#define	kData_56	3
#define	kData_57	12
#define	kData_58	7
#define	kData_59	9
#define	kData_60	48
#define	kData_61	9
#define	kData_62	21
#define	kData_63	31
#define	kData_64	59
#define	kData_65	57
#define	kData_66	16
#define	kData_67	26
#define	kData_68	59
#define	kData_69	25
#define	kData_70	50
#define	kData_71	61
#define	kData_72	20
#define	kData_73	7
#define	kData_74	47
#define	kData_75	31
#define	kData_76	13
#define	kData_77	29
#define	kData_78	29
#define	kData_79	40
#define	kData_80	55
#define	kData_81	36
#define	kData_82	26
#define	kData_83	9
#define	kData_84	21
#define	kData_85	24
#define	kData_86	50
#define	kData_87	24
#define	kData_88	24
#define	kData_89	5
#define	kData_90	37
#define	kData_91	41
#define	kData_92	27
#define	kData_93	6
#define	kData_94	11
#define	kData_95	3
#define	kData_96	62
#define	kData_97	3
#define	kData_98	49
#define	kData_99	49
#define	kData_100	47
#define	kData_101	26
#define	kData_102	52
#define	kData_103	3
#define	kData_104	24
#define	kData_105	30
#define	kData_106	59
#define	kData_107	9

#define	kData_108	25
#define	kData_109	39
#define	kData_110	54
#define	kData_111	7
#define	kData_112	1
#define	kData_113	56
#define	kData_114	41
#define	kData_115	57
#define	kData_116	42
#define	kData_117	17
#define	kData_118	11
#define	kData_119	38
#define	kData_120	11
#define	kData_121	13
#define	kData_122	4
#define	kData_123	37
#define	kData_124	13
#define	kData_125	44
#define	kData_126	23
#define	kData_127	36
#define	kData_128	3
#define	kData_129	60
#define	kData_130	61
#define	kData_131	6
#define	kData_132	0
#define	kData_133	48
#define	kData_134	60
#define	kData_135	37
#define	kData_136	48
#define	kData_137	36
#define	kData_138	16
#define	kData_139	48
#define	kData_140	35
#define	kData_141	55
#define	kData_142	35
#define	kData_143	30
#define	kData_144	53
#define	kData_145	53
#define	kData_146	23
#define	kData_147	37
#define	kData_148	52
#define	kData_149	57
#define	kData_150	21
#define	kData_151	4
#define	kData_152	36
#define	kData_153	32
#define	kData_154	47
#define	kData_155	39
#define	kData_156	14
#define	kData_157	43
#define	kData_158	1
#define	kData_159	60
#define	kData_160	31
#define	kData_161	9
#define	kData_162	4
#define	kData_163	18
#define	kData_164	36
#define	kData_165	2
#define	kData_166	8
#define	kData_167	13
#define	kData_168	4
#define	kData_169	12
#define	kData_170	44
#define	kData_171	27
#define	kData_172	33
#define	kData_173	55
#define	kData_174	49
#define	kData_175	12
#define	kData_176	13
#define	kData_177	36
#define	kData_178	17
#define	kData_179	35
#define	kData_180	4
#define	kData_181	11
#define	kData_182	15
#define	kData_183	40
#define	kData_184	60
#define	kData_185	35
#define	kData_186	44
#define	kData_187	61
#define	kData_188	24

#define	kData_189	53
#define	kData_190	30
#define	kData_191	24
#define	kData_192	27
#define	kData_193	14
#define	kData_194	35
#define	kData_195	22
#define	kData_196	8
#define	kData_197	3
#define	kData_198	1
#define	kData_199	18
#define	kData_200	24
#define	kData_201	3
#define	kData_202	33
#define	kData_203	19
#define	kData_204	8
#define	kData_205	50
#define	kData_206	29
#define	kData_207	53
#define	kData_208	62
#define	kData_209	4
#define	kData_210	26
#define	kData_211	8
#define	kData_212	11
#define	kData_213	27
#define	kData_214	51
#define	kData_215	27
#define	kData_216	38
#define	kData_217	17
#define	kData_218	57
#define	kData_219	3
#define	kData_220	20
#define	kData_221	10
#define	kData_222	4
#define	kData_223	29
#define	kData_224	10
#define	kData_225	11
#define	kData_226	58
#define	kData_227	12
#define	kData_228	55
#define	kData_229	30
#define	kData_230	22
#define	kData_231	21
#define	kData_232	42
#define	kData_233	47
#define	kData_234	44
#define	kData_235	16
#define	kData_236	61
#define	kData_237	31
#define	kData_238	14
#define	kData_239	37
#define	kData_240	17
#define	kData_241	29
#define	kData_242	43
#define	kData_243	51
#define	kData_244	27
#define	kData_245	4
#define	kData_246	22
#define	kData_247	53
#define	kData_248	59
#define	kData_249	43
#define	kData_250	30
#define	kData_251	42
#define	kData_252	11
#define	kData_253	59
#define	kData_254	24
#define	kData_255	20
#define	kData_256	30
#define	kData_257	45
#define	kData_258	45
#define	kData_259	19
#define	kData_260	60
#define	kData_261	42
#define	kData_262	10
#define	kData_263	60
#define	kData_264	39
#define	kData_265	1
#define	kData_266	17
#define	kData_267	36
#define	kData_268	55
#define	kData_269	33

#define	kData_270	36
#define	kData_271	11
#define	kData_272	1
#define	kData_273	62
#define	kData_274	54
#define	kData_275	41
#define	kData_276	25
#define	kData_277	10
#define	kData_278	40
#define	kData_279	8
#define	kData_280	10
#define	kData_281	49
#define	kData_282	62
#define	kData_283	50
#define	kData_284	15
#define	kData_285	22
#define	kData_286	51
#define	kData_287	0
#define	kData_288	4
#define	kData_289	30
#define	kData_290	38
#define	kData_291	33
#define	kData_292	28
#define	kData_293	0
#define	kData_294	59
#define	kData_295	0
#define	kData_296	23
#define	kData_297	53
#define	kData_298	7
#define	kData_299	28
#define	kData_300	40
#define	kData_301	9
#define	kData_302	52
#define	kData_303	42
#define	kData_304	40
#define	kData_305	8
#define	kData_306	45
#define	kData_307	17
#define	kData_308	50
#define	kData_309	41
#define	kData_310	11
#define	kData_311	9
#define	kData_312	25
#define	kData_313	27
#define	kData_314	1
#define	kData_315	20
#define	kData_316	52
#define	kData_317	24
#define	kData_318	33
#define	kData_319	13
#define	kData_320	59
#define	kData_321	62
#define	kData_322	55
#define	kData_323	33
#define	kData_324	51
#define	kData_325	31
#define	kData_326	54
#define	kData_327	9
#define	kData_328	19
#define	kData_329	35
#define	kData_330	33
#define	kData_331	61
#define	kData_332	48
#define	kData_333	23
#define	kData_334	12
#define	kData_335	41
#define	kData_336	5
#define	kData_337	34
#define	kData_338	11
#define	kData_339	29
#define	kData_340	39
#define	kData_341	27
#define	kData_342	42
#define	kData_343	30
#define	kData_344	1
#define	kData_345	52
#define	kData_346	33
#define	kData_347	12
#define	kData_348	8
#define	kData_349	56
#define	kData_350	41

```

#define      kData_351      43
#define      kData_352      24
#define      kData_353      21
#define      kData_354      33
#define      kData_355      29
#define      kData_356      57
#define      kData_357      24
#define      kData_358      19
#define      kData_359      22
#define      kData_360      34
#define      kData_361      22
#define      kData_362      19
#define      kData_363      23
#define      kData_364      27
#define      kData_365      18
#define      kData_366      28
#define      kData_367      33
#define      kData_368      16
#define      kData_369      55
#define      kData_370      29
#define      kData_371      58
#define      kData_372      45
#define      kData_373      35
#define      kData_374      9
#define      kData_375      5
#define      kData_376      50
#define      kData_377      19
#define      kData_378      2
#define      kData_379      23
#define      kData_380      2
#define      kData_381      61
#define      kData_382      34
#define      kData_383      21
#define      kData_384      56
#define      kData_385      1
#define      kData_386      5
#define      kData_387      21
#define      kData_388      22

#endif

// ***** Plants a fixed CRC value (just some random number) if the CRC funtion is turned off.
#ifdef NO_CALC_CRC
#define      kTest_CRC_01      0x003A      // 0b1001101001010101
#define      kTest_CRC_02      0x003A      // 0b1001101001010101
#endif

//=====
// Input Params:      None
// Output Params:      None
//-----
// Description:      Loads into memory a Data Frame in preparation for
//                  transmission. This routine only has to be called once, but
//                  it must be called BEFORE calling the routine TX_Data_Frame.
//=====

Initialize_Data_Frame:

        st      r0, sp, 0
        add     sp, sp, 1

        // Load Data Frame into memory

        mov     r0, kData_Frame_Control
        st      r0, v_Tx_Data_Frame_Control

        mov     r0, kDuration_ID_field
        st      r0, v_Tx_Data_Duration_ID

        // Load Address 1 (Destination Address) into Memory
        // Can preload all but the last 16 bit address for the Destination Address,
        // because first two words are same for all stations in this WLAN

#ifdef STATION_1
        mov     r0, kSA_Address_STATION_01
        st      r0, v_Tx_Data_Address_2

```

```

#endif

#ifdef STATION_2
    mov    r0, kSA_Address_STATION_02
    st     r0, v_Tx_Data_Address_2

#endif

#ifdef STATION_3
    mov    r0, kSA_Address_STATION_03
    st     r0, v_Tx_Data_Address_2

#endif

#ifdef STATION_4
    mov    r0, kSA_Address_STATION_04
    st     r0, v_Tx_Data_Address_2

#endif

// Load Address 3 (BBSSID) into memory
    mov    r0, kBBSSID
    st     r0, v_Tx_Data_Address_3

// Loading random picked data into memory
#ifdef TELEMETRY
    mov    r0, kData_01
    st     r0, a_Tx_Data_Frame_Data + 0

    mov    r0, kData_02
    st     r0, a_Tx_Data_Frame_Data + 1

    mov    r0, kData_03
    st     r0, a_Tx_Data_Frame_Data + 2

    mov    r0, kData_04
    st     r0, a_Tx_Data_Frame_Data + 3

    mov    r0, kData_05
    st     r0, a_Tx_Data_Frame_Data + 4

    mov    r0, kData_06
    st     r0, a_Tx_Data_Frame_Data + 5

    mov    r0, kData_07
    st     r0, a_Tx_Data_Frame_Data + 6

    mov    r0, kData_08
    st     r0, a_Tx_Data_Frame_Data + 7

    mov    r0, kData_09
    st     r0, a_Tx_Data_Frame_Data + 8

    mov    r0, kData_10
    st     r0, a_Tx_Data_Frame_Data + 9

    mov    r0, kData_11
    st     r0, a_Tx_Data_Frame_Data + 10

    mov    r0, kData_12
    st     r0, a_Tx_Data_Frame_Data + 11

    mov    r0, kData_13
    st     r0, a_Tx_Data_Frame_Data + 12

    mov    r0, kData_14
    st     r0, a_Tx_Data_Frame_Data + 13

    mov    r0, kData_15
    st     r0, a_Tx_Data_Frame_Data + 14

    mov    r0, kData_16
    st     r0, a_Tx_Data_Frame_Data + 15

    mov    r0, kData_17
    st     r0, a_Tx_Data_Frame_Data + 16

    mov    r0, kData_18
    st     r0, a_Tx_Data_Frame_Data + 17

```

```

mov    r0, kData_19
st     r0, a_Tx_Data_Frame_Data + 18

mov    r0, kData_20
st     r0, a_Tx_Data_Frame_Data + 19

mov    r0, kData_21
st     r0, a_Tx_Data_Frame_Data + 20

mov    r0, kData_22
st     r0, a_Tx_Data_Frame_Data + 21

mov    r0, kData_23
st     r0, a_Tx_Data_Frame_Data + 22

mov    r0, kData_24
st     r0, a_Tx_Data_Frame_Data + 23

mov    r0, kData_25
st     r0, a_Tx_Data_Frame_Data + 24

mov    r0, kData_26
st     r0, a_Tx_Data_Frame_Data + 25

mov    r0, kData_27
st     r0, a_Tx_Data_Frame_Data + 26

mov    r0, kData_28
st     r0, a_Tx_Data_Frame_Data + 27

mov    r0, kData_29
st     r0, a_Tx_Data_Frame_Data + 28

mov    r0, kData_30
st     r0, a_Tx_Data_Frame_Data + 29

mov    r0, kData_31
st     r0, a_Tx_Data_Frame_Data + 30

mov    r0, kData_32
st     r0, a_Tx_Data_Frame_Data + 31

mov    r0, kData_33
st     r0, a_Tx_Data_Frame_Data + 32

mov    r0, kData_34
st     r0, a_Tx_Data_Frame_Data + 33

mov    r0, kData_35
st     r0, a_Tx_Data_Frame_Data + 34

mov    r0, kData_36
st     r0, a_Tx_Data_Frame_Data + 35

mov    r0, kData_37
st     r0, a_Tx_Data_Frame_Data + 36

mov    r0, kData_38
st     r0, a_Tx_Data_Frame_Data + 37

mov    r0, kData_39
st     r0, a_Tx_Data_Frame_Data + 38

mov    r0, kData_40
st     r0, a_Tx_Data_Frame_Data + 39

mov    r0, kData_41
st     r0, a_Tx_Data_Frame_Data + 40

mov    r0, kData_42
st     r0, a_Tx_Data_Frame_Data + 41

#endif

#ifdef AVIONICS

mov    r0, kData_01
st     r0, a_Tx_Data_Frame_Data + 0

mov    r0, kData_02
st     r0, a_Tx_Data_Frame_Data + 1

```

```

mov    r0, kData_03
st     r0, a_Tx_Data_Frame_Data + 2

mov    r0, kData_04
st     r0, a_Tx_Data_Frame_Data + 3

mov    r0, kData_05
st     r0, a_Tx_Data_Frame_Data + 4

mov    r0, kData_06
st     r0, a_Tx_Data_Frame_Data + 5

mov    r0, kData_07
st     r0, a_Tx_Data_Frame_Data + 6

mov    r0, kData_08
st     r0, a_Tx_Data_Frame_Data + 7

mov    r0, kData_09
st     r0, a_Tx_Data_Frame_Data + 8

mov    r0, kData_10
st     r0, a_Tx_Data_Frame_Data + 9

mov    r0, kData_11
st     r0, a_Tx_Data_Frame_Data + 10

mov    r0, kData_12
st     r0, a_Tx_Data_Frame_Data + 11

mov    r0, kData_13
st     r0, a_Tx_Data_Frame_Data + 12

mov    r0, kData_14
st     r0, a_Tx_Data_Frame_Data + 13

mov    r0, kData_15
st     r0, a_Tx_Data_Frame_Data + 14

mov    r0, kData_16
st     r0, a_Tx_Data_Frame_Data + 15

mov    r0, kData_17
st     r0, a_Tx_Data_Frame_Data + 16

mov    r0, kData_18
st     r0, a_Tx_Data_Frame_Data + 17

mov    r0, kData_19
st     r0, a_Tx_Data_Frame_Data + 18

mov    r0, kData_20
st     r0, a_Tx_Data_Frame_Data + 19

mov    r0, kData_21
st     r0, a_Tx_Data_Frame_Data + 20

mov    r0, kData_22
st     r0, a_Tx_Data_Frame_Data + 21

mov    r0, kData_23
st     r0, a_Tx_Data_Frame_Data + 22

mov    r0, kData_24
st     r0, a_Tx_Data_Frame_Data + 23

mov    r0, kData_25
st     r0, a_Tx_Data_Frame_Data + 24

mov    r0, kData_26
st     r0, a_Tx_Data_Frame_Data + 25

mov    r0, kData_27
st     r0, a_Tx_Data_Frame_Data + 26

mov    r0, kData_28
st     r0, a_Tx_Data_Frame_Data + 27

mov    r0, kData_29
st     r0, a_Tx_Data_Frame_Data + 28

```

```

mov    r0, kData_30
st     r0, a_Tx_Data_Frame_Data + 29

mov    r0, kData_31
st     r0, a_Tx_Data_Frame_Data + 30

mov    r0, kData_32
st     r0, a_Tx_Data_Frame_Data + 31

mov    r0, kData_33
st     r0, a_Tx_Data_Frame_Data + 32

mov    r0, kData_34
st     r0, a_Tx_Data_Frame_Data + 33

mov    r0, kData_35
st     r0, a_Tx_Data_Frame_Data + 34

mov    r0, kData_36
st     r0, a_Tx_Data_Frame_Data + 35

mov    r0, kData_37
st     r0, a_Tx_Data_Frame_Data + 36

mov    r0, kData_38
st     r0, a_Tx_Data_Frame_Data + 37

mov    r0, kData_39
st     r0, a_Tx_Data_Frame_Data + 38

mov    r0, kData_40
st     r0, a_Tx_Data_Frame_Data + 39

mov    r0, kData_41
st     r0, a_Tx_Data_Frame_Data + 40

mov    r0, kData_42
st     r0, a_Tx_Data_Frame_Data + 41

mov    r0, kData_43
st     r0, a_Tx_Data_Frame_Data + 42

mov    r0, kData_44
st     r0, a_Tx_Data_Frame_Data + 43

mov    r0, kData_45
st     r0, a_Tx_Data_Frame_Data + 44

mov    r0, kData_46
st     r0, a_Tx_Data_Frame_Data + 45

mov    r0, kData_47
st     r0, a_Tx_Data_Frame_Data + 46

mov    r0, kData_48
st     r0, a_Tx_Data_Frame_Data + 47

mov    r0, kData_49
st     r0, a_Tx_Data_Frame_Data + 48

mov    r0, kData_50
st     r0, a_Tx_Data_Frame_Data + 49

mov    r0, kData_51
st     r0, a_Tx_Data_Frame_Data + 50

mov    r0, kData_52
st     r0, a_Tx_Data_Frame_Data + 51

mov    r0, kData_53
st     r0, a_Tx_Data_Frame_Data + 52

mov    r0, kData_54
st     r0, a_Tx_Data_Frame_Data + 53

mov    r0, kData_55
st     r0, a_Tx_Data_Frame_Data + 54

mov    r0, kData_56
st     r0, a_Tx_Data_Frame_Data + 55

```

```

mov    r0, kData_57
st     r0, a_Tx_Data_Frame_Data + 56

mov    r0, kData_58
st     r0, a_Tx_Data_Frame_Data + 57

mov    r0, kData_59
st     r0, a_Tx_Data_Frame_Data + 58

mov    r0, kData_60
st     r0, a_Tx_Data_Frame_Data + 59

mov    r0, kData_61
st     r0, a_Tx_Data_Frame_Data + 60

mov    r0, kData_62
st     r0, a_Tx_Data_Frame_Data + 61

mov    r0, kData_63
st     r0, a_Tx_Data_Frame_Data + 62

mov    r0, kData_64
st     r0, a_Tx_Data_Frame_Data + 63

mov    r0, kData_65
st     r0, a_Tx_Data_Frame_Data + 64

mov    r0, kData_66
st     r0, a_Tx_Data_Frame_Data + 65

mov    r0, kData_67
st     r0, a_Tx_Data_Frame_Data + 66

mov    r0, kData_68
st     r0, a_Tx_Data_Frame_Data + 67

mov    r0, kData_69
st     r0, a_Tx_Data_Frame_Data + 68

mov    r0, kData_70
st     r0, a_Tx_Data_Frame_Data + 69

mov    r0, kData_71
st     r0, a_Tx_Data_Frame_Data + 70

mov    r0, kData_72
st     r0, a_Tx_Data_Frame_Data + 71

mov    r0, kData_73
st     r0, a_Tx_Data_Frame_Data + 72

mov    r0, kData_74
st     r0, a_Tx_Data_Frame_Data + 73

mov    r0, kData_75
st     r0, a_Tx_Data_Frame_Data + 74

mov    r0, kData_76
st     r0, a_Tx_Data_Frame_Data + 75

mov    r0, kData_77
st     r0, a_Tx_Data_Frame_Data + 76

mov    r0, kData_78
st     r0, a_Tx_Data_Frame_Data + 77

mov    r0, kData_79
st     r0, a_Tx_Data_Frame_Data + 78

mov    r0, kData_80
st     r0, a_Tx_Data_Frame_Data + 79

mov    r0, kData_81
st     r0, a_Tx_Data_Frame_Data + 80

mov    r0, kData_82
st     r0, a_Tx_Data_Frame_Data + 81

mov    r0, kData_83
st     r0, a_Tx_Data_Frame_Data + 82

```

```

mov    r0, kData_84
st     r0, a_Tx_Data_Frame_Data + 83

mov    r0, kData_85
st     r0, a_Tx_Data_Frame_Data + 84

mov    r0, kData_86
st     r0, a_Tx_Data_Frame_Data + 85

mov    r0, kData_87
st     r0, a_Tx_Data_Frame_Data + 86

mov    r0, kData_88
st     r0, a_Tx_Data_Frame_Data + 87

mov    r0, kData_89
st     r0, a_Tx_Data_Frame_Data + 88

mov    r0, kData_90
st     r0, a_Tx_Data_Frame_Data + 89

mov    r0, kData_91
st     r0, a_Tx_Data_Frame_Data + 90

mov    r0, kData_92
st     r0, a_Tx_Data_Frame_Data + 91

mov    r0, kData_93
st     r0, a_Tx_Data_Frame_Data + 92

mov    r0, kData_94
st     r0, a_Tx_Data_Frame_Data + 93

mov    r0, kData_95
st     r0, a_Tx_Data_Frame_Data + 94

mov    r0, kData_96
st     r0, a_Tx_Data_Frame_Data + 95

mov    r0, kData_97
st     r0, a_Tx_Data_Frame_Data + 96

mov    r0, kData_98
st     r0, a_Tx_Data_Frame_Data + 97

mov    r0, kData_99
st     r0, a_Tx_Data_Frame_Data + 98

mov    r0, kData_100
st     r0, a_Tx_Data_Frame_Data + 99

mov    r0, kData_101
st     r0, a_Tx_Data_Frame_Data + 100

mov    r0, kData_102
st     r0, a_Tx_Data_Frame_Data + 101

mov    r0, kData_103
st     r0, a_Tx_Data_Frame_Data + 102

mov    r0, kData_104
st     r0, a_Tx_Data_Frame_Data + 103

mov    r0, kData_105
st     r0, a_Tx_Data_Frame_Data + 104

mov    r0, kData_106
st     r0, a_Tx_Data_Frame_Data + 105

mov    r0, kData_107
st     r0, a_Tx_Data_Frame_Data + 106

mov    r0, kData_108
st     r0, a_Tx_Data_Frame_Data + 107

mov    r0, kData_109
st     r0, a_Tx_Data_Frame_Data + 108

mov    r0, kData_110
st     r0, a_Tx_Data_Frame_Data + 109

```

```

mov    r0, kData_111
st     r0, a_Tx_Data_Frame_Data + 110

mov    r0, kData_112
st     r0, a_Tx_Data_Frame_Data + 111

mov    r0, kData_113
st     r0, a_Tx_Data_Frame_Data + 112

mov    r0, kData_114
st     r0, a_Tx_Data_Frame_Data + 113

mov    r0, kData_115
st     r0, a_Tx_Data_Frame_Data + 114

mov    r0, kData_116
st     r0, a_Tx_Data_Frame_Data + 115

mov    r0, kData_117
st     r0, a_Tx_Data_Frame_Data + 116

mov    r0, kData_118
st     r0, a_Tx_Data_Frame_Data + 117

mov    r0, kData_119
st     r0, a_Tx_Data_Frame_Data + 118

mov    r0, kData_120
st     r0, a_Tx_Data_Frame_Data + 119

mov    r0, kData_121
st     r0, a_Tx_Data_Frame_Data + 120

mov    r0, kData_122
st     r0, a_Tx_Data_Frame_Data + 121

mov    r0, kData_123
st     r0, a_Tx_Data_Frame_Data + 122

mov    r0, kData_124
st     r0, a_Tx_Data_Frame_Data + 123

mov    r0, kData_125
st     r0, a_Tx_Data_Frame_Data + 124

mov    r0, kData_126
st     r0, a_Tx_Data_Frame_Data + 125

mov    r0, kData_127
st     r0, a_Tx_Data_Frame_Data + 126

mov    r0, kData_128
st     r0, a_Tx_Data_Frame_Data + 127

mov    r0, kData_129
st     r0, a_Tx_Data_Frame_Data + 128

mov    r0, kData_130
st     r0, a_Tx_Data_Frame_Data + 129

mov    r0, kData_131
st     r0, a_Tx_Data_Frame_Data + 130

mov    r0, kData_132
st     r0, a_Tx_Data_Frame_Data + 131

mov    r0, kData_133
st     r0, a_Tx_Data_Frame_Data + 132

mov    r0, kData_134
st     r0, a_Tx_Data_Frame_Data + 133

mov    r0, kData_135
st     r0, a_Tx_Data_Frame_Data + 134

mov    r0, kData_136
st     r0, a_Tx_Data_Frame_Data + 135

mov    r0, kData_137
st     r0, a_Tx_Data_Frame_Data + 136

```

```

mov    r0, kData_138
st     r0, a_Tx_Data_Frame_Data + 137

mov    r0, kData_139
st     r0, a_Tx_Data_Frame_Data + 138

mov    r0, kData_140
st     r0, a_Tx_Data_Frame_Data + 139

mov    r0, kData_141
st     r0, a_Tx_Data_Frame_Data + 140

mov    r0, kData_142
st     r0, a_Tx_Data_Frame_Data + 141

mov    r0, kData_143
st     r0, a_Tx_Data_Frame_Data + 142

mov    r0, kData_144
st     r0, a_Tx_Data_Frame_Data + 143

mov    r0, kData_145
st     r0, a_Tx_Data_Frame_Data + 144

mov    r0, kData_146
st     r0, a_Tx_Data_Frame_Data + 145

mov    r0, kData_147
st     r0, a_Tx_Data_Frame_Data + 146

mov    r0, kData_148
st     r0, a_Tx_Data_Frame_Data + 147

mov    r0, kData_149
st     r0, a_Tx_Data_Frame_Data + 148

mov    r0, kData_150
st     r0, a_Tx_Data_Frame_Data + 149

mov    r0, kData_151
st     r0, a_Tx_Data_Frame_Data + 150

mov    r0, kData_152
st     r0, a_Tx_Data_Frame_Data + 151

mov    r0, kData_153
st     r0, a_Tx_Data_Frame_Data + 152

mov    r0, kData_154
st     r0, a_Tx_Data_Frame_Data + 153

mov    r0, kData_155
st     r0, a_Tx_Data_Frame_Data + 154

mov    r0, kData_156
st     r0, a_Tx_Data_Frame_Data + 155

mov    r0, kData_157
st     r0, a_Tx_Data_Frame_Data + 156

mov    r0, kData_158
st     r0, a_Tx_Data_Frame_Data + 157

mov    r0, kData_159
st     r0, a_Tx_Data_Frame_Data + 158

mov    r0, kData_160
st     r0, a_Tx_Data_Frame_Data + 159

mov    r0, kData_161
st     r0, a_Tx_Data_Frame_Data + 160

mov    r0, kData_162
st     r0, a_Tx_Data_Frame_Data + 161

mov    r0, kData_163
st     r0, a_Tx_Data_Frame_Data + 162

mov    r0, kData_164
st     r0, a_Tx_Data_Frame_Data + 163

```

```

mov    r0, kData_165
st     r0, a_Tx_Data_Frame_Data + 164

mov    r0, kData_166
st     r0, a_Tx_Data_Frame_Data + 165

mov    r0, kData_167
st     r0, a_Tx_Data_Frame_Data + 166

mov    r0, kData_168
st     r0, a_Tx_Data_Frame_Data + 167

mov    r0, kData_169
st     r0, a_Tx_Data_Frame_Data + 168

mov    r0, kData_170
st     r0, a_Tx_Data_Frame_Data + 169

mov    r0, kData_171
st     r0, a_Tx_Data_Frame_Data + 170

mov    r0, kData_172
st     r0, a_Tx_Data_Frame_Data + 171

mov    r0, kData_173
st     r0, a_Tx_Data_Frame_Data + 172

mov    r0, kData_174
st     r0, a_Tx_Data_Frame_Data + 173

mov    r0, kData_175
st     r0, a_Tx_Data_Frame_Data + 174

mov    r0, kData_176
st     r0, a_Tx_Data_Frame_Data + 175

mov    r0, kData_177
st     r0, a_Tx_Data_Frame_Data + 176

mov    r0, kData_178
st     r0, a_Tx_Data_Frame_Data + 177

mov    r0, kData_179
st     r0, a_Tx_Data_Frame_Data + 178

mov    r0, kData_180
st     r0, a_Tx_Data_Frame_Data + 179

mov    r0, kData_181
st     r0, a_Tx_Data_Frame_Data + 180

mov    r0, kData_182
st     r0, a_Tx_Data_Frame_Data + 181

mov    r0, kData_183
st     r0, a_Tx_Data_Frame_Data + 182

mov    r0, kData_184
st     r0, a_Tx_Data_Frame_Data + 183

mov    r0, kData_185
st     r0, a_Tx_Data_Frame_Data + 184

mov    r0, kData_186
st     r0, a_Tx_Data_Frame_Data + 185

mov    r0, kData_187
st     r0, a_Tx_Data_Frame_Data + 186

mov    r0, kData_188
st     r0, a_Tx_Data_Frame_Data + 187

mov    r0, kData_189
st     r0, a_Tx_Data_Frame_Data + 188

mov    r0, kData_190
st     r0, a_Tx_Data_Frame_Data + 189

mov    r0, kData_191
st     r0, a_Tx_Data_Frame_Data + 190

```

```

mov    r0, kData_192
st     r0, a_Tx_Data_Frame_Data + 191

mov    r0, kData_193
st     r0, a_Tx_Data_Frame_Data + 192

mov    r0, kData_194
st     r0, a_Tx_Data_Frame_Data + 193

mov    r0, kData_195
st     r0, a_Tx_Data_Frame_Data + 194

mov    r0, kData_196
st     r0, a_Tx_Data_Frame_Data + 195

mov    r0, kData_197
st     r0, a_Tx_Data_Frame_Data + 196

mov    r0, kData_198
st     r0, a_Tx_Data_Frame_Data + 197

mov    r0, kData_199
st     r0, a_Tx_Data_Frame_Data + 198

mov    r0, kData_200
st     r0, a_Tx_Data_Frame_Data + 199

mov    r0, kData_201
st     r0, a_Tx_Data_Frame_Data + 200

mov    r0, kData_202
st     r0, a_Tx_Data_Frame_Data + 201

mov    r0, kData_203
st     r0, a_Tx_Data_Frame_Data + 202

mov    r0, kData_204
st     r0, a_Tx_Data_Frame_Data + 203

mov    r0, kData_205
st     r0, a_Tx_Data_Frame_Data + 204

mov    r0, kData_206
st     r0, a_Tx_Data_Frame_Data + 205

mov    r0, kData_207
st     r0, a_Tx_Data_Frame_Data + 206

mov    r0, kData_208
st     r0, a_Tx_Data_Frame_Data + 207

mov    r0, kData_209
st     r0, a_Tx_Data_Frame_Data + 208

mov    r0, kData_210
st     r0, a_Tx_Data_Frame_Data + 209

mov    r0, kData_211
st     r0, a_Tx_Data_Frame_Data + 210

mov    r0, kData_212
st     r0, a_Tx_Data_Frame_Data + 211

mov    r0, kData_213
st     r0, a_Tx_Data_Frame_Data + 212

mov    r0, kData_214
st     r0, a_Tx_Data_Frame_Data + 213

mov    r0, kData_215
st     r0, a_Tx_Data_Frame_Data + 214

mov    r0, kData_216
st     r0, a_Tx_Data_Frame_Data + 215

mov    r0, kData_217
st     r0, a_Tx_Data_Frame_Data + 216

mov    r0, kData_218
st     r0, a_Tx_Data_Frame_Data + 217

```

```

mov    r0, kData_219
st     r0, a_Tx_Data_Frame_Data + 218

mov    r0, kData_220
st     r0, a_Tx_Data_Frame_Data + 219

mov    r0, kData_221
st     r0, a_Tx_Data_Frame_Data + 220

mov    r0, kData_222
st     r0, a_Tx_Data_Frame_Data + 221

mov    r0, kData_223
st     r0, a_Tx_Data_Frame_Data + 222

mov    r0, kData_224
st     r0, a_Tx_Data_Frame_Data + 223

mov    r0, kData_225
st     r0, a_Tx_Data_Frame_Data + 224

mov    r0, kData_226
st     r0, a_Tx_Data_Frame_Data + 225

mov    r0, kData_227
st     r0, a_Tx_Data_Frame_Data + 226

mov    r0, kData_228
st     r0, a_Tx_Data_Frame_Data + 227

mov    r0, kData_229
st     r0, a_Tx_Data_Frame_Data + 228

mov    r0, kData_230
st     r0, a_Tx_Data_Frame_Data + 229

mov    r0, kData_231
st     r0, a_Tx_Data_Frame_Data + 230

mov    r0, kData_232
st     r0, a_Tx_Data_Frame_Data + 231

mov    r0, kData_233
st     r0, a_Tx_Data_Frame_Data + 232

mov    r0, kData_234
st     r0, a_Tx_Data_Frame_Data + 233

mov    r0, kData_235
st     r0, a_Tx_Data_Frame_Data + 234

mov    r0, kData_236
st     r0, a_Tx_Data_Frame_Data + 235

mov    r0, kData_237
st     r0, a_Tx_Data_Frame_Data + 236

mov    r0, kData_238
st     r0, a_Tx_Data_Frame_Data + 237

mov    r0, kData_239
st     r0, a_Tx_Data_Frame_Data + 238

mov    r0, kData_240
st     r0, a_Tx_Data_Frame_Data + 239

mov    r0, kData_241
st     r0, a_Tx_Data_Frame_Data + 240

mov    r0, kData_242
st     r0, a_Tx_Data_Frame_Data + 241

mov    r0, kData_243
st     r0, a_Tx_Data_Frame_Data + 242

mov    r0, kData_244
st     r0, a_Tx_Data_Frame_Data + 243

mov    r0, kData_245
st     r0, a_Tx_Data_Frame_Data + 244

```

```

mov    r0, kData_246
st     r0, a_Tx_Data_Frame_Data + 245

mov    r0, kData_247
st     r0, a_Tx_Data_Frame_Data + 246

mov    r0, kData_248
st     r0, a_Tx_Data_Frame_Data + 247

mov    r0, kData_249
st     r0, a_Tx_Data_Frame_Data + 248

mov    r0, kData_250
st     r0, a_Tx_Data_Frame_Data + 249

mov    r0, kData_251
st     r0, a_Tx_Data_Frame_Data + 250

mov    r0, kData_252
st     r0, a_Tx_Data_Frame_Data + 251

mov    r0, kData_253
st     r0, a_Tx_Data_Frame_Data + 252

mov    r0, kData_254
st     r0, a_Tx_Data_Frame_Data + 253

mov    r0, kData_255
st     r0, a_Tx_Data_Frame_Data + 254

mov    r0, kData_256
st     r0, a_Tx_Data_Frame_Data + 255

mov    r0, kData_257
st     r0, a_Tx_Data_Frame_Data + 256

mov    r0, kData_258
st     r0, a_Tx_Data_Frame_Data + 257

mov    r0, kData_259
st     r0, a_Tx_Data_Frame_Data + 258

mov    r0, kData_260
st     r0, a_Tx_Data_Frame_Data + 259

mov    r0, kData_261
st     r0, a_Tx_Data_Frame_Data + 260

mov    r0, kData_262
st     r0, a_Tx_Data_Frame_Data + 261

mov    r0, kData_263
st     r0, a_Tx_Data_Frame_Data + 262

mov    r0, kData_264
st     r0, a_Tx_Data_Frame_Data + 263

mov    r0, kData_265
st     r0, a_Tx_Data_Frame_Data + 264

mov    r0, kData_266
st     r0, a_Tx_Data_Frame_Data + 265

mov    r0, kData_267
st     r0, a_Tx_Data_Frame_Data + 266

mov    r0, kData_268
st     r0, a_Tx_Data_Frame_Data + 267

mov    r0, kData_269
st     r0, a_Tx_Data_Frame_Data + 268

mov    r0, kData_270
st     r0, a_Tx_Data_Frame_Data + 269

mov    r0, kData_271
st     r0, a_Tx_Data_Frame_Data + 270

mov    r0, kData_272
st     r0, a_Tx_Data_Frame_Data + 271

```

```

mov    r0, kData_273
st     r0, a_Tx_Data_Frame_Data + 272

mov    r0, kData_274
st     r0, a_Tx_Data_Frame_Data + 273

mov    r0, kData_275
st     r0, a_Tx_Data_Frame_Data + 274

mov    r0, kData_276
st     r0, a_Tx_Data_Frame_Data + 275

mov    r0, kData_277
st     r0, a_Tx_Data_Frame_Data + 276

mov    r0, kData_278
st     r0, a_Tx_Data_Frame_Data + 277

mov    r0, kData_279
st     r0, a_Tx_Data_Frame_Data + 278

mov    r0, kData_280
st     r0, a_Tx_Data_Frame_Data + 279

mov    r0, kData_281
st     r0, a_Tx_Data_Frame_Data + 280

mov    r0, kData_282
st     r0, a_Tx_Data_Frame_Data + 281

mov    r0, kData_283
st     r0, a_Tx_Data_Frame_Data + 282

mov    r0, kData_284
st     r0, a_Tx_Data_Frame_Data + 283

mov    r0, kData_285
st     r0, a_Tx_Data_Frame_Data + 284

mov    r0, kData_286
st     r0, a_Tx_Data_Frame_Data + 285

mov    r0, kData_287
st     r0, a_Tx_Data_Frame_Data + 286

mov    r0, kData_288
st     r0, a_Tx_Data_Frame_Data + 287

mov    r0, kData_289
st     r0, a_Tx_Data_Frame_Data + 288

mov    r0, kData_290
st     r0, a_Tx_Data_Frame_Data + 289

mov    r0, kData_291
st     r0, a_Tx_Data_Frame_Data + 290

mov    r0, kData_292
st     r0, a_Tx_Data_Frame_Data + 291

mov    r0, kData_293
st     r0, a_Tx_Data_Frame_Data + 292

mov    r0, kData_294
st     r0, a_Tx_Data_Frame_Data + 293

mov    r0, kData_295
st     r0, a_Tx_Data_Frame_Data + 294

mov    r0, kData_296
st     r0, a_Tx_Data_Frame_Data + 295

mov    r0, kData_297
st     r0, a_Tx_Data_Frame_Data + 296

mov    r0, kData_298
st     r0, a_Tx_Data_Frame_Data + 297

mov    r0, kData_299
st     r0, a_Tx_Data_Frame_Data + 298

```

```

mov    r0, kData_300
st     r0, a_Tx_Data_Frame_Data + 299

mov    r0, kData_301
st     r0, a_Tx_Data_Frame_Data + 300

mov    r0, kData_302
st     r0, a_Tx_Data_Frame_Data + 301

mov    r0, kData_303
st     r0, a_Tx_Data_Frame_Data + 302

mov    r0, kData_304
st     r0, a_Tx_Data_Frame_Data + 303

mov    r0, kData_305
st     r0, a_Tx_Data_Frame_Data + 304

mov    r0, kData_306
st     r0, a_Tx_Data_Frame_Data + 305

mov    r0, kData_307
st     r0, a_Tx_Data_Frame_Data + 306

mov    r0, kData_308
st     r0, a_Tx_Data_Frame_Data + 307

mov    r0, kData_309
st     r0, a_Tx_Data_Frame_Data + 308

mov    r0, kData_310
st     r0, a_Tx_Data_Frame_Data + 309

mov    r0, kData_311
st     r0, a_Tx_Data_Frame_Data + 310

mov    r0, kData_312
st     r0, a_Tx_Data_Frame_Data + 311

mov    r0, kData_313
st     r0, a_Tx_Data_Frame_Data + 312

mov    r0, kData_314
st     r0, a_Tx_Data_Frame_Data + 313

mov    r0, kData_315
st     r0, a_Tx_Data_Frame_Data + 314

mov    r0, kData_316
st     r0, a_Tx_Data_Frame_Data + 315

mov    r0, kData_317
st     r0, a_Tx_Data_Frame_Data + 316

mov    r0, kData_318
st     r0, a_Tx_Data_Frame_Data + 317

mov    r0, kData_319
st     r0, a_Tx_Data_Frame_Data + 318

mov    r0, kData_320
st     r0, a_Tx_Data_Frame_Data + 319

mov    r0, kData_321
st     r0, a_Tx_Data_Frame_Data + 320

mov    r0, kData_322
st     r0, a_Tx_Data_Frame_Data + 321

mov    r0, kData_323
st     r0, a_Tx_Data_Frame_Data + 322

mov    r0, kData_324
st     r0, a_Tx_Data_Frame_Data + 323

mov    r0, kData_325
st     r0, a_Tx_Data_Frame_Data + 324

mov    r0, kData_326
st     r0, a_Tx_Data_Frame_Data + 325

```

```

mov    r0, kData_327
st     r0, a_Tx_Data_Frame_Data + 326

mov    r0, kData_328
st     r0, a_Tx_Data_Frame_Data + 327

mov    r0, kData_329
st     r0, a_Tx_Data_Frame_Data + 328

mov    r0, kData_330
st     r0, a_Tx_Data_Frame_Data + 329

mov    r0, kData_331
st     r0, a_Tx_Data_Frame_Data + 330

mov    r0, kData_332
st     r0, a_Tx_Data_Frame_Data + 331

mov    r0, kData_333
st     r0, a_Tx_Data_Frame_Data + 332

mov    r0, kData_334
st     r0, a_Tx_Data_Frame_Data + 333

mov    r0, kData_335
st     r0, a_Tx_Data_Frame_Data + 334

mov    r0, kData_336
st     r0, a_Tx_Data_Frame_Data + 335

mov    r0, kData_337
st     r0, a_Tx_Data_Frame_Data + 336

mov    r0, kData_338
st     r0, a_Tx_Data_Frame_Data + 337

mov    r0, kData_339
st     r0, a_Tx_Data_Frame_Data + 338

mov    r0, kData_340
st     r0, a_Tx_Data_Frame_Data + 339

mov    r0, kData_341
st     r0, a_Tx_Data_Frame_Data + 340

mov    r0, kData_342
st     r0, a_Tx_Data_Frame_Data + 341

mov    r0, kData_343
st     r0, a_Tx_Data_Frame_Data + 342

mov    r0, kData_344
st     r0, a_Tx_Data_Frame_Data + 343

mov    r0, kData_345
st     r0, a_Tx_Data_Frame_Data + 344

mov    r0, kData_346
st     r0, a_Tx_Data_Frame_Data + 345

mov    r0, kData_347
st     r0, a_Tx_Data_Frame_Data + 346

mov    r0, kData_348
st     r0, a_Tx_Data_Frame_Data + 347

mov    r0, kData_349
st     r0, a_Tx_Data_Frame_Data + 348

mov    r0, kData_350
st     r0, a_Tx_Data_Frame_Data + 349

mov    r0, kData_351
st     r0, a_Tx_Data_Frame_Data + 350

mov    r0, kData_352
st     r0, a_Tx_Data_Frame_Data + 351

mov    r0, kData_353
st     r0, a_Tx_Data_Frame_Data + 352

```

```

mov    r0, kData_354
st     r0, a_Tx_Data_Frame_Data + 353

mov    r0, kData_355
st     r0, a_Tx_Data_Frame_Data + 354

mov    r0, kData_356
st     r0, a_Tx_Data_Frame_Data + 355

mov    r0, kData_357
st     r0, a_Tx_Data_Frame_Data + 356

mov    r0, kData_358
st     r0, a_Tx_Data_Frame_Data + 357

mov    r0, kData_359
st     r0, a_Tx_Data_Frame_Data + 358

mov    r0, kData_360
st     r0, a_Tx_Data_Frame_Data + 359

mov    r0, kData_361
st     r0, a_Tx_Data_Frame_Data + 360

mov    r0, kData_362
st     r0, a_Tx_Data_Frame_Data + 361

mov    r0, kData_363
st     r0, a_Tx_Data_Frame_Data + 362

mov    r0, kData_364
st     r0, a_Tx_Data_Frame_Data + 363

mov    r0, kData_365
st     r0, a_Tx_Data_Frame_Data + 364

mov    r0, kData_366
st     r0, a_Tx_Data_Frame_Data + 365

mov    r0, kData_367
st     r0, a_Tx_Data_Frame_Data + 366

mov    r0, kData_368
st     r0, a_Tx_Data_Frame_Data + 367

mov    r0, kData_369
st     r0, a_Tx_Data_Frame_Data + 368

mov    r0, kData_370
st     r0, a_Tx_Data_Frame_Data + 369

mov    r0, kData_371
st     r0, a_Tx_Data_Frame_Data + 370

mov    r0, kData_372
st     r0, a_Tx_Data_Frame_Data + 371

mov    r0, kData_373
st     r0, a_Tx_Data_Frame_Data + 372

mov    r0, kData_374
st     r0, a_Tx_Data_Frame_Data + 373

mov    r0, kData_375
st     r0, a_Tx_Data_Frame_Data + 374

mov    r0, kData_376
st     r0, a_Tx_Data_Frame_Data + 375

mov    r0, kData_377
st     r0, a_Tx_Data_Frame_Data + 376

mov    r0, kData_378
st     r0, a_Tx_Data_Frame_Data + 377

mov    r0, kData_379
st     r0, a_Tx_Data_Frame_Data + 378

mov    r0, kData_380
st     r0, a_Tx_Data_Frame_Data + 379

```

```

        mov    r0, kData_381
        st     r0, a_Tx_Data_Frame_Data + 380

        mov    r0, kData_382
        st     r0, a_Tx_Data_Frame_Data + 381

        mov    r0, kData_383
        st     r0, a_Tx_Data_Frame_Data + 382

        mov    r0, kData_384
        st     r0, a_Tx_Data_Frame_Data + 383

        mov    r0, kData_385
        st     r0, a_Tx_Data_Frame_Data + 384

        mov    r0, kData_386
        st     r0, a_Tx_Data_Frame_Data + 385

        mov    r0, kData_387
        st     r0, a_Tx_Data_Frame_Data + 386

        mov    r0, kData_388
        st     r0, a_Tx_Data_Frame_Data + 387

#ifdefif

        // If the CRC calculator is NOT on, put it an arbitrary CRC
        // Used mostly for testing purposes
        #ifdef NO_CALC_CRC

                mov    r0, kTest_CRC_01
                st     r0, a_Tx_Data_FCS + 0

                mov    r0, kTest_CRC_02
                st     r0, a_Tx_Data_FCS + 1

        #endif

        sub     sp, sp, 1
        ld     r0, sp, 0

        jsr    r6, r6

//=====
// Input Params:      None
// Output Params:     None
//-----
// Description:       Loads into memory a Data Frame in preparation for
//                    transmission. This routine only has to be called once, but
//                    it must be called BEFORE calling the routine TX_Data_Frame.
//=====

Initialize_TwT_Data_Frame:

        st     r0, sp, 0
        add    sp, sp, 1

        // Load Data Frame into memory

        mov    r0, kData_Frame_Control
        st     r0, v_Tx_Data_Frame_Control

        mov    r0, 0
        st     r0, v_Tx_Data_Duration_ID

        // Load Address 1 (Destination Address) into Memory
        // Can preload all but the last 16 bit address for the Destination Address,
        // because first two words are same for all stations in this WLAN

        #ifdef STATION_1
                mov    r0, kSA_Address_STATION_01
                st     r0, v_Tx_Data_Address_2
        #endif

        #ifdef STATION_2
                mov    r0, kSA_Address_STATION_02
                st     r0, v_Tx_Data_Address_2

```

```

#endif

#ifdef STATION_3
    mov    r0, kSA_Address_STATION_03
    st     r0, v_Tx_Data_Address_2
#endif

#ifdef STATION_4
    mov    r0, kSA_Address_STATION_04
    st     r0, v_Tx_Data_Address_2
#endif

// Load Address 3 (BBSSID) into memory
    mov    r0, kBBSSID
    st     r0, v_Tx_Data_Address_3

// If the CRC calculator is NOT on, put it an arbitrary CRC
// Used mostly for testing purposes
#ifdef NO_CALC_CRC

    mov    r0, kTest_CRC_01
    st     r0, a_Tx_Data_FCS + 0

    mov    r0, kTest_CRC_02
    st     r0, a_Tx_Data_FCS + 1

#endif

    sub    sp, sp, 1
    ld     r0, sp, 0

    jsr    r6, r6

//=====
// Input Params:    None
// Output Params:   None
//-----
// Description:     Loads into memory an ACK Frame in preparation for
//                  transmission. This routine only has to be called once, but
//                  it must be called BEFORE calling the routine TX_ACK_Frame.
//=====

Initialize_ACK_Frame:

    st     r0, sp, 0
    add    sp, sp, 1

    // Load Data Frame into memory

    mov    r0, kACK_Frame_Control
    st     r0, v_Tx_ACK_Frame_Control

    mov    r0, kDuration_ID_field
    st     r0, v_Tx_ACK_Duration_ID

    // Load Address 2 (Destination Address) into Memory
    // Can preload all but the last 16 bit address for the Destination Address,
    // because first two words are same for all stations in this WLAN

    mov    r0, kSA_Address_1st_16_bits
    st     r0, a_Tx_ACK_Address_2 + 0

    mov    r0, kSA_Address_2nd_16_bits
    st     r0, a_Tx_ACK_Address_2 + 1

#ifdef NO_CALC_CRC

    mov    r0, kTest_CRC_01
    st     r0, a_Tx_ACK_FCS + 0

    mov    r0, kTest_CRC_02
    st     r0, a_Tx_ACK_FCS + 1

#endif

    sub    sp, sp, 1
    ld     r0, sp, 0

```

```
jsr    r6, r6  
  
#endif
```

C.7. Init.asm

XInC library file included with the development kit. The library file Init.asm Initialization code that is run on thread 0 after XInC is powered on. This code sets up the Program Counters and Stack Pointers of all threads.

```
//*****
//***** (C) 2002 by Eleven Engineering Incorporated *****
//*****
//**
//**      Tabs: This file looks best with tab stops set every 6 spaces.
//**
//*****
//**
//** File:      Init.asm
//** Created: 25 Jun 2002 by Ryan Northcott
//** Revised: 25 Jun 2002 by Ryan Northcott
//**
//** Description: Initialization code that is run on thread 0 after XInC is
//**              powered on. This code sets up the Program Counters and Stack
//**              Pointers of all threads.
//**
//** Disclaimer: You may incorporate this sample source code into your
//**              program(s) without restriction. This sample source code has
//**              been provided "AS IS" and the responsibility for its
//**              operation is yours. You are not permitted to redistribute
//**              this sample source code as "Eleven sample source code" after
//**              having made changes. If you're going to re-distribute the
//**              source, we require that you make it clear in the source that
//**              the code was descended from Eleven sample source code, but
//**              that you've made changes.
//**
//*****
//*****

// Program the EEPROM
        bra    ProgramEEPROM
        0x8009

// Clear Resource Vector (Hardware Semaphores)
        inp    r0, SCUsrc
        outp    r0, SCUup

        mov    r0, 0x00FF

#ifdef __T0__
        // Setup Thread 0's Stack Pointer
        mov    r7, T0_SP
        bic    r0, r0, 0
#endif

#ifdef __T1__
        // Setup Thread 1's Program Counter & Stack Pointer
        mov    r1, 7 | (1<<3)
        outp    r1, SCUpntr
        mov    r1, T1_SP
        outp    r1, SCUreg
        mov    r1, Thread1
        outp    r1, SCUpc
        bic    r0, r0, 1
#endif

#ifdef __T2__
        // Setup Thread 2's Program Counter & Stack Pointer
        mov    r1, 7 | (2<<3)
        outp    r1, SCUpntr
        mov    r1, T2_SP
        outp    r1, SCUreg
        mov    r1, Thread2
        outp    r1, SCUpc
        bic    r0, r0, 2
#endif

#ifdef __T3__
        // Setup Thread 3's Program Counter & Stack Pointer
```

```

        mov     r1, 7 | (3<<3)
        outp    r1, SCUpntr
        mov     r1, T3_SP
        outp    r1, SCUreg
        mov     r1, Thread3
        outp    r1, SCUpc
        bic     r0, r0, 3
#endif

#ifdef __T4__      // Setup Thread 4's Program Counter & Stack Pointer
        mov     r1, 7 | (4<<3)
        outp    r1, SCUpntr
        mov     r1, T4_SP
        outp    r1, SCUreg
        mov     r1, Thread4
        outp    r1, SCUpc
        bic     r0, r0, 4
#endif

#ifdef __T5__      // Setup Thread 5's Program Counter & Stack Pointer
        mov     r1, 7 | (5<<3)
        outp    r1, SCUpntr
        mov     r1, T5_SP
        outp    r1, SCUreg
        mov     r1, Thread5
        outp    r1, SCUpc
        bic     r0, r0, 5
#endif

#ifdef __T6__      // Setup Thread 6's Program Counter & Stack Pointer
        mov     r1, 7 | (6<<3)
        outp    r1, SCUpntr
        mov     r1, T6_SP
        outp    r1, SCUreg
        mov     r1, Thread6
        outp    r1, SCUpc
        bic     r0, r0, 6
#endif

#ifdef __T7__      // Setup Thread 7's Program Counter & Stack Pointer
        mov     r1, 7 | (7<<3)
        outp    r1, SCUpntr
        mov     r1, T7_SP
        outp    r1, SCUreg
        mov     r1, Thread7
        outp    r1, SCUpc
        bic     r0, r0, 7
#endif

        outp    r0, SCUstop // Enable the desired threads

```

C.8. LEDs.asm

XInC library file included with the development kit. The library file defines routines for using the LEDs on the XInC Development Board.

```
//*****
//***** (C) 2002 by Eleven Engineering Incorporated *****
//*****
/**
/**      Tabs: This file looks best with tab stops set every 6 spaces.
/**
/**
//*****
//*****
/**
/**      $RCSfile: LEDs.asm,v $
/**      $Revision: 1.4 $
/**      Tag $Name: $
/**      $Date: 2003/02/12 21:17:11 $
/**      $Author: eleven $
/**
/**      Project: XInC Library
/**      Description: Routines for using the LEDs on the XInC Development Board.
/**
/**      NOTE: To use these routines in your project, you must assign
/**              kDevLEDs_Semaphore to one of your hardware semaphores.
/**
/**      Disclaimer: You may incorporate this sample source code into your
/**                  program(s) without restriction. This sample source code has
/**                  been provided "AS IS" and the responsibility for its
/**                  operation is yours. You are not permitted to redistribute
/**                  this sample source code as "Eleven sample source code" after
/**                  having made changes. If you're going to re-distribute the
/**                  source, we require that you make it clear in the source that
/**                  the code was descended from Eleven sample source code, but
/**                  that you've made changes.
/**
//*****
//*****
/**
/**      InitializeLEDs
/**      TurnOnLEDs
/**      TurnOffLEDs
/**      ToggleLEDs
/**      SetLEDs
/**
//*****
//*****

#ifndef __LED_UTILS__
#define __LED_UTILS__

#define      DevLED_Port1_Cfg          GPFcfg
#define      DevLED_Port2_Cfg          GPCCfg
#define      DevLED_Port1              GPFout
#define      DevLED_Port2              GPCout
#define      DevLED_Port1_Init         0xFFFF
#define      DevLED_Port2_Init         0xFFFF

//=====
// Input Params:      None
// Output Params:      None
//-----
// Description:      LED initialization
//=====
InitializeLEDs:
        st        r0, sp, 0
        st        r1, sp, 1
        st        r6, sp, 2
        add       sp, sp, 3

        mov       r0, DevLED_Port1_Init
        outp      r0, DevLED_Port1_Cfg

        mov       r0, DevLED_Port2_Init
```

```

        outp    r0, DevLED_Port2_Cfg

        mov     r1, 0
        jsr     r6, SetLEDs

InitializeLEDs_END:
        sub     sp, sp, 3
        ld      r0, sp, 0
        ld      r1, sp, 1
        ld      r6, sp, 2
        jsr     r6, r6

//=====
// Input Params:    r1 = LED Vector
// Output Params:    None
//-----
// Description:      Turns on the LEDs that are specified by the bits set in r1.
//                   The first LED is controlled by bit 0, the second by bit 1,
//                   etc. All 16 bits map to LEDs since there are 16 LEDs on the
//                   DevKit Board.
//=====
TurnOnLEDs:
        st      r0, sp, 0
        st      r1, sp, 1
        st      r2, sp, 2

        xor     r1, r1, 0xFFFF    // Invert r1 (LEDs are active low)
        and     r2, r1, 0x00FF    // Mask the bits for LED Port 1
        rol     r1, r1, -8
        and     r1, r1, 0x00FF    // Mask the bits for LED Port 2

        mov     r0, kDevLEDs_Semaphore
        outp    r0, SCUdown

        inp     r0, DevLED_Port1
        and     r0, r0, r2        // Clear the bits for the LEDs we want to turn on on LED Port 1
        outp    r0, DevLED_Port1

        inp     r0, DevLED_Port2
        and     r0, r0, r1        // Clear the bits for the LEDs we want to turn on on LED Port 2
        outp    r0, DevLED_Port2

        mov     r0, kDevLEDs_Semaphore
        outp    r0, SCUup

TurnOnLEDs_END:
        ld      r0, sp, 0
        ld      r1, sp, 1
        ld      r2, sp, 2
        jsr     r6, r6

//=====
// Input Params:    r1 = LED Vector
// Output Params:    None
//-----
// Description:      Turns off the LEDs that are specified by the bits set in r1.
//                   The first LED is controlled by bit 0, the second by bit 1,
//                   etc. All 16 bits map to LEDs since there are 16 LEDs on the
//                   DevKit Board.
//=====
TurnOffLEDs:
        st      r0, sp, 0
        st      r1, sp, 1
        st      r2, sp, 2

        and     r2, r1, 0x00FF    // Mask the bits for LED Port 1
        rol     r1, r1, -8
        and     r1, r1, 0x00FF    // Mask the bits for LED Port 2

        mov     r0, kDevLEDs_Semaphore
        outp    r0, SCUdown

        inp     r0, DevLED_Port1
        ior     r0, r0, r2        // Set the bits for the LEDs we want to turn off on LED Port 1
        outp    r0, DevLED_Port1

        inp     r0, DevLED_Port2

```

```

        ior    r0, r0, r1    // Set the bits for the LEDs we want to turn off on LED Port 2
        outp   r0, DevLED_Port2

        mov    r0, kDevLEDs_Semaphore
        outp   r0, SCUUp

TurnOffLEDs_END:
        ld     r0, sp, 0
        ld     r1, sp, 1
        ld     r2, sp, 2
        jsr    r6, r6

//=====
// Input Params:    r1 = LED Vector
// Output Params:   None
//-----
// Description:     Toggles the LEDs that are specified by the bits set in r1.
//                  The first LED is controlled by bit 0, the second by bit 1,
//                  etc. All 16 bits map to LEDs since there are 16 LEDs on the
//                  DevKit Board.
//=====
ToggleLEDs:
        st     r0, sp, 0
        st     r1, sp, 1
        st     r2, sp, 2

        and    r2, r1, 0x00FF    // Mask the bits for LED Port 1
        rol    r1, r1, -8
        and    r1, r1, 0x00FF    // Mask the bits for LED Port 2

        mov    r0, kDevLEDs_Semaphore
        outp   r0, SCUdown

        inp    r0, DevLED_Port1
        xor    r0, r0, r2    // Toggle the bits for the LEDs we want to toggle on LED Port 1
        outp   r0, DevLED_Port1

        inp    r0, DevLED_Port2
        xor    r0, r0, r1    // Toggle the bits for the LEDs we want to toggle on LED Port 2
        outp   r0, DevLED_Port2

        mov    r0, kDevLEDs_Semaphore
        outp   r0, SCUUp

ToggleLEDs_END:
        ld     r0, sp, 0
        ld     r1, sp, 1
        ld     r2, sp, 2
        jsr    r6, r6

//=====
// Input Params:    r1 = LED Vector
// Output Params:   None
//-----
// Description:     Turns on the LEDs that are specified by bits set in r1 and
//                  turns off the LEDs that are specified by bits cleared in r1.
//                  The first LED is controlled by bit 0, the second by bit 1,
//                  etc. All 16 bits map to LEDs since there are 16 LEDs on the
//                  DevKit Board.
//=====
SetLEDs:
        st     r0, sp, 0
        st     r1, sp, 1
        st     r2, sp, 2

        xor    r1, r1, 0xFFFF    // Invert r1 (LEDs are active low)
        and    r2, r1, 0x00FF    // Mask the bits for LED Port 1
        rol    r1, r1, -8
        and    r1, r1, 0x00FF    // Mask the bits for LED Port 2

        mov    r0, kDevLEDs_Semaphore
        outp   r0, SCUdown

        outp   r2, DevLED_Port1    // Set the bits for the LEDs we want to turn on on LED Port 1
        outp   r1, DevLED_Port2    // Set the bits for the LEDs we want to turn on on LED Port 2

        mov    r0, kDevLEDs_Semaphore

```

```

        outp    r0, SCUUp

SetLEDS_END:
        ld      r0, sp, 0
        ld      r1, sp, 1
        ld      r2, sp, 2
        jsr     r6, r6

#endif

```

C.9. Long_Data.asm

The file includes any data variables, arrays, or strings for the program. The file also sets up stack pointers for each thread. The location of the stack pointer is initialized in the Init.asm file.

```

//*****
//***** (C) 2002 by Eleven Engineering Incorporated *****
//*****
//**
//**      Tabs:  This file looks best with tab stops set every 6 spaces.
//**
//*****
//*****
//**
//** File:          Long_Data.asm
//**
//** Project: IEEE 802.11 MAC emulator.  It can send to multiple (1-4) stations
//** Created: 1 June 2004 by Capt Joshua D. Green
//**
//** Description: Contains the data (memory variables and tables) used by Two-Way
//**               Text Messaging Application.  All data should be stored in this
//**               file or in the "Short_Data.asm" file to ensure that it is stored
//**               in a separate 2kWord memory block from all code.
//**
//** Disclaimer: You may incorporate this sample source code into your
//**               program(s) without restriction.  This sample source code has
//**               been provided "AS IS" and the responsibility for its
//**               operation is yours.  You are not permitted to redistribute
//**               this sample source code as "Eleven sample source code" after
//**               having made changes.  If you're going to re-distribute the
//**               source, we require that you make it clear in the source that
//**               the code was descended from Eleven sample source code, but
//**               that you've made changes.
//**
//*****
//*****

v_Number_of_ACKs_Sent:      @ = @ + 1
v_T7_Number_of_ACKs_Sent:   @ = @ + 1


SCU_PNTR:                   @ = @ + 1

v_TEMP:                     @ = @ + 1


// Flags
v_Medium_Idle_Flag:         @ = @ + 1
v_Received_ACK_Packet_FLAG: @ = @ + 1
v_Received_Stuff_FLAG:      @ = @ + 1


// Variables
v_Number_of_RX:              @ = @ + 1
v_PacketStartTime:           @ = @ + 1
v_Number_of_Retransmissions: @ = @ + 1
v_Delay_Time:                @ = @ + 1
v_BV_Slots:                  @ = @ + 1
v_RN:                        @ = @ + 1
v_Received_Stuff:            @ = @ + 1
v_Packets_in_Que:            @ = @ + 1

```

```

v_Thread_0_packet_que_number:    @ = @ + 1
v_Thread_5_packet_que_number:    @ = @ + 1
v_Thread_6_packet_que_number:    @ = @ + 1
v_Queued_Packets:                @ = @ + 1
v_ACKs_Received:                 @ = @ + 1
v_Number_of_tests:               @ = @ + 1

v_T7_Sent_Packets:               @ = @ + 1
v_T7_Queued_Packets:             @ = @ + 1
v_T7_Number_of_TX:               @ = @ + 1
v_T7_Number_of_Failed_TX:        @ = @ + 1
v_T7_ACKs_Received:              @ = @ + 1

// Arrays
a_Time:                          @ = @ + 3
a_Start_Time:                    @ = @ + 3
a_End_Time:                      @ = @ + 3
a_Recorded_TX:
    v_Number_of_TX:              @ = @ + 1
    v_Number_of_Failed_TX:       @ = @ + 1
    a_Number_of_ReTX:            @ = @ + (kMaxReTransmit - 1)
a_T7_Number_of_ReTX:             @ = @ + (kMaxReTransmit - 1)
a_T7_Mean_Delay_Time:            @ = @ + 3

// Timing Arrays
a_BEGIN_Time_Seconds:            @ = @ + kTransmitter_Buffer_Size
a_BEGIN_Time_Microseconds:       @ = @ + kTransmitter_Buffer_Size
a_BEGIN_Time_Milliseconds:       @ = @ + kTransmitter_Buffer_Size

a_Thread_6_BEGIN_Times:          @ = @ + 3
a_Thread_6_END_Times:            @ = @ + 3

a_Mean_Delay_Time:               @ = @ + 3

// TX Packet Que
v_Tx_Data_Frame_Control:         @ = @ + 1
v_Tx_Data_Duration_ID:           @ = @ + 1 // Same for ALL packets
a_Tx_Data_Address_1:             @ = @ + kTransmitter_Buffer_Size
v_Tx_Data_Address_2:             @ = @ + 1 // Same for ALL packets
v_Tx_Data_Address_3:             @ = @ + 1 // Same for ALL packets
v_Tx_Data_Sequence_Number:       @ = @ + 1 // Same for ALL packet
// Frame Data
#ifdef TELEMETRY
a_Tx_Data_Frame_Data:           @ = @ + 42 // 0-41
#endif
#ifdef AVIONICS
a_Tx_Data_Frame_Data:           @ = @ + 388 // 0-387
#endif

// FCS
a_Tx_Data_FCS:                   @ = @ + 2 // Same for ALL packets
// Timing for Frame (used to calculate mean delay)
a_Tx_Data_Frame_Start_Time_sec:   @ = @ + kTransmitter_Buffer_Size
a_Tx_Data_Frame_Start_Time_ms:    @ = @ + kTransmitter_Buffer_Size
a_Tx_Data_Frame_Start_Time_us:    @ = @ + kTransmitter_Buffer_Size

a_Rx_Data_Frame:
v_Rx_Data_Frame_Control:         @ = @ + 1 // 0
v_Rx_Data_Duration_ID:           @ = @ + 1 // 1
a_Rx_Data_Address_1:             @ = @ + 3 // 2,3,4
a_Rx_Data_Address_2:             @ = @ + 3 // 5,6,7
a_Rx_Data_Address_3:             @ = @ + 3 // 8,9,10
v_Rx_Data_Sequence_Number:       @ = @ + 1 // 11
#ifdef TELEMETRY
a_Rx_Data_Frame_Data:           @ = @ + 42 // 12-53
#endif
#ifdef AVIONICS
a_Rx_Data_Frame_Data:           @ = @ + 388 // 12-399
#endif
a_Rx_Data_FCS:                   @ = @ + 2 // 29,30 or 54,55 or 400,401

a_Tx_ACK_Frame:
v_Tx_ACK_Frame_Control:          @ = @ + 1
v_Tx_ACK_Duration_ID:            @ = @ + 1
a_Tx_ACK_Address_2:              @ = @ + 3
a_Tx_ACK_FCS:                    @ = @ + 2

a_Rx_ACK_Frame:
v_Rx_ACK_Frame_Control:          @ = @ + 1

```

```

        v_Rx_ACK_Duration_ID:      @ = @ + 1
        a_Rx_ACK_Address_2:        @ = @ + 3
        a_Rx_ACK_FCS:              @ = @ + 2

a_Tx_Sequence_Numbers:
    v_Tx_Sequence_Number_Station_1:  @ = @ + 1
    v_Tx_Sequence_Number_Station_2:  @ = @ + 1
    v_Tx_Sequence_Number_Station_3:  @ = @ + 1
    v_Tx_Sequence_Number_Station_4:  @ = @ + 1

a_Rx_Sequence_Numbers:
    v_Rx_Sequence_Number_Station_1:  @ = @ + 1
    v_Rx_Sequence_Number_Station_2:  @ = @ + 1
    v_Rx_Sequence_Number_Station_3:  @ = @ + 1
    v_Rx_Sequence_Number_Station_4:  @ = @ + 1

// Messages
MSG_DOT:      ".", EOS
MSG_READY_2:  "Press any key to start Transmitting.", CR, LF, EOS
MSG_TX_START_1: "Started Transmitting. To stop hit the 'd' key", CR, LF, EOS
MSG_TX_START_2: "Press any other key again to start recording.", CR, LF, EOS
MSG_TX_STOPPED: "---Stopped transmitting---", CR, LF, EOS
MSG_DATA_DUMP_1: "Delay|# of |Test |Paket|      | 1 | 2 | 3 |      |ACKs |---Mean Delay---|",
CR, LF, EOS
MSG_DATA_DUMP_2: "(mil)|slots|Time |Qued |TX      |ReTX |ReTX |ReTX |F-TX |RX      |(Sec)|(ms) |(mil)|",
CR, LF, EOS
MSG_RECORDING:  "****Recording Started****", CR, LF, EOS

#ifdef STATION_1
MSG_CURRENT_STATION: "This is Station #1", CR,LF, EOS
MSG_STATION_NUMBER: "Choose Station # (2-4): ", EOS
#endif

#ifdef STATION_2
MSG_CURRENT_STATION: "This is Station #2", CR,LF, EOS
MSG_STATION_NUMBER: "Choose Station # (1, or 3-4): ", EOS
#endif

#ifdef STATION_3
MSG_CURRENT_STATION: "This is Station #3", CR,LF, EOS
MSG_STATION_NUMBER: "Choose Station # (1-2 or 4): ", EOS
#endif

#ifdef STATION_4
MSG_CURRENT_STATION: "This is Station #4", CR,LF, EOS
MSG_STATION_NUMBER: "Choose Station # (1-3): ", EOS
#endif

#ifdef PrintErrors
MSG_CORRUPTPACKET: "Corrupt Packet!", CR, LF, EOS
MSG_HUNTERROR:     "Hunt Error!", CR, LF, EOS
#endif

#ifdef __T0__
T0_SP: @ = @ + kStackSize
#endif

#ifdef __T1__
T1_SP: @ = @ + kStackSize
#endif

#ifdef __T2__
T2_SP: @ = @ + kStackSize
#endif

#ifdef __T3__
T3_SP: @ = @ + kStackSize
#endif

#ifdef __T4__
T4_SP: @ = @ + kStackSize
#endif

#ifdef __T5__
T5_SP: @ = @ + kStackSize
#endif

```

```

#ifdef __T6__
T6_SP: @ = @ + kStackSize
#endif

#ifdef __T7__
T7_SP: @ = @ + kStackSize
#endif

```

C.10. Math.asm

XInC library file included with the development kit. The library file defines firmware routines for doing math not available as a single XInC instruction.

```

//*****
//***** (C) 2002 by Eleven Engineering Incorporated *****
//*****
/**
/**      Tabs: This file looks best with tab stops set every 6 spaces.
/**
/**
//*****
//*****
/**
/**      $RCSfile: Math.asm,v $
/**      $Revision: 1.2 $
/**      Tag $Name: $
/**      $Date: 2003/02/12 21:17:11 $
/**      $Author: eleven $
/**
/**      Project: XInC Library
/**      Description: Firmware routines for doing math not available as a single
/**                    XInC instruction.
/**
/**      Disclaimer: You may incorporate this sample source code into your
/**                    program(s) without restriction. This sample source code has
/**                    been provided "AS IS" and the responsibility for its
/**                    operation is yours. You are not permitted to redistribute
/**                    this sample source code as "Eleven sample source code" after
/**                    having made changes. If you're going to re-distribute the
/**                    source, we require that you make it clear in the source that
/**                    the code was descended from Eleven sample source code, but
/**                    that you've made changes.
/**
//*****
//*****
/**
/**      Routines:
/**
/**      IntegerDivide
/**
//*****
//*****

#ifdef __MATH__
#define __MATH__

//=====
// Input Params:    r1 = Numerator (Unsigned 16-bit Integer)
//                  r2 = Divisor (Unsigned 16-bit Integer)
// Output Params:   r1 = Result
//                  r2 = Remainder
//-----
// Description:     Performs the unsigned integer division of one 16-Bit unsigned
//                  integer by another 16-bit unsigned integer.
//
//                  Note: x/0 is treated as x/1 to prevent an infinite loop.
//
//                  There is some optimization in the register usage to be done
//                  but this routine is compatible with the old IntegerDivide
//                  routine. This version has some speed optimizations over the
//                  previous version.
//=====
IntegerDivide:

```

```

    st    r3, sp, 0
    st    r4, sp, 1
    st    r5, sp, 2
    add   sp, sp, 3

    // r1 = dividend //numerator// result
    // r2 = remainder
    // r3 = divisor
    // r4 = loop counter
    // r5 = carry

    mov    r4, 17                // Setup loop counter
    add    r3, r2, 0             // mov r3 = r2
    mov    r2, 0                 // Clear remainder
    mov    r5, 0                 // Clear carry

IntegerDivide_loop:
    sub    r4, r4, 1             // Decrement loop counter
    bc     ZS, IntegerDivide_done

    add    r1, r1, r1            // Shift left dividend into carry
    bc     CS, IntegerDivide_carryset

IntegerDivide_carryclear:
    add    r1, r1, r5            // Add carry-in
    add    r2, r2, r2            // Shift left remainder with no carry
    sub    r2, r2, r3            // Subtract divisor from remainder

    bc     ULT, IntegerDivide_undo // Check for negative result(CS)

    mov    r5, 1                 // Set carry

    bra    IntegerDivide_loop

IntegerDivide_undo:
    add    r2, r2, r3            // Add back divisor
    mov    r5, 0                 // Clear carry

    bra    IntegerDivide_loop

IntegerDivide_carryset:
    add    r1, r1, r5            // Add carry-in
    add    r2, r2, r2            // Shift left remainder
    add    r2, r2, 1             // Add carry
    sub    r2, r2, r3            // Subtract divisor from remainder
    bc     ULT, IntegerDivide_undo // Check for negative result(CS)
    mov    r5, 1                 // Set carry

    bra    IntegerDivide_loop

IntegerDivide_done:
    add    r1, r1, r1            // Shift left dividend
    add    r1, r1, r5            // Add carry-in

IntegerDivide_END:
    sub    sp, sp, 3
    ld     r3, sp, 0
    ld     r4, sp, 1
    ld     r5, sp, 2
    jsr    r6, r6

#endif

```

C.11. RFWaves.asm

XInC library file included with the development kit. The library file defines routines for using the RFW RF Module.

```
//*****
//***** (C) 2002 by Eleven Engineering Incorporated *****
//*****
/**
**      Tabs: This file looks best with tab stops set every 6 spaces.
**
**
//*****
//*****
/**
**      File: RFWaves.asm
/** Created: 24 July 2003 by Ryan Northcott
/** Revised: 1 June 2004 by Capt Joshua D. Green
/**
/**      Project: IEEE 802.11 MAC emulator. It can send to multiple (1-4) stations
/**      Description: Routines for using the RFW RF Module.
/**
/**      Disclaimer: This code was descended from Eleven Engineering sample
/**                  source code, but changes were made by Capt Joshua D. Green
/**
/** NOTE: Be sure to select speed via #ifdef RFWaves1Mbps, or #ifdef RFWaves3Mbps
/**
//*****
//*****
/**
** RF Waves RADIO ROUTINES:
/**
/**      RFW_Initialize
/**
/**      RFW_SwitchOn
/**      RFW_SwitchOff
/**
/**      RFW_EnterReceiveMode
/**      RFW_EnterTransmitMode
/**
/**      RFW_DelayRxCal
/**      RFW_DelayTxCal
/**
/**      RFW_SendPacketPreamble
/**      RFW_Send3Words616
/**      RFW_Send3Bytes616
/**
/**      RFW_Send_6_Bits_616
/**      RFW_Send_16_Bits_Unencoded
/**
/**      WiFi_Send_Data_Packet
/**      WiFi_Send_ACK_Packet
/**
/**      RFW_SendPacketPostamble
/**      RFW_Receive3Words616
/**      RFW_Receive3Bytes616
/**
/**      WiFi_Receive_Packet
/**
/**      RFW_Send16Chips
/**
//*****
//*****

//-----
// BBU Rate Table ROM Locations

                #define      kRate4TableROMAddress      0x0020
                #define      kRate6TableROMAddress      0x0030

//-----
// RFW IO Definitions

                #define      RFWConfigPort              GPACfg
                #define      RFWDataPort                 GPAout
```

```

#define      kRFXCENBit          0          // Output
#define      kRFXONBit           1          // Output

#define      kRFXHardErrorBit    14
#define      kRFXHuntBit         15

#define      kStation_01         49
#define      kStation_02         50
#define      kStation_03         51
#define      kStation_04         52
#define      kStation_05         53

//=====
// Input Params:      none
// Output Params:     none
//-----
// Description:       Initialize the RFX port/radio
//=====
RFX_Initialize:
                                st      r1, sp, 0
                                add     sp, sp, 1

                                // The following settings give us just under 1Mbps:
                                // transmit mode:      BBURgb = 20969 =~1999855.042 bps =~ 99927.5208 bps(real bit rate)
                                // receive mode:       BBURgb = 10484 =~99927.5208 bps

                                // ***My Try***
                                // transmit mode:      BBURgb = 20971 =~1999855.042 bps =~ 99927.5208 bps(real bit rate)
                                // receive mode:       BBURgb = 10485 =~99927.5208 bps
                                //
                                // for 2 Mbps
                                // transmit mode:      BBURgb = 41942 = 4,000,000 bps = 2,000,000 bps(real bit rate)
                                // receive mode:       BBURgb = 20971 = 2,000,000 bps

                                // Reset BBU
                                mov     r1, 0x03
                                outp    r1, BBUcfig                      // Enable the BBU

#ifdef RFXWaves1Mbps
                                mov     r1, 10485 //20969                // should be for 50 MHz
#endif
#ifdef RFXWaves2Mbps
                                mov     r1, 20971                        // should be for 50 MHz
#endif
#ifdef RFXWaves3Mbps
                                mov     r1, 32768                        // should be for 50 MHz
#endif
                                outp    r1, BBURgb                       // Setup the Baud Rate Generator

                                // Initialize the RFX GPIO Port
                                inp     r1, RFXConfigPort
                                bis     r1, r1, kRFXRXONBit + 8
                                bis     r1, r1, kRFXCENBit + 8
                                outp    r1, RFXConfigPort

RFX_Initialize_END:
                                sub     sp, sp, 1
                                ld      r1, sp, 0
                                jsr     r6, r6

//=====
// Input Params:      none
// Output Params:     none
//-----
// Description:       Switches the RFX chip on and puts it into Rx mode
//=====
RFX_SwitchOn:
                                st      r1, sp, 0

                                inp     r1, RFXDataPort
                                bis     r1, r1, kRFXRXONBit
                                bis     r1, r1, kRFXCENBit
                                outp    r1, RFXDataPort

RFX_SwitchOn_END:
                                ld      r1, sp, 0

                                jsr     r6, r6

```

```

//=====
// Input Params:      none
// Output Params:     none
//-----
// Description:       Switches the RFW chip off and puts it into Rx mode
//=====
RFW_SwitchOff:
                st      r1, sp, 0

                inp     r1, RFWDataPort
                bic     r1, r1, kRFWRXONBit
                bic     r1, r1, kRFWCENBit
                outp    r1, RFWDataPort

RFW_SwitchOff_END:
                ld      r1, sp, 0
                jsr     r6, r6

//=====
// Input Params:      none
// Output Params:     none
//-----
// Description:       Switches the RFW chip off and puts it into Rx mode
//=====
RFW_EnterReceiveMode:
                st      r1, sp, 0

                // Receive mode:      BBUbrg = 10484 =~999927.5208 bps

                // Reset BBU
                mov     r1, 0x01
                outp    r1, BBUcfg                // Enable the BBU

#ifdef RFWaves1Mbps
                mov     r1, 10485 //20969          // should be for 50 MHz
#endif
#ifdef RFWaves2Mbps
                mov     r1, 20971                  // should be for 50 MHz
#endif
#ifdef RFWaves3Mbps
                mov     r1, 32768                  // should be for 50 MHz
#endif

                outp    r1, BBUbrg                // Setup the Baud Rate Generator

                inp     r1, RFWDataPort
                bic     r1, r1, kRFWRXONBit
                bis     r1, r1, kRFWCENBit
                outp    r1, RFWDataPort

RFW_EnterReceiveMode_END:
                ld      r1, sp, 0
                jsr     r6, r6

//=====
// Input Params:      none
// Output Params:     none
//-----
// Description:       Switches the RFW chip off and puts it into Tx mode
//=====
RFW_EnterTransmitMode:
                st      r1, sp, 0

                // Transmit mode: BBUbrg = 20969 = ~1999855.042 bps = ~999927.5208 bps(real bit rate)
                // Reset BBU
                mov     r1, 0x03
                outp    r1, BBUcfg                // Enable the BBU

#ifdef RFWaves1Mbps
                mov     r1, 20971 //20969          // should be for 50 MHz
#endif
#ifdef RFWaves2Mbps
                mov     r1, 41942                  // should be for 50 MHz
#endif
#ifdef RFWaves3Mbps
                mov     r1, 65535                  // should be for 50 MHz
#endif
#endif

```

```

                                outp    r1, BBUBrg                // Setup the Baud Rate Generator

                                inp      r1, RFWDataPort
                                bis      r1, r1, kRFWRXONBit
                                bis      r1, r1, kRFWXCENBit
                                outp     r1, RFWDataPort

RFW_EnterTransmitMode_END:
                                ld       r1, sp, 0
                                jsr      r6, r6

//=====
// Input Params:      none
// Output Params:     none
//-----
// Description:       Switches the RFW chip off and puts it into Rx mode
//=====
RFW_DelayRxCal:
                                st       r1, sp, 0
                                st       r2, sp, 1

                                inp      r1, BBUtime

#ifdef RFWaves1Mbps
                                add      r1, r1, 20
#endif
#ifdef RFWaves2Mbps
                                add      r1, r1, 40
#endif
#ifdef RFWaves3Mbps
                                add      r1, r1, 60
#endif
                                RFW_DelayRxCal_loop:
                                inp      r2, BBUtime
                                sub      r2, r2, r1
                                bc       NS, RFW_DelayRxCal_loop

RFW_DelayRxCal_END:
                                ld       r1, sp, 0
                                ld       r2, sp, 1
                                jsr      r6, r6

//=====
// Input Params:      none
// Output Params:     none
//-----
// Description:       Switches the RFW chip off and puts it into Tx mode
//=====
RFW_DelayTxCal:
                                st       r1, sp, 0
                                st       r2, sp, 1

                                inp      r1, BBUtime

#ifdef RFWaves1Mbps
                                add      r1, r1, 40
#endif
#ifdef RFWaves3Mbps
                                add      r1, r1, 80
#endif
#ifdef RFWaves3Mbps
                                add      r1, r1, 120
#endif
                                RFW_DelayTxCal_loop:
                                inp      r2, BBUtime
                                sub      r2, r2, r1
                                bc       NS, RFW_DelayTxCal_loop

RFW_DelayTxCal_END:
                                ld       r1, sp, 0
                                ld       r2, sp, 1
                                jsr      r6, r6

//=====
// Input Params:      None
// Output Params:     None
//-----

```

```

// Description:      Sends a training sequence to calibrate the data bit slicer of
//                  the receiving radio and then sends a start code to
//                  establish word synchronization. The second start code is
//                  sent to decrease the likelihood of the receiving radio
//                  thinking that random noise is the start of a packet.
//=====
RFW_SendPacketPreamble:
                                st      r0, sp, 0
                                st      r1, sp, 1
                                st      r2, sp, 2
                                st      r6, sp, 3

                                mov      r2,0x5555
                                jsr      r6,RFW_Send16Chips      //put preamble
                                mov      r2,0x5555
                                jsr      r6,RFW_Send16Chips      //put preamble
                                mov      r2,0x5555
                                jsr      r6,RFW_Send16Chips      //put preamble
                                mov      r2,0x5555
                                jsr      r6,RFW_Send16Chips      //put preamble
                                mov      r2,0x5555
                                jsr      r6,RFW_Send16Chips      //put preamble
                                mov      r2,0x217B
                                jsr      r6,RFW_Send16Chips      //put start word 1
                                mov      r2,0x217B
                                jsr      r6,RFW_Send16Chips      //put start word 2

RFW_SendPacketPreamble_END:
                                ld      r0, sp, 0
                                ld      r1, sp, 1
                                ld      r2, sp, 2
                                ld      r6, sp, 3
                                jsr      r6, r6

//=====
// Input Params:      r0 = The first word to transmit
//                  r1 = The second word to transmit
//                  r2 = The third word to transmit
// Output Params:      r0 = Garbage
//                  r1 = Garbage
//                  r2 = Garbage
//                  r3 = Garbage
//                  r4 = Garbage
//                  r5 = Garbage
//-----
// Description:      Transmits the 3 specified words using the 616 Rate Table.
//=====
RFW_Send3Words616:
                                // Send the first 6 bits
                                rol      r0, r0, 6
                                and      r3, r0, 0b00111111
                                ld      r3, r3, kRate6TableROMAddress

                                and      r4,r3,0x000F
                                ld      r4,r4,rxNibbleTable
                                rol      r3,r3,-4
                                and      r5,r3,0x000F
                                ld      r5,r5,rxNibbleTable
                                rol      r5,r5,8                //shift left
                                ior      r4,r4,r5
                                outp     r4,BBUTx // 1

                                rol      r3,r3,-4
                                and      r4,r3,0x000F
                                ld      r4,r4,rxNibbleTable
                                rol      r3,r3,-4
                                and      r5,r3,0x000F
                                ld      r5,r5,rxNibbleTable
                                rol      r5,r5,8                //shift left
                                ior      r4,r4,r5
                                outp     r4,BBUTx // 2

                                // Send the second 6 bits
                                rol      r0, r0, 6
                                and      r3, r0, 0b00111111
                                ld      r3, r3, kRate6TableROMAddress

                                and      r4,r3,0x000F
                                ld      r4,r4,rxNibbleTable
                                rol      r3,r3,-4
                                and      r5,r3,0x000F
                                ld      r5,r5,rxNibbleTable
                                rol      r5,r5,8                //shift left

```

```

ior    r4,r4,r5
outp   r4,BBUTx // 3

rol    r3,r3,-4
and    r4,r3,0x000F
ld     r4,r4,rxNibbleTable
rol    r3,r3,-4
and    r5,r3,0x000F
ld     r5,r5,rxNibbleTable
rol    r5,r5,8                      //shift left
ior    r4,r4,r5
outp   r4,BBUTx // 4

        // Send the third 6 bits
        rol    r0, r0, 6
        and    r3, r0, 0b00111100
        rol    r1, r1, 2
        and    r0, r1, 0b00000011
        ior    r3, r3, r0
        ld     r3, r3, kRate6TableROMAddress

and    r4,r3,0x000F
ld     r4,r4,rxNibbleTable
rol    r3,r3,-4
and    r5,r3,0x000F
ld     r5,r5,rxNibbleTable
rol    r5,r5,8                      //shift left
ior    r4,r4,r5
outp   r4,BBUTx // 5

rol    r3,r3,-4
and    r4,r3,0x000F
ld     r4,r4,rxNibbleTable
rol    r3,r3,-4
and    r5,r3,0x000F
ld     r5,r5,rxNibbleTable
rol    r5,r5,8                      //shift left
ior    r4,r4,r5
outp   r4,BBUTx // 6

        // Send the fourth 6 bits
        rol    r1, r1, 6
        and    r3, r1, 0b00111111
        ld     r3, r3, kRate6TableROMAddress

and    r4,r3,0x000F
ld     r4,r4,rxNibbleTable
rol    r3,r3,-4
and    r5,r3,0x000F
ld     r5,r5,rxNibbleTable
rol    r5,r5,8                      //shift left
ior    r4,r4,r5
outp   r4,BBUTx // 7

rol    r3,r3,-4
and    r4,r3,0x000F
ld     r4,r4,rxNibbleTable
rol    r3,r3,-4
and    r5,r3,0x000F
ld     r5,r5,rxNibbleTable
rol    r5,r5,8                      //shift left
ior    r4,r4,r5
outp   r4,BBUTx // 8

        // Send the fifth 6 bits
        rol    r1, r1, 6
        and    r3, r1, 0b00111111
        ld     r3, r3, kRate6TableROMAddress

and    r4,r3,0x000F
ld     r4,r4,rxNibbleTable
rol    r3,r3,-4
and    r5,r3,0x000F
ld     r5,r5,rxNibbleTable
rol    r5,r5,8                      //shift left
ior    r4,r4,r5
outp   r4,BBUTx // 9

rol    r3,r3,-4
and    r4,r3,0x000F
ld     r4,r4,rxNibbleTable
rol    r3,r3,-4

```

```

and    r5,r3,0x000F
ld     r5,r5,rxNibbleTable
rol    r5,r5,8           //shift left
ior    r4,r4,r5
outp   r4,BBUTx // 10

        // Send the sixth 6 bits
        rol    r1, r1, 6
        and    r3, r1, 0b00110000
        rol    r2, r2, 4
        and    r1, r2, 0b00001111
        ior    r3, r3, r1
        ld     r3, r3, kRate6TableROMAddress

and    r4,r3,0x000F
ld     r4,r4,rxNibbleTable
rol    r3,r3,-4
and    r5,r3,0x000F
ld     r5,r5,rxNibbleTable
rol    r5,r5,8           //shift left
ior    r4,r4,r5
outp   r4,BBUTx // 11

rol    r3,r3,-4
and    r4,r3,0x000F
ld     r4,r4,rxNibbleTable
rol    r3,r3,-4
and    r5,r3,0x000F
ld     r5,r5,rxNibbleTable
rol    r5,r5,8           //shift left
ior    r4,r4,r5
outp   r4,BBUTx // 12

        // Send the seventh 6 bits
        rol    r2, r2, 6
        and    r3, r2, 0b00111111
        ld     r3, r3, kRate6TableROMAddress

and    r4,r3,0x000F
ld     r4,r4,rxNibbleTable
rol    r3,r3,-4
and    r5,r3,0x000F
ld     r5,r5,rxNibbleTable
rol    r5,r5,8           //shift left
ior    r4,r4,r5
outp   r4,BBUTx // 13

rol    r3,r3,-4
and    r4,r3,0x000F
ld     r4,r4,rxNibbleTable
rol    r3,r3,-4
and    r5,r3,0x000F
ld     r5,r5,rxNibbleTable
rol    r5,r5,8           //shift left
ior    r4,r4,r5
outp   r4,BBUTx // 14

        // Send the eighth 6 bits
        rol    r2, r2, 6
        and    r3, r2, 0b00111111
        ld     r3, r3, kRate6TableROMAddress

and    r4,r3,0x000F
ld     r4,r4,rxNibbleTable
rol    r3,r3,-4
and    r5,r3,0x000F
ld     r5,r5,rxNibbleTable
rol    r5,r5,8           //shift left
ior    r4,r4,r5
outp   r4,BBUTx // 15

rol    r3,r3,-4
and    r4,r3,0x000F
ld     r4,r4,rxNibbleTable
rol    r3,r3,-4
and    r5,r3,0x000F
ld     r5,r5,rxNibbleTable
rol    r5,r5,8           //shift left
ior    r4,r4,r5
outp   r4,BBUTx // 16

```

```

RFW_Send3Words616_END:
                                jsr    r6, r6

//=====
// Input Params:    r0 = The first byte to transmit
//                  r1 = The second byte to transmit
//                  r2 = The third byte to transmit
// Output Params:   r0 = Garbage
//                  r1 = Garbage
//                  r2 = Garbage
//                  r3 = Garbage
//-----
// Description:      Transmits the 3 specified bytes using the 616 Rate Table.
//=====
RFW_Send3Bytes616:
                                st      r4, sp, 0
                                st      r5, sp, 1

                                // Send the first 6 bits
                                rol      r0, r0, 8
                                ior      r0, r0, r1    // merge r0 and r1 into 1 word
                                rol      r0, r0, 6
                                and      r3, r0, 0b00111111
                                ld        r3, r3, kRate6TableROMAddress

                                and      r4, r3, 0x000F
                                ld        r4, r4, rxNibbleTable
                                rol      r3, r3, -4
                                and      r5, r3, 0x000F
                                ld        r5, r5, rxNibbleTable
                                rol      r5, r5, 8                //shift left
                                ior      r4, r4, r5
                                outp      r4, BBUTx

                                rol      r3, r3, -4
                                and      r4, r3, 0x000F
                                ld        r4, r4, rxNibbleTable
                                rol      r3, r3, -4
                                and      r5, r3, 0x000F
                                ld        r5, r5, rxNibbleTable
                                rol      r5, r5, 8                //shift left
                                ior      r4, r4, r5
                                outp      r4, BBUTx

                                // Send the second 6 bits
                                rol      r0, r0, 6
                                and      r3, r0, 0b00111111
                                ld        r3, r3, kRate6TableROMAddress

                                and      r4, r3, 0x000F
                                ld        r4, r4, rxNibbleTable
                                rol      r3, r3, -4
                                and      r5, r3, 0x000F
                                ld        r5, r5, rxNibbleTable
                                rol      r5, r5, 8                //shift left
                                ior      r4, r4, r5
                                outp      r4, BBUTx

                                rol      r3, r3, -4
                                and      r4, r3, 0x000F
                                ld        r4, r4, rxNibbleTable
                                rol      r3, r3, -4
                                and      r5, r3, 0x000F
                                ld        r5, r5, rxNibbleTable
                                rol      r5, r5, 8                //shift left
                                ior      r4, r4, r5
                                outp      r4, BBUTx

                                and      r2, r2, 0x00FF

                                // Send the third 6 bits
                                and      r0, r0, 0xF000
                                rol      r2, r2, 4
                                ior      r0, r0, r2    // merge r0 and r2

                                rol      r0, r0, 6
                                and      r3, r0, 0b00111111
                                ld        r3, r3, kRate6TableROMAddress

                                and      r4, r3, 0x000F
                                ld        r4, r4, rxNibbleTable
                                rol      r3, r3, -4
                                and      r5, r3, 0x000F
                                ld        r5, r5, rxNibbleTable

```

```

        rol    r5,r5,8                //shift left
        ior    r4,r4,r5
        outp   r4,BBUTx

        rol    r3,r3,-4
        and    r4,r3,0x000F
        ld     r4,r4,rxNibbleTable
        rol    r3,r3,-4
        and    r5,r3,0x000F
        ld     r5,r5,rxNibbleTable
        rol    r5,r5,8                //shift left
        ior    r4,r4,r5
        outp   r4,BBUTx

        // Send the fourth 6 bits
        rol    r0, r0, 6
        and    r3, r0, 0b00111111
        ld     r3, r3, kRate6TableROMAddress

        and    r4,r3,0x000F
        ld     r4,r4,rxNibbleTable
        rol    r3,r3,-4
        and    r5,r3,0x000F
        ld     r5,r5,rxNibbleTable
        rol    r5,r5,8                //shift left
        ior    r4,r4,r5
        outp   r4,BBUTx

        rol    r3,r3,-4
        and    r4,r3,0x000F
        ld     r4,r4,rxNibbleTable
        rol    r3,r3,-4
        and    r5,r3,0x000F
        ld     r5,r5,rxNibbleTable
        rol    r5,r5,8                //shift left
        ior    r3,r4,r5
ld     r4, sp, 0
ld     r5, sp, 1
        outp   r3,BBUTx

RFW_Send3Bytes616_END:
        jsr    r6, r6

//=====
// Input Params:    r3 = The 6 bits to transmit
// Output Params:   None
//-----
// Description:     Transmits the 6 specified bits using the 616 Rate Table.
//=====

RFW_Send_6_Bits_616:

        st     r5, sp, 0
        add    sp, sp, 1

        // Send the 6 bits
        and    r3, r3, 0b00111111

        ld     r3, r3, kRate6TableROMAddress
        and    r4, r3, 0x000F
        ld     r4, r4, rxNibbleTable
        rol    r3, r3, -4
        and    r5, r3, 0x000F
        ld     r5, r5, rxNibbleTable
        rol    r5, r5, 8                //shift left
        ior    r4, r4, r5
        outp   r4, BBUTx                // Transmitting 16 bits

// The reason for transmitting 6 bits this way has to do with the way that this
// particular radio actually operates. On Tx the radio sends a pulse when ever
// it sees a rising edge in the bitstream. We use a table (rxNibbleTable) to do a
// transformation of NRZ encoding into a form that the radio requires. For
// example, a '0' gets encoded as '00' and a '1' gets encoded as '01'. This
// encoded waveform looks like the 3 Mbps signal that the radio expects.

        rol    r3, r3, -4
        and    r4, r3, 0x000F
        ld     r4, r4, rxNibbleTable
        rol    r3, r3, -4
        and    r5, r3, 0x000F

```

```

        ld    r5, r5, rxNibbleTable
        rol   r5, r5, 8      //shift left
        ior   r4, r4, r5
        outp  r4, BBUTx      // Transmitting second 8 bits

RFW_Send_6_Bits_616_End:

        sub   sp, sp, 1
        ld    r5, sp, 0

        jsr   r6, r6

//=====
// Input Params:    r3 = The 16 bit word to transmit
// Output Params:    None
//-----
// Description:      Transmits the 16 specified bits.
//=====

RFW_Send_16_Bits_Unencoded:

        // Send the 6 bits
//
        mov   r3, r0
        and   r4, r3, 0x000F
        ld    r4, r4, rxNibbleTable
        rol   r3, r3, -4
        and   r5, r3, 0x000F
        ld    r5, r5, rxNibbleTable
        rol   r5, r5, 8      //shift left
        ior   r4, r4, r5
        outp  r4, BBUTx      // Transmitting first 8 bits

// The reason for transmitting 6 bits this way has to do with the way that this
// particular radio actually operates. On Tx the radio sends a pulse when ever
// it sees a rising edge in the bitstream. We use a table (rxNibbleTable) to do a
// transformation of NRZ encoding into a form that the radio requires. For
// example, a '0' gets encoded as '00' and a '1' gets encoded as '01'. This
// encoded waveform looks like the 3 Mbps signal that the radio expects.

        rol   r3, r3, -4
        and   r4, r3, 0x000F
        ld    r4, r4, rxNibbleTable
        rol   r3, r3, -4
        and   r5, r3, 0x000F
        ld    r5, r5, rxNibbleTable
        rol   r5, r5, 8      //shift left
        ior   r4, r4, r5
        outp  r4, BBUTx      // Transmitting second 8 bits

RFW_Send_16_Bits_Unencoded_End:

        jsr   r6, r6

//=====
// Input Params:    r5 - Packet Number in que Transmitting
// Output Params:    None
//-----
// Description:      Transmits a WiFi Data Frame. The frame has a fixed length of
//                   84 bytes long (for the Data)
// Note:             This routine was written by Capt Green
//=====

WiFi_Send_Data_Packet:

        st    r6, sp, 0
        add   sp, sp, 1

        // Transmitting Frame Control
        // Tells distant end wether packet is an ACK or Data Packet
        // NOTE: Uses 6/16 encoding - sends out 16 bits for 6 bits of data
        // NOTE: Using the 6/16 encoding differes from IEEE 802.11 standard.
        // NOTE: It is done here strictly for experimental purposes
        ld    r3, v_Tx_Data_Frame_Control
        jsr   r6, RFW_Send_6_Bits_616      // Frame Octet 1-2

        // Transmitting Duration/ID
        // NOTE: Uses 6/16 encoding - sends out 16 bits for 6 bits of data
        // NOTE: Using the 6/16 encoding differes from IEEE 802.11 standard.

```

```

        // NOTE: It is done here strictly for experimental purposes

        ld    r3, v_Tx_Data_Duration_ID
        jsr   r6, RFW_Send_6_Bits_616    // Frame Octet 3-4

// Transmitting Address 1
// Frame Octets 5-10
    // NOTE: Uses 6/16 encoding - sends out 16 bits for 6 bits of data
    // NOTE: Using the 6/16 encoding differs from IEEE 802.11 standard.
    // NOTE: The order also is different from the IEEE 802.11 standard
    // NOTE: It is done here strictly for experimental purposes

        ld    r5, v_Thread_0_packet_que_number
        mov   r1, 1<<kTx_Data_Address_1_SEMAPHORE
        outp  r1, SCUdown
        ld    r2, r5, a_Tx_Data_Address_1    // Address 1 - Destination Address

of Frame
only)        outp  r1, SCUUp                    // (last 6 bits of the address

        mov   r0, 3
WiFi_Send_Data_Packet_Transmitting_Address_1_LOOP:
    // Transmits the address three times
        mov   r3, r2
        jsr   r6, RFW_Send_6_Bits_616
        sub   r0, r0, 1
        bc    ZC, WiFi_Send_Data_Packet_Transmitting_Address_1_LOOP

// Transmitting Address 2
// Frame Octets 11-16
    // NOTE: Uses 6/16 encoding - sends out 16 bits for 6 bits of data
    // NOTE: Using the 6/16 encoding differs from IEEE 802.11 standard.
    // NOTE: The order also is different from the IEEE 802.11 standard
    // NOTE: It is done here strictly for experimental purposes

    // Address 2 - Sending Station Address
        ld    r2, v_Tx_Data_Address_2

    // (last 6 bits of the address only)
        mov   r0, 3
WiFi_Send_Data_Packet_Transmitting_Address_2_LOOP:
    // Transmits the address three times
        mov   r3, r2
        jsr   r6, RFW_Send_6_Bits_616
        sub   r0, r0, 1
        bc    ZC, WiFi_Send_Data_Packet_Transmitting_Address_2_LOOP

// Transmitting Address 3 - BSSID
// Frame Octets 21-22
    // NOTE: This is different from the IEEE 802.11 standard
    // There would normally be 6 Octets for the BSSID
    // The BSSID is not used in this experiment, so it is not big deal
    // NOTE: Uses 6/16 encoding - sends out 16 bits for 6 bits of data

    // Address 2 - Sending Station Address
        ld    r2, v_Tx_Data_Address_3
// (last 6 bits of the address only)
        mov   r0, 3

WiFi_Send_Data_Packet_Transmitting_Address_3_LOOP:
    // Transmits the address three times
        mov   r3, r2
        jsr   r6, RFW_Send_6_Bits_616
        sub   r0, r0, 1
        bc    ZC, WiFi_Send_Data_Packet_Transmitting_Address_3_LOOP

// Transmitting Sequence Number
// Frame Octets 23-24
        ld    r3, v_Tx_Data_Sequence_Number // a_Tx_Data_Frame + 11
        jsr   r6, RFW_Send_6_Bits_616

// Transmitting Frame Data
        mov   r1, 0
WiFi_Send_Data_Packet_Data_LOOP:
        ld    r3, r1, a_Tx_Data_Frame_Data
        jsr   r6, RFW_Send_6_Bits_616
        add   r1, r1, 1

#ifdef TELEMETRY
        sub   r0, r1, 42
#endif
#endif

```

```

#ifdef AVIONICS
    sub    r0, r1, 388
#endif

    bc     NE, WiFi_Send_Data_Packet_Data_LOOP

// Transmitting FCS (frame check sequence)
    ld     r3, a_Tx_Data_FCS + 0
    jsr    r6, RFW_Send_6_Bits_616

    ld     r3, a_Tx_Data_FCS + 1
    jsr    r6, RFW_Send_6_Bits_616

WiFi_Send_Data_Packet_END:
    sub    sp, sp, 1
    ld     r6, sp, 0

    jsr    r6, r6

//=====
// Input Params:    r0 = Address 2 - Sending Station Address (last 6 bits of the address only)
// Output Params:    None
//-----
// Description:      Transmits a WiFi ACK Frame.
// Note:             This routine was written by Capt Green
//=====

WiFi_Send_ACK_Packet:

    st     r6, sp, 0
    add    sp, sp, 1

// Transmitting Frame Control
// Tells distant end whether packet is an ACK or Data Packet
// NOTE: Uses 6/16 encoding - sends out 16 bits for 6 bits of data
// NOTE: Using the 6/16 encoding differs from IEEE 802.11 standard.
// NOTE: It is done here strictly for experimental purposes
    ld     r3, a_Tx_ACK_Frame + 0
    jsr    r6, RFW_Send_6_Bits_616    // Frame Octet 1-2

// Transmitting Duration/ID
// NOTE: Uses 6/16 encoding - sends out 16 bits for 6 bits of data
// NOTE: Using the 6/16 encoding differs from IEEE 802.11 standard.
// NOTE: It is done here strictly for experimental purposes
    ld     r3, a_Tx_ACK_Frame + 1
    jsr    r6, RFW_Send_6_Bits_616    // Frame Octet 3-4

// Transmitting Received Address 2 - Sending Station Address
// NOTE: Uses 6/16 encoding - sends out 16 bits for 6 bits of data
// NOTE: Using the 6/16 encoding differs from IEEE 802.11 standard.
// NOTE: It is done here strictly for experimental purposes

    ld     r3, a_Tx_ACK_Frame + 2
    jsr    r6, RFW_Send_6_Bits_616    // Frame Octet 5-6

    ld     r3, a_Tx_ACK_Frame + 3
    jsr    r6, RFW_Send_6_Bits_616    // Frame Octet 7-8

    mov    r3, r0 // r0 = Destination Station
                // (last 16 bits of the address only)

    jsr    r6, RFW_Send_6_Bits_616    // Frame Octet 9-10

// Transmitting FCS (frame check sequence)
    ld     r3, a_Tx_ACK_Frame + 5
    jsr    r6, RFW_Send_6_Bits_616

    ld     r3, a_Tx_ACK_Frame + 6
    jsr    r6, RFW_Send_6_Bits_616

WiFi_Send_ACK_Packet_END:
    sub    sp, sp, 1
    ld     r6, sp, 0

    jsr    r6, r6

//=====

```

```

// Input Params:      None
// Output Params:     r0 = Garbage
//-----
// Description:       Sends a training sequence to reset the BBU of the receiving
//                   radio.
//=====
RFW_SendPacketPostamble:
                                st      r6, sp, 0
                                add     sp, sp, 1

                                mov     r2, 0x5555
                                jsr     r6, RFW_Send16Chips           //flush transmit pipe

                                sub     sp, sp, 1
                                ld      r6, sp, 0

                                jsr     r6, r6

//=====
// Input Params:      r1 = Pointer to 3 Word Array to Store Received Data
// Output Params:     r0 = Error (0 = No Error, 1 = Hunt Error, 2 = Hard Error)
//                   r2 = Garbage
//                   r3 = Garbage
//-----
// Description:       Receives 3 words using the 616 Rate Table and places them
//                   into the array pointed to by r1. If the routine does not
//                   receives all three words successfully, it returns an error
//                   code in r0.
//
//                   There must not be more than 13 instruction times between
//                   successive calls to this routine when receiving a packet.
//=====
RFW_Receive3Words616:
                                // Receive the first 6 bits
                                inp     r2, BBURx6
                                bc      NS, RFW_Receive3Words616_HuntError // Abort if no data detected
                                bic     r2, r2, kRFWHardErrorBit
                                bc      VS, RFW_Receive3Words616_HardError // Abort if hard error detected
                                and     r2, r2, 0b00111111
                                rol     r0, r2, 10

                                // Receive the second 6 bits
                                inp     r2, BBURx6
                                bc      NS, RFW_Receive3Words616_HuntError
                                bic     r2, r2, kRFWHardErrorBit
                                bc      VS, RFW_Receive3Words616_HardError
                                and     r2, r2, 0b00111111
                                rol     r2, r2, 4
                                ior     r0, r0, r2

                                // Receive the third 6 bits
                                inp     r2, BBURx6
                                bc      NS, RFW_Receive3Words616_HuntError
                                bic     r2, r2, kRFWHardErrorBit
                                bc      VS, RFW_Receive3Words616_HardError
                                and     r2, r2, 0b00111111
                                rol     r2, r2, -2
                                and     r3, r2, 0b1100000000000000
                                and     r2, r2, 0b0000000000000111
                                ior     r0, r0, r2
                                st      r0, r1, 0           // Store 1st word of data

                                // Receive the fourth 6 bits
                                inp     r2, BBURx6
                                bc      NS, RFW_Receive3Words616_HuntError
                                bic     r2, r2, kRFWHardErrorBit
                                bc      VS, RFW_Receive3Words616_HardError
                                and     r2, r2, 0b00111111
                                rol     r2, r2, 8
                                ior     r3, r3, r2

                                // Receive the fifth 6 bits
                                inp     r2, BBURx6
                                bc      NS, RFW_Receive3Words616_HuntError
                                bic     r2, r2, kRFWHardErrorBit
                                bc      VS, RFW_Receive3Words616_HardError
                                and     r2, r2, 0b00111111
                                rol     r2, r2, 2
                                ior     r3, r3, r2

```

```

        // Receive the sixth 6 bits
        inp    r2, BBURx6
        bc     NS, RFW_Receive3Words616_HuntError
        bic    r2, r2, kRFWHardErrorBit
        bc     VS, RFW_Receive3Words616_HardError
        and    r2, r2, 0b00111111
        rol    r2, r2, -4
        and    r0, r2, 0b0000000000000011
        ior    r3, r3, r0
        and    r0, r2, 0b1111000000000000
        st     r3, r1, 1    // Store 2nd word of data

        // Receive the seventh 6 bits
        inp    r2, BBURx6
        bc     NS, RFW_Receive3Words616_HuntError
        bic    r2, r2, kRFWHardErrorBit
        bc     VS, RFW_Receive3Words616_HardError
        and    r2, r2, 0b00111111
        rol    r2, r2, 6
        ior    r0, r0, r2

        // Receive the eighth 6 bits
        inp    r2, BBURx6
        bc     NS, RFW_Receive3Words616_HuntError
        bic    r2, r2, kRFWHardErrorBit
        bc     VS, RFW_Receive3Words616_HardError
        and    r2, r2, 0b00111111
        ior    r0, r0, r2
        st     r0, r1, 2    // Store 3rd word of data

        mov    r0, 0
        bra    RFW_Receive3Words616_END

RFW_Receive3Words616_HuntError:
        mov    r0, 1
        bra    RFW_Receive3Words616_END

RFW_Receive3Words616_HardError:
        mov    r0, 2

RFW_Receive3Words616_END:
        jsr    r6, r6

//=====
// Input Params:    r1 = Pointer to 3 Word Array to Store Received Data
// Output Params:   r0 = Error (0 = No Error, 1 = Hunt Error, 2 = Hard Error)
//                  r2 = Garbage
//                  r3 = Garbage
//-----
// Description:      Receives 3 words using the 616 Rate Table and places them
//                  into the array pointed to by r1. If the routine does not
//                  receives all three words successfully, it returns an error
//                  code in r0.
//
//                  There must not be more than 13 instruction times between
//                  successive calls to this routine when receiving a packet.
//=====
RFW_Receive3Bytes616:

        // Receive the first 6 bits
        inp    r2, BBURx6
        bc     NS, RFW_Receive3Bytes616_HuntError// Abort if no data detected
        bic    r2, r2, kRFWHardErrorBit
        bc     VS, RFW_Receive3Bytes616_HardError// Abort if hard error detected
        and    r2, r2, 0b00111111
        rol    r0, r2, 10

mov    r3, 0b00111111

        // Receive the second 6 bits
        inp    r2, BBURx6
        bc     NS, RFW_Receive3Bytes616_HuntError
        bic    r2, r2, kRFWHardErrorBit
        bc     VS, RFW_Receive3Bytes616_HardError
        and    r2, r2, r3
        rol    r2, r2, 4
        ior    r0, r0, r2

// extract first byte
        and    r2, r0, 0xFF00
        rol    r2, r2, 8

```

```

        st        r2, r1, 0

// pack remaining data in high bits
        rol      r0, r0, 8
        and      r0, r0, 0xF000

// Receive the third 6 bits
        inp      r2, BBURx6
        bc       NS, RFW_Receive3Bytes616_HuntError
        bic      r2, r2, kRFWHardErrorBit
        bc       VS, RFW_Receive3Bytes616_HardError
        rol      r2, r2, 6
        ior      r0, r0, r2

// extract second byte
        and      r2, r0, 0xFF00
        rol      r2, r2, 8
        st       r2, r1, 1
and      r0, r0, 0x00C0

mov      r3, 0x00FF

// Receive the third 6 bits
        inp      r2, BBURx6
        bc       NS, RFW_Receive3Bytes616_HuntError
        bic      r2, r2, kRFWHardErrorBit
        bc       VS, RFW_Receive3Bytes616_HardError
        ior      r0, r0, r2

// extract third byte
        and      r2, r0, r3
        st       r2, r1, 2

        mov      r0, 0
        bra      RFW_Receive3Bytes616_END

RFW_Receive3Bytes616_HuntError:
        mov      r0, 1
        bra      RFW_Receive3Bytes616_END

RFW_Receive3Bytes616_HardError:

        mov      r0, 2

RFW_Receive3Bytes616_END:
        jsr      r6, r6

//=====
// Input Params:      None
// Output Params:     r0 = (0 = No Error, 1 = Hunt Error, 2 = CRC Error)
//-----
// Description: Received a Data Frame and stores all received data in a Rx_Data_Frame
//=====
WiFi_Received_Data_Frame:

        mov      r5, 0
        mov      r0, 1
WiFi_Received_Data_Frame_LOOP:
        inp      r2, BBURx6

        and      r3, r2, 0b00111111
        st       r3, r0, a_Rx_Data_Frame

        bic      r2, r2, kRFWHuntBit
// Abort if no data detected
        bc       VS, WiFi_Received_Data_Frame_HuntError
        bic      r2, r2, kRFWHardErrorBit
// Abort if hard error detected
        bc       VS, WiFi_Received_Data_Frame_HardError

        add      r0, r0, 1

#ifdef TELEMETRY
        sub      r1, r0, 56
#endif
#ifdef AVIONICS
        sub      r1, r0, 402
#endif
        bc       NE, WiFi_Received_Data_Frame_LOOP

```

```

        bra    WiFi_Received_Data_Frame_END

//*****
// Errors received
        WiFi_Received_Data_Frame_HuntError:
            // r0 = 0 = NO Error
            // r0 = 1 = **Hunt Error**
            // r0 = 2 = **Hard Error**
            mov    r5, 1

            add    r0, r0, 1

#ifdef TELEMETRY
            sub    r1, r0, 56
#endif
#ifdef AVIONICS
            sub    r1, r0, 402
#endif
            bc     NE, WiFi_Received_Data_Frame_LOOP

        bra    WiFi_Received_Data_Frame_END

        WiFi_Received_Data_Frame_HardError:
            // r0 = 0 = NO Error
            // r0 = 1 = **Hunt Error**
            // r0 = 2 = **Hard Error**
            mov    r5, 2

            add    r0, r0, 1

#ifdef TELEMETRY
            sub    r1, r0, 56
#endif
#ifdef AVIONICS
            sub    r1, r0, 402
#endif
            bc     NE, WiFi_Received_Data_Frame_LOOP

        bra    WiFi_Received_Data_Frame_END
//*****
// Frame Received. Jump back from subroutine.
        WiFi_Received_Data_Frame_END:

            bic    r5, r5, 0 // If lsb is set, received a HUNT error
            bc     VS, WiFi_Received_Data_Frame_END_Hunt_Error
            // If bit 1 is set (with bit 0 being lsb), received HARD error
            bic    r5, r5, 1
            bc     VS, WiFi_Received_Data_Frame_END_Hard_Error
            // If neither bit 0 or bit 1 in r5 were set, then the frame was
            // received without errors
            // r0 = 0 = NO Error
            // r0 = 1 = **Hunt Error**
            // r0 = 2 = **Hard Error**
            mov    r0, 0

            jsr    r6, r6

        WiFi_Received_Data_Frame_END_Hunt_Error:
            // r0 = 0 = NO Error
            // r0 = 1 = **Hunt Error**
            // r0 = 2 = **Hard Error**
            mov    r0, 1

            jsr    r6, r6

        WiFi_Received_Data_Frame_END_Hard_Error:
            // r0 = 0 = NO Error
            // r0 = 1 = **Hunt Error**
            // r0 = 2 = **Hard Error**
            mov    r0, 2

            jsr    r6, r6

//=====
// Input Params:    None
// Output Params:   r0 = (0 = No Error, 1 = Hunt Error, 2 = CRC Error)
//-----
// Description: Received an ACK Frame and stores all received data in a Rx_ACK_Frame
//=====

```

```

WiFi_Received_ACK_Frame:

        mov     r5, 0
        mov     r0, 1
WiFi_Received_ACK_Frame_LOOP:
        inp     r2, BBURx6

        and     r3, r2, 0b00111111
        st      r3, r0, a_Rx_ACK_Frame

        bic     r2, r2, kRFWHuntBit
// Abort if no ACK detected
        bc     VS, WiFi_Received_ACK_Frame_HuntError
        bic     r2, r2, kRFWHardErrorBit
// Abort if hard error detected
        bc     VS, WiFi_Received_ACK_Frame_HardError

        add     r0, r0, 1

        sub     r1, r0, 6

        bc     NE, WiFi_Received_ACK_Frame_LOOP

        bra     WiFi_Received_ACK_Frame_END

//*****
// Errors received
        WiFi_Received_ACK_Frame_HuntError:
// r0 = 0 = NO Error
// r0 = 1 = **Hunt Error**
// r0 = 2 = **Hard Error**
        mov     r5, 1

        add     r0, r0, 1

        sub     r1, r0, 6
        bc     NE, WiFi_Received_ACK_Frame_LOOP

        bra     WiFi_Received_ACK_Frame_END

        WiFi_Received_ACK_Frame_HardError:
// r0 = 0 = NO Error
// r0 = 1 = **Hunt Error**
// r0 = 2 = **Hard Error**
        mov     r5, 2

        add     r0, r0, 1

        sub     r1, r0, 6
        bc     NE, WiFi_Received_ACK_Frame_LOOP

        bra     WiFi_Received_ACK_Frame_END
//*****
// Frame Received. Jump back from subroutine.
        WiFi_Received_ACK_Frame_END:

        bic     r5, r5, 0 // If lsb is set, received a HUNT error
        bc     VS, WiFi_Received_ACK_Frame_END_Hunt_Error
// If bit 1 is set (with bit 0 being lsb), received HARD error
        bic     r5, r5, 1
        bc     VS, WiFi_Received_ACK_Frame_END_Hard_Error
// If neither bit 0 or bit 1 in r5 were set,
// then the frame was received without errors

        // r0 = 0 = NO Error
        // r0 = 1 = **Hunt Error**
        // r0 = 2 = **Hard Error**
        mov     r0, 0

        jsr     r6, r6

        WiFi_Received_ACK_Frame_END_Hunt_Error:
// r0 = 0 = NO Error
// r0 = 1 = **Hunt Error**
// r0 = 2 = **Hard Error**
        mov     r0, 1

        jsr     r6, r6

        WiFi_Received_ACK_Frame_END_Hard_Error:
// r0 = 0 = NO Error

```

```

        // r0 = 1 = **Hunt Error**
        // r0 = 2 = **Hard Error**
        mov     r0, 2

        jsr     r6, r6

//=====
// Input Params:    r2
//                  r0 = scrambled
//                  r1 = scrambled
//                  r2 = 16chips to send
//                  r6 = return address
// Output Params:   none
//-----
// Description:     Takes "16 chips" and sends out 32 chips double speed to generate bit structure
//=====
RFW_Send16Chips:
        and     r0, r2, 0x000F
        ld      r0, r0, rxNibbleTable
        rol     r2, r2, -4
        and     r1, r2, 0x000F
        ld      r1, r1, rxNibbleTable
        rol     r1, r1, 8                //shift left
        ior     r0, r0, r1
        outp    r0, BBUTx

        rol     r2, r2, -4
        and     r0, r2, 0x000F
        ld      r0, r0, rxNibbleTable
        rol     r2, r2, -4
        and     r1, r2, 0x000F
        ld      r1, r1, rxNibbleTable
        rol     r1, r1, 8                //shift left
        ior     r0, r0, r1
        outp    r0, BBUTx

RFW_Send16Chips_END:
        jsr     r6, r6

```

C.12. *RFWaves_data.asm*

Short Data File used by Routines for using the RFW RF Module. It contains all data that is time critical for the RFW RF Module routines.

```
//*****
//***** (C) 2002 by Eleven Engineering Incorporated *****
//*****
/**
/**      Tabs:  This file looks best with tab stops set every 6 spaces.
/**
/**
//*****
//*****
/**      $RCSfile: RFWaves_data.asm,v $
/**      $Revision: 1.1 $
/**      Tag $Name:  $
/**      $Date: 2003/07/24 00:01:35 $
/**      $Author: eleven $
/**
/**      Project: XInC Library
/**      Description: Short Data File used by Routines for using the RFW RF Module.
/**
//*****
//*****

// LookupTable that could go in short address space.
// It just does the following transformation:
//      0 -> 00
//      1 -> 01
//
// Data is sent out of the BBU lsb first.
// A 2^8 table could be used if time is a problem.

rxNibbleTable:
    0b00000000
    0b00000001
    0b00000100
    0b00000101
    0b00010000
    0b00010001
    0b00010100
    0b00010101
    0b01000000
    0b01000001
    0b01000100
    0b01000101
    0b01010000
    0b01010001
    0b01010100
    0b01010101
```

C.13. Short_Data.asm

Contains any data (memory variables and tables) that must be accessible with a single word instruction.

```
//*****
//***** (C) 2002 by Eleven Engineering Incorporated *****
//*****
/**
**      Tabs:  This file looks best with tab stops set every 6 spaces.
**
**
//*****
//*****
/**
** File:      Short_Data.asm
** Project:  XInC Dev Kit - Sample Empty Project
** Created:  25 Jun 2002 by Ryan Northcott
** Revised:  25 Jun 2002 by Ryan Northcott
**
/**
** Description:  Contains any data (memory variables and tables) that must be
**               accessible with a single word instruction.  There is only
**               enough room for 127 words of data in the Short Address space.
**               All other data should be stored in the "Long_Data.asm" file
**               to ensure that it is stored in a separate 2kWord memory block
**               from all code.
**
/**
** Disclaimer:  This code was descended from Eleven Engineering sample
**               source code, but changes were made by Capt Joshua D. Green
**
//*****
//*****

rxBVTable:
    0x001F
    0x003F
    0x007F
    0x00FF
    0x01FF
    0x03FF

#ifdef THROUGHPUT_2_STATIONS
    #ifdef STATION_2
        rxDA_Station_Number:
            Station_03 // 0
            Station_03 // 1
            Station_03 // 2
            Station_03 // 3
    #endif

    #ifdef STATION_3
        rxDA_Station_Number:
            Station_02 // 0
            Station_02 // 1
            Station_02 // 2
            Station_02 // 3
    #endif
#endif

#ifdef THROUGHPUT_3_STATIONS
    #ifdef STATION_1
        rxDA_Station_Number:
            Station_02 // 0
            Station_02 // 1
            Station_03 // 2
            Station_03 // 3
    #endif

    #ifdef STATION_2
        rxDA_Station_Number:
            Station_01 // 0
            Station_01 // 1
            Station_03 // 2
```

```

        Station_03 // 3
    #endif

    #ifdef STATION_3
        rxDA_Station_Number:
            Station_01 // 0
            Station_01 // 1
            Station_02 // 2
            Station_02 // 3
    #endif
#endif

#ifdef THROUGHPUT_4_STATIONS
    #ifdef STATION_1
        rxDA_Station_Number:
            Station_02 // 0 - Will never be used
            Station_02 // 1
            Station_03 // 2
            Station_04 // 3
    #endif

    #ifdef STATION_2
        rxDA_Station_Number:
            Station_01 // 0 - Will never be used
            Station_01 // 1
            Station_03 // 2
            Station_04 // 3
    #endif

    #ifdef STATION_3
        rxDA_Station_Number:
            Station_01 // 0 - Will never be used
            Station_01 // 1
            Station_02 // 2
            Station_04 // 3
    #endif

    #ifdef STATION_4
        rxDA_Station_Number:
            Station_01 // 0 - Will never be used
            Station_01 // 1
            Station_02 // 2
            Station_03 // 3
    #endif
#endif
#endif

```

C.14. Thread0.asm

The main thread running the IEEE 802.11 protocol. It also handled all packet transmission.

```
//*****
//***** (C) 2002 by Eleven Engineering Incorporated *****
//*****
/**
**      Tabs: This file looks best with tab stops set every 6 spaces.
**
**
//*****
//*****
/**
** File:      Thread0.asm
**
/**
/** Project: IEEE 802.11 MAC emulator. It can send to multiple (1-4) stations
/** Created: 1 June 2004 by Capt Joshua D. Green
/**
/** Description: Code that is run by Thread 0. IEEE 802.11 MAC layer Protocol.
/**
/** Disclaimer: This code was descended from Eleven Engineering sample
/**              source code, but changes were made by Capt Joshua D. Green
/**
//*****
//*****

_T0_Initialization:

    // Initialize the Tokaido Radio
    jsr    r6, RFW_Initialize

    // Initialize LEDs
    jsr    r6, InitializeLEDs

    // Turn Off Thread2
    mov    r0, kStop_Thread_2
    outp    r0, SCUstop

    // Initialize the Data and ACK frames
    jsr    r6, Initialize_Data_Frame
    jsr    r6, Initialize_ACK_Frame

    // Initialize some variables
    mov    r0, 0
    st     r0, a_Rx_Sequence_Numbers + 0
    st     r0, a_Rx_Sequence_Numbers + 1
    st     r0, a_Rx_Sequence_Numbers + 2
    st     r0, a_Rx_Sequence_Numbers + 3
    st     r0, v_Medium_Idle_Flag
    st     r0, v_Delay_Time
    sub    r0, r0, 1
    st     r0, v_Thread_0_packet_que_number

    mov    r0, 1
    st     r0, v_Number_of_Retransmissions
    st     r0, a_Tx_Sequence_Numbers + 0
    st     r0, a_Tx_Sequence_Numbers + 1
    st     r0, a_Tx_Sequence_Numbers + 2
    st     r0, a_Tx_Sequence_Numbers + 3

    // Put Tokaido into receive mode
    jsr    r6, RFW_EnterReceiveMode

//*****
//*****
WiFi_Main_Loop:
    // Check to see if waiting for an ACK (is kACK_SEMAPHORE HIGH?)
    // If YES, jump to Waiting_ACK_Timeout.
    inp     r1, SCUsrc
    bis     r1, r1, kACK_SEMAPHORE
    bc      VS, Waiting_ACK_Timeout
```

```

        // Start Thread 2
        jsr    r6, Start_RX_Thread

Wait_for_Transmission_Request:
// Loop while waiting for:
// A packet from Application layer (indicated by v_Packets_in_Queue being > 0)
// --or--
// Thread 2 to detect a transmission (indicated by kReceived_TX_SEMAPHORE going HIGH)
        mov    r0, 1<<kPackets_in_Queue_SEMAPHORE
        outp   r0, SCUdown
        ld     r1, v_Packets_in_Queue
        outp   r0, SCUup
        bc     ZC, Transmit_Frame_HIGH

        inp    r1, SCUsrc
        bis    r1, r1, kReceived_TX_SEMAPHORE
        bc     VS, Receive_Frame_with_Transmit_Frame_LOW

        bra    Wait_for_Transmission_Request

Transmit_Frame_HIGH:
// Begin frame transmission process

Transmit_Frame_HIGH_CALC:
// Dummy command to get the timing just right
        mov    r5, 0

        // Calculating DIFS Period
        inp    r5, SCUtime
        add    r5, r5, (kDIFSTime - 300)
        st     r5, v_Delay_Time

Transmit_Frame_DIFS_LOOP:
        ld     r5, v_Delay_Time

Transmit_Frame_DIFS_LOOP_2:
// If kReceived_TX_SEMAPHORE goes HIGH, jump out to a separate loop
        inp    r1, SCUsrc
        bis    r1, r1, kReceived_TX_SEMAPHORE
        bc     VS, Receive_Frame_in_Transmit_DIFS_window

        inp    r1, SCUtime
        sub    r1, r1, r5
        bc     LT0, Transmit_Frame_DIFS_LOOP_2

        // Check to see if any frame was received during the DIFS Period.
        // Basically, was the channel clear during the entire DIFS period?
        // If YES, then wait a Random Backoff Time interval and transmit.
        // If NO, then just transmit without waiting a Random Backoff Time interval.
        // Note: v_Medium_Idle_Flag is 1 or greater = YES,
        //       v_Medium_Idle_Flag is zero = NO

        ld     r1, v_Medium_Idle_Flag
        sub    r1, r1, 0
        bc     EQ, Transmit_Frame_DONE

Transmit_Frame_BV_START:
// Calculating Slot Time
        inp    r5, SCUtime
        add    r5, r5, (kSlotTime - 100)
        st     r5, v_Delay_Time

Transmit_Frame_BV_LOOP:
// Calculating BV Period
        ld     r4, v_BV_Slots
        ld     r5, v_Delay_Time

Transmit_Frame_BV_LOOP_2:
// If kReceived_TX_SEMAPHORE goes HIGH, jump out to a separate loop
        inp    r1, SCUsrc
        bis    r1, r1, kReceived_TX_SEMAPHORE
        bc     VS, Receive_Frame_in_Transmit_BV_window

        inp    r1, SCUtime
        sub    r2, r1, r5
        bc     LT0, Transmit_Frame_BV_LOOP_2

        // Dummy Load to get timing right
        mov    r0, 0
        mov    r0, 0

```

```

        mov     r0, 0
        mov     r0, 0

        // Decrement by one v_BV_Slots
        sub     r4, r4, 1
        st      r4, v_BV_Slots

        // If v_BV_Slots does NOT equal zero, loop back
        // If v_BV_Slots IS equal to zero, re-transmit frame
        bc      ZC, Transmit_Frame_BV_START

Transmit_Frame_DONE:

        bra     Transmit_Frame

//*****
//*****
// Waiting for ACK Routines
Waiting_ACK_Timeout:

        // Start Thread 2
        jsr     r6, Start_RX_Thread

        // Loop listening to the channel for an ACK for the last frame sent.
        // If don't receive one before the ACK Timeout period, retransmitt the frame
        inp     r5, SCUtime
        add     r5, r5, (kACK_Timeout - kDIFSTime_Adjustment)
        st      r5, v_Delay_Time

Waiting_ACK_Timeout_LOOP:
        ld      r5, v_Delay_Time

Waiting_ACK_Timeout_LOOP_2:
        // Loop while kReceived_TX_SEMAPHORE is LOW or still in DIFS window
        inp     r1, SCUtime
        bis     r1, r1, kReceived_TX_SEMAPHORE
        bc      VS, Receive_Frame_waiting_for_ACK

        inp     r1, SCUtime
        sub     r1, r1, r5
        bc      LT0, Waiting_ACK_Timeout_LOOP_2

DONE_Waiting_ACK_Timeout:
// Retransmitting frame.

#ifdef DEBUG_LEDS // Dummy loads for timing purposes
        mov     r0, 0xFFFF
        mov     r0, 0xFFFF
        mov     r0, 0xFFFF
#endif

        // Check to see if any frame was received during the ACK Timeout Period.
        // Basically, was the channel clear during the entire ACK Timeout period?
        // If NO, wait a DIFS, then a Random Backoff Time interval, then retransmit.
        // If YES, then wait a Random Backoff Time interval and then retransmit
        // Note: v_Medium_Idle_Flag HIGH (true) = YES,
        //       v_Medium_Idle_Flag LOW (false) = NO
        ld      r1, v_Medium_Idle_Flag
        sub     r1, r1, 0
        bc      EQ, ACKTimeout_Done_BV_START

ACKTimeout_Done_DIFS_CALC:
        // Dummy command to get the timing just right
        mov     r5, 0

        // Calculating DISF Period
        inp     r5, SCUtime
        add     r5, r5, (kDIFSTime - 300)
        st      r5, v_Delay_Time

ACKTimeout_Done_DIFS_LOOP:
        ld      r5, v_Delay_Time

ACKTimeout_Done_DIFS_LOOP_2:
        // If kReceived_TX_SEMAPHORE goes HIGH, jump out to a separate loop
        inp     r1, SCUtime
        bis     r1, r1, kReceived_TX_SEMAPHORE
        bc      VS, Receive_Frame_in_ACK_DIFS_window

        inp     r1, SCUtime
        sub     r1, r1, r5

```

```

        bc      LT0, ACKTimeout_Done_DIFS_LOOP_2

ACKTimeout_Done_BV_START:
    // Calculating Slot Time
    inp    r5, SCUtime
    add    r5, r5, (kSlotTime - 100)
    st     r5, v_Delay_Time

ACKTimeout_Done_BV_LOOP:
    // Calculating BV Period
    ld     r4, v_BV_Slots
    ld     r5, v_Delay_Time

ACKTimeout_Done_BV_LOOP_2:
    // If kReceived_TX_SEMAPHORE goes HIGH, jump out to a separate loop
    inp    r1, SCUsrc
    bis    r1, r1, kReceived_TX_SEMAPHORE
    bc     VS, Receive_Frame_in_ACK_BV_window

    inp    r1, SCUtime
    sub    r2, r1, r5
    bc     LT0, ACKTimeout_Done_BV_LOOP_2

    // Dummy Load to get timing right
    mov    r0, 0
    mov    r0, 0
    mov    r0, 0
    mov    r0, 0

    // Decrement by one v_BV_Slots
    sub    r4, r4, 1
    st     r4, v_BV_Slots

    // If v_BV_Slots does NOT equal zero, loop back
    // If v_BV_Slots IS equal to zero, re-transmit frame
    bc     ZC, ACKTimeout_Done_BV_START

ACKTimeout_Done_BV_DONE:
    bra     Retransmit_Frame

//*****
//*****
// Received a Frame commands
Receive_Frame_with_Transmit_Frame_LOW:
    // --WAIT-- till Thread 2 stops processing the Received Frame
    inp    r1, SCUsrc
    bis    r1, r1, kReceived_TX_SEMAPHORE
    bc     VS, Receive_Frame_with_Transmit_Frame_LOW

    // v_Received_Stuff_FLAG = 0 = Received JUNK
    // v_Received_Stuff_FLAG = 1 = Received DATA packet
    // v_Received_Stuff_FLAG = 2 = Received ACK
    // v_Received_Stuff_FLAG = 3 = Received frame NOT for this station
    ld     r1, v_Received_Stuff_FLAG
    sub    r1, r1, 1

    // Wait_for_Transmission_Request // r1 = -1 = Received JUNK
    bc     NS, EIFS_Period
    bc     ZS, Send_ACK // r1 = 0 = Received DATA packet

    // r1 = 1 or 2 = Received ACK or frame NOT for this station
    bra     WiFi_Main_Loop

Receive_Frame_in_Transmit_DIFS_window:
    // --WAIT-- till Thread 2 stops processing the Received Frame
    inp    r1, SCUsrc
    bis    r1, r1, kReceived_TX_SEMAPHORE
    bc     VS, Receive_Frame_in_Transmit_DIFS_window

    // v_Received_Stuff_FLAG = 0 = Received JUNK
    // v_Received_Stuff_FLAG = 1 = Received DATA packet
    // v_Received_Stuff_FLAG = 2 = Received ACK
    // v_Received_Stuff_FLAG = 3 = Received frame NOT for this station
    ld     r1, v_Received_Stuff_FLAG
    sub    r1, r1, 1

    // Transmit_Frame_DIFS_LOOP // r1 = -1 = Received JUNK
    bc     NS, EIFS_Period
    bc     ZS, Send_ACK // r1 = 0 = Received DATA packet

```

```

        sub    r1, r1, 1
// r1 = 0 = Received ACK packet (should NOT happen - so reset)
        bc     ZS, WiFi_Main_Loop

// r1 = 1 = frame NOT for this station
// increment v_Medium_Idle_Flag by 1
        ld     r0, v_Medium_Idle_Flag
        add    r0, r0, 1
        st     r0, v_Medium_Idle_Flag

// Create a new BV
        mov    r1, 1<<kCreate_RN_BV_SEMAPHORE
        outp   r1, SCUdown
        outp   r1, SCUdown

        bra    Transmit_Frame_HIGH_CALC

Receive_Frame_in_Transmit_BV_window:
// --WAIT-- till Thread 6 stops processing the Received Frame
        inp    r1, SCUsrc
        bis    r1, r1, kReceived_TX_SEMAPHORE
        bc     VS, Receive_Frame_in_Transmit_BV_window

// v_Received_Stuff_FLAG = 0 = Received JUNK
// v_Received_Stuff_FLAG = 1 = Received DATA packet
// v_Received_Stuff_FLAG = 2 = Received ACK
// v_Received_Stuff_FLAG = 3 = Received frame NOT for this station
        ld     r1, v_Received_Stuff_FLAG
        sub    r1, r1, 1
// Transmit_Frame_BV_LOOP // r1 = -1 = Received JUNK
        bc     NS, EIFS_Period
        bc     ZS, Send_ACK // r1 = 0 = Received DATA packet

        sub    r1, r1, 1
// r1 = 0 = Received ACK packet (should NOT happen - so reset)
        bc     ZS, WiFi_Main_Loop

// r1 = 1 = frame NOT for this station
// increment v_Medium_Idle_Flag by 1
        ld     r0, v_Medium_Idle_Flag
        add    r0, r0, 1
        st     r0, v_Medium_Idle_Flag

// Create a new BV
        mov    r1, 1<<kCreate_RN_BV_SEMAPHORE
        outp   r1, SCUdown
        outp   r1, SCUdown

        bra    Transmit_Frame_HIGH_CALC

Receive_Frame_waiting_for_ACK:
// --WAIT-- till Thread 6 stops processing the Received Frame
        inp    r1, SCUsrc
        bis    r1, r1, kReceived_TX_SEMAPHORE
        bc     VS, Receive_Frame_waiting_for_ACK

// v_Received_Stuff_FLAG = 0 = Received JUNK
// v_Received_Stuff_FLAG = 1 = Received DATA packet
// v_Received_Stuff_FLAG = 2 = Received ACK
// v_Received_Stuff_FLAG = 3 = Received frame NOT for this station
        ld     r1, v_Received_Stuff_FLAG
        sub    r1, r1, 1
// r1 = -1 = Received JUNK
        bc     NS, EIFS_Period // Waiting_ACK_Timeout_LOOP
// r1 = 0 = Received DATA packet (should NOT happen)
        bc     ZS, Send_ACK

        sub    r1, r1, 1
        bc     ZS, WiFi_Main_Loop // r1 = 0 = Received ACK packet

// r1 = 1 = frame NOT for this station
// increment v_Medium_Idle_Flag by 1
        ld     r0, v_Medium_Idle_Flag
        add    r0, r0, 1
        st     r0, v_Medium_Idle_Flag

// Create a new BV
        mov    r1, 1<<kCreate_RN_BV_SEMAPHORE
        outp   r1, SCUdown

```

```

        outp    r1, SCUdown

        bra     Waiting_ACK_Timeout_LOOP

Receive_Frame_in_ACK_DIFS_window:
    // --WAIT-- till Thread 6 stops processing the Received Frame
        inp     r1, SCUsrc
        bis     r1, r1, kReceived_TX_SEMAPHORE
        bc      VS, Receive_Frame_in_ACK_DIFS_window

    // v_Received_Stuff_FLAG = 0 = Received JUNK
    // v_Received_Stuff_FLAG = 1 = Received DATA packet
    // v_Received_Stuff_FLAG = 2 = Received ACK
    // v_Received_Stuff_FLAG = 3 = Received frame NOT for this station
        ld      r1, v_Received_Stuff_FLAG
        sub     r1, r1, 1
        // ACKTimeout_Done_DIFS_LOOP // r1 = -1 = Received JUNK
        bc      NS, EIFS_Period
        bc      ZS, Send_ACK // r1 = 0 = Received DATA packet

        sub     r1, r1, 1
        bc      ZS, WiFi_Main_Loop // r1 = 0 = Received ACK

    // r1 = 1 = frame NOT for this station
    // increment v_Medium_Idle_Flag by 1
        ld      r0, v_Medium_Idle_Flag
        add     r0, r0, 1
        st      r0, v_Medium_Idle_Flag

    // Create a new BV
        mov     r1, 1<<kCreate_RN_BV_SEMAPHORE
        outp    r1, SCUdown
        outp    r1, SCUdown

    // r1 = 1 = Received frame NOT for this station
        bra     ACKTimeout_Done_DIFS_CALC

Receive_Frame_in_ACK_BV_window:
    // --WAIT-- till Thread 6 stops processing the Received Frame
        inp     r1, SCUsrc
        bis     r1, r1, kReceived_TX_SEMAPHORE
        bc      VS, Receive_Frame_in_ACK_BV_window

    // v_Received_Stuff_FLAG = 0 = Received JUNK
    // v_Received_Stuff_FLAG = 1 = Received DATA packet
    // v_Received_Stuff_FLAG = 2 = Received ACK
    // v_Received_Stuff_FLAG = 3 = Received frame NOT for this station
        ld      r1, v_Received_Stuff_FLAG
        sub     r1, r1, 1
        // ACKTimeout_Done_BV_LOOP // r1 = -1 = Received JUNK
        bc      NS, EIFS_Period
        bc      ZS, Send_ACK // r1 = 0 = Received DATA packet

        sub     r1, r1, 1
        bc      ZS, WiFi_Main_Loop // r1 = 0 = Received ACK

    // r1 = 1 = frame NOT for this station
    // increment v_Medium_Idle_Flag by 1
        ld      r0, v_Medium_Idle_Flag
        add     r0, r0, 1
        st      r0, v_Medium_Idle_Flag

    // Create a new BV
        mov     r1, 1<<kCreate_RN_BV_SEMAPHORE
        outp    r1, SCUdown
        outp    r1, SCUdown

        bra     ACKTimeout_Done_DIFS_CALC

//*****
//*****
Transmit_Frame:

Calculate_BV:
    // Starts process to generate another RN BV
        mov     r1, 1<<kCreate_RN_BV_SEMAPHORE
        outp    r1, SCUdown

```

```

Initialize_for_Send_Packet:
    // Reset the Maximum Number of Retransmissions
    mov    r1, 1
    st     r1, v_Number_of_Retransmissions

    // Increment v_Theard_0_packet_que_number
    ld     r5, v_Thread_0_packet_que_number
    add    r5, r5, 1
    and    r5, r5, (kTransmitter_Buffer_Size - 1) // Creates proper mask for
Buffer Size
    st     r5, v_Thread_0_packet_que_number

    // load into sending array the sequance number for the packet
    mov    r0, 1<<kTx_Data_Address_1_SEMAPHORE
    outp   r0, SCUdown
    ld     r1, r5, a_Tx_Data_Address_1 // Destination Address
    outp   r0, SCUup

    // Now r0 = 0 if sending station is Station 1,
    // r0 = 1 if sending station is Station 2, ect.
    sub    r1, r1, Station_01

    // Will load:
    // a_Tx_Sequence_Numbers + 0 for Station 1
    // a_Tx_Sequence_Numbers + 1 for Station 2
    // etc.
    ld     r0, r1, a_Tx_Sequence_Numbers
    // Stores sequance number in the transmitting array
    st     r0, v_Tx_Data_Sequence_Number

    // If recording started, increment v_Number_of_Failed_TX
    inp    r0, SCUsrc
    bis    r0, r0, kStart_Stop_SEMAPHORE
    bc     VC, transmit_SendPacket_Start

    ld     r0, v_Number_of_TX
    add    r0, r0, 1
    st     r0, v_Number_of_TX

transmit_SendPacket_Start:
// LED Toggle
#ifdef DEBUG_LEDS
    mov    r1, 0 // Dummy load for timing purposes
    mov    r1, 0x0080
    jsr    r6, ToggleLEDS
#endif

    // Start transmitting procedure
    mov    r0, 1<<kPacket_Start_Time_SEMAPHORE
    outp   r0, SCUdown
    ld     r4, v_PacketStartTime
    outp   r0, SCUup

transmit_SendPacket_wait:
    // Wait until the right time to send
    inp    r0, SCUtime
    sub    r0, r0, r4
    bc     LT0, transmit_SendPacket_wait

    jsr    r6, RFW_EnterTransmitMode // Calibrate the transmitter

transmit_SendPacket:
    // Send the packet preamble
    jsr    r6, RFW_SendPacketPreamble

    // Send the packet
    jsr    r6, WiFi_Send_Data_Packet

    // Send the packet postamble
    jsr    r6, RFW_SendPacketPostamble

    // Calibrate the receiver
    jsr    r6, RFW_EnterReceiveMode

transmit_Set_Flags:
    // Set kACK_SEMAPHORE HIGH
    mov    r0, 1<<kACK_SEMAPHORE
    outp   r0, SCUdown

```

```

        // Prepare to send the next packet
        bra    Waiting_ACK_Timeout

//*****
//*****
Retransmit_Frame:

    Retransmit_Calculate_BV:
        // Starts process to generate another RN BV
        mov    r1, 1<<kCreate_RN_BV_SEMAPHORE
        outp   r1, SCUdown

    Retransmit_Initialize_for_Send_Packet:
        // If reached maximum number of retransmissions, give it up and go back to the main loop
        ld     r1, v_Number_of_Retransmissions
        sub    r4, r1, kMaxReTransmit
        bc     EQ, Retransmit_Give_Up

        // If max retransmissions not reached, increment the counter and store
        add    r1, r1, 1
        st     r1, v_Number_of_Retransmissions

        // If recording started, increment v_Number_of_Failed_TX
        inp    r0, SCUsrc
        bis    r0, r0, kStart_Stop_SEMAPHORE
        bc     VC, Retransmit_SendPacket_Start

        ld     r0, r1, a_Recorded_TX
        add    r0, r0, 1
        st     r0, r1, a_Recorded_TX

    Retransmit_SendPacket_Start:
// LED Toggle
#ifdef DEBUG_LEDs
        mov    r1, 0x0040
        jsr    r6, ToggleLEds
#endif

        mov    r0, 1<<kPacket_Start_Time_SEMAPHORE
        outp   r0, SCUdown
        ld     r4, v_PacketStartTime
        outp   r0, SCUup

    Retransmit_SendPacket_wait:
        // Wait until the right time to send
        inp    r0, SCUtime
        sub    r0, r0, r4
        bc     LT0, Retransmit_SendPacket_wait

        jsr    r6, RFW_EnterTransmitMode    // Calibrate the transmitter

    Retransmit_SendPacket:
        // Send the packet preamble
        jsr    r6, RFW_SendPacketPreamble

        // Send the packet
        jsr    r6, WiFi_Send_Data_Packet

        // Send the packet postamble
        jsr    r6, RFW_SendPacketPostamble

        // Calibrate the receiver
        jsr    r6, RFW_EnterReceiveMode

    Retransmit_SendPacket_Done:
        // Prepare to send the next packet
        bra    Waiting_ACK_Timeout

    Retransmit_Give_Up:

        // If recording started, increment v_Number_of_Failed_TX
        inp    r0, SCUsrc
        bis    r0, r0, kStart_Stop_SEMAPHORE
        bc     VC, Retransmit_Give_Up_ACK_HIGH

        ld     r0, v_Number_of_Failed_TX
        add    r0, r0, 1
        st     r0, v_Number_of_Failed_TX

```

```

Retransmit_Give_Up_ACK_HIGH:
    // Increment Sequence_Number
    ld    r5, v_Thread_0_packet_que_number

    mov    r0, 1<<kTx_Data_Address_1_SEMAPHORE
    outp   r0, SCUdown
    ld     r1, r5, a_Tx_Data_Address_1 // Destination Address
    outp   r0, SCUup
    // Now r0 = 0 if sending station is Station 1,
    // r0 = 1 if sending station is Station 2, ect.
    sub    r1, r1, Station_01

    // Will load:
    // a_Tx_Sequence_Numbers + 0 for Station 1
    // a_Tx_Sequence_Numbers + 1 for Station 2
    // etc.
    ld     r0, r1, a_Tx_Sequence_Numbers

    // Increment Sequence_Number for that particular station
    add    r0, r0, 1
    // Stores sequence number in the Sequence_Numbers array
    st     r0, r1, a_Tx_Sequence_Numbers

//////////
#ifdef DEBUG_LEDs
    mov    r2, 1
    rol    r1, r2, r1
    jsr    r6, ToggleLEds
#endif
//////////

Retransmit_Give_Up_Done:
    // Decrement v_Packets_in_Queue by 1
    mov    r0, 1<<kPackets_in_Queue_SEMAPHORE
    outp   r0, SCUdown
    ld     r1, v_Packets_in_Queue
    sub    r1, r1, 1
    bc     NC, Retransmit_Packets_in_Queue_OK
    // If Negative is set, something is broken and must reset v_Packets_in_Queue to zero
    mov    r1, 0
Retransmit_Packets_in_Queue_OK:
    st     r1, v_Packets_in_Queue
    outp   r0, SCUup

    // If recording started, increment v_Number_of_Failed_TX
    inp    r0, SCUsrc

    mov    r1, 1<<kACK_SEMAPHORE
    mov    r3, 1<<kFailed_TX_SEMAPHORE

    // Reset ACK Flag
    outp   r1, SCUup

    bis    r0, r0, kStart_Stop_SEMAPHORE
    bc     VC, Retransmit_Give_Up_ACK_HIGH_2
    outp   r3, SCUdown

Retransmit_Give_Up_ACK_HIGH_2:

    bra    WiFi_Main_Loop

//*****
//*****
Send_ACK:
    mov    r1, (kSIFSTime - kSIFSTime_Adjustment)
    SIFS_Delay_for_ACK:
        // wait one SIFS period before transmitting ACK
        sub    r1, r1, 1
        bc     ZC, SIFS_Delay_for_ACK

// LED Toggle
#ifdef DEBUG_LEDs
    mov    r1, 0x0020
    jsr    r6, ToggleLEds
#else
    // Delay to compensate for not using LEDs
    mov    r1, 9
    jsr    r6, Delay

```

```

#endif

        mov     r0, 1<<kPacket_Start_Time_SEMAPHORE
        outp    r0, SCUdown
        ld      r4, v_PacketStartTime
        outp    r0, SCUup

Send_ACK_SendPacket_wait:
        // Wait until the right time to send
        inp     r0, SCUtime
        sub     r0, r0, r4
        bc      LT0, Send_ACK_SendPacket_wait

        jsr     r6, RFW_EnterTransmitMode           // Calibrate the transmitter

Send_ACK_SendPacket:
        // Send the packet preamble
        jsr     r6, RFW_SendPacketPreamble

        // Send the packet.
        // r0 = Address 2 - Sending Station Address (last 6 bits of the address only)
        ld      r0, a_Rx_Data_Address_2 + 2
        jsr     r6, WiFi_Send_ACK_Packet

        // Send the packet postamble
        jsr     r6, RFW_SendPacketPostamble

        // Calibrate the receiver
        jsr     r6, RFW_EnterReceiveMode

        // If recording started, increment v_Number_of_ACKs_Sent
        inp     r0, SCUsrc
        bis     r0, r0, kStart_Stop_SEMAPHORE
        bc      VC, Send_ACK_SendPacket_Done

        ld      r0, v_Number_of_ACKs_Sent
        add     r0, r0, 1
        st      r0, v_Number_of_ACKs_Sent

Send_ACK_SendPacket_Done:
        bra     WiFi_Main_Loop

Start_RX_Thread:
//=====
// Input Params:    none
// Output Params:   r1 = Junk
//                  r2 = Junk
//                  r3 = Junk
//-----
// Description:      Use these commands to turn a thread back on and make it start
//                  at a specified place. In this case, the Thread being turned
//                  back on is Thread 2, and the routine Thread 2 will start on
//                  is "BBUrx6_SEMAPHORE_LOW_LOOP"
//=====
        // Turn Off Thread2
        mov     r0, kStop_Thread_2
        outp    r0, SCUstop

        // Reset v_Received_Stuff to zero
        mov     r1, 0
        st      r1, v_Received_Stuff

        // Reset the SEMAPHORE kReceived_TX_SEMAPHORE to zero
        mov     r1, 1<<kReceived_TX_SEMAPHORE
        outp    r1, SCUup

        mov     r3, 0b010000 // Prepping to set SCU Pointer so when
                                // turn Thread 2 back on, the thread will
                                // start on the command of our choosing.
                                // See XInC User Guide, p. 34, for what
                                // 0b010000 means
        outp    r3, SCUpntr

        mov     r2, BBUrx6_SEMAPHORE_LOW_LOOP
        // Routine in Thread 2 to start with
        // Set memory location were Thread 2 will start from
        outp    r2, SCUreg
        // Start Thread 2

```

```

        mov    r1, kStart_Thread_2
        outp   r1, SCUstop

        jsr    r6, r6

//*****
//*****
EIFS_Period:
    // Extended Interframe Space Time (EIFS) = 1068 µsec or 53400 SCUtime cycles
    // This function puts the Station in Rx only mode for an EIFS period

        mov    r0, 1349

EIFS_Period_Loop:
        inp    r1, SCUsrc
        bis    r1, r1, kReceived_TX_SEMAPHORE
        bc     VS, Receive_Frame_in{EIFS

        sub    r0, r0, 1
        bc     ZC, EIFS_Period_Loop

        bra    WiFi_Main_Loop

Receive_Frame_in{EIFS:
    // --WAIT-- till Thread 2 stops processing the Received Frame
        inp    r1, SCUsrc
        bis    r1, r1, kReceived_TX_SEMAPHORE
        bc     VS, Receive_Frame_in{EIFS

    // v_Received_Stuff_FLAG = 0 = Received JUNK
    // v_Received_Stuff_FLAG = 1 = Received DATA packet
    // v_Received_Stuff_FLAG = 2 = Received ACK
    // v_Received_Stuff_FLAG = 3 = Received frame NOT for this station
        ld     r1, v_Received_Stuff_FLAG
        sub    r1, r1, 1
        // Wait_for_Transmission_Request // r1 = -1 = Received JUNK
        bc     NS, EIFS_Period
        bc     ZS, Send_ACK // r1 = 0 = Received DATA packet

        // r1 = 1 or 2 = Received ACK or frame NOT for this station
        bra    WiFi_Main_Loop

        sub    r1, r1, 1
        bc     ZS, WiFi_Main_Loop // r1 = 0 = Received ACK

    // r1 = 1 = frame NOT for this station
    // increment v_Medium_Idle_Flag by 1
        ld     r2, v_Medium_Idle_Flag
        add    r2, r2, 1
        st     r2, v_Medium_Idle_Flag

    // Create a new BV
        mov    r1, 1<<kCreate_RN_BV_SEMAPHORE
        outp   r1, SCUdown
        outp   r1, SCUdown

        sub    r0, r0, 5

        // r1 = 1 or 2 = Received ACK or frame NOT for this station
        bra    WiFi_Main_Loop

```

C.15. Thread1.asm

Polling thread. This thread runs a clock that tells Thread 0 when it can transmit a packet.

Thread 1 creates the slotted the channel in accordance with IEEE 802.11.

```
//*****
//***** (C) 2002 by Eleven Engineering Incorporated *****
//*****
/**
**      Tabs: This file looks best with tab stops set every 6 spaces.
**
**
//*****
//*****
/**
** File:      Thread1.asm
**
/**
/** Project: IEEE 802.11 MAC emulator. It can send to multiple (1-4) stations
/** Created: 1 June 2004 by Capt Joshua D. Green
/**
/** Description: Code that is run by Thread 1. Polling thread. This thread runs
/**              a clock that tells Thread 0 when it can transmit a packet.
/**              Thread 1 creates the slotted the channel in accordance with
/**              IEEE 802.11.
/**
/** Disclaimer: This code was descended from Eleven Engineering sample
/**              source code, but changes were made by Capt Joshua D. Green
/**
//*****
//*****

_T1_Initialization:

                                inp    r1, SCUtime
                                add     r1, r1, (kSlotTime - 60)

Poling_Main_Loop:

    Determine_Start_Time:
        // Last, determine if it is time to advance the variable v_PacketStartTime
        // If not, loop back to Determine_Start_Time
                                inp     r2, SCUtime
                                sub      r3, r2, r1
                                bc       LT0, Determine_Start_Time

                                add      r5, r2, kSlotTime

                                inp      r1, SCUtime
                                add      r1, r1, (kSlotTime - 60)

    Set_Packet_Start_Time_END:
                                mov      r0, 1<<kPacket_Start_Time_SEMAPHORE
                                outp     r0, SCUdown
                                st       r5, v_PacketStartTime
                                outp     r0, SCUup

                                bra      Poling_Main_Loop
```

C.16. Thread2

Receiver thread. Receives all packets transmitted on the medium, determines if the packets are for the node, in the proper order, and without errors. It also communicates to Thread 0 whenever the medium is sensed busy.

```
//*****
//***** (C) 2002 by Eleven Engineering Incorporated *****
//*****
/**
/**      Tabs: This file looks best with tab stops set every 6 spaces.
/**
//*****
/** File:      Thread2.asm
/**
/** Project: IEEE 802.11 MAC emulator. It can send to multiple (1-4) stations
/** Created: 1 June 2004 by Capt Joshua D. Green
/**
/** Description: Code that is run by Thread 2. Receiver thread. Receives all
/**                packets transmitted on the medium, determines if the packets
/**                are for the node, in the proper order, and without errors.
/**                It also communicates to Thread 0 whenever the medium is
/**                sensed busy.
/**
/** Disclaimer: This code was descended from Eleven Engineering sample
/**                source code, but changes were made by Capt Joshua D. Green
/**
//*****
//*****

_T2_Initialization:

                                mov    r1, 0
                                st     r1, v_Received_Stuff

                                mov    r1, 1<<kReceived_TX_SEMAPHORE
                                outp   r1, SCUup

BBUrx6_SEMAPHORE_LOW_LOOP:

    Looking_For_Preamble:
                                inp    r1, BBUrx6
                                bc     NC, BBUrx6_SEMAPHORE_LOW_LOOP

    Looking_For_END_of_Preamble:
                                inp    r0, BBUrx6
                                bc     NS, Looking_For_END_of_Preamble

    Checking_for_START:
                                and     r0, r0, 0b00111111
                                sub     r0, r0, 0
                                bc     NE, BBUrx6_SEMAPHORE_LOW_LOOP

                                inp    r0, BBUrx6
                                bc     NS, Looking_For_END_of_Preamble    // Loop while Hunt bit set

    Checking_for_START_2:
                                and     r0, r0, 0b00111111
                                sub     r0, r0, 0
                                bc     NE, BBUrx6_SEMAPHORE_LOW_LOOP // Looking_For_Preamble

    Receive_Stuff:
                                // Set kReceived_TX_SEMAPHORE LOW
                                mov     r1, 1<<kReceived_TX_SEMAPHORE
                                outp    r1, SCUdown

                                // Receive the first 6 bits - they are encoded in 6-16 format
                                // Received Frame Control
                                // Tells distant end wether packet is an ACK or Data Packet
                                // NOTE: Uses 6/16 encoding - sends out 16 bits for 6 bits of data
```

```

        // NOTE: Using the 6/16 encoding differs from IEEE 802.11 standard.
        // NOTE: It is done here strictly for experimental purposes

        inp    r0, BBURx6
        // Abort if no data detected
        bc     NS, Receive_Packet_HuntError_2
        bic    r0, r0, kRFWHardErrorBit
        // Abort if hard error detected
        bc     VS, Receive_Packet_HardError_2

        and    r0, r0, 0b00111111

        sub    r1, r0, kACK_Frame_Control
        bc     EQ, Received_ACK_Frame

        sub    r1, r0, kData_Frame_Control // If Data
        bc     EQ, Received_Data_Frame

        // Abort if no data detected
        bra    Receive_Packet_HuntError_2

Received_ACK_Frame:

        // Move data into a Rx ACK Frame
        st     r0, v_Rx_ACK_Frame_Control // Frame Octet 1-2

        // Go get the rest of the data from the ACK frame
        jsr    r6, WiFi_Received_ACK_Frame
        // r0 = 0 = NO Error
        // r0 = 1 = **Hunt Error**
        // r0 = 2 = **Hard Error**
        sub    r0, r0, 1
        bc     NS, ACK_Received_Successfully
        bc     ZS, Receive_Packet_HuntError

        bra    Receive_Packet_HardError

Received_Data_Frame:

        // Move data into a Rx Data Frame
        st     r0, v_Rx_Data_Frame_Control // Frame Octet 1-2

        // Go get the rest of the data from the Data Frame
        jsr    r6, WiFi_Received_Data_Frame
        // r0 = 0 = NO Error
        // r0 = 1 = **Hunt Error**
        // r0 = 2 = **Hard Error**
        sub    r0, r0, 1
        bc     NS, Data_Received_Successfully
        bc     ZS, Receive_Packet_HuntError

        bra    Receive_Packet_HardError

Receive_Packet_HardError:

#ifdef PrintErrors
        mov    r1, MSG_CORRUPTPACKET
        jsr    r6, XPD_EchoString
#endif

        // v_Received_Stuff_FLAG = 0 = Received JUNK
        // v_Received_Stuff_FLAG = 1 = Received DATA packet
        // v_Received_Stuff_FLAG = 2 = Received ACK
        // v_Received_Stuff_FLAG = 3 = Received frame NOT for this station
        mov    r1, 0 // 0 = Received JUNK
        st     r1, v_Received_Stuff

        // Reset kReceived_TX_SEMAPHORE HIGH and kReceived_TX_DONE_SEMAPHORE LOW
        mov    r0, 1<<kReceived_TX_SEMAPHORE
        outp    r0, SCUUp

        bra    BBURx6_SEMAPHORE_LOW_LOOP

Receive_Packet_HuntError:

#ifdef PrintErrors
        mov    r1, MSG_HUNTERERROR
        jsr    r6, XPD_EchoString
#endif

        // v_Received_Stuff_FLAG = 0 = Received JUNK

```

```

        // v_Received_Stuff_FLAG = 1 = Received DATA packet
        // v_Received_Stuff_FLAG = 2 = Received ACK
        // v_Received_Stuff_FLAG = 3 = Received frame NOT for this station
        mov     r1, 0 // 0 = Received JUNK
        st      r1, v_Received_Stuff

        // Reset kReceived_TX_SEMAPHORE HIGH and kReceived_TX_DONE_SEMAPHORE LOW
        mov     r0, 1<<kReceived_TX_SEMAPHORE
        outp    r0, SCUUp

        bra     BBURx6_SEMAPHORE_LOW_LOOP

Receive_Packet_HuntError_2:
#ifdef PrintErrors
    mov     r1, MSG_HUNTERERROR
    jsr     r6, XPD_EchoString
#endif

#ifdef THROUGHPUT
#ifdef TELEMETRY
    mov     r1, 3185 // 3190
#endif

#ifdef AVIONICS
    mov     r1, 20485 // 20490
#endif
#endif

    jsr     r6, Delay
    // v_Received_Stuff_FLAG = 0 = Received JUNK
    // v_Received_Stuff_FLAG = 1 = Received DATA packet
    // v_Received_Stuff_FLAG = 2 = Received ACK
    // v_Received_Stuff_FLAG = 3 = Received frame NOT for this station
    mov     r1, 0 // 0 = Received JUNK
    st      r1, v_Received_Stuff

    // Reset kReceived_TX_SEMAPHORE HIGH
    mov     r0, 1<<kReceived_TX_SEMAPHORE
    outp    r0, SCUUp

    bra     BBURx6_SEMAPHORE_LOW_LOOP

Receive_Packet_HardError_2:
#ifdef PrintErrors
    mov     r1, MSG_CORRUPTPACKET
    jsr     r6, XPD_EchoString
#endif

#ifdef THROUGHPUT
#ifdef TELEMETRY
    mov     r1, 3185 // 3190
#endif

#ifdef AVIONICS
    mov     r1, 20485 // 20490
#endif
#endif

    jsr     r6, Delay
    // v_Received_Stuff_FLAG = 0 = Received JUNK
    // v_Received_Stuff_FLAG = 1 = Received DATA packet
    // v_Received_Stuff_FLAG = 2 = Received ACK
    // v_Received_Stuff_FLAG = 3 = Received frame NOT for this station
    mov     r1, 0 // 0 = Received JUNK
    st      r1, v_Received_Stuff

    // Reset kReceived_TX_SEMAPHORE HIGH
    mov     r0, 1<<kReceived_TX_SEMAPHORE
    outp    r0, SCUUp

    bra     BBURx6_SEMAPHORE_LOW_LOOP

//*****
//*****
Data_Received_Successfully:

    // Load last stored Sequence_Number from sending station
    ld      r1, a_Rx_Data_Address_2 + 2 // Sending Station Address
    // Now r0 = 0 if sending station is Station 1,
    // r0 = 1 if sending station is Station 2, ect.
    sub     r1, r1, Station_01

```

```

        // Will load into r0:
        // a_Rx_Sequence_Numbers + 0 for Station 1
        // a_Rx_Sequence_Numbers + 1 for Station 2
        // etc.
        ld    r0, r1, a_Rx_Sequence_Numbers
        ld    r1, v_Rx_Data_Sequence_Number
        ld    r2, a_Rx_Data_Address_1 + 2

// If NOT for this station, discard packet

#ifdef STATION_1
    sub    r2, r2, Station_01
    bc     NE, Data_Received_but_NOT_for_me
#endif

#ifdef STATION_2
    sub    r2, r2, Station_02
    bc     NE, Data_Received_but_NOT_for_me
#endif

#ifdef STATION_3
    sub    r2, r2, Station_03
    bc     NE, Data_Received_but_NOT_for_me
#endif

#ifdef STATION_4
    sub    r2, r2, Station_04
    bc     NE, Data_Received_but_NOT_for_me
#endif

Data_Received_Successfully_2:
    // Check to see if received packet before. If yes, send another ACK
    // by indicating v_Received_Stuff_FLAG = 1 = Received DATA packet
    sub    r4, r1, r0
    bc     EQ, Data_Received_Successfully_DONE

    // Store v_Rx_Data_Sequence_Number to appropriate
    ld     r1, a_Rx_Data_Address_2 + 2 // Sending Station Address

    // Now r0 = 0 if sending station is Station 1,
    // r0 = 1 if sending station is Station 2, ect.
    sub    r1, r1, Station_01
    // Will load into r0:
    // a_Rx_Sequence_Numbers + 0 for Station 1
    // a_Rx_Sequence_Numbers + 1 for Station 2
    // etc.
    ld     r0, r1, a_Rx_Sequence_Numbers
    add    r0, r0, 1 // Increment Sequence Number for that station

    // Will store:
    // a_Rx_Sequence_Numbers + 0 for Station 1
    // a_Rx_Sequence_Numbers + 1 for Station 2
    // etc.
    st     r0, r1, a_Rx_Sequence_Numbers

Data_Received_Successfully_DONE:
    // v_Received_Stuff_FLAG = 0 = Received JUNK
    // v_Received_Stuff_FLAG = 1 = Received DATA packet
    // v_Received_Stuff_FLAG = 2 = Received ACK
    // v_Received_Stuff_FLAG = 3 = Received frame NOT for this station
    mov    r1, 1 // v_Received_Stuff_FLAG = 1 = Received DATA packet
    st     r1, v_Received_Stuff_FLAG

#ifdef Two_Way_Text
    // Set kReceived_some_text_SEMAPHORE HIGH
    mov    r1, 1 << kReceived_some_text_SEMAPHORE
    outp   r1, SCUdown
#endif

    // Reset kReceived_TX_SEMAPHORE HIGH and kReceived_TX_DONE_SEMAPHORE LOW
    mov    r0, 1 << kReceived_TX_SEMAPHORE
    outp   r0, SCUup

    bra    BBURx6_SEMAPHORE_LOW_LOOP

Data_Received_but_NOT_for_me:
    // v_Received_Stuff_FLAG = 0 = Received JUNK
    // v_Received_Stuff_FLAG = 1 = Received DATA packet

```

```

// v_Received_Stuff_FLAG = 2 = Received ACK
// v_Received_Stuff_FLAG = 3 = Received frame NOT for this station
// v_Received_Stuff_FLAG = 3 = Received frame NOT for this station
    mov    r1, 3
    st     r1, v_Received_Stuff_FLAG

// Delay of 80 clock ticks to even up timing with Data_Received_Successfully_DONE
    mov    r1, 1
    jsr    r6, Delay

// Reset kReceived_TX_SEMAPHORE HIGH and kReceived_TX_DONE_SEMAPHORE LOW
    mov    r0, 1<<kReceived_TX_SEMAPHORE
    outp   r0, SCUUp

    bra    BBURx6_SEMAPHORE_LOW_LOOP

//*****
//*****
ACK_Received_Successfully:
    // If NOT for this station, disgard packet and return to main loop
    ld     r2, a_Rx_ACK_Address_2 + 2

#ifdef STATION_1
    sub    r2, r2, Station_01
    bc     NE, ACK_Received_but_NOT_for_me
#endif

#ifdef STATION_2
    sub    r2, r2, Station_02
    bc     NE, ACK_Received_but_NOT_for_me
#endif

#ifdef STATION_3
    sub    r2, r2, Station_03
    bc     NE, ACK_Received_but_NOT_for_me
#endif

#ifdef STATION_4
    sub    r2, r2, Station_04
    bc     NE, ACK_Received_but_NOT_for_me
#endif

// See if recording started
    inp    r0, SCUsrc
    bis    r0, r0, kStart_Stop_SEMAPHORE
    bc     VC, Set_ACK_LOW

// Store the last transmitted Packet number in v_Theard_6_packet_que_number
    ld     r5, v_Thread_0_packet_que_number
    st     r5, v_Thread_6_packet_que_number

// Get End Times for just completed transmission
    mov    r0, 1<<kTime_SEMAPHORE
    outp   r0, SCUdown
    ld     r1, a_Time + 0 // Seconds
    ld     r2, a_Time + 1 // ms
    ld     r3, a_Time + 2 // µs
    outp   r0, SCUUp

// Store times in array used ONLY by Thread 6
    st     r1, a_Thread_6_END_Times + 0 // sec
    st     r2, a_Thread_6_END_Times + 1 // ms
    st     r3, a_Thread_6_END_Times + 2 // µs

    ld     r0, v_ACKs_Received
    add    r0, r0, 1
    st     r0, v_ACKs_Received

Set_ACK_LOW:

    mov    r0, 1<<kACK_SEMAPHORE
    outp   r0, SCUUp

// Increment Sequence_Number
    ld     r5, v_Thread_0_packet_que_number

```

```

        mov    r0, 1<<kTx_Data_Address_1_SEMAPHORE
        outp   r0, SCUdown
        ld     r1, r5, a_Tx_Data_Address_1 // Destination Address
        outp   r0, SCUup

// Now r0 = 0 if sending station is Station 1,
// r0 = 1 if sending station is Station 2, ect.
        sub    r1, r1, Station_01

// Will load:
// a_Tx_Sequence_Numbers + 0 for Station 1
// a_Tx_Sequence_Numbers + 1 for Station 2
// etc.
        ld     r0, r1, a_Tx_Sequence_Numbers

// Increment Sequence Number for that particular station
        add    r0, r0, 1

// Stores sequence number in the Sequence_Numbers array
        st     r0, r1, a_Tx_Sequence_Numbers

Receive_ACK_Done:
// Decrement v_Packets_in_Queue by 1
        mov    r0, 1<<kPackets_in_Queue_SEMAPHORE
        outp   r0, SCUdown
        ld     r1, v_Packets_in_Queue
        sub    r1, r1, 1
        bc     NC, Receive_ACK_Packets_in_Queue_OK
// If Negative is set, something is broken and must reset v_Packets_in_Queue to zero
        mov    r1, 0
Receive_ACK_Packets_in_Queue_OK:
        st     r1, v_Packets_in_Queue
        outp   r0, SCUup

// Set v_Medium_Idle_Flag back to zero
        mov    r1, 0
        st     r1, v_Medium_Idle_Flag

// v_Received_Stuff_FLAG = 0 = Received JUNK
// v_Received_Stuff_FLAG = 1 = Received DATA packet
// v_Received_Stuff_FLAG = 2 = Received ACK
// v_Received_Stuff_FLAG = 3 = Received frame NOT for this station
        mov    r1, 2 // v_Received_Stuff_FLAG = 2 = Received ACK packet
        st     r1, v_Received_Stuff_FLAG

// Reset kReceived_TX_SEMAPHORE HIGH and kReceived_TX_DONE_SEMAPHORE LOW
        mov    r0, 1<<kReceived_TX_SEMAPHORE
        outp   r0, SCUup

bra        BBURx6_SEMAPHORE_LOW_LOOP

ACK_Received_but_NOT_for_me:
// v_Received_Stuff_FLAG = 0 = Received JUNK
// v_Received_Stuff_FLAG = 1 = Received DATA packet
// v_Received_Stuff_FLAG = 2 = Received ACK
// v_Received_Stuff_FLAG = 3 = Received frame NOT for this station
        mov    r1, 3 // v_Received_Stuff_FLAG = 0 = Received JUNK
        st     r1, v_Received_Stuff_FLAG

// Delay of 456 clock cycles to sync up with Receive_ACK_Packets_in_Queue_OK routine
        mov    r1, 24
        mov    r1, 24
        jsr    r6, Delay

// Reset kReceived_TX_SEMAPHORE HIGH and kReceived_TX_DONE_SEMAPHORE LOW
        mov    r0, 1<<kReceived_TX_SEMAPHORE
        outp   r0, SCUup

bra        BBURx6_SEMAPHORE_LOW_LOOP

```

C.17. Thread3.asm

Random Number Generator. Using a 16-bit linear shift register, this thread produces uniform random numbers. The random numbers are used by Thread 3 to calculate backoff values for the IEEE 802.11 protocol in Thread 0.

```
//*****
//***** (C) 2002 by Eleven Engineering Incorporated *****
//*****
/**
/**      Tabs:  This file looks best with tab stops set every 6 spaces.
/**
//*****
//** File:          Thread3.asm
//**
//** Project: IEEE 802.11 MAC emulator.  It can send to multiple (1-4) stations
//** Created: 1 June 2004 by Capt Joshua D. Green
//**
//** Description: Code that is run by Thread 3.  Random Number Generator.  Using
//**              a 16-bit linear shift register, this thread produces uniform
//**              random numbers.  The random numbers are used by Thread 3 to
//**              calculate backoff values for the IEEE 802.11 protocol in Thread 0.
//**
//** Disclaimer: This code was descended from Eleven Engineering sample
//**              source code, but changes were made by Capt Joshua D. Green
//**
//*****
//*****

// RNG Constants
#ifdef STATION_1
#define      kSeed          0xB3CD
#endif

#ifdef STATION_2
#define      kSeed          0x2B0B
#endif

#ifdef STATION_3
#define      kSeed          0xDE23
#endif

#ifdef STATION_4
#define      kSeed          0xC236
#endif

#define      kXOR_Bit_3      0x0008
#define      kXOR_Bit_12     0x1000
#define      kXOR_Bit_14     0x4000
#define      kXOR_Bit_15     0x8000

_T3_Initialization:

                                mov     r0, kSeed

Main_Loop_Backoff_Value:

                                and     r2, r0, kXOR_Bit_3
                                rol     r2, r2, -3 // Role bit to the LSB position
                                and     r3, r0, kXOR_Bit_12
                                rol     r3, r3, -12 // Role bit to the LSB position
                                xor     r2, r2, r3
                                and     r4, r0, kXOR_Bit_14
                                rol     r4, r4, -14 // Role bit to the LSB position
                                and     r5, r0, kXOR_Bit_15
                                rol     r5, r5, -15 // Role bit to the LSB position
                                xor     r4, r4, r5
```

```

        xor    r3, r2, r4
        rol    r0, r0, 1
        bic    r0, r0, 0    // clear lsb
        ior    r0, r0, r3    // new RN is now in r0

        mov    r1, 1<<kRN_SEMAPHORE
        outp    r1, SCUdown
    // Stores the RN in a variable so other threads can get at it
        st     r0, v_RN
        outp    r1, SCUup

Is_Create_RN_BV_Flag_HIGH:
    // When kCreate_RN_BV_SEMAPHORE goes HIGH,
    // store the results from above in v_BV_Slots.
    //
    // The results will be increased as the number of
    // unsuccessful tries to transmit on the medium
    // (held in the variable v_Medium_Idle_Flag) goes high
    //
    // The final BV (representing the number of BV slots
    // to wait before transmitting) is stored in v_BV_Slots

        inp    r1, SCUsrc
        bis    r1, r1, kCreate_RN_BV_SEMAPHORE
        bc     VC, Main_Loop_Backoff_Value

Get_Backoff_Value:

        ld     r2, v_Medium_Idle_Flag

        sub    r3, r2, 5
    // Reached CWinMAX?
    // If NO, use rxBVTable to find right mask for RN
    // If Yes, use 0x03FF for mask of RN
        bc     GE, Get_Backoff_Value_MAX
    // Loads the mask needed for making out RN
        ld     r2, rxBVTable

Get_Backoff_Value_2:

        and    r3, r0, r2    // Chop off part of RN required
        add    r3, r3, 1    // Add one to BV so loop in Thread0 works right

        st     r3, v_BV_Slots

Get_Backoff_Value_END:
    // Reset kCreate_RN_BV_SEMAPHORE
        mov    r1, 1<<kCreate_RN_BV_SEMAPHORE
        outp    r1, SCUup

        bra    Main_Loop_Backoff_Value

Get_Backoff_Value_MAX:

        ld     r2, rxBVTable + 5    // mask of RN used if CWinMAX reached

        bra    Get_Backoff_Value_2

```

C.18. Thread4.asm

Timing Thread. The thread runs a clock storing the time in seconds, milliseconds, and microseconds. This is necessary because the running clock on the boards roles over after only 1.31 ms.

```
//*****
//***** (C) 2002 by Eleven Engineering Incorporated *****
//*****
/**
/**      Tabs: This file looks best with tab stops set every 6 spaces.
/**
//*****
//*****
/** File:      Thread4.asm
/**
/** Project: IEEE 802.11 MAC emulator. It can send to multiple (1-4) stations
/** Created: 1 June 2004 by Capt Joshua D. Green
/**
/** Description: Code that is run by Thread 4. Timing thread (clock counting
/**              out  $\mu$ s, ms, and sec). The thread runs a clock storing the
/**              time in seconds, milliseconds, and microseconds. This is
/**              necessary because the running clock on the boards roles over
/**              after only 1.31 ms.
/**
//*****
//*****

_T4_Initialization:

        mov     r0, 0
        st      r0, a_Time + 0 // sec
        st      r0, a_Time + 1 // ms
        st      r0, a_Time + 2 //  $\mu$ s - NOTE: clock counts in multiplies of 4

        mov     r3, 0 //  $\mu$ s counter - NOTE: clock counts in multiplies of 4
        mov     r4, 0 // ms counter
        mov     r5, 0 // sec counter

        mov     r1, 7 // Gives a delay of exactly 4  $\mu$ s between commands for first run
through

        Timing_Loop_for_MICRO_Sec:
                // Counts out 0-996  $\mu$ s in 4  $\mu$ s intervals. Stores them in a_Time + 2
                mov     r0, 996

                Timing_Loop_1:
                        sub     r1, r1, 1
                        bc      ZC, Timing_Loop_1

                        add     r3, r3, 4

                        mov     r2, 1<<kTime_SEMAPHORE
                        outp     r2, SCUdown
                        st      r3, a_Time + 2
                        outp     r2, SCUup

                        sub     r2, r3, r0
                        bc      EQ, Timing_Loop_for_MILA_Sec

                        mov     r1, 6

                bra     Timing_Loop_for_MICRO_Sec

        Timing_Loop_for_MILA_Sec:
                // Counts out 0-999 ms in 1 ms intervals. Stores them in a_Time + 1
                // Also resets  $\mu$ s to 0. Stores this in a_Time + 2
                mov     r0, 999
```

```

        mov     r1, 5

Timing_Loop_2:
        sub     r1, r1, 1
        bc     ZC, Timing_Loop_2

        mov     r3, 0
        add     r4, r4, 1
        mov     r2, 1<<kTime_SEMAPHORE
        outp    r2, SCUdown
        st      r3, a_Time + 2
        st      r4, a_Time + 1
        outp    r2, SCUup

        sub     r1, r4, r0
        bc     EQ, Timing_Loop_for_SECONDS

        mov     r1, 6

bra     Timing_Loop_for_MICRO_Sec

Timing_Loop_for_SECONDS:
        // Counts out sec in 1 sec intervals. Stores them in a_Time + 0
        // Also resets us and ms to 0. Stores them in a_Time + 2 and a_Time + 1 respectively

        mov     r1, 6 //Dummy load to get timing correct
        mov     r1, 6

Timing_Loop_for_SECONDS_2:
        // Counts out 0-996 us in 4 us intervals. Stores them in a_Time + 2
        mov     r0, 996

        Timing_Loop_3:
        sub     r1, r1, 1
        bc     ZC, Timing_Loop_3

        add     r3, r3, 4

        mov     r2, 1<<kTime_SEMAPHORE
        outp    r2, SCUdown
        st      r3, a_Time + 2
        outp    r2, SCUup

        sub     r2, r3, r0
        bc     EQ, Timing_Loop_for_SECONDS_3

        mov     r1, 6

bra     Timing_Loop_for_SECONDS_2

Timing_Loop_for_SECONDS_3:

        mov     r1, 5

Timing_Loop_4:
        sub     r1, r1, 1
        bc     ZC, Timing_Loop_4

        mov     r3, 0
        mov     r4, 0
        add     r5, r5, 1

        mov     r2, 1<<kTime_SEMAPHORE
        outp    r2, SCUdown
        st      r3, a_Time + 2
        st      r4, a_Time + 1
        st      r5, a_Time + 0
        outp    r2, SCUup

        mov     r1, 7

bra     Timing_Loop_for_MICRO_Sec

```

C.19. Thread5.asm

Packet Generation. Offers packets to the MAC layer's queue. It takes a random number generated by Thread 3 and uses it in conjunction this clock from Thread 4 to randomly offer packets to the queue. The thread also randomly chooses the destination address of the packet it loads into the queue. If the queue is determined full, it discards the packet.

```
//*****
//***** (C) 2002 by Eleven Engineering Incorporated *****
//*****
/**
/**      Tabs: This file looks best with tab stops set every 6 spaces.
/**
//*****
//*****
/** File:      Thread5.asm
/**
/** Project: IEEE 802.11 MAC emulator. It can send to multiple (1-4) stations
/** Created: 1 June 2004 by Capt Joshua D. Green
/**
/** Description: Code that is run by Thread 5. Packet Generation. Offers packets
/**              to the MAC layer's queue. It takes a random number generated by
/**              Thread 3 and uses it in conjunction this clock from Thread 4 to
/**              randomly offer packets to the queue. The thread also randomly
/**              chooses the destination address of the packet it loads into the
/**              queue. If the queue is determined full, it discards the packet.
/**
//*****
//*****

_T5_Initialization:
                                mov     r0, 0
                                sub      r0, r0, 1
                                st       r0, v_Thread_5_packet_que_number

                                jsr      r6, DelayLong
                                jsr      r6, DelayLong

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

Hold_Loop:
                                // WAIT till kGO_SEMAPHORE goes LOW, then start queing packets
                                mov      r0, 1<<kGO_SEMAPHORE
                                outp     r0, SCUdown
                                mov      r0, 1<<kGO_SEMAPHORE
                                outp     r0, SCUdown

                                mov      r5, (kDelay_Between_Tx + (kDelay_Between_Tx/4 - 1))
                                // r5 holds the number of loops so kDelay_Between_Tx delay will be whatever it is set

                                Delay_Between_TX:
                                //
                                mov      r0, 1<<kRN_SEMAPHORE
                                outp     r0, SCUdown
                                ld       r4, v_RN
                                outp     r0, SCUup

                                and      r4, r4, (kDelay_Between_TX_MASK-1)
                                add      r4, r4, 1

                                Delay_Between_TX_LOOP_1:
                                mov      r2, 1
                                mov      r2, 0xFFFF
                                sub      r4, r4, 1
                                bc       LE0, Que_Packet
```

```

        mov    r1, r5 // (kDelay_Between_Tx + (kDelay_Between_Tx/4 - 1))

Delay_Between_TX_LOOP_2:
        mov    r2, 0xFFF
        sub    r1, r1, 1
        bc     LE0, Delay_Between_TX_LOOP_1

        bra    Delay_Between_TX_LOOP_2

Que_Packet:

        // If recording started, increment v_Number_Packets_put_in_Que
        inp    r0, SCUsrc
        bis    r0, r0, kStart_Stop_SEMAPHORE
        bc     VC, Check_for_Buffer_Overflow

Increment_Queued_Packets_variable:
        ld     r1, v_Queued_Packets
        add    r1, r1, 1
        st     r1, v_Queued_Packets

Check_for_Buffer_Overflow:
        // Compare the Packet Queue Number for threads 0 and 5
        // If there difference is NOT zero,
        // they are NOT pointing to the same memory address
        // and the thread can load up another pack into the que
        ld     r0, v_Thread_0_packet_que_number
        ld     r1, v_Thread_5_packet_que_number
        sub    r0, r0, r1
        bc     NE, Determine_DA_Address

        // If the difference between the Packet Queue Number for threads 0 and 5 **IS** zero,
        // then they are pointing to the same address. This is OK when the number of packets
        // in the queue is zero. If the number of packets in the queue is NOT zero,
        // must throw out packet request to prevent buffer overflow.
        // In this case, we just skip to the "Check_for_Stop" routine
        mov    r0, 1<<kPackets_in_Que_SEMAPHORE
        outp   r0, SCUdown
        ld     r1, v_Packets_in_Que
        outp   r0, SCUup
        sub    r1, r1, (kTransmitter_Buffer_Size - 1)
        bc     EQ, Check_for_Stop

Determine_DA_Address:

        // Determin Address 1 (Destination Address)
Get_RN:
        mov    r1, 1<<kRN_SEMAPHORE
        outp   r1, SCUdown
        ld     r2, v_RN
        outp   r1, SCUup
        and    r2, r2, 0x00FF
        xor    r2, r2, 0x4200
        outp   r2, SFUpack
        inp    r2, SFUpack

#ifdef THROUGHPUT_4_STATIONS
        sub    r1, r2, 0
        bc     EQ, Get_RN
#endif

Determine_DA_Address_END:
        ld     r4, v_Thread_5_packet_que_number
        add    r4, r4, 1
        // Creates proper mask for Buffer Size
        and    r4, r4, (kTransmitter_Buffer_Size - 1)
        st     r4, v_Thread_5_packet_que_number

        // use lookup table to determine station to send to
        ld     r1, r2, rxDa_Station_Number
        mov    r0, 1<<kTx_Data_Address_1_SEMAPHORE
        outp   r0, SCUdown
        st     r1, r4, a_Tx_Data_Address_1
        outp   r0, SCUup

Que_Packet_2:
        // Que a packet
        mov    r0, 1<<kPackets_in_Que_SEMAPHORE
        outp   r0, SCUdown
        ld     r1, v_Packets_in_Que

```

```

        add    r1, r1, 1
        st     r1, v_Packets_in_Queue
        outp   r0, SCUUp

// If recording started, increment v_Number_Packets_put_in_Queue
        inp    r0, SCUsrc
        bis    r0, r0, kStart_Stop_SEMAPHORE
        bc     VC, Check_for_Stop

// Record packet Start Time
        mov    r0, 1<<kTime_SEMAPHORE
        outp   r0, SCUdown
        ld     r1, a_Time + 0 // Seconds
        ld     r2, a_Time + 1 // ms
        ld     r3, a_Time + 2 // µs
        outp   r0, SCUUp

        ld     r4, v_Thread_5_packet_queue_number
        st     r1, r4, a_BEGIN_Time_Seconds
        st     r2, r4, a_BEGIN_Time_Microseconds
        st     r3, r4, a_BEGIN_Time_Milliseconds

        inp    r0, SCUsrc

Check_for_Stop:
        // Loop until kGO_SEMAPHORE goes HIGH
        bis    r0, r0, kGO_SEMAPHORE
        bc     VC, Shut_Queue_Down

        bra    Delay_Between_TX

Shut_Queue_Down:
        // Set packet Queue to zero
        mov    r1, 0
        mov    r0, 1<<kPackets_in_Queue_SEMAPHORE
        outp   r0, SCUdown
        st     r1, v_Packets_in_Queue
        outp   r0, SCUUp

        bra    Hold_Loop

```

C.20. Thread6.asm

Testing and Recording. Starts and stops the testing for each trial. The thread also records all the information gathered from each trail.

```
//*****
//***** (C) 2002 by Eleven Engineering Incorporated *****
//*****
/**
**          Tabs: This file looks best with tab stops set every 6 spaces.
**
**
//*****
//*****
/**
** File:          Thread6.asm
**
/**
** Project: IEEE 802.11 MAC emulator. It can send to multiple (1-4) stations
** Created: 1 June 2004 by Capt Joshua D. Green
/**
** Description: Code that is run by Thread 6. Testing and Recording. Starts
**              and stops the testing for each trial. The thread also records
**              all the information gathered from each trail.
/**
//*****
//*****
```

_T6_Initialization:

```
Hold_T6_Loop_1:
    mov     r1, 1
    jsr     r6, Delay
    jsr     r6, XPD_ReadByteWithTimeout
    sub     r3, r1, 0xFFFF
    bc      ZS, Hold_T6_Loop_1
```

Start_Transmitting:

```
    mov     r1, MSG_NEWLINE
    jsr     r6, XPD_EchoString

    mov     r1, MSG_CURRENT_STATION
    jsr     r6, XPD_EchoString

    mov     r1, MSG_READY_2
    jsr     r6, XPD_EchoString
```

```
Hold_T6_Loop_2:
    mov     r1, 1
    jsr     r6, XPD_ReadByteWithTimeout
    sub     r3, r1, 0xFFFF
    bc      ZS, Hold_T6_Loop_2

    mov     r1, MSG_NEWLINE
    jsr     r6, XPD_EchoString

    mov     r1, MSG_TX_START_1
    jsr     r6, XPD_EchoString

    mov     r1, MSG_TX_START_2
    jsr     r6, XPD_EchoString

    mov     r0, 1<<kGO_SEMAPHORE
    outp    r0, SCUup
```

Start_Recording:

```
Hold_T6_Loop_3:
    mov     r1, 1
    jsr     r6, XPD_ReadByteWithTimeout
    sub     r3, r1, 0xFFFF
    bc      ZS, Hold_T6_Loop_3

    and     r1, r1, 0x00FF
```

```

sub    r1, r1, 'd' // Check to see if the character typed is a 'd'
bc     EQ, Stop_Transmitting

mov    r1, kNumber_of_tests
st     r1, v_Number_of_tests

mov    r1, MSG_RECORDING // ***Recording Started***
jsr    r6, XPD_EchoString

#ifdef MULT_TESTS
//MSG_DATA_DUMP_1:  "Delay|# of |Test |Paket|      | 1 | 2 | 3 |      |ACKs |---Mean Delay---|", CR, LF,
EOS
//MSG_DATA_DUMP_2:  "(mil)|slots|Time |Qued |TX      |ReTX |ReTX |ReTX |F-TX |RX      |(Sec)|(ms) |(mil)|", CR, LF,
EOS

mov    r1, MSG_DATA_DUMP_1
jsr    r6, XPD_EchoString

mov    r1, MSG_DATA_DUMP_2
jsr    r6, XPD_EchoString

#endif

Keep_Recording:

// Reset variables and the array a_Mean_Delay_Time
mov    r0, 0
st     r0, a_Mean_Delay_Time + 0
st     r0, a_Mean_Delay_Time + 1
st     r0, a_Mean_Delay_Time + 2
st     r0, v_ACKs_Received
st     r0, v_Queue_Packets
st     r0, v_Number_of_ACKs_Sent
//
st     r0, v_Thread_6_Number_of_Failed_TX

//a_Recorded_TX:
//  v_Number_of_TX:      @ = @ + 1
//  v_Number_of_Failed_TX: @ = @ + 1
//  a_Number_of_ReTX:    @ = @ + kMaxRetransmit
st     r0, v_Number_of_TX
st     r0, v_Number_of_Failed_TX
st     r0, a_Number_of_ReTX + 0
st     r0, a_Number_of_ReTX + 1
st     r0, a_Number_of_ReTX + 2

#ifdef DEBUG_LEDS
#ifdef STATION_1
// Turn ON LED #1
mov    r1, 0b0001
jsr    r6, ToggleLEDS // TurnOffLEDS //
#endif

#ifdef STATION_2
// Turn ON LED #2
mov    r1, 0b0010
jsr    r6, ToggleLEDS // TurnOffLEDS //
#endif

#ifdef STATION_3
// Turn ON LED #3
mov    r1, 0b0100
jsr    r6, ToggleLEDS // TurnOffLEDS //
#endif

#ifdef STATION_4
// Turn ON LED #4
mov    r1, 0b1000
jsr    r6, ToggleLEDS // TurnOffLEDS //
#endif
#endif

// Turn on recorder
mov    r1, 1<<kStart_Stop_SEMAPHORE
outp   r1, SCUdown

Main_Loop_T6:

mov    r3, 1<<kTime_SEMAPHORE
outp   r3, SCUdown
ld     r4, a_Time + 0 // sec
ld     r5, a_Time + 1 // ms

```

```

        outp    r3, SCUup

        st      r4, a_Start_Time + 0 // sec
        st      r5, a_Start_Time + 1 // ms

        add     r4, r4, kTime_of_Testing_Period

        st      r4, a_End_Time + 0 // sec
        st      r5, a_End_Time + 1 // ms

Loop_1_T6:
        // Delay for about 50  $\mu$ s before checking if reached number of sec yet
        mov     r5, 50 // Delay for loop

        Delay_Loop_1:

                inp     r1, SCUsrc
                bis     r1, r1, kACK_SEMAPHORE
                bc      VS, Calculate_Mean_Delay_from_Loop_1

                sub     r5, r5, 1
                bc      LT0, Delay_Loop_1

                mov     r3, 1<<kTime_SEMAPHORE
                outp    r3, SCUdown
                ld      r2, a_Time + 0 // sec
                outp    r3, SCUup

                ld      r4, a_End_Time + 0 // sec

                sub     r2, r4, r2
                bc      LE0, Loop_2_T6

        bra     Loop_1_T6

Calculate_Mean_Delay_from_Loop_1:
        //Wait while ACK Semaphore is HIGH.
        mov     r0, 1<<kACK_SEMAPHORE
        outp    r0, SCUdown

        // Release kACK_SEMAPHORE
        mov     r0, 1<<kACK_SEMAPHORE
        outp    r0, SCUup

        // Calculate Mean Delay of the packet just transmitted
        jsr     r6, Record_Data

        bra     Delay_Loop_1

Loop_2_T6:
        // Delay for about 50  $\mu$ s before checking if reached number of sec yet
        mov     r5, 50 // Delay for loop

        Delay_Loop_2:

                inp     r1, SCUsrc
                bis     r1, r1, kACK_SEMAPHORE
                bc      VS, Calculate_Mean_Delay_from_Loop_2

                sub     r5, r5, 1
                bc      LT0, Delay_Loop_2

                mov     r3, 1<<kTime_SEMAPHORE
                outp    r3, SCUdown
                ld      r2, a_Time + 1 // ms
                outp    r3, SCUup

                ld      r4, a_End_Time + 1 // ms

                sub     r2, r4, r2
                bc      LE0, Turn_Off

        bra     Loop_2_T6

Calculate_Mean_Delay_from_Loop_2:
        //Wait while ACK Semaphore is HIGH.
        mov     r0, 1<<kACK_SEMAPHORE
        outp    r0, SCUdown

```

```

        // Release kACK_SEMAPHORE
        mov    r0, 1<<kACK_SEMAPHORE
        outp   r0, SCUUp

        // Calculate Mean Delay of just transmitted packet
        jsr    r6, Record_Data

        bra    Delay_Loop_2

Turn_Off:
        // Turn on recorder
        mov    r1, 1<<kStart_Stop_SEMAPHORE
        outp   r1, SCUUp

#ifdef Pretty_Stuff
        mov    r1, MSG_NEWLINE
        jsr    r6, XPD_EchoString
#endif

Calculate_and_print_Results:

        ld     r1, v_Queued_Packets
        ld     r2, v_Number_of_TX
        ld     r3, a_Number_of_ReTX + 0
        ld     r4, a_Number_of_ReTX + 1
        ld     r5, a_Number_of_ReTX + 2

        st     r1, v_T7_Queued_Packets
        st     r2, v_T7_Number_of_TX
        st     r3, a_T7_Number_of_ReTX + 0
        st     r4, a_T7_Number_of_ReTX + 1
        st     r5, a_T7_Number_of_ReTX + 2

        ld     r0, v_Number_of_Failed_TX
        ld     r1, v_ACKs_Received
        ld     r2, a_Mean_Delay_Time + 0
        ld     r3, a_Mean_Delay_Time + 1
        ld     r4, a_Mean_Delay_Time + 2
ld     r5, v_Number_of_ACKs_Sent

        st     r0, v_T7_Number_of_Failed_TX
        st     r1, v_T7_ACKs_Received
        st     r2, a_T7_Mean_Delay_Time + 0
        st     r3, a_T7_Mean_Delay_Time + 1
        st     r4, a_T7_Mean_Delay_Time + 2
st     r5, v_T7_Number_of_ACKs_Sent

#ifdef MULT_TESTS
        mov    r0, 1<<kData_Dump_SEMAPHORE
        outp   r0, SCUUp
#endif

#ifdef MULT_TESTS
Multi_Test_1:
        // If recording mutiple tests, decrement test counter
        ld     r0, v_Number_of_tests
        sub    r0, r0, 1
        st     r0, v_Number_of_tests
        bc     LE0, Multi_Test_END

        mov    r0, 1<<kData_Dump_SEMAPHORE
        outp   r0, SCUUp

        bra    Keep_Recording

Multi_Test_END:
        mov    r0, 1<<kData_Dump_SEMAPHORE
        outp   r0, SCUUp

        mov    r1, 0xFFFF
        outp   r0, SCUdown

        mov    r1, MSG_LONGLINE
        jsr    r6, XPD_EchoString

        mov    r1, MSG_NEWLINE
        jsr    r6, XPD_EchoString

```

```

#endif

#ifdef STATION_1
// Turn OFF LED #1
    mov    r1, 0b0001
    jsr    r6, ToggleLEDs // TurnOffLEDs //
#endif

#ifdef STATION_2
// Turn OFF LED #2
    mov    r1, 0b0010
    jsr    r6, ToggleLEDs // TurnOffLEDs //
#endif

#ifdef STATION_3
// Turn OFF LED #3
    mov    r1, 0b0100
    jsr    r6, ToggleLEDs // TurnOffLEDs //
#endif

#ifdef STATION_4
// Turn OFF LED #4
    mov    r1, 0b1000
    jsr    r6, ToggleLEDs // TurnOffLEDs //
#endif

    bra    Start_Recording

Stop_Transmitting:
    mov    r0, 1<<kGO_SEMAPHORE
    outp   r0, SCUup

    mov    r1, MSG_NEWLINE
    jsr    r6, XPD_EchoString

    mov    r1, MSG_TX_STOPPED
    jsr    r6, XPD_EchoString

    bra    Start_Transmitting

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

Record_Data:
    // Save the contexts of r5 in the stack pointer (sp)
    // so it can be used again later.
    st     r5, sp, 0
    add    sp, sp, 1

    mov    r1, 0xFFFF
    mov    r1, 0xFFFF

    // Check to see if this was a failed transmittion
    inp    r1, SCUrsrc
    bis    r1, r1, kFailed_TX_SEMAPHORE
    bc     VS, Record_Data_END_2

    // Transfer the Packet Queue Number from Thread 1 to Thread 6
    ld     r1, v_Thread_6_packet_que_number

    // Transfer BEGIN times to arrays used only by Thread 6
    ld     r2, r1, a_BEGIN_Time_Seconds // sec
    ld     r3, r1, a_BEGIN_Time_Microseconds // ms
    ld     r4, r1, a_BEGIN_Time_Milliseconds // µs

    st     r2, a_Thread_6_BEGIN_Times + 0 // sec
    st     r3, a_Thread_6_BEGIN_Times + 1 // ms
    st     r4, a_Thread_6_BEGIN_Times + 2 // µs

Calculate_MicroSecond_Difference:
    // Find difference between the two times in µs
    ld     r5, a_Thread_6_END_Times + 2 // µs
    ld     r3, a_Thread_6_BEGIN_Times + 2 // µs

    sub    r0, r5, r3
    // If the difference between the Begin time and the End time is > 0, store it and
move on
    // If NOT, then must preform a carry function with the ms
    bc     LT0, Carry_MicroSeconds

```

```

        // Add the results to the total Mean Delay Time for  $\mu$ s and store
        ld    r1, a_Mean_Delay_Time + 2 //  $\mu$ s
        add   r1, r1, r0
        st    r1, a_Mean_Delay_Time + 2 //  $\mu$ s
        // If the addition had a carry (results >  $2^{16}$ ),
        // branch to increment the ms part of a_Mean_Delay_Time
        bc    CS, Mean_Delay_Carry_ms

        bra    Calculate_MilliSecond_Difference

Carry_MicroSeconds:
        // Add 1000 to the End time  $\mu$ s
        add   r5, r5, 1000
        // Subtract 1 from End time ms
        ld    r4, a_Thread_6_END_Times + 1 // ms
        sub   r4, r4, 1
        st    r4, a_Thread_6_END_Times + 1 // ms
        // Subtract End time from Beginning time again
        sub   r0, r3, r5
        // Add the results to the total Mean Delay Time for  $\mu$ s and store
        ld    r1, a_Mean_Delay_Time + 2 //  $\mu$ s
        add   r1, r0, r1
        st    r1, a_Mean_Delay_Time + 2 //  $\mu$ s
        // If the addition had a carry (results >  $2^{16}$ ),
        // branch to increment the ms part of a_Mean_Delay_Time
        bc    CS, Mean_Delay_Carry_ms

        bra    Calculate_MilliSecond_Difference

Mean_Delay_Carry_ms:
        // Called if when adding to a_Mean_Delay_Time + 2 rolls over
        // and sets the Carry Bit HIGH.
        // Increment a_Mean_Delay_Time + 2 ( $\mu$ s) by 535 ( $2^{16} - 65,000$   $\mu$ s)
        ld    r0, a_Mean_Delay_Time + 2 //  $\mu$ s
        add   r0, r0, 536
        st    r0, a_Mean_Delay_Time + 2 //  $\mu$ s
        // Increment a_Mean_Delay_Time + 1 (ms) by 65 (65 ms = 65,000  $\mu$ s)
        ld    r0, a_Mean_Delay_Time + 1 // ms
        add   r0, r0, 65
        st    r0, a_Mean_Delay_Time + 1 // ms
        // If the addition had a carry (results >  $2^{16}$ ),
        // branch to increment the seconds part of a_Mean_Delay_Time
        bc    CS, Mean_Delay_Carry_ms_Sec

        bra    Calculate_MilliSecond_Difference

Mean_Delay_Carry_ms_Sec:
        // Called if when adding to a_Mean_Delay_Time + 1 rolls over
        // and sets the Carry Bit HIGH.
        // Increment a_Mean_Delay_Time + 0 (sec) by 1
        ld    r0, a_Mean_Delay_Time + 0 // ms
        add   r0, r0, 1
        st    r0, a_Mean_Delay_Time + 0 // ms

Calculate_MilliSecond_Difference:
        // Find difference between the two times in ms
        ld    r2, a_Thread_6_BEGIN_Times + 0 // sec
        ld    r3, a_Thread_6_BEGIN_Times + 1 // ms

        ld    r4, a_Thread_6_END_Times + 0 // sec
        ld    r5, a_Thread_6_END_Times + 1 // ms

        sub   r0, r5, r3
        // If the difference between the Begin time and the End time is > 0, store it and
        // If NOT, then must perform a carry function with the sec
        bc    LT0, Carry_MilliSeconds

        // Add the results to the total Mean Delay Time for ms and store
        ld    r1, a_Mean_Delay_Time + 1 // ms
        add   r0, r0, r1
        st    r0, a_Mean_Delay_Time + 1 // ms
        // If the addition had a carry (results >  $2^{16}$ ),
        // branch to increment the Seconds part of a_Mean_Delay_Time
        bc    CS, Mean_Delay_Carry_SEC

        bra    Calculate_Second_Difference

Carry_MilliSeconds:
        // Add 1000 to the End time ms
        add   r5, r5, 1000

```

```

        // Subtract 1 from End time sec
        sub    r4, r4, 1
        // Subtract End time from Beginning time again and store
        sub    r0, r5, r3
        // Add the results to the total Mean Delay Time for ms and store
        ld     r1, a_Mean_Delay_Time + 1 // ms
        add    r1, r1, r0
        st     r1, a_Mean_Delay_Time + 1 // ms
        // If the addition had a carry (results > 2^16),
        // branch to increment the Seconds part of a_Mean_Delay_Time
        bc     CS, Mean_Delay_Carry_SEC

bra        Calculate_Second_Difference

Mean_Delay_Carry_SEC:
        // Called if when adding to a_Mean_Delay_Time + 1 rolls over
        // and sets the Carry Bit HIGH.
        // Increment a_Mean_Delay_Time + 1 (ms) by 536 (2^16 - 65,000 ms)
        ld     r0, a_Mean_Delay_Time + 1 // ms
        add    r0, r0, 536
        st     r0, a_Mean_Delay_Time + 1 // ms
        // Increment a_Mean_Delay_Time + 0 (sec) by 65 (65 sec = 65,000 ms)
        ld     r0, a_Mean_Delay_Time + 0 // ms
        add    r0, r0, 65
        st     r0, a_Mean_Delay_Time + 0 // ms

Calculate_Second_Difference:
        // Find difference between the two times in seconds
        sub    r0, r4, r2

        // Add the results to the total Mean Delay Time for ms and store
        ld     r1, a_Mean_Delay_Time + 0 // sec
        add    r1, r1, r0
        st     r1, a_Mean_Delay_Time + 0 // sec

Record_Data_END:
        sub    sp, sp, 1
        ld     r5, sp, 0

        jsr    r6, r6

Record_Data_END_2:
        mov    r0, 1<<kFailed_TX_SEMAPHORE
        outp    r0, SCUUp

        sub    sp, sp, 1
        ld     r5, sp, 0

        jsr    r6, r6

```

C.21. Thread7.asm

Print to Screen. This thread takes the data recorded by Thread 6 and displays it on computer attached to the boards. The data is then manually copied and saved to disk.

```
//*****
//***** (C) 2002 by Eleven Engineering Incorporated *****
//*****
/**
**      Tabs: This file looks best with tab stops set every 6 spaces.
**
**
//*****
//*****
/**
** File:      Thread6.asm
**
/**
/** Project: IEEE 802.11 MAC emulator. It can send to multiple (1-4) stations
/** Created: 1 June 2004 by Capt Joshua D. Green
/**
/** Description: Code that is run by Thread 6. Print to Screen. This thread takes
/** the data recorded by Thread 6 and displays it on computer attached
/** to the boards. The data is then manually copied and saved to disk.
/**
//*****
//*****

_T7_Initialization:

        mov     r0, 1<<kData_Dump_SEMAPHORE
        outp    r0, SCUdown
        outp    r0, SCUdown

#ifdef Pretty_Stuff

//MSG_TEST_COMPLETE:      ****Test Completed***", CR, LF, EOS
//MSG_DELAY:              " - Min delay in Milliseconds between sending packets: ", EOS
//MSG_SENT_1:              " - Sent ", EOS
//MSG_SENT_2:              " packets in ", EOS
//MSG_SENT_3:              " seconds.", CR, LF, EOS
//MSG_SENT_4:              " - Placed ", EOS
//MSG_SENT_5:              " in the TX queue.", CR, LF, EOS
//MSG_SENT_6:              " - Number of Re-TX: ", EOS
//MSG_SENT_7:              " - Total Mean Delay (Seconds, Microseconds, Milliseconds): ", EOS
//MSG_READY_1:             "Ready to start recording.", CR, LF, EOS
//MSG_READY_2:             "Press any key to start Transmitting.", CR, LF, EOS
//MSG_TX_START_1:          "Started Transmitting. To stop hit the 'd' key", CR, LF, EOS
//MSG_TX_START_2:          "Press any other key again to start recording.", CR, LF, EOS
//MSG_TX_STOPPED:          "---Stopped transmitting---", CR, LF, EOS

        mov     r1, MSG_NEWLINE
        jsr     r6, XPD_EchoString

        mov     r1, MSG_TEST_COMPLETE      // ****Test Completed***
        jsr     r6, XPD_EchoString

        mov     r1, MSG_NEWLINE
        jsr     r6, XPD_EchoString
        jsr     r6, XPD_EchoString

        mov     r1, MSG_RESULTS // Results of test:
        jsr     r6, XPD_EchoString

        mov     r1, MSG_DELAY // - Min delay in Milliseconds between sending
packets:
        jsr     r6, XPD_EchoString

        mov     r1, kDelay_Between_Tx
        jsr     r6, XPD_EchoUnsignedDec

        mov     r1, MSG_NEWLINE
        jsr     r6, XPD_EchoString
```

```

mov    r1, MSG_DELAY_MASK //    - Mask for Queuing Delay Random Number:
jsr    r6, XPD_EchoString

mov    r1, kDelay_Between_TX_MASK
jsr    r6, XPD_EchoHex

mov    r1, MSG_NEWLINE
jsr    r6, XPD_EchoString

mov    r1, MSG_SENT_1 //    - Sent
jsr    r6, XPD_EchoString

ld     r1, v_T7_Sent_Packets
jsr    r6, XPD_EchoUnsignedDec

mov    r1, MSG_SENT_2 //    packets in
jsr    r6, XPD_EchoString

mov    r1, kTime_of_Testing_Period
jsr    r6, XPD_EchoUnsignedDecNLZ

mov    r1, MSG_SENT_3 //    seconds.
jsr    r6, XPD_EchoString

mov    r1, MSG_SENT_4 //    - Placed
jsr    r6, XPD_EchoString

ld     r1, v_T7_Queued_Packets
jsr    r6, XPD_EchoUnsignedDec

mov    r1, MSG_SENT_5 //    in the TX queue.
jsr    r6, XPD_EchoString

mov    r1, MSG_SENT_6 //    - Number of Re-TX:
jsr    r6, XPD_EchoString

ld     r1, v_Number_of_Failed_TX
jsr    r6, XPD_EchoUnsignedDec

mov    r1, MSG_NEWLINE
jsr    r6, XPD_EchoString

mov    r1, MSG_SENT_7 //    - Total Mean Delay (Seconds, Microseconds,
jsr    r6, XPD_EchoString

ld     r1, a_Mean_Delay_Time + 0
jsr    r6, XPD_EchoUnsignedDec

mov    r1, MSG_COMMA
jsr    r6, XPD_EchoString

ld     r1, a_Mean_Delay_Time + 1
jsr    r6, XPD_EchoUnsignedDec

mov    r1, MSG_COMMA
jsr    r6, XPD_EchoString

ld     r1, a_Mean_Delay_Time + 2
jsr    r6, XPD_EchoUnsignedDec

mov    r1, MSG_NEWLINE
jsr    r6, XPD_EchoString

mov    r1, MSG_NEWLINE
jsr    r6, XPD_EchoString

Milliseconds):

mov    r1, MSG_NEWLINE
jsr    r6, XPD_EchoString

mov    r1, MSG_LONGLINE
jsr    r6, XPD_EchoString

//MSG_DATA_DUMP_1:  "Delay|# of |Test |Paket|      | 1 | 2 | 3 |      |ACKs |---Mean Delay---|", CR, LF,
EOS
//MSG_DATA_DUMP_2:  "(mil)|slots|Time |Qued |TX      |ReTX |ReTX |ReTX |F-TX |RX      |(Sec)|(ms) |(mil)|", CR, LF,
EOS
mov    r1, MSG_DATA_DUMP_1

```

```

                                jsr    r6, XPD_EchoString

                                mov    r1, MSG_DATA_DUMP_2
                                jsr    r6, XPD_EchoString

#endif
//                                mov    r1, kDelay_Between_Tx // Delay
//                                jsr    r6, XPD_EchoUnsignedDec

ld    r1, v_T7_Number_of_ACKs_Sent
jsr   r6, XPD_EchoUnsignedDec

                                mov    r1, MSG_SPACE
                                jsr    r6, XPD_EchoString

                                mov    r1, kDelay_Between_TX_MASK //slots
                                jsr    r6, XPD_EchoUnsignedDec

                                mov    r1, MSG_SPACE
                                jsr    r6, XPD_EchoString

                                mov    r1, kTime_of_Testing_Period // Time
                                jsr    r6, XPD_EchoUnsignedDec

                                mov    r1,MSG_SPACE
                                jsr    r6, XPD_EchoString

                                ld     r1, v_T7_Queued_Packets // Queued
                                jsr    r6, XPD_EchoUnsignedDec

                                mov    r1, MSG_SPACE
                                jsr    r6, XPD_EchoString

                                ld     r1, v_T7_Number_of_TX // TX
                                jsr    r6, XPD_EchoUnsignedDec

                                mov    r1, MSG_SPACE
                                jsr    r6, XPD_EchoString

                                ld     r1, a_T7_Number_of_ReTX + 0 // 1 Re-TX
                                jsr    r6, XPD_EchoUnsignedDec

                                mov    r1, MSG_SPACE
                                jsr    r6, XPD_EchoString

                                ld     r1, a_T7_Number_of_ReTX + 1 // 2 Re-TX
                                jsr    r6, XPD_EchoUnsignedDec

                                mov    r1, MSG_SPACE
                                jsr    r6, XPD_EchoString

                                ld     r1, a_T7_Number_of_ReTX + 2 // 3 Re-TX
                                jsr    r6, XPD_EchoUnsignedDec

                                mov    r1, MSG_SPACE
                                jsr    r6, XPD_EchoString

                                ld     r1, v_T7_Number_of_Failed_TX // F-TX
                                jsr    r6, XPD_EchoUnsignedDec

                                mov    r1, MSG_SPACE
                                jsr    r6, XPD_EchoString

                                ld     r1, v_T7_ACKs_Received // ACKS Rx
                                jsr    r6, XPD_EchoUnsignedDec

                                mov    r1, MSG_SPACE
                                jsr    r6, XPD_EchoString

                                ld     r1, a_T7_Mean_Delay_Time + 0 // MD(Sec)
                                jsr    r6, XPD_EchoUnsignedDec

                                mov    r1, MSG_SPACE
                                jsr    r6, XPD_EchoString

                                ld     r1, a_T7_Mean_Delay_Time + 1 // MD(ms)
                                jsr    r6, XPD_EchoUnsignedDec

                                mov    r1, MSG_SPACE
                                jsr    r6, XPD_EchoString

                                ld     r1, a_T7_Mean_Delay_Time + 2 // MD(mil-sec)

```

```

                                jsr    r6, XPD_EchoUnsignedDec

                                mov     r1, MSG_NEWLINE
                                jsr     r6, XPD_EchoString

#ifdef Pretty_Stuff
                                mov     r1, MSG_LONGLINE
                                jsr     r6, XPD_EchoString

                                mov     r1, MSG_NEWLINE
                                jsr     r6, XPD_EchoString
#endif

                                mov     r0, 1<<kData_Dump_SEMAPHORE
                                outp    r0, SCUup

                                bra     _T7_Initialization

```

C.22. XInC.c

XInC library file included with the development kit. The library file XInX.c defines Constants used for XInC Assembly programming.

```

//*****
//***** (C) 2002 by Eleven Engineering Incorporated *****
//*****
//**
//**      Tabs:  This file looks best with tab stops set every 6 spaces.
//**
//**
//*****
//*****
//**
//**      $RCSfile: XInC.h,v $
//**      $Revision: 1.7 $
//**      Tag $Name: $
//**      $Date: 2003/02/12 21:17:11 $
//**      $Author: eleven $
//**
//**      Project: XInC Library
//**      Description: Constants used for XInC Assembly programming.
//**
//**      Disclaimer: You may incorporate this sample source code into your
//**                  program(s) without restriction. This sample source code has
//**                  been provided "AS IS" and the responsibility for its
//**                  operation is yours. You are not permitted to redistribute
//**                  this sample source code as "Eleven sample source code" after
//**                  having made changes. If you're going to re-distribute the
//**                  source, we require that you make it clear in the source that
//**                  the code was descended from Eleven sample source code, but
//**                  that you've made changes.
//**
//*****
//*****

#ifndef __XINC_H_FILE__
#define __XINC_H_FILE__

//=====
// Register Set
//=====

#define    r0                %0
#define    r1                %1
#define    r2                %2
#define    r3                %3
#define    r4                %4
#define    r5                %5
#define    r6                %6
#define    r7                %7
#define    sp                %7

```

```

//=====
// Conditional Branch Tests
//=====

// Test NZVC Bits (Clear or Set)
#define NC 0xB
#define NS 0x3
#define ZC 0xA
#define ZS 0x2
#define VC 0x9
#define VS 0x1
#define CC 0x8
#define CS 0x0

// Comparison
#define EQ 0x2
#define NE 0xA

#define LT0 0x3
#define LE0 0x7
#define GE0 0xB
#define GT0 0xF

// Signed Comparison
#define LT 0x5
#define LE 0x6
#define GE 0xD
#define GT 0xE

// Unsigned Comparison
#define ULT 0x0
#define ULE 0x4
#define UGE 0x8
#define UGT 0xC

//=====
// I/O Peripheral Addresses
//=====

// SCU (Supervisory Control Unit)
#define SCUreg 0x00
#define SCUpc 0x01
#define SCUcc 0x02
#define SCUtime 0x03
#define SCUpntr 0x03
#define SCUbktpt 0x04
#define SCUstop 0x04
#define SCUwait 0x05
#define SCUsrc 0x06
#define SCUup 0x06
#define SCUver 0x07
#define SCUdown 0x07

// SCX (Supervisory Control Extensions)
#define SCXioCfgP 0x08
#define SCXioCfgD 0x09
#define SCXclkCfg 0x0A
#define SCXclkBuf 0x0B

// SFU (Shared Functional Units)
#define SFUpack 0x11 // Pack Bits
#define SFUpop 0x12 // Population Count
#define SFUls1 0x13 // Least Significant 1
#define SFUmul0 0x15 // Multiply Source 1, Result LS 16 Bits
#define SFUmul1 0x16 // Multiply Source 2, Result MS 16 Bits
#define SFUrev 0x17 // Bit Reverse

// SPI/ADC
#define SPI0rx 0x20
#define SPI0tx 0x20
#define SPI0cfg 0x21

#define SPI1rx 0x22
#define SPI1tx 0x22
#define SPI1cfg 0x23

#define ADCcfg 0x24
#define ADCdata 0x25

// BBU
#define BBUcfg 0x28

```

```

#define      BBUstatus      0x28
#define      BBUTx          0x29
#define      BBURx          0x29
#define      BBUBrg         0x2A
#define      BBUtime        0x2B
#define      BBURx4         0x2C
#define      BBURx6         0x2D

// GPIO
#define      GPAin          0x60
#define      GPAout         0x60
#define      GPACfg         0x61

#define      GPBin          0x62
#define      GPBout         0x62
#define      GPCfg         0x63

#define      GPCin          0x64
#define      GPCout         0x64
#define      GPCCfg         0x65

#define      GPDin          0x66
#define      GPDout         0x66
#define      GPDcfg         0x67

#define      GPEin          0x68
#define      GPEout         0x68
#define      GPECfg         0x69

#define      GPFin          0x6A
#define      GPFout         0x6A
#define      GPCfg         0x6B

#define      GPGin          0x6C
#define      GPGout         0x6C
#define      GPGcfg         0x6D

#define      GPHin          0x6E
#define      GPHout         0x6E
#define      GPHcfg         0x6F

#define      GPIin          0x70
#define      GPIout         0x70
#define      GPIcfg         0x71

#define      GPJin          0x72
#define      GPJout         0x72
#define      GPJcfg         0x73

//=====
// Memory Configuration
//=====

// ROM routines
#define      HardReset      0x0000
#define      SoftReset      0x0002
#define      PeripheralReset 0x0004
#define      ShowTerminationCode 0x0006
#define      ExpansionModule 0x0008
#define      ProgramEEPROM   0x000A
#define      ManufacturerTest 0x000C
#define      ArchitectureTest 0x000E

// RAM configuration
#define      kRAM_Block0_Start 0xC000
#define      kRAM_Block1_Start 0xC800
#define      kRAM_Block2_Start 0xD000
#define      kRAM_Block3_Start 0xD800
#define      kRAM_Block4_Start 0xE000
#define      kRAM_Block5_Start 0xE800
#define      kRAM_Block6_Start 0xF000
#define      kRAM_Block7_Start 0xF800

#define      kRAM_End        0xFFFF      // 16K words of RAM

//=====
// Boolean Logic
//=====

#define      true            1
#define      false          0

```

```
//=====
// Hardware Semaphores
//=====

#define      kHardwareSemaphore0  1 << 0
#define      kHardwareSemaphore1  1 << 1
#define      kHardwareSemaphore2  1 << 2
#define      kHardwareSemaphore3  1 << 3
#define      kHardwareSemaphore4  1 << 4
#define      kHardwareSemaphore5  1 << 5
#define      kHardwareSemaphore6  1 << 6
#define      kHardwareSemaphore7  1 << 7
#define      kHardwareSemaphore8  1 << 8
#define      kHardwareSemaphore9  1 << 9
#define      kHardwareSemaphore10 1 << 10
#define      kHardwareSemaphore11 1 << 11
#define      kHardwareSemaphore12 1 << 12
#define      kHardwareSemaphore13 1 << 13
#define      kHardwareSemaphore14 1 << 14
#define      kHardwareSemaphore15 1 << 15

#endif
```

C.23. XPD_Echo.asm

XInC library file included with the development kit. The firmware subroutines to echo ASCII messages to a terminal program connected to the XInC Program / Debug Port.

```
//*****
//***** (C) 2002 by Eleven Engineering Incorporated *****
//*****
//**
//**      Tabs: This file looks best with tab stops set every 6 spaces.
//**
//*****
//*****
//**
//**      $RCSfile: XPD_Echo.asm,v $
//**      $Revision: 1.5 $
//**      Tag $Name: $
//**      $Date: 2003/02/12 21:17:11 $
//**      $Author: eleven $
//**
//**      Project: XInC Library
//**      Description: Firmware subroutines to echo ASCII messages to a terminal
//**                    program connected to the XInC Program / Debug Port.
//**
//**      NOTE: To use these routines in your project, you must include the
//**            file "XPD_Echo_Data.asm" in your "LongData.asm" file.
//**
//**      Disclaimer: You may incorporate this sample source code into your
//**                    program(s) without restriction. This sample source code has
//**                    been provided "AS IS" and the responsibility for its
//**                    operation is yours. You are not permitted to redistribute
//**                    this sample source code as "Eleven sample source code" after
//**                    having made changes. If you're going to re-distribute the
//**                    source, we require that you make it clear in the source that
//**                    the code was descended from Eleven sample source code, but
//**                    that you've made changes.
//**
//*****
//*****
//**
//**      Routines:
//**
//**      XPD_EchoString
//**      XPD_EchoUnsignedDec
//**      XPD_EchoUnsignedDecNLZ
//**      XPD_EchoSignedDec
//**      XPD_EchoSignedDecNLZ
//**      XPD_EchoHex
```

```

/** XPD_EchoSetBitList
/** XPD_EchoBlock
/**
/** *****
/** *****

#ifndef __XPD_ECHO__
#define __XPD_ECHO__

#include "Math.asm"
#include "XPD_Serial.asm"

// ASCII Constants
#define CR 13
#define LF 10
#define EOS 0

//=====
// Input Params: r1 = Pointer to a Null Terminated String
// Output Params: None
//-----
// Description: Used to echo ASCII Strings to a computer terminal for
// debugging. Newlines and other control characters can be
// embedded in the string. Also strings must be
// Null-terminated.
//=====
XPD_EchoString:
    st r1, sp, 0
    st r2, sp, 1
    st r6, sp, 2
    add sp, sp, 3

    add r2, r1, 0 // Copy r1 to r2
XPD_EchoString_loop1:
    ld r1, r2, 0 // Read in character
    bc CC, XPD_EchoString_END
    jsr r6, XPD_WriteByte
    add r2, r2, 1
    bra XPD_EchoString_loop1

XPD_EchoString_END:
    sub sp, sp, 3
    ld r1, sp, 0
    ld r2, sp, 1
    ld r6, sp, 2
    jsr r6, r6

//=====
// Input Params: r1 = 16-bit Unsigned Integer
// Output Params: None
//-----
// Description: Echos a 16-bit unsigned integer to the terminal. Leading
// zeros are output if necessary to pad the output to 5 digits.
//=====
XPD_EchoUnsignedDec:
    st r1, sp, 0
    st r2, sp, 1
    st r6, sp, 2
    add sp, sp, 3

    // Determine 10000's digit
    mov r2, 10000
    jsr r6, IntegerDivide
    add r1, r1, '0'
    jsr r6, XPD_WriteByte // Echo result

    // Determine 1000's digit
    add r1, r2, 0 // Copy remainder to r1
    mov r2, 1000
    jsr r6, IntegerDivide
    add r1, r1, '0'
    jsr r6, XPD_WriteByte // Echo result

    // Determine 100's digit
    add r1, r2, 0 // Copy remainder to r1
    mov r2, 100
    jsr r6, IntegerDivide
    add r1, r1, '0'
    jsr r6, XPD_WriteByte // Echo result

```

```

        // Determine 10's digit
        add    r1, r2, 0
        mov    r2, 10
        jsr    r6, IntegerDivide
        add    r1, r1, '0'
        jsr    r6, XPD_WriteByte
        // Copy remainder to r1
        // Echo result

        // Determine 1's digit
        add    r1, r2, 0
        add    r1, r1, '0'
        jsr    r6, XPD_WriteByte
        // Remainder = 1's digit
        // Echo result

XPD_EchoUnsignedDec_END:
        sub    sp, sp, 3
        ld     r1, sp, 0
        ld     r2, sp, 1
        ld     r6, sp, 2
        jsr    r6, r6

//=====
// Input Params:    r1 = 16-bit Unsigned Integer
// Output Params:   None
//-----
// Description:     Echos a 16-bit unsigned integer to the terminal.  No leading
//                  zeros are ever output.
//=====
XPD_EchoUnsignedDecNLZ:
        st     r1, sp, 0
        st     r2, sp, 1
        st     r6, sp, 2
        add    sp, sp, 3

        // Determine 10000's digit
        mov    r2, 10000
        jsr    r6, IntegerDivide
        add    r1, r1, 0
        bc     ZS, XPD_EchoUnsignedDecNLZ_1000
        add    r1, r1, '0'
        jsr    r6, XPD_WriteByte
        // Echo result

XPD_EchoUnsignedDecNLZ_1000:
        // Determine 1000's digit
        add    r1, r2, 0
        mov    r2, 1000
        jsr    r6, IntegerDivide
        add    r1, r1, 0
        bc     ZS, XPD_EchoUnsignedDecNLZ_100
        add    r1, r1, '0'
        jsr    r6, XPD_WriteByte
        // Echo result

XPD_EchoUnsignedDecNLZ_100:
        // Determine 100's digit
        add    r1, r2, 0
        mov    r2, 100
        jsr    r6, IntegerDivide
        add    r1, r1, 0
        bc     ZS, XPD_EchoUnsignedDecNLZ_10
        add    r1, r1, '0'
        jsr    r6, XPD_WriteByte
        // Echo result

XPD_EchoUnsignedDecNLZ_10:
        // Determine 10's digit
        add    r1, r2, 0
        mov    r2, 10
        jsr    r6, IntegerDivide
        add    r1, r1, 0
        bc     ZS, XPD_EchoUnsignedDecNLZ_1
        add    r1, r1, '0'
        jsr    r6, XPD_WriteByte
        // Echo result

XPD_EchoUnsignedDecNLZ_1:
        // Determine 1's digit
        add    r1, r2, 0
        add    r1, r1, '0'
        jsr    r6, XPD_WriteByte
        // Copy remainder to r1 (1's digit)
        // Echo result

XPD_EchoUnsignedDecNLZ_END:
        sub    sp, sp, 3
        ld     r1, sp, 0
        ld     r2, sp, 1

```

```

        ld     r6, sp, 2
        jsr    r6, r6

//=====
// Input Params:    r1 = 16-bit Signed Integer
// Output Params:    None
//-----
// Description:      Echos a 16-bit signed integer to the terminal. Leading
//                    zeros are output if necessary to pad the output to 5 digits.
//                    In total, 6 characters are output: 1 sign and 5 digits.
//=====
XPD_EchoSignedDec:
        st     r1, sp, 0
        st     r2, sp, 1
        st     r6, sp, 2
        add    sp, sp, 3

        // Determine the sign character
        add    r1, r1, 0
        bc     ZS, XPD_EchoSignedDec_Zero
        bc     NC, XPD_EchoSignedDec_Positive

XPD_EchoSignedDec_Negative:
        mov    r1, '-'
        jsr    r6, XPD_WriteByte
        ld     r1, sp, -3                // Reload the integer
        mov    r2, 0
        sub    r1, r2, r1                // Convert to positive representation
        bra    XPD_EchoSignedDec_Digits

XPD_EchoSignedDec_Positive:
        mov    r1, '+'
        jsr    r6, XPD_WriteByte
        ld     r1, sp, -3                // Reload the integer
        bra    XPD_EchoSignedDec_Digits

XPD_EchoSignedDec_Zero:
        mov    r1, '0'
        jsr    r6, XPD_WriteByte
        ld     r1, sp, -3                // Reload the integer

XPD_EchoSignedDec_Digits:

        // Determine 10000's digit
        mov    r2, 10000
        jsr    r6, IntegerDivide
        add    r1, r1, '0'
        jsr    r6, XPD_WriteByte        // Echo result

        // Determine 1000's digit
        add    r1, r2, 0                // Copy remainder to r1
        mov    r2, 1000
        jsr    r6, IntegerDivide
        add    r1, r1, '0'
        jsr    r6, XPD_WriteByte        // Echo result

        // Determine 100's digit
        add    r1, r2, 0                // Copy remainder to r1
        mov    r2, 100
        jsr    r6, IntegerDivide
        add    r1, r1, '0'
        jsr    r6, XPD_WriteByte        // Echo result

        // Determine 10's digit
        add    r1, r2, 0                // Copy remainder to r1
        mov    r2, 10
        jsr    r6, IntegerDivide
        add    r1, r1, '0'
        jsr    r6, XPD_WriteByte        // Echo result

        // Determine 1's digit
        add    r1, r2, 0                // Remainder = 1's digit
        add    r1, r1, '0'
        jsr    r6, XPD_WriteByte        // Echo result

XPD_EchoSignedDec_END:
        sub    sp, sp, 3
        ld     r1, sp, 0
        ld     r2, sp, 1

```

```

        ld     r6, sp, 2
        jsr    r6, r6

//=====
// Input Params:    r1 = 16-bit Signed Integer
// Output Params:    None
//-----
// Description:      Echos a 16-bit signed integer to the terminal. No leading
//                   zeros are ever output.
//=====
XPD_EchoSignedDecNLZ:
        st     r1, sp, 0
        st     r2, sp, 1
        st     r6, sp, 2
        add    sp, sp, 3

        // Determine the sign character
        add    r1, r1, 0
        bc     ZS, XPD_EchoSignedDecNLZ_Zero
        bc     NC, XPD_EchoSignedDecNLZ_Positive

XPD_EchoSignedDecNLZ_Negative:
        mov    r1, '-'
        jsr    r6, XPD_WriteByte
        ld     r1, sp, -3                // Reload the integer
        mov    r2, 0
        sub    r1, r2, r1                // Convert to positive representation
        bra    XPD_EchoSignedDecNLZ_Digits

XPD_EchoSignedDecNLZ_Positive:
        mov    r1, '+'
        jsr    r6, XPD_WriteByte
        ld     r1, sp, -3                // Reload the integer
        bra    XPD_EchoSignedDecNLZ_Digits

XPD_EchoSignedDecNLZ_Zero:
        mov    r1, '0'
        jsr    r6, XPD_WriteByte
        ld     r1, sp, -3                // Reload the integer

XPD_EchoSignedDecNLZ_Digits:
        // Determine 10000's digit
        mov    r2, 10000
        jsr    r6, IntegerDivide
        add    r1, r1, 0
        bc     ZS, XPD_EchoSignedDecNLZ_1000
        add    r1, r1, '0'
        jsr    r6, XPD_WriteByte        // Echo result

XPD_EchoSignedDecNLZ_1000:
        // Determine 1000's digit
        add    r1, r2, 0                // Copy remainder to r1
        mov    r2, 1000
        jsr    r6, IntegerDivide
        add    r1, r1, 0
        bc     ZS, XPD_EchoSignedDecNLZ_100
        add    r1, r1, '0'
        jsr    r6, XPD_WriteByte        // Echo result

XPD_EchoSignedDecNLZ_100:
        // Determine 100's digit
        add    r1, r2, 0                // Copy remainder to r1
        mov    r2, 100
        jsr    r6, IntegerDivide
        add    r1, r1, 0
        bc     ZS, XPD_EchoSignedDecNLZ_10
        add    r1, r1, '0'
        jsr    r6, XPD_WriteByte        // Echo result

XPD_EchoSignedDecNLZ_10:
        // Determine 10's digit
        add    r1, r2, 0                // Copy remainder to r1
        mov    r2, 10
        jsr    r6, IntegerDivide
        add    r1, r1, 0
        bc     ZS, XPD_EchoSignedDecNLZ_1
        add    r1, r1, '0'
        jsr    r6, XPD_WriteByte        // Echo result

```

```

XPD_EchoSignedDecNLZ_1:
    // Determine 1's digit
    add    r1, r2, 0                // Copy remainder to r1 (1's digit)
    add    r1, r1, '0'
    jsr    r6, XPD_WriteByte        // Echo result

XPD_EchoSignedDecNLZ_END:
    sub    sp, sp, 3
    ld     r1, sp, 0
    ld     r2, sp, 1
    ld     r6, sp, 2
    jsr    r6, r6

//=====
// Input Params:    r1 = 16-bit Number
// Output Params:   None
//-----
// Description:     Echos a 16-bit number to the terminal formatted as a
//                  hexadecimal integer with format 0xABCD where ABCD are hex
//                  digits. Uses R2 for temp, divisor, remainder. Subroutines
//                  use R0 as scratch.
//=====
XPD_EchoHex:
    st     r1, sp, 0
    st     r2, sp, 1
    st     r6, sp, 2
    add    sp, sp, 3

    add    r2, r1, 0                // Copy r1 to r2

    mov    r1, '0'
    jsr    r6, XPD_WriteByte        // Echo leading 0
    mov    r1, 'x'
    jsr    r6, XPD_WriteByte        // Echo leading x

    rol    r1, r2, 4
    and    r1, r1, 0x000F
    ld     r1, r1, table_bintohex    // Convert MSD
    jsr    r6, XPD_WriteByte        // Echo to stdout

    rol    r1, r2, 8
    and    r1, r1, 0x000F
    ld     r1, r1, table_bintohex    // Convert next digit
    jsr    r6, XPD_WriteByte        // Echo to stdout

    rol    r1, r2, 12
    and    r1, r1, 0x000F
    ld     r1, r1, table_bintohex    // Convert next digit
    jsr    r6, XPD_WriteByte        // Echo to stdout

    rol    r1, r2, 0
    and    r1, r1, 0x000F
    ld     r1, r1, table_bintohex    // Convert LSD
    jsr    r6, XPD_WriteByte        // Echo to stdout

XPD_EchoHex_END:
    sub    sp, sp, 3
    ld     r1, sp, 0
    ld     r2, sp, 1
    ld     r6, sp, 2
    jsr    r6, r6

//=====
// Input Params:    r1 = 16-Bit Vector
// Output Params:   None
//-----
// Description:     Echos to the terminal a comma delimited list of the bits
//                  that are set in a 16-bit vector.
//=====
XPD_EchoSetBitList:
    st     r0, sp, 0
    st     r1, sp, 1
    st     r2, sp, 2
    st     r3, sp, 3
    st     r4, sp, 4
    st     r5, sp, 5
    st     r6, sp, 6
    add    sp, sp, 7

```

```

        mov     r2, 0                      // Previous Bit = FALSE
        mov     r3, 15
        mov     r4, 0                      // i = 0
        add     r0, r1, 0                  // r0 = r1

XPD_EchoSetBitList_loop:
        sub     r1, r3, r4
        rol     r1, r0, r1                // Test Bit i
        bc     NC, XPD_EchoSetBitList_loop_end

        add     r2, r2, 0                  // Test For Previous Bit
        bc     ZS, XPD_EchoSetBitList_output

        mov     r1, MSG_COMMA              // Output ", "
        jsr     r6, XPD_EchoString

XPD_EchoSetBitList_output:
        mov     r2, 1                      // Previous Bit = TRUE
        add     r1, r4, 0                  // Output i
        jsr     r6, XPD_EchoUnsignedDecNLZ

XPD_EchoSetBitList_loop_end:
        add     r4, r4, 1                  // i++
        sub     r5, r4, 16
        bc     ZC, XPD_EchoSetBitList_loop

XPD_EchoSetBitList_END:
        sub     sp, sp, 7
        ld      r0, sp, 0
        ld      r1, sp, 1
        ld      r2, sp, 2
        ld      r3, sp, 3
        ld      r4, sp, 4
        ld      r5, sp, 5
        ld      r6, sp, 6
        jsr     r6, r6

//=====
// Input Params:    r5 = Start address of the block
//                  r4 = Number of words to display
// Output Params:    None
//-----
// Description:      Echos to the terminal a given number of words of data in
//                  hex format starting at a given memory address. The output is
//                  formatted with 8 words per line and a space inbetween each
//                  word.
//=====
XPD_EchoBlock:
        st      r0, sp, 0
        st      r1, sp, 1
        st      r2, sp, 2
        st      r3, sp, 3
        st      r4, sp, 4
        st      r5, sp, 5
        st      r6, sp, 6
        add     sp, sp, 7

XPD_EchoBlock_lineLoop:
        mov     r3, 8                      // r3 = words on this line
XPD_EchoBlock_wordLoop:
        ld      r1, r5, 0
        jsr     r6, XPD_EchoHex
        mov     r1, ' '
        jsr     r6, XPD_WriteByte
        add     r5, r5, 1                  // Increment address
        sub     r4, r4, 1                  // Decrement total
        bc     ZS, XPD_EchoBlock_END
        sub     r3, r3, 1                  // Decrement words on this line
        bc     ZC, XPD_EchoBlock_wordLoop
        mov     r1, MSG_NEWLINE            // Start new line
        jsr     r6, XPD_EchoString
        bra     XPD_EchoBlock_lineLoop

XPD_EchoBlock_END:
        sub     sp, sp, 7
        ld      r0, sp, 0
        ld      r1, sp, 1
        ld      r2, sp, 2
        ld      r3, sp, 3
        ld      r4, sp, 4

```

```
        ld    r5, sp, 5
        ld    r6, sp, 6
        jsr   r6, r6
#endif
```

C.24. XPD_Echo_Data.asm

XInC library file included with the development kit. Data file used by XPD_Echo.asm.

```
//*****
//***** (C) 2002 by Eleven Engineering Incorporated *****
//*****
//**
//**      Tabs:  This file looks best with tab stops set every 6 spaces.
//**
//*****
//*****
//**
//**      $RCSfile: XPD_Echo_Data.asm,v $
//**      $Revision: 1.3 $
//**      Tag $Name: $
//**      $Date: 2003/02/12 21:17:11 $
//**      $Author: eleven $
//**
//**      Project: XInC Library
//**      Description: Data used by XPD_Echo.asm.
//**
//**      Disclaimer: You may incorporate this sample source code into your
//**                  program(s) without restriction.  This sample source code has
//**                  been provided "AS IS" and the responsibility for its
//**                  operation is yours.  You are not permitted to redistribute
//**                  this sample source code as "Eleven sample source code" after
//**                  having made changes.  If you're going to re-distribute the
//**                  source, we require that you make it clear in the source that
//**                  the code was descended from Eleven sample source code, but
//**                  that you've made changes.
//**
//*****
//*****

table_bintohex:
    "0123456789ABCDEF"

MSG_COMMA:
    ", ", EOS
MSG_NEWLINE:
    CR, LF, EOS

MSG_8SPACES:
    " "
MSG_7SPACES:
    " "
MSG_6SPACES:
    " "
MSG_5SPACES:
    " "
MSG_4SPACES:
    " "
MSG_3SPACES:
    " "
MSG_2SPACES:
    " "
MSG_SPACE:
    " ", EOS
MSG_DASH:
    "-", EOS
MSG_LONGLINE:
    "-----", CR, LF, EOS
```

C.25. XPD_Serial.asm

XInC library file included with the development kit. The firmware subroutines are used to configure, read data, and write data using the XInC Program / Debug Port.

```
//*****
//***** (C) 2002 by Eleven Engineering Incorporated *****
//*****
/**
    Tabs: This file looks best with tab stops set every 6 spaces.
/**
//*****
/**
$RCSfile: XPD_Serial.asm,v $
$Revision: 1.4 $
Tag $Name: $
$Date: 2003/02/12 21:17:11 $
$Author: eleven $
/**
Project: XInC Library
Description: Firmware subroutines to configure, read data, and write data
            using the XInC Program / Debug Port.
/**
NOTE: To use these routines in your project, you must assign
      kSPI0CS_Semaphore to one of your hardware semaphores.
/**
Disclaimer: You may incorporate this sample source code into your
            program(s) without restriction. This sample source code has
            been provided "AS IS" and the responsibility for its
            operation is yours. You are not permitted to redistribute
            this sample source code as "Eleven sample source code" after
            having made changes. If you're going to re-distribute the
            source, we require that you make it clear in the source that
            the code was descended from Eleven sample source code, but
            that you've made changes.
/**
//*****
//*****
/**
High Level Routines:
/**
XPD_Configure
XPD_ReadConfigWord
/**
XPD_WriteByte
XPD_ReadByte
XPD_ReadByteWithTimeout
XPD_ReadWriteByte
/**
Low Level Routines:
/**
XPD_ShiftInOut
/**
//*****
//*****

#ifndef __XPD_SERIAL__
#define __XPD_SERIAL__

//-----
// XPD Port Configuration Constants
//-----

// Baud Rate Constants
#define kXPD_BaudRate_230400    0x0
#define kXPD_BaudRate_115200    0x1
#define kXPD_BaudRate_76800     0x8
#define kXPD_BaudRate_57600     0x2
#define kXPD_BaudRate_38400     0x9
#define kXPD_BaudRate_28800     0x3
#define kXPD_BaudRate_19200     0xA
#define kXPD_BaudRate_14400     0x4
#define kXPD_BaudRate_9600      0xB
```

```

#define      kXPD_BaudRate_7200      0x5
#define      kXPD_BaudRate_4800      0xC
#define      kXPD_BaudRate_3600      0x6
#define      kXPD_BaudRate_2400      0xD
#define      kXPD_BaudRate_1800      0x7
#define      kXPD_BaudRate_1200      0xE
#define      kXPD_BaudRate_600       0xF

// Protocol Constants
#define      kXPD_Use7DataBits        1 << 4
#define      kXPD_EnableParityBits    1 << 5
#define      kXPD_Use2StopBits        1 << 6
#define      kXPD_Enable_IrDA_Timing  1 << 7
#define      kXPD_Shutdown            1 << 12
#define      kXPD_DisableFIFO         1 << 13

// XInC Clock Speed Constants
#define      kXPD_ClockLE_3MHz        0 << 8
#define      kXPD_ClockLE_6MHz        1 << 8
#define      kXPD_ClockLE_12MHz       2 << 8
#define      kXPD_ClockLE_24MHz       3 << 8
#define      kXPD_ClockLE_48MHz       4 << 8
#define      kXPD_ClockLE_96MHz       5 << 8
#define      kXPD_ClockLE_192MHz      6 << 8
#define      kXPD_ClockLE_384MHz      7 << 8

//-----
// XPD Port Control Constants
//-----
#define      kXPD_ParityBit           8
#define      kXPD_CTS_RTS_Bit         9
#define      kXPD_ErrorBit            10
#define      kXPD_TransmitDoneBit      14
#define      kXPD_DataReceivedBit     15

//=====
// Input Params:    r1 = Configuration Word (Sum of Configuration Constants)
// Output Params:   r1 = Configuration Succeeded (true or false)
//-----
// Description:     Used to configure the XPD Port.
//
//                 This routine always sets up SPI0 for polarity=0, phase=0, and
//                 mode=master.
//
//                 The default settings are:
//                 Baud Rate:      230.4k
//                 Data Bits:      8
//                 Parity Bits:    None
//                 Stop Bits:      1
//                 Timing:         Standard
//                 Running:        True
//                 FIFO:           Enabled
//                 Clock:          3MHz or less
//
//                 To change these settings, add the desired constants
//                 to the Configuration Word.
//
//                 At 3.3V the maximum SPI0 clock supported by the MAX3100
//                 SPI-UART is 1.5MHz.  Therefore, to get the fastest possible
//                 data rate on the SPI, you should add the first XInC Clock
//                 Speed Constant that is faster than the actual speed of the
//                 XInC clock to your Configuration Word.
//=====
XPD_Configure:
    st     r6, sp, 0
    st     r1, sp, 1
    add    sp, sp, 2

// Setup the SEM Address Mux
// GPB[0:3] = SPI as output
// GPB0      = SPI ROM CS
// GPB[1:2]  = SPI MUX Address decoding
// 0b001    = SPI XPD Port
    mov    r1, 0x0707
    outp   r1, GPBcfg

// Derive a config word for the SPI0 Port from the XPD Config Word
    ld     r1, sp, -1
    rol    r1, r1, -6           // Move the "Clock Speed Constant" into its appropriate position
    and    r1, r1, 0b0000000001011100
    bis    r1, r1, 1           // Set the "Master SPI" bit

```

```

        bic    r1, r1, 6                // Test the "Shutdown" bit
        bc     VS, XPD_Configure_Disable
        bis    r1, r1, 0                // Set the "Enable SPI" bit

XPD_Configure_Disable:
    outp     r1, SPI0cfg

    // Test if the UART hardware exists by configuring it with a non-zero dummy baudrate
    mov      r1, 0xC00F                // "Write Config" command in upper two bits
    jsr      r6, XPD_ShiftInOut        // Write the configuration word

    mov      r1, 0x4000                // "Read Config" command in upper two bits
    jsr      r6, XPD_ShiftInOut        // Read the configuration word

    sub      r1, r1, 0x400F            // Check for correct baud rate
    bc       EQ, XPD_Configure_UART_Attached
    mov      r1, false                 // Return false
    bra      XPD_Configure_END

XPD_Configure_UART_Attached:
    ld       r1, sp, -1
    and      r1, r1, 0b111110001111111
    ior      r1, r1, 0xC000            // "Write Config" command in upper two bits
    jsr      r6, XPD_ShiftInOut        // Write config, also clears receive FIFO

    mov      r1, true                  // Return true

XPD_Configure_END:
    sub      sp, sp, 2
    // Don't restore r1
    ld       r6, sp, 0

    jsr      r6, r6

//=====
// Input Params:  None
// Output Params: r1 = Configuration Word
//-----
// Description:   Reads config and status data from the SPI-UART. Can be used
//                to determine the status of the transmit and receive buffers
//                by checking the transmit and receive bits.
//=====
XPD_ReadConfigWord:
    st       r6, sp, 0
    add      sp, sp, 1

    mov      r1, 0x4000                // "Read Config" command in upper two bits
    jsr      r6, XPD_ShiftInOut        // Read the configuration word

    sub      sp, sp, 1
    ld       r6, sp, 0

    jsr      r6, r6

//=====
// Input Params:  r1 = The byte to write
// Output Params:  None
//-----
// Description:   Used to shift a data byte out to the SPI-UART. The data byte
//                shifted in is discarded. The data is always in the LSB of
//                the word.
//=====
XPD_WriteByte:
    st       r6, sp, 0
    st       r1, sp, 1                // Push r1 because XPD_ReadConfigWord uses it
    add      sp, sp, 2

XPD_WriteByte_LOOP:
    jsr      r6, XPD_ReadConfigWord
    bic      r1, r1, kXPD_TransmitDoneBit // Is transmit buffer empty?
    bc       VC, XPD_WriteByte_LOOP
    ld       r1, sp, -1                // Reload r1 from the stack
    and      r1, r1, 0x07FF
    bis      r1, r1, 15
    jsr      r6, XPD_ShiftInOut

    sub      sp, sp, 2

```

```

        ld    r6, sp, 0
        ld    r1, sp, 1

        jsr   r6, r6

//=====
// Input Params:    None
// Output Params:   r1 = The byte read from the SPI-UART
//-----
// Description:     Used to shift a data byte in from the SPI-UART. A zero byte
//                  is shifted out. This subroutine does not return until a byte
//                  has been received. The data is always in the LSB of the
//                  word.
//=====
XPD_ReadByte:
        st    r6, sp, 0
        add   sp, sp, 1

        XPD_ReadByte_LOOP:
        mov   r1, 0
        jsr   r6, XPD_ShiftInOut
        bis   r1, r1, kXPD_DataReceivedBit    // Has byte arrived?
        bc    VC, XPD_ReadByte_LOOP
        and   r1, r1, 0x07FF

        sub   sp, sp, 1
        ld    r6, sp, 0

        jsr   r6, r6

//=====
// Input Params:    r1 = The maximum number of read attempts
// Output Params:   r1 = The byte read from the SPI-UART
//-----
// Description:     Used to read a data byte from the SPI-UART. A zero byte is
//                  shifted out. This subroutine does not return until a byte
//                  has been received or the maximum number of attempts has been
//                  reached. The data is always in the LSB of the word.
//=====
XPD_ReadByteWithTimeout:
        st    r2, sp, 0
        st    r6, sp, 1
        add   sp, sp, 2

        add   r2, r1, 0                        // r2 = counter
        XPD_ReadByteWithTimeout_LOOP:
        bc    ZS, XPD_ReadByteWithTimeout_FAIL
        mov   r1, 0
        jsr   r6, XPD_ShiftInOut
        bis   r1, r1, kXPD_DataReceivedBit    // Has byte arrived?
        bc    VS, XPD_ReadByteWithTimeout_SUCCESS
        sub   r2, r2, 1
        bra   XPD_ReadByteWithTimeout_LOOP

        XPD_ReadByteWithTimeout_SUCCESS:
        and   r1, r1, 0x07FF
        bra   XPD_ReadByteWithTimeout_END

        XPD_ReadByteWithTimeout_FAIL:
        mov   r1, 0xFFFF

        XPD_ReadByteWithTimeout_END:
        sub   sp, sp, 2
        ld    r2, sp, 0
        ld    r6, sp, 1

        jsr   r6, r6

//=====
// Input Params:    r1 = The byte to write
// Output Params:   r1 = The byte read back
//-----
// Description:     Used to shift out a data byte to the SPI-UART and to shift
//                  back in another byte from the SPI-UART. The data is always
//                  in the LSB of the word.

```

```

//=====
XPD_ReadWriteByte:
    st    r6, sp, 0
    st    r1, sp, 1                // Push r1 because XPD_ReadConfigWord uses it
    add   sp, sp, 2

    XPD_ReadWriteByte_LOOP:
        jsr    r6, XPD_ReadConfigWord
        bic    r1, r1, kXPD_TransmitDoneBit    // Is transmit buffer empty?
        bc     VC, XPD_ReadWriteByte_LOOP
        sub    sp, sp, 1                // Pop r1
        ld     r1, sp, 0
        and    r1, r1, 0x07FF
        bis    r1, r1, 15
        jsr    r6, XPD_ShiftInOut

        sub    sp, sp, 1
        ld     r6, sp, 0

        jsr    r6, r6

//=====
// Input Params:    r1 = 16-bit word to write to the SPI-UART
// Output Params:   r1 = 16-bit word read back from the SPI-UART
//-----
// Description:     Used to shift out the word in r1 to the SPI-UART and to
//                  read back a word into r1. The MSB of the word is a control
//                  byte and the LSB is a data byte.
//=====
XPD_ShiftInOut:
    st     r0, sp, 0
    st     r2, sp, 1

    mov    r2, 1<<kSPI0CS_Semaphore
    outp   r2, SCUdown                // Resource down (semaphore)

    inp    r0, GPBin
    bic    r0, r0, 1
    bic    r0, r0, 2
    outp   r0, GPBout                // Assert SPI-UART chip select

    rol    r0, r1, 8                // Move MSbyte to r0
    outp   r0, SPI0tx                // Output MSbyte
    inp    r0, SPI0rx                // Get received MSbyte
    outp   r1, SPI0tx                // Output LSbyte
    rol    r1, r0, 8                // Move MSbyte received into data register
    inp    r0, SPI0rx                // Get received LSbyte
    ior    r1, r1, r0                // Move received LSbyte to data register

    inp    r0, GPBin
    bis    r0, r0, 1
    bis    r0, r0, 2
    outp   r0, GPBout                // Negate SPI-UART chip select

    outp   r2, SCUup                // Resource up (semaphore)

    ld     r0, sp, 0
    ld     r2, sp, 1

    jsr    r6, r6

#endif

```

Appendix D - Experimental Data

The experimental data collected during this research is in the possession of:

Rusty Baldwin, PhD, PE
Associate Professor of Computer Engineering
Department of Electrical and Computer Engineering
Air Force Institute of Technology
AFIT/ENG
Building 642
2950 Hobson Way
Wright-Patterson AFB, OH 45433-7765

voice: (937) 255-6565 ext. 4445 (DSN 785)
fax: (937) 656-4055 (DSN 986)
e-mail: rusty.baldwin@afit.edu
web site: <http://en.afit.edu/issa/faculty/drbbaldwin.htm>

Bibliography

- [Abr70] Abramson, Norman. "The ALOHA SYSTEM-Another Alternative For Computer Communications." *Proceedings of the 1970 Fall Joint Computer Conference. Houston, TX*, pages 281-285. American Federal Information Processing Societies, (November 1970).
- [Abr77] Abramson, Norman. "The Throughput of Packet Broadcasting Channels." *IEEE Transactions on Communications*, Vol. COM-25, No. 1: 117-128 (January 1977).
- [Bal99] Baldwin, Rusty O. *Improving the Real-Time Performance of a Wireless Local Area Network*. PhD dissertation. Virginia Polytechnic Institute and State University, Blacksburg VA, 1999.
- [BFO96] Bianchi, G., L. Fratta, and M. Oliveri. "Performance Evaluation and Enhancement of the CSMA/CA MAC Protocol for 802.11 wireless LANs." *The Seventh IEEE International Symposium on Personal, Indoor and Mobile Radio Communications PIMRC '96*, pages 392-396, October 1996.
- [BG92] Bertsekas, D. and R. Gallager. *Data Networks*. Prentice-Hall, NJ, 1992.
- [Bia00] Bianchi, Giuseppe. "Performance Analysis of the IEEE 802.11 Distributed Coordination Function." *IEEE Journal on Selected Areas in Communications*, Vol. 18(3):535-547 (March 2000).
- [CCG00] Cali, Frederico, Marco Conti, and Enrico Gregori. "Dynamic Tuning of the IEEE 802.11 Protocol to Achieve a Theoretical Throughput Limit." *IEEE/ACM Transactions on Networking*, Volume 8 , Issue 6, pages 785 – 799. Institute of Electrical and Electronics Engineers, 2000.
- [CN95] Crisler, K. and M. Needham. "Throughput Analysis of Reservation ALOHA Multiple Access." *Electronic Letters*, 31(2):87-89 (January 1995).
- [EE04] Eleven Engineering Inc., Address: 10150-100 Street, Suite 900, Edmonton, Alberta, Canada, T5J 0P6. Phone: (780) 425-6511. Fax: (780) 425-7006. <http://www.elevenengineering.com>.
- [IEEE99] IEEE Standard. *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*. 1999.
- [LARO02] LaRocca, James and Ruth LaRocca. *802.11 Demystified*. New York, Chicago, San Fransisco: McGraw Hill, 2002. ISBN: 0-07-138528-2.

- [Leu95] Leung, Y. W. "Generalised Multi-Copy ALOHA." *Electronic Letters*, 31(2):82-83 (January 1995).
- [LL03] Lin, An-Tai and Shie-Jue Lee. "A Modified Distributed Coordination Function for Real-Time Traffic in IEEE 802.11 Wireless LAN." *Proceedings of the 17th International Conference on Advanced Information Networking and Applications (AINA'03)*. Xidian University, Xian, China. (27-29 March 2003).
- [KL99] Kim, Jae Hyun and Jong Kyu Lee. "Capture effects of wireless CSMA/CA protocols in Rayleigh and shadow fading channels." *IEEE Transactions on Vehicular Technology*, Volume 48 , Issue 4, pages 1277 – 1286. Institute of Electrical and Electronics Engineers, July 1999.
- [KT75] Kleinrock, L. and F. A. Tobagi. "Packet Switching in Radio Channels: Part I: CSMA Modes and Their Throughput-Delay Characteristics." *IEEE Transactions on Communications*, COM-23(12):1400-1416 (December 1975).
- [KT85] Takagi, H. and L. Kleinrock. "Throughput Analysis for Persistent CSMA Systems." *IEEE Transactions on Communications*, Volume 33 , Issue 7, pages 627 – 638. Institute of Electrical and Electronics Engineers, 1985.
- [PDO02] Pal, Abhishek, Atakan Doğan, and Füsün Özgüner. "MAC Layer Protocols for Real-time Traffic in Ad-hoc Wireless Networks." *Proceedings of the International Conference on Parallel Processing (ICPP'02)* Vancouver, B.C., Canada. p. 539-546 (18 - 21 August 2002).
- [SK99] Sobrinho, João L. and A. S. Krishnakumar, "Quality-of-Service in Ad Hoc Carrier Sense Multiple Access Wireless Networks." *IEEE Journal on Selected Areas in Communications*, Vol. 17, No. 8, p. 1353-1368 (August 1999).
- [Tan89] Tanenbaum, Andrew S. *Computer Networks 2nd edition*. New Jersey: Prentice-Hall, Inc, 1989.
- [VZ95] Visser, M. A. and M. El Zarki. "Voice and Data Transmission Over an 802.11 Wireless Network." *6th IEEE International Symposium on Personal, Indoor, and Mobile Radio Communications*, Vol. 2, pages 648-652, 1995.
- [ZiA02] Ziouva, Eustathia and Theodore Antonakopoulos. "The Effect of Finite Population on IEEE 802.11 Wireless LANs Throughput/Delay Performance." *Electrotechnical Conference, 2002. MELECON 2002*, pages 95 – 99, 2002.

Vita

Captain Joshua D. Green, USAF, graduated from Foxborough High School in 1991. He attended Rensselaer Polytechnic Institute in Troy, NY for his undergraduate studies, graduating in May of 1995 with a Bachelor of Science Degree in Electrical Engineering.

His first assignment was to the 38th Engineering and Installation Wing, Tinker AFB, OK, where he first served as a Secure Computer Systems Engineer. He then served as the Systems Telecommunications Engineering Manager – Base Level (STEM-B) for Kuwait, where he supervised all communication planning and installation in Kuwait.

Capt Green's next assignment was to the 48th Communications Squadron at RAF Lakenheath, UK, where he served as a Flight Commander. He first commanded a Tactical Communications Unit. He then deployed to Vincenza, Italy and commanded a Mission Systems Flight. Finally, back at RAF Lakenheath, he commanded the Information Systems Flight.

He reported to the Air Force Institute of Technology (AFIT), Wright Patterson AFB, OH, in August of 2002. Upon completion of his Masters in Electrical Engineering, he will report to Head Quarter, Air Combat Command (HQ ACC) at Langley AFB, VA, to work as a Staff Officer.

“Things turn out best for those that make the best of the way things turn out.”
- Art Linkletter

This page intentionally left blank

REPORT DOCUMENTATION PAGE				Form Approved OMB No. 074-0188	
The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of the collection of information, including suggestions for reducing this burden to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to a penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.					
1. REPORT DATE (DD-MM-YYYY) 14-09-2004		2. REPORT TYPE Master's Thesis		3. DATES COVERED (From – To) May 2003 – June 2004	
4. TITLE AND SUBTITLE IMPLEMENTING INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS (IEEE) 802.11 STANDARD MEDIUM ACCESS CONTROL PROTOCOL FOR WIRELESS LOCAL AREA NETWORKS (LANS) ON A LABORATORY HARDWARE PROTOTYPE				5a. CONTRACT NUMBER	
				5b. GRANT NUMBER	
				5c. PROGRAM ELEMENT NUMBER	
6. AUTHOR(S) Green, Joshua, D., Captain, USAF				5d. PROJECT NUMBER	
				5e. TASK NUMBER	
				5f. WORK UNIT NUMBER	
7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(S) Air Force Institute of Technology Graduate School of Engineering and Management (AFIT/EN) 2950 Hobson Way WPAFB OH 45433-7765				8. PERFORMING ORGANIZATION REPORT NUMBER AFIT/GE/ENG/04-11	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) NSA/R5 Attn: Mr. William Kroah Ft. Meade, MD 20755 Comm: (301) 688-0348				10. SPONSOR/MONITOR'S ACRONYM(S)	
				11. SPONSOR/MONITOR'S REPORT NUMBER(S)	
12. DISTRIBUTION/AVAILABILITY STATEMENT APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED					
13. SUPPLEMENTARY NOTES					
14. ABSTRACT Wireless Local Area Networks (LANs) are extremely convenient, flexible, and easy to deploy. All LANs in which multiple hosts must access the same medium use a Medium Access Control (MAC) protocol to coordinate channel access. The MAC is part of the Data Link Layer of the Open Systems Interconnection (OSI) Reference Model. One MAC protocol in extensive use today is the Institute of Electrical and Electronics Engineers (IEEE) 802.11 standard. Since IEEE 802.11 devices are so prevalent in today's world, many researcher are exploring modifications and enhancements to the protocol. There are several well developed analytical and simulation models for IEEE 802.11 available to researchers, yet one significant obstacle remains: the lack of a means to obtain experimental data based on proposed protocol changes. Without real world experimental data, researchers lack the ability to test out their proposals in a real world environment. To fill this need, this thesis created a hardware prototype from which researchers can obtain experimental data about IEEE 802.11. This hardware prototype can now be used by researchers to gain real world data on their proposed modifications to IEEE 802.11.					
15. SUBJECT TERMS Local Area Networks, Wireless Communications, Computer Networks, Network Analysis, LAN					
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT UU	18. NUMBER OF PAGES 218	19a. NAME OF RESPONSIBLE PERSON Dr. Rusty O. Baldwin, PhD, PE (ENG)
REPORT U	ABSTRACT U	c. THIS PAGE U			19b. TELEPHONE NUMBER (Include area code) (937) 255-6565 ext. 4445; e-mail: Rusty.Baldwin@afit.edu

Standard Form 298 (Rev: 8-98)

Prescribed by ANSI Std. Z39-18