A WIRELESS CALL BUTTON NETWORK DESIGN

by

Punit Mukhija

Thesis submitted to the Faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

in

Electrical Engineering

June 11, 1999

Blacksburg, Virginia

Key Words: Wireless Networks, Controller Area Network, CAN, Call Button Network

A WIRELESS CALL BUTTON NETWORK DESIGN

Punit Mukhija

ABSTRACT

Traditional call button networks that control elevator systems utilize a wired connection for communication. The communication cables are run through the elevator shaft from one call button to another and finally to the controller on the roof. Installing this wired link is highly time consuming. In this thesis, we propose the design for a wireless call button network. Two important features of this wireless network design are low cost and low power consumption. Controller Area Network (CAN) is a widely used protocol for wired networks and has been proposed for use in next generation elevator control systems. A modified CAN for wireless (MCANW) protocol has been developed for the wireless call button network. The wireless link will be implemented via the use of data radios. A modified form of traditional Binary Phase Shift Keying (BPSK) modulation scheme for the radios is proposed. The proposed modulation scheme, like differential BPSK, can be detected non-coherently but it offers better performance than differential BPSK. Its implementation includes an innovative tracking algorithm to maintain synchronization at the receiver.

Acknowledgments

I would like to thank my advisor Dr. Bostian for his support and guidance. I would also like to thank Mr. Farley, Dr. Sweeney and Dr. W. Davis for reviewing this thesis report. I have enjoyed working with everyone at the Center for Wireless Telecommunications during my two year stay at Virginia Tech.

I would also like to thank Rusty Baldwin and Scott Marshall for always being ready to help me during the course of my thesis work.

I wish to thank my parents and my brother for their constant support and encouragement. Last but not least, I would like to thank my fiancée for allowing me to spend so much time working on my thesis.

Contents

List of Figures

List of Tables

Chapter 1
Introduction

An elevator call button network traditionally uses a wired link to communicate with a central controller. When a call button in a floor hall is depressed, the request for an elevator is sent to the central controller. The central controller then processes this request and directs an elevator to the requested floor. The communication link between the controller and the call buttons is formed by cables that permanently reside in the elevator shaft. The main problem with such a system is that installation of this wired network is highly time consuming. This thesis proposes a design for a wireless call button network. The proposed network would not require the wired link between the call buttons and the central controller. The wireless network design will utilize the elevator shaft for radio frequency (RF) propagation to minimize the network's power consumption. We will also study a low cost data radio design to implement the wireless link. A modified form of Binary Phase Shift Keying (Modified BPSK or MBPSK) is proposed for the data radios. This modulation scheme, like differential BPSK (DBPSK), allows the receiver to perform non-coherent detection. However, unlike DBPSK, MBPSK does not require a training sequence and offers a bit error rate half that of DBPSK. A tracking algorithm which allows the receiver to maintain synchronization is proposed. The modulation-demodulation algorithms and the tracking algorithms for MBPSK are presented in detail. The data radio design includes the use of low cost 1-bit A/D converters. Two popular methods that are used to achieve multi-bit performance from 1-bit analog-to-digital (A/D) converters are analyzed and compared.

The contributions of this thesis are :

1) A conceptual design for a wireless call button network.

2) Design and testing of a modified form of BPSK (MBPSK) that can be implemented in a DSP and is particularly suited to the call button application.

1.1 Thesis Organization

The thesis is organized in six different chapters. This chapter introduces the thesis and explains the organization of this document.

Chapter 2 discusses the design of a wireless call button network and proposes a protocol for this network. The proposed protocol is based on the Controller Area Network (CAN) protocol. CAN protocol is used to implement the current wired call button network.

Chapter 3 will look at some key aspects of the wireless link. A suitable frequency band for the deployment of the link is studied. We will also look at the advantages of using the elevator shaft as the propagation path for the radio waves. A low cost data radio design will be analyzed and a suitable data rate for the network is proposed.

Chapter 4 will discuss the MBPSK modulation-demodulation scheme for the data radios in detail. A tracking algorithm which allows the receiver to maintain synchronization is proposed. The strength of this algorithm is that it is not computationally intensive. An MBPSK transmitter and receiver have been implemented using two DSP evaluation boards. The receiver utilizes a non-coherent detector implemented in software. This chapter will also discuss how a software detector's performance can be tested under various signal-to-noise ratios. Performance results of the implemented software detector are presented as well.

Chapter 5 proposes a 1-bit A/D converter be used instead of a more expensive multi-bit A/D converter. This chapter will study two different methods that allow a 1-bit A/D converter to achieve the performance of a multi-bit A/D converter. We will analyze the advantages and disadvantages of both these techniques. Based on this analysis, one of the two techniques is selected for our data radios.

The final chapter will summarize the work that has been completed towards implementing a wireless call button network. We will also look at what work remains to be done in this project.

Chapter 2

Call Button Network

Controller Area Network (CAN) is a protocol used widely in wired networks. The objective of this chapter is to analyze CAN and to modify it for a wireless call button network. This chapter has been divided into three sections. Section 2.1 provides an overview of CAN. Section 2.2 proposes a wireless call button network and a new protocol for such a network based on CAN. Section 2.3 discusses the conditions required for the two protocols (i.e. standard CAN and the modified CAN for wireless networks) to work transparently in one system.

2.1 Controller Area Network

CAN was originally developed by Bosch GmBh for the automotive industry and has traditionally been used for wired systems. Each CAN message has a unique identifier that defines the type of data contained in the message. This identifier is also used to specify the priority level of the message. The message is transmitted to every node in the system via the shared serial bus. Based on the identifier in the message, a node decides whether to accept or reject the message. Wired network systems using CAN utilize a common serial bus that acts as a wired-AND gate with the various nodes on the network. The serial bus is high when data is not being transmitted. This means that, if several nodes transmit at the same time, then the bit stream seen on the serial bus will be the result of AND-ing all the individual bit streams from the various transmitters. The wired-AND implementation means that a logical '0' is the dominant state while a logical '1' is the recessive state. For instance if Node A transmits the bit stream [0 0 0 ...] and at the same time Node B transmits the bit stream [1 1 1 ...], the bit stream seen on the serial bus will be [0 0 0 ...]. The wired-AND implementation means that when a transmitting node outputs a data stream, this data stream will not necessarily show up on the shared serial bus. Each transmitting node also monitors the serial bus to see whether or not its transmitted data stream appears on the bus.

If the bus is idle, any node is allowed to start transmitting. What happens if two nodes start transmitting at the same time? The identifier in each message is also used to assign priority. The message with the lowest number identifier is given priority in a wired-AND implementation. For instance let us consider the situation when Node A and Node B start transmitting at the same time. Node A is assigned higher priority than Node B. Node A is given the identifier [0 ...] while Node B has the identifier [1 ...]. When Node A transmits a binary 0 and Node B transmits a binary 1, the wired-AND implementation causes the serial bus to output a binary 0. Both the transmitters are also monitoring the serial bus at this point. Node B realizes that its output did not show up on the serial bus and therefore, there must be a node with a higher priority trying to transmit at the same time. This causes Node B to stop transmitting until the bus is inactive. In other words, CAN's arbitration method utilizes a form of Carrier Sense Multiple Access with Collision Detection (CSMA/CD). CAN's form of CSMA/CD is different from the traditional CSMA/CD because with CAN there is one transmitting node whose identifier will be dominant. The non-dominant nodes will stop transmitting, allowing the node with the lowest identifier to take control of the serial bus.

Figure 2.1 illustrates the CAN 2.0A message frame.

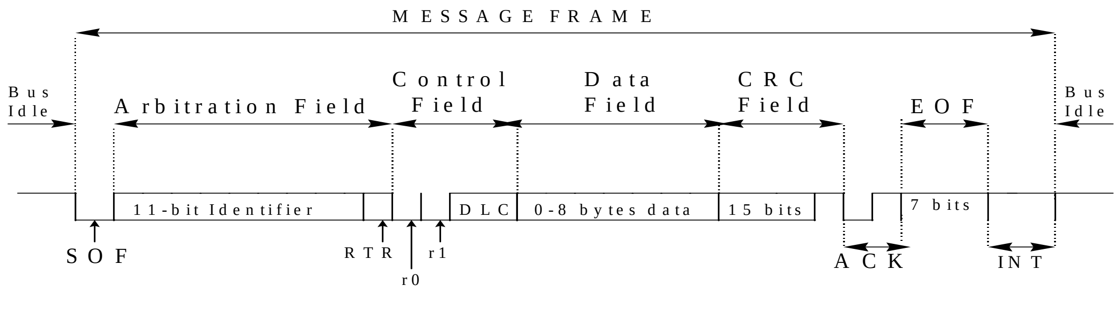


Figure 2.1 CAN 2.0A message frame

A Standard CAN protocol (version 2.0A) message frame consists of seven different bit fields. A description of the seven bit fields follows.

1. The *Start of Frame (SOF) Field* indicates the beginning of the message frame. This field consists of one bit, which is set low.
2. The *Arbitration Field* consists of an 11-bit identifier and the Remote Transmission Request (RTR) bit. The 11-bit identifier can be used to identify more than two thousand unique nodes. The RTR bit indicates whether the message frame contains data or if data is being requested. When data is requested, the RTR bit is set high and the Data Field is omitted from the frame. If data is being requested then the frame is called a remote frame.
3. The *Control Field* consists of six bits. Two bits (r0 and r1, both reserved for future use) are set low. A four bit Data Length Code (DLC) indicates the number of bytes in the Data Field.
4. The *Data Field* can be 0 to 8 bytes long.
5. The *CRC Field* contains a fifteen bit Cyclic Redundancy Check (CRC) and a delimiter bit, which is set high.
6. The *ACKnowledge (ACK) Field* consists of two bits. The first bit is the Slot Bit that is transmitted as high. In a wired link, when a receiving node gets the message it sets the Slot Bit low. Each transmitter monitors the bus while transmitting. When the Slot Bit is set to low by a receiving node, the transmitting node knows that the message has been received and acknowledged [Sch98]. The second bit in the ACKnowledge Field is a delimiter bit, which is set high.
7. The *End of Frame (EOF) Field* consists of seven high bits.

The EOF Field is followed by an intermission consisting of three high bits. When there is no message, the idle bus remains high.

2.1.1 Error Detection in CAN

CAN provides excellent error detection capabilities. Wired networks using CAN utilize a total of five error detection mechanisms; three at the message level and two at the bit level [Sch98]. The first mechanism at the message level is a 15-bit CRC. The CRC is computed by the transmitter and is based on the transmitted message. The receivers compute a CRC based on the message received and compare this with the received CRC to check for errors. A second error detection scheme at the message level is a frame check scheme. Certain predefined bit values are transmitted at certain points in each message frame [Sch98]. If the receiver does not receive all of these preset bits, then it knows that an error has occurred. The SOF Field, the bits r0 and r1 in the Control Field, both of the bits in the ACKnowledge Field, and the entire EOF Field act as frame checks. The third error detection mechanism at the message level is an acknowledgment error check. If the transmitted message is not acknowledged then an ACK error is flagged by the transmitter.

The first bit level error detection scheme is bit monitoring. The transmitter unit continuously monitors the bus during transmission. When the transmitted bit is not the same as the actual bit level then a bit error is flagged. If this happens while the Arbitration Field is being transmitted, then the transmitter stops transmitting and no error is detected. (The Arbitration Field carries the 11-bit identifier which is used to assign priority levels.) Bit monitoring also does not flag an error during the ACK Slot when the receiver is acknowledging a message. The second bit level error detection scheme is bit stuffing. After five consecutive identical bit levels have been transmitted, the transmitter inserts (or stuffs) a bit of the opposite polarity into the bit stream. Bit stuffing is not used to replace the CRC delimiter bit or in the ACK and EOF fields [Sch98]. Receivers automatically delete (or de-stuff) such bits before processing the message in any way. If a receiver detects more than five consecutive bits of the same level, it detects an error. When any unit on the network detects an error, an error frame is transmitted on to the serial bus. The error frame can be a maximum of 19 bits long. All the receiving nodes on the network use the rising and falling edges of the transmitted bits to achieve synchronization. Since all the nodes are synchronized to each other, several nodes can transmit the error message at the same time.

As stated before CAN utilizes a form of CSMA/CD to arbitrate access to the bus. Collision detection implies that a transmitted bit is continuously monitored by the transmitting node. Bit monitoring is also used to acknowledge message reception and check for global errors. The wireless channel does not allow a transmitting node to monitor the state of a bit at the receiving end. This means that CAN would require some modifications before it can be used in our wireless network. The next section will present the design of the wireless call button network and propose a modified version of CAN suitable for this network.

2.2 Wireless Call Button Network

The wireless call button network will operate in a master-slave configuration. Data radios will be used to connect the various nodes to the wireless link. The central controller will act as the master, with the call button radios playing the role of the slaves. The central controller will communicate with all the call buttons while the call buttons will only communicate with the central controller. For example, consider a situation in which a single central controller communicates with *Y* number of call button radios (y1, y2, y3,…, y*Y*). The central controller will first communicate with the call button radio y1. The controller is able to poll y1 by placing an address uniquely associated with that call button in the Arbitration Field of the message frame. Using the 11-bit identifier in the Arbitration Field, more than 2,000 unique addresses can be generated. The controller can place a unique identifier associated with a slave radio in the Arbitration Field, and it can set the Remote Transmission Request (RTR) bit high to request information. This message frame will be received by every slave radio on the network. Using the address in the Arbitration Field, all radios will know that the message is intended for only y1. The only slave that will be allowed to talk is y1. y1 will be required to respond to the central controller's poll within a fixed time if it wants to request an elevator. If y1 responds and requests an elevator, then the central controller will send a message to y1 acknowledging this request. If y1 does not respond, the controller will assume that y1 does not require an elevator. Anytime a radio call button does not respond within the allotted time, the controller will move on to poll the next call button radio. Next, the controller will poll y2, and so on. Table 2.1 shows the sequence of tasks that will be performed in the wireless call button network.

Table 2.1 Call button network task sequence

1) controller polls *y1*
2) *y*1 requests elevator
3) controller acknowledges message from *y*1

In Table 2.1, tasks 2 and 3 are only performed if the call button desires an elevator. A call button radio will normally remain in a dormant state to minimize its power consumption. When the UP or DOWN button is depressed, the radio will be awakened. Once awakened, the call button radio will go into receive mode and wait until it is polled. Tasks 2 and 3 are performed when the call button receives the controller's poll. Section 2.3.2 describes what happens if an error is detected at any stage in Table 2.1.

CAN supports a Data Field that can be up to 8-bytes long and it uses a 4-bit Data Length Code (DLC) to indicate the length of the Data Field. However, the tasks in Table 2.1 can all be accomplished using a 1-byte long Data Field. (This will be illustrated later in this chapter.) In this situation, using a DLC to allow a variable Data Field is a waste of bandwidth. CAN message frames also use an ACKnowledge Field that is virtually useless in a wireless network. With these two factors in mind, a Modified CAN for Wireless (MCANW) systems is proposed.

2.2.1 Modified CAN for Wireless (MCANW)

The only difference between CAN and MCANW is that MCANW does not use the Control Field and the ACKnowledge Field. (The Control Field in CAN holds the DLC and allows the Data Field to be variable.) MCANW will utilize two different frame formats. Figure 2.2 shows the message frame format that will be used by the master to poll the slaves. This format will be referred to as the short frame format. The short frame is 39 bits long.

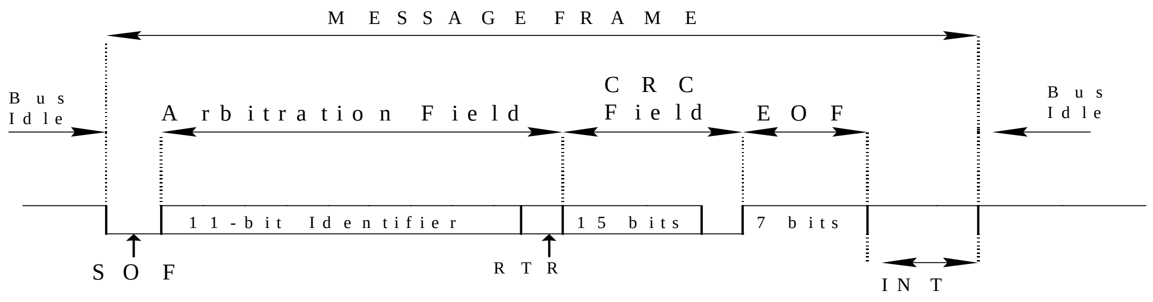


Figure 2.2 MCANW short frame format (39 bits)

As shown in Figure 2.2, the 11-bit identifier allows more than 2,000 unique nodes to be addressed on the network. This identifier will be used to indicate which slave is being polled. A 15-bit CRC (Cyclic Redundancy Check) code is used for error detection. As Figure 2.2 shows, the master will not use a Data Field while polling the slaves. The RTR (Remote Transmission Request) bit will be set high when the master is polling a slave. The short frame is identical to a CAN remote frame but with the Control Field and the ACKnowledge Field stripped off.

Figure 2.3 shows the message frame format that will be used for tasks 2 and 3 in Table 1. This format will be referred to as the long frame format. The long frame is 47 bits long.

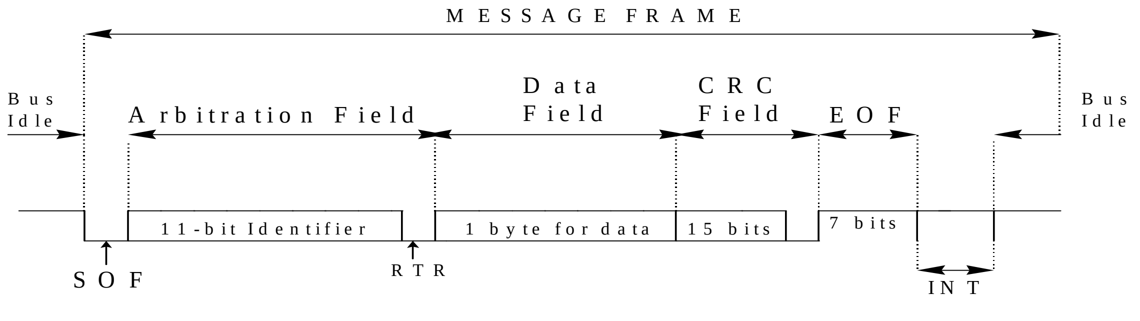


Figure 2.3 MCANW long frame format (47 bits)

The 11-bit identifier will be used to identify the slave in question. This means that the address of the call button radio will be placed in the 11-bit identifier when it requests an elevator and when the master acknowledges this request. The Data Field in a long frame is set to be 1-byte long. When the slave is requesting an elevator, one bit in the Data Field will be used to indicate the sequence number of request. The first time a call button requests an elevator the sequence number 0 is used. If an acknowledgment from the master is not received by the next poll, the call button will send another request with sequence number 0. The sequence number is changed by the slave when an acknowledgment for an elevator request is received. However, the next request will only be sent when the call button is depressed after the promised elevator arrives. Requests are never sent in the time period in which an acknowledgment has been received but the elevator has not yet arrived. The sequence numbers will allow the master to know if another elevator needs to be sent out for the latest request. The master only dispatches an elevator if the latest sequence number is different from the last sequence number for which it sent an elevator. On receiving an elevator request, the master always transmits an acknowledgment message. Only one bit in the Data Field is needed to acknowledge an elevator request. This bit indicates the sequence number for the request which is being acknowledged. One bit is needed to indicated whether the UP or DOWN call button was depressed. Next generation call buttons will also be used to monitor some safety and emergency switches. The remaining six bits in the Data Field allow each call button to monitor seven switches. Figure 2.4 shows the sub-fields in the Data Field.

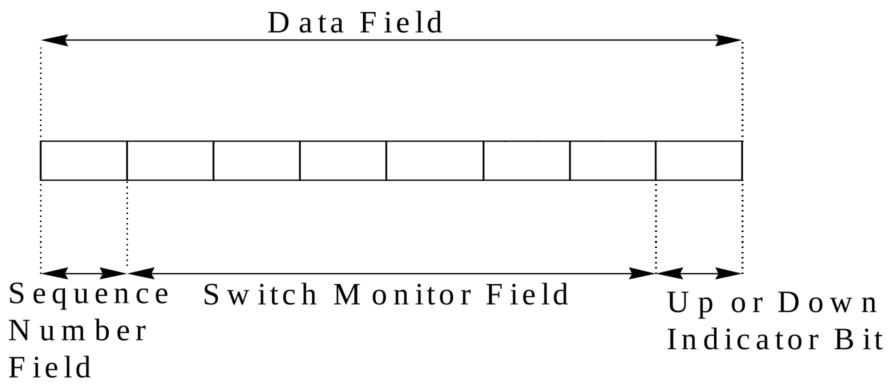


Figure 2.4 Sub-fields in a MCANW Data Field

The address of the master never appears in the message frame. A slave node only communicates with a single master. Anytime a slave receives a message with its address as the identifier, it knows that the message was sent by the master.

2.2.2 Error Detection in MCANW

As mentioned in section 2.1.1, CAN utilizes five error detection mechanisms. MCANW will be able to employ three of these five error detection mechanisms. MCANW will use the Cyclic Redundancy Check (CRC), frame checking and bit stuffing to detect errors.

The CRC will be MCANW's most powerful error detection scheme. The CRC generator polynomial, g(x), used by CAN is shown in Equation 2.1.

$$g(x) = x^{15} + x^{14} + x^{10} + x^{8} + x^{7} + x^{4} + x^{3} + 1 \qquad (2.1)$$

The generator polynomial divides the unstuffed SOF, Arbitration Field, Control Field, and the Data Field (if present). The 15-bit remainder acts as the checksum and is appended to the information bits. This CRC is best suited for a total code length of 127 or less [Wic95]. The generator polynomial, g(x), provides a minimum distance 6 code. A minimum distance of 6 implies that 5 or fewer random bit errors can always be detected. CAN's CRC is used for error detection only and error correction is not performed.

CRC codes provide very powerful burst error detection capabilities as well. A q-ary cyclic code detects at least the fraction of burst errors shown in Equation 2.2.

$$\text{fraction of burst errors always detected} = 1 - \left(\frac{q^{1-r}}{q-1} \right) \qquad (2.2)$$

where r = degree of g(x)

q = 2 (for a binary cyclic code)

CAN's CRC is able to detect the fraction ($1 - 6.10 \cdot 10^{-5}$) or more of all burst errors. The probability of undetected error is approximated by Equation 2.3. Equation 2.3 evaluates the probability that 6 bit errors occur and that the error burst is undetected by the CRC.

$$\text{Probability of undetected error} \cong (6.10 \cdot 10^{-5}) \binom{n}{6} (P_e)^6 (1-P_e)^{n-6} \qquad (2.3)$$

where n = code length

P_e = probability of single bit error

The code length is the sum of the information length and the checksum length. MCANW frames can be either 49-bits or 57-bits long. Figure 2.5 plots the bit error ratio (BER) versus the probability of undetected error. In Figure 2.5, the code length, n, is assumed to be 57.

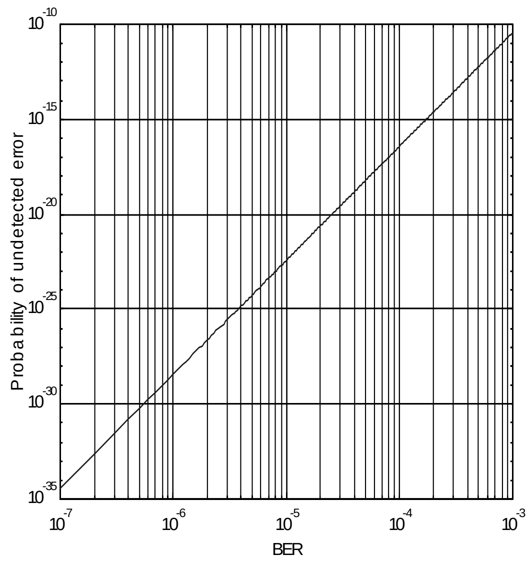


Figure 2.5 BER versus probability of undetected bit error

Figure 2.5 shows that even if the receiver's BER is only 10^{3}, the probability of an error being undetected by the CRC is 10^{-10}.

2.3 Interfacing CAN and MCANW

The existing wired call button network utilizes CAN. CAN micro-controllers monitor the call buttons and operate the central controller. As stated previously, data radios will be utilized to make this network wireless. Figure 2.6 illustrates the proposed network.

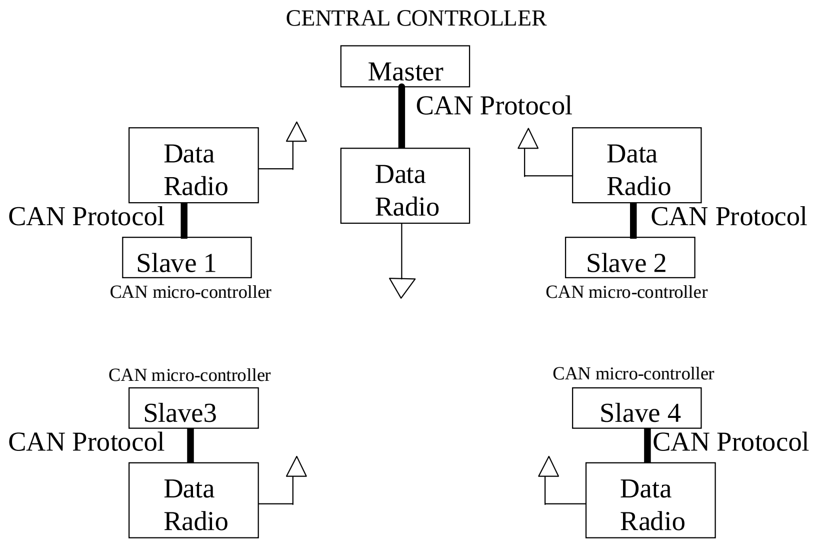


Figure 2.6 Proposed wireless call button network

The previous section showed that the wireless call button network could function with MCANW. This means that the data radios will only need to transmit MCANW frames. However, the slaves and the master controller still only communicate using CAN. This means that when the master controller wants to poll a call button, the master will generate a CAN frame. This frame will be read by a DSP or micro-controller in the data radio. The radio will strip off the Control Field and the Acknowledge Field and transmit the MCANW frame. All the data radios know that there is no Data Field in a short frame and that a long frame has a one byte long Data Field. This will allow a receiving radio to insert the appropriate Control Field and ACKnowledge Field and convert the MCANW frame into a valid CAN frame. The data radio will then pass this CAN frame onto its CAN micro-controller.

2.3.1 Fooling a CAN Micro-controller

A CAN transmitter is always monitoring the transmitted message. If a receiver does not acknowledge the message, the transmitter will believe an error has occurred. To avoid this situation the DSP in the data radio must 'fool' the transmitter into believing that the message has been correctly received. This would require the serial line between the data radio's DSP and the CAN transmitter to implement a wired-AND gate. When the transmitter places the recessive ACK Slot Bit on the line, the DSP must put out a dominant bit to change the logic level on the bus. When a CAN unit sees the dominant bit in the ACK Slot, it will believe that the message has been successfully received. At the receiving end, when a message is sent from the data radio to a CAN micro-controller, the CAN micro-controller will change the ACK Slot Bit to indicate that the message has been received correctly. At the receiving end, the wired-AND gate implementation of the serial bus between the data radio and the CAN micro-controller would be required to prevent a short circuit.

2.3.2 Procedure Upon Detecting an Error

When a CAN micro-controller detects an error in the received message an error frame is generated. This error message is never transmitted by the data radio connected to that particular CAN micro-controller. However, MCANW follows the protocol shown below in the case of an error being detected.

A call button CAN micro-controller receives an error corrupted poll from the master. The slave CAN micro-controller will assume that the poll was meant for some other slave and will not respond. The slave CAN micro-controller will be allowed to transmit only when it receives an error free poll.

The master CAN micro-controller receives an elevator request with errors. The master will retransmit the last polling message. This will allow a slave to repeat the request for an elevator. An upper bound will be set such that the master re-polls a slave only a limited number of times in one round of polling.

The call button CAN micro-controller detects an error in a message from the master acknowledging the request for an elevator. The call button CAN micro-controller will assume that the master controller was not able to receive the request for an elevator. The call button will wait till it is polled again, and then request an elevator. Each request will include a sequence number of 0 or 1. The first time a call button requests an elevator the sequence number 0 is used. If an error free acknowledgment from the master is not received by the next poll, the call button will send another request with sequence number 0. The sequence numbers will allow the master to know if another elevator needs to be sent out for the latest request. If the controller has already sent out an elevator for request number 0 for a particular call button, then it will transmit an acknowledgment but not dispatch another elevator. On the very next request, the controller will only send an elevator to this call button if the sequence number 1 is used. This procedure is illustrated in Figure 2.7. In Figure 2.7, the call button requesting an elevator is Y1. The variable Y1_last is originally initialized to 1.

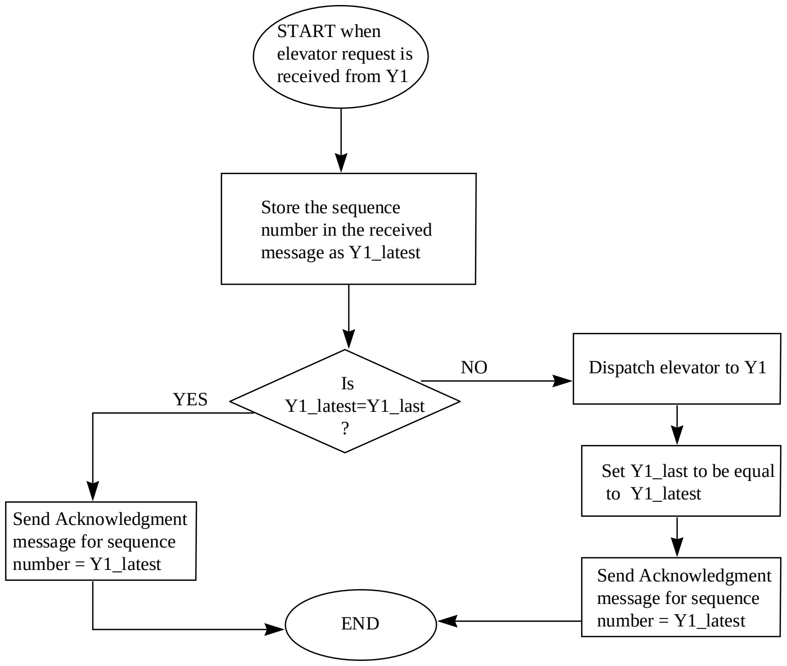

START when elevator request is received from Y1
Store the sequence number in the received message as Y1_latest
Is Y1_latest=Y1_last ?
YES
NO
Dispatch elevator to Y1
Set Y1_last to be equal to Y1_latest
Send Acknowledgment message for sequence number = Y1_latest
Send Acknowledgment message for sequence number = Y1_latest
END

Figure 2.7 Dispatch elevator algorithm

The call button sends a request for an elevator after its UP or DOWN button has been depressed. The call button will keep on transmitting the same request until it receives an acknowledgment from the controller. However, even if the UP or DOWN button is depressed again, no more requests are sent after the acknowledgment is received and before the promised elevator arrives.

2.4 Summary

A wireless call button network design has been proposed. The network will use MCANW. MCANW frames are CAN frames without the Control and Acknowledgment Fields. MCANW frames are either 39-bits long or 47-bits long. The corresponding frames in CAN are 47-bits long and 55-bits long respectively. Therefore, CAN in this system would only utilize the bandwidth with an efficiency of 83 or 85.5 percent in comparison to MCANW. Transmit data from a CAN micro-controller can be fed into a data radio at *R* bps, buffered, and transmitted wirelessly at rate of approximately 0.85*R* bps. MCANW will fully utilize CAN's 15-bit CRC. Methods to allow the existing wired CAN network to be transformed into a wireless MCANW network have also been proposed. The design of the wireless call button network aims to minimize the power consumption. The slave data radios stay in a dormant state until a call button is depressed.

Chapter 3

The Wireless Link

Chapter 2 discussed the design of a wireless call button network. Chapter 3 will analyze several design parameters critical to the wireless link. Section 3.1 proposes a suitable frequency for the deployment of the wireless network. Section 3.2 makes a strong argument in favor of using the elevator shaft for RF propagation. Section 3.3 will outline the design of the data radio and discuss the data rates of the system.

3.1 Choosing a Frequency Band

A suitable frequency band that allows the network to be deployed in the United States, Europe and Japan must be chosen. This means that the selected frequency band must adhere not only to the US regulations, but also to those of Europe and Japan. Table 3.1 shows the unlicensed frequency allocation power limits for US, Europe and Japan. It shows that a 12.5 MHz wide spectrum is available from 2471-2483.5 MHz in all of these locations.

Table 3.1 Unlicensed frequency allocation RF power limits

BAND	FCC (US)	ETSI (EUROPE)	MPT (JAPAN)
902-928 MHz	<1W	N/A	N/A
2400-2483.5 MHz	<1W	<100mW	N/A
2471-2497 MHz	N/A	N/A	<10mW/MHz
5725-5850 MHz	<1W	<100mW	N/A

Source : Adapted from [Zyr98]

3.1.1 Microwave Oven Interference in the 2.4 GHz Band

The 2.4 GHz band can be used world wide for the deployment of the wireless call button network. However, the FCC (Federal Communications Commission) does not set a limit on the radiated energy in the 2.45 GHz ISM band for ISM equipment like microwave ovens. Fortunately, the FDA (Food and Drug Administration) does set a limit for the power density for microwave oven emissions. This maximum power density, P_d, is 5 mW/cm^2 at any point 5 cm or more away from the external surface of the oven. However, FDA tests [Far93] found that actual microwave emissions are under 2mW/ cm^2. Equation 3.1 can be used to calculate the effective isotropic radiated power (EIRP).

$$EIRP = (P_d)(\text{area}) \tag{3.1}$$

We will assume that an average microwave oven has the dimensions (70cm x 50cm x 50cm). These dimensions give the microwave oven a total surface area of 1.7m^2. Using Equation 3.1, with area = 1.7m^2 and P_d = 2 mW/cm^2, the maximum *EIRP* is calculated to be 34 W. We will assume that approximately 5% of the total maximum radiated power is picked up by our receiver antenna. Therefore, the power of the interference from the microwave oven is approximated to be at 1.7 W or 32.3 dBm. Reports [NTIA1], [NTIA2] from the National Telecommunications and Information Administration (NTIA) also show that the maximum EIRP for residential microwave ovens lies between 16 and 33 dBm. Therefore, we will assume that the EIRP from a microwave oven is 33 dBm. The next step is to analyze the effect of microwave oven emissions on the operation of our wireless network. Using Equation 3.2, we can find the path loss seen by the microwave interference in an indoor environment. The parameter *d* represents the total distance between the transmitter and the receiver. d_0 is the distance between the transmitter and the first obstruction in the signal path.

$$L_{indoor}(d) = L(d_0) + 10n\log_{10}\left(\frac{d}{d_0}\right) \tag{3.2}$$

where n = Path loss exponent; typical range of n is $3 \le n \le 5$

d = Distance between transmit and receive antennas (m)

d_0 = Initial free space propagation distance (m)

L = Propagation loss for the free space path (dB)

L_{indoor} = Total propagation loss indoors (dB)

The path loss suffered by a RF signal in free space is given by Equation 3.3.

$$L(d) = 20\log\left(\frac{4\pi d}{\lambda}\right) \tag{3.3}$$

where L = free space path loss (dB)

d = distance between the transmitter and the receiver (m)

λ = wavelength of transmission (m)

We will assume that the microwave oven and the data radio receiver are on the same floor, and therefore, use a path loss exponent, n, of 3. Furthermore, we will assume that d = 10m and d_0 = 3m. The path loss under these conditions is found to be more than 75 dB. Section 3.2 will propose that the elevator shaft be used as a path for the RF propagation in the call button network. Any external interference will suffer at least 8 dB of loss when it passes through the walls of the elevator shaft. This means that if the microwave oven does output 33 dBm of power, then the interference seen by the data radio is at (33-75-8) = -50 dBm. We will assume that the microwave oven interference occupies the entire band and, therefore, it is not possible to avoid the interference. If the noise power is approximately -50 dBm and an SNR of 10 dB or above is desired, the signal power must be at least -40 dBm.

It is important to note that in the above observations and calculations, several conservative estimates have been used. First of all, most microwave oven models do not radiate 33 dBm of EIRP. Secondly, the microwave oven interference consists of a CW tone at about 2.45 GHz and broadband noise over tens of megahertz wide. We have correctly assumed here that the broadband noise occupies our entire band. However, the broadband noise carries a lot less energy than the CW signal. For our conservative estimates we have assumed that the broadband noise is also 33 dBm strong. Thirdly, the path loss model uses n = 3, which assumes that the microwave oven is on the same floor as the depressed call button.

In spite of our conservative estimates, for the moment let us assume that the desired signal strength at the receiver is as high as -40 dBm. The elevator shaft acts as a *waveguide* and therefore, offers low path loss. We will assume that the path loss is 2 dB/floor and also use a 20 dB fade margin. Under these conditions, a 0 dBm transmitted signal from 10 floors away would have the required signal strength of -40 dBm. In other words, even with our conservative estimates, only 1 mW of transmit power is required if the receiver and transmitter are 10 floors apart.

Our analysis showed that the call button network can overcome the interference due to a microwave oven when the oven is 10 meters away from a call button radio. The question that remains to be answered is what happens if the microwave oven is placed right next to a depressed call button radio or if there are several microwave ovens in operation? In such a situation, the error detection capabilities of the network are utilized. CAN's CRC will allow the call button radio to detect errors if noise or interference causes substantial signal degradation. Figure 3.1, shows the probability of an undetected error under varying SNRs for DBPSK[1] modulation schemes. (Chapter 4 will discuss the modulation scheme to be used by the data radios in more detail. For now we will assume that radios will utilize DBPSK.)

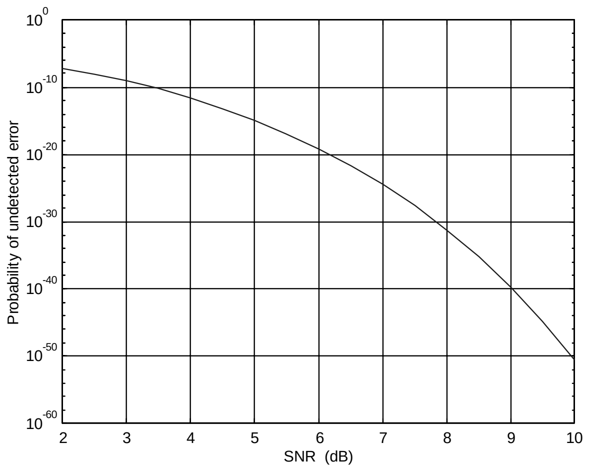


Figure 3.1 SNR (dB) versus probability of undetected error using CAN's CRC

[1] In BPSK and DBPSK modulation Eb/No = 2SNR

3.2 RF Propagation Path

One of the goals of this thesis is to design a wireless network which minimizes power consumption. Figure 3.2 compares the elevator shaft path loss with the path loss expected in an indoor non-line of sight (NLOS) channel and the path loss encountered in a line of sight (LOS) channel.

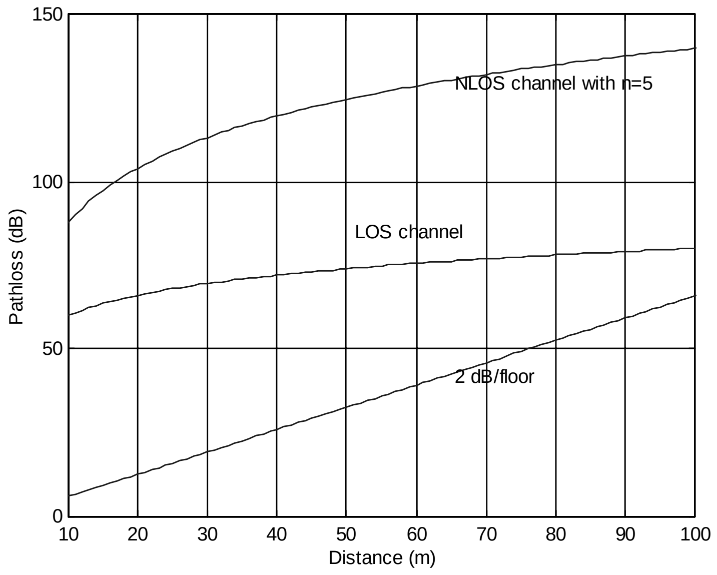


Figure 3.2 Path loss at 2.4 GHz for various propagation paths

Two important observations can be made from Figure 3.2. First of all, it shows that using an indoor NLOS channel for RF communication at 2.4 GHz is highly power intensive. In a 10 floor building, with consecutive floors being about 3 meters apart, the central controller and one of its slaves could be separated by 30 meters. Figure 3.2 shows that at 30 meters, the NLOS channel path loss would be almost 120 dB. If an NLOS channel was employed, the central controller would not be able to directly communicate with all of its slaves even if a 1W transmitter was used. When the controller needs to poll a call button radio 10 floors away, the controller would have to use radios on intermediate floors as repeaters before the message would reach its destination. Thus, the controller would have to form a chain, in which it would transmit to one data radio, which would then transmit to another, and so on until the message is finally transmitted to the intended call button radio. This scheme greatly increases the power consumption of the system and the probability of error.

Figure 3.2 also illustrates the low path loss suffered by a signal in the elevator shaft. The reason for this low path loss is because the elevator shaft acts in a similar fashion to a waveguide. A waveguide can fully contain a propagated wave within its bounds. If the elevator shaft is modeled as a rectangular waveguide, then the cutoff frequency, Fc, of the dominant waveguide mode would be given by Equation 3.4.

$$Fc = \frac{C}{2a} \tag{3.4}$$

where C = velocity of light in free space (m/s)

a = longer side wall of the rectangular waveguide (m)

Fc = cutoff frequency (Hz)

Equation 3.4 shows that even if the long dimension of the rectangle is only 1 meter (i.e. a=1), then the cutoff frequency is 150 MHz. This explains why the propagation studies found the path loss at 2.4 GHz in the elevator shaft to be less then the path loss in free space. However, because the elevator shaft is not an ideal rectangular waveguide, there are losses even at frequencies above the cutoff frequency. We have estimated these losses to be approximately 2 dB/floor. Figure 3.2 strongly suggests that the elevator shaft be utilized as the propagation path to minimize the power consumption of the network.

We must consider the situation when the elevator shaft is used for propagation but the elevator is situated in between the transmitter and the radio. It is likely that the elevator will greatly attenuate the signal in such a case. One way to get the signal to the receiver would be through the use of a passive repeater. Figure 3.3 illustrates this concept.

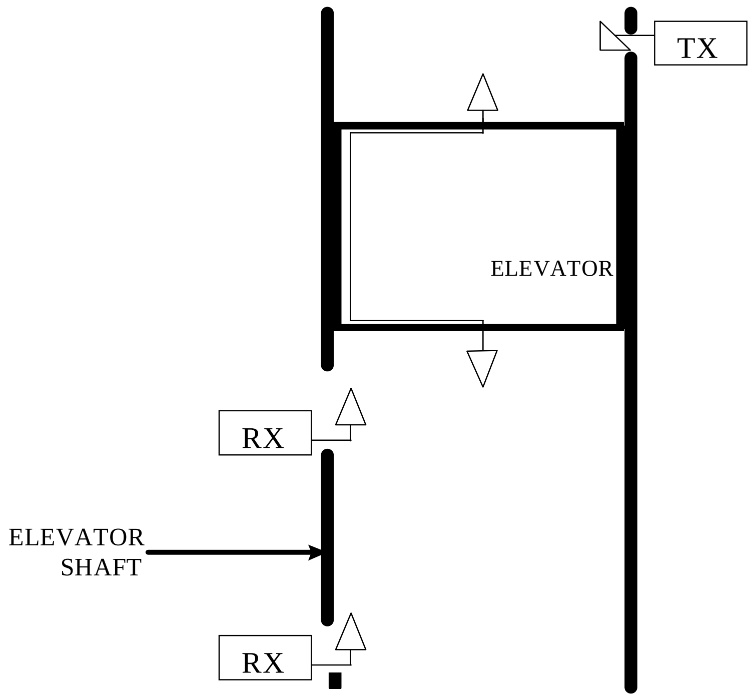


Figure 3.3 The use of a passive repeater to transmit past the elevator

Figure 3.3 shows two antennas connected to the elevator, one on the top of the elevator and the other on the bottom. A coaxial cable, which acts as a transmission line, is used to connect these antennas. Any signal from a transmitter would be picked up by an antenna on one side of the elevator, and then it would be transmitted out from the antenna on the other side of the elevator.

3.3 Data Radio

This section will outline a simple yet practical design for the data radios and discuss the data rates of the network. A form of Binary Phase Shift Keying (BPSK) modulation will be used because of its performance and ease of implementation. BPSK signals do not offer the best bandwidth utilization. However, this particular drawback of BPSK is not a severe limitation since we will be operating at relatively low data rates. (Section 3.3.1 will discuss the data rates of the system in more detail.)

The wireless call button network proposed in Chapter 2 will work satisfactorily with half duplex data radios. The data radios will utilize *smart frequency hopping*. The premise behind smart hopping is to utilize only those frequency channels that offer favorable SNRs. At the most basic level of smart hopping, any node wishing to transmit would first be required to go into receive mode and scan the entire band. The node would then know which channels are relatively noise and interference free. The node would then transmit a test sequence in one of the vacant channels. In the meantime, all the other nodes would be scanning the band, looking for this test sequence. This scheme does not offer protection against localized noise sources near receivers. The design and implementation of sophisticated smart hopping algorithms will require additional research and will not be pursued further in this thesis. The smart frequency hopping algorithms will be developed in the next stage of this project and are not part of this thesis.

A DSP will be needed in the data radio for base band processing. Texas Instruments' C54x series of DSPs is aimed at the wireless market. C54x DSPs come in speeds up to 100 million instructions per second (MIPS). C54x DSPs also offer varying sizes of on-chip memory. Most importantly, the low cost and low power consumption of these DSPs make them ideal for our network. Several versions of the C54x DSPs cost less than \$10. The C5402 costs \$5.00 in large quantities and operates at 100 MIPS. The chief features of the C5402 are highlighted in Appendix A.

Figure 3.4 shows the design of the data radios. CAN frames to be transmitted are fed into the DSP. The DSP will output the wireless data which will be modulated using a BPSK modulator. The BPSK signal will be fed into a power amplifier (PA) and transmitted via a transmit/receive (T/R) switch.

On the receiver end, the signal is passed through a T/R switch, low noise amplifier (LNA) and a BPSK demodulator. The demodulator output is fed into the DSP for detection. A programmable frequency synthesizer allows the DSP to control the frequency hopping.

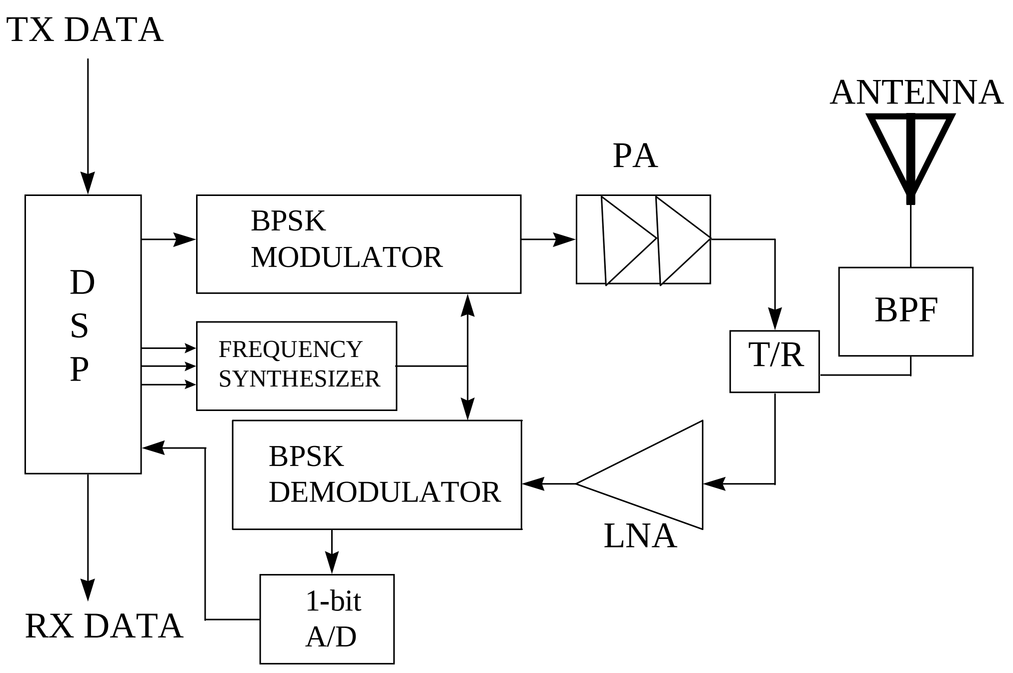


Figure 3.4 Frequency hopping data radio

While the architecture shown in Figure 3.4 looks conventional, this radio design has two unique features. Firstly, the modulation scheme utilized is not traditional BPSK or DBPSK and will be discussed further in Chapter 4. Secondly, a 1-bit A/D converter is used instead of a more expensive multi-bit A/D converter. Chapter 5 will discuss A/D conversion practices that use a 1-bit converter but achieve multi-bit converter performance.

3.3.1 Data Rate

A low data rate, which translates to a smaller bandwidth, is preferable in wireless systems for two reasons. Firstly, a smaller bandwidth means that the IF filter in the receiver will allow less noise into the detector. A wider bandwidth would require us to increase the transmit power. Secondly, in the case of frequency selective fades, there is a higher probability that equalization will not be required if the signal of interest has a small bandwidth. Since our data radios do not perform equalization, a smaller bandwidth is very attractive for our system. If our bandwidth is small enough, we may be able to transmit in a deep faded channel as long as the transmit power is sufficient. However, a lower data rate also means that the tasks in the network take longer to accomplish. The lower the data rate, the longer it takes for the elevator to come to a particular floor after the call button has been depressed.

The high level block diagram of the wireless network (which was first described in Chapter 2) is shown again in Figure 3.5. Figure 3.5 shows that each data radio will communicate with a CAN micro-controller using CAN protocol. The data radios will then convert CAN frames to MCANW frames, add overhead bits for smart hopping and transmit this data through the wireless channel.

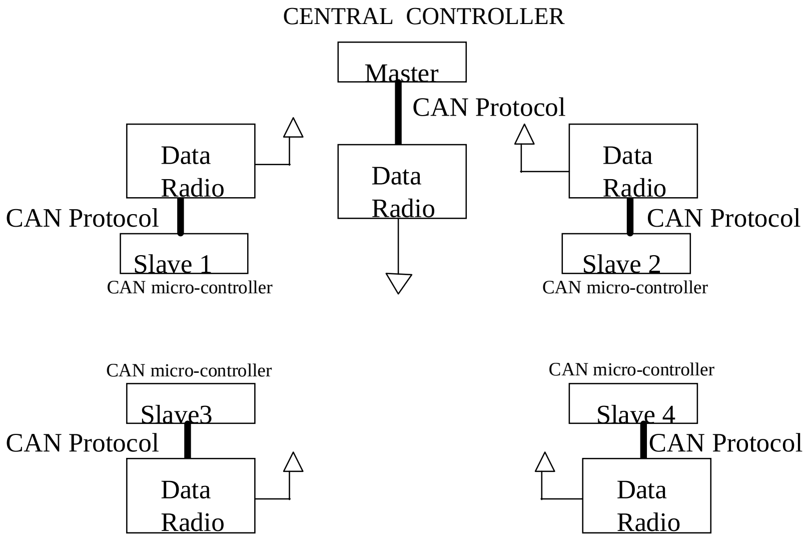


Figure 3.5 Wireless call button network

We will assume that the CAN micro-controllers communicate with their data radios at 50 kbits/s. Chapter 2 showed that CAN utilizes the bandwidth with only 85 percent efficiency in comparison to MCANW. Therefore, MCANW would be operating at 42.5 kbits/s when CAN operates at 50 kbits/s. Due to the smart hopping scheme, overhead bits will be added to the MCANW frames to allow synchronization in time and frequency. This will mean that the wireless rate of transmission may be considerably greater than 42.5 kbits/s. It is not yet possible to know accurately as to how much of the bandwidth will be consumed by the smart hopping schemes. However, we will assume that the wireless data rate will be less than 80 kbits/s after the addition of the overhead bits.

Implementation may require that the data radios communicate with their CAN micro-controllers at a high data rate. For instance, CAN data may be fed into the DSP at 1 Mbits/s, buffered, and converted into the data to be transmitted wirelessly. The rate of CAN data is not critical to our analysis at this point. The two key transmission rates that we are focusing on are the rates of the MCANW data and the rate of the wireless data. The MCANW data will influence the delays in our network while the wireless data rate will dictate the transmitter power needed.

The next section will analyze the required receiver performance given that the wireless data rate is 80 kbits/s. In this section, we will look at the delays in the network assuming that the MCANW data rate is 42.5 kbits/s. Table 3.2 shows the tasks performed in the network. In this table, y1 is one of the call button slaves in the network.

Table 3.2 Call button network task sequence

1) controller polls *y1*
2) *y*1 requests elevator
3) controller acknowledges *y*1's message

Task 1 will be performed for each call button. However, tasks 2 and 3 are only performed if the call button desires an elevator. As shown in Chapter 2, it takes one MCANW short frame to accomplish task 1 and one long frame each for tasks 2 and 3. The length of a short frame is 39 bits while a long frame is 47 bits long. When task 3 is completed, the call button will light the indicator bulb showing that an elevator is on its way. The maximum time between a call button being depressed and the indicator being lit is the time it takes to accomplish the tasks in Table 3.2 for all the call buttons in the network. Equation 3.4 calculates this maximum time lapse.

$$T_{max} = n\left[\left(\frac{39Y}{R}\right)+\left(\frac{2(47)X}{R}\right)\right] \qquad (3.4)$$

where Y = total number of slaves on the network

X = number of slaves that request an elevator

n = number of times on average that each task has to be repeated due to errors being detected at the receiver

R = MCANW data rate (42.5 kbits/s)

T_{max} = maximum time (in seconds) between call button being depressed and the indicator being lit

We will assume that 20% of the call button slaves request an elevator and that on average each task has to be repeated twice (i.e. X=0.2Y and n=2). Figure 3.6 plots the maximum time lapse between a call button being depressed and the indicator being lit versus the size of the network.

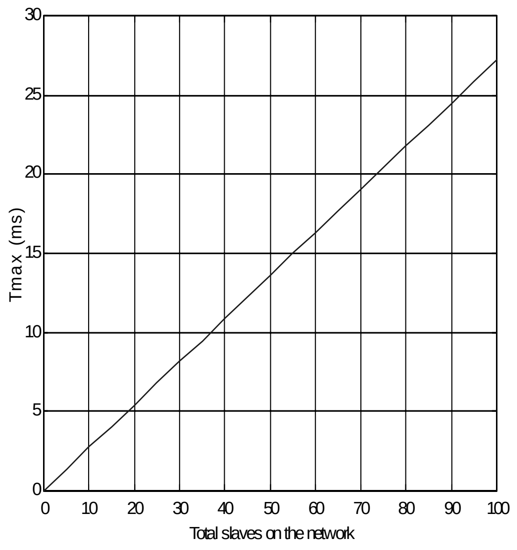


Figure 3.6 T_{max} versus Y with R = 42.5 kbits/s, n = 2, and X=0.2Y

Figure 3.6 shows that under the given conditions, T_{max} is less than 30 ms in a network that has one hundred slaves. Now consider the situation when half the call buttons on the network request an elevator, and due to in-band noise or interference, each task has to be repeated an average of five times. Figure 3.7 plots the time period between a call button being depressed and the indicator being lit versus the size of the network when X=0.5Y and n=5.

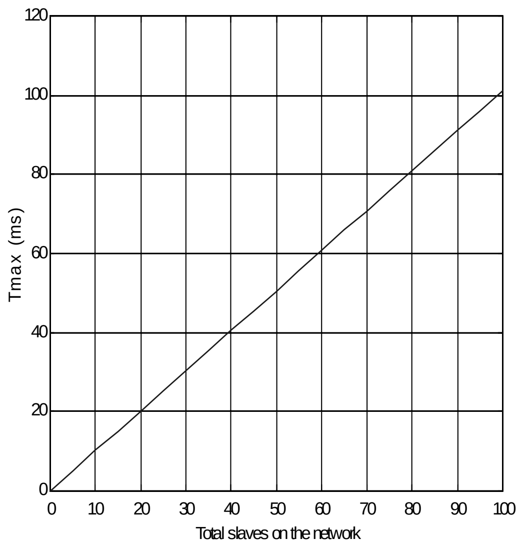


Figure 3.7 T_{max} versus Y with R = 42.5 kbits/s, n = 5, and X=0.5Y

Figure 3.7 shows that under these demanding conditions, T_{max} is approximately 100 ms in a network that has one hundred slaves. Therefore, it would not take more than 0.1 second for the indicator light to come on after the call button has been depressed. In other words, operating MCANW at 42.5 kbits/s will not cause any noticeable delays in the network, even under unrealistically adverse conditions.

3.3.2 Receiver Performance

To estimate the maximum separation distance between the transmitter and the receiver, we must analyze the receiver's performance. In the previous section we made several conservative estimates regarding microwave oven transmissions. In this section we will be more realistic and assume that we can hop around the in-band noise and interference.

Equation 3.5 shows the received signal power calculated in terms of the receiver noise floor and the system's SNR.

$$(\textit{Received Power}) = (\textit{Noise Floor}) + (\textit{SNR}) \text{ (dBm)} \qquad (3.5)$$

where $(\textit{Noise Floor})$ = receiver noise floor (dBm)

$(\textit{SNR})$ = signal-to-noise ratio for the system (dB)

The noise floor (in dBm) of the receiver can be calculated using Equation 3.6.

$$(\textit{Noise Floor}) = 10\log_{10}(KT_A) + NF + 10\log_{10}(B) + 30 \qquad \text{(dBm)} \qquad (3.6)$$

where K = Boltzmann's constant ($1.38\text{x}10^{-23}$ J/K)

T_A = antenna temperature

(set at 290°K assuming antenna is seeing ambient thermal noise)

NF = receiver Noise Figure (dB)

B = intermediate frequency (IF) bandwidth (Hz)

We will assume that the wireless data rate is 80 kbits/s. Setting the IF bandwidth at 160 KHz, the noise floor can be calculated in terms of the receiver noise figure. This is done in Equation 3.7.

$$(\textit{Noise Floor}) = (-122 + NF) \qquad \text{(dBm)} \qquad (3.7)$$

Substituting Equation 3.7 in Equation 3.5 produces Equation 3.8.

$$(\textit{Received Power}) = (-122 + NF) + (\textit{SNR}) \quad \text{(dBm)} \qquad (3.8)$$

An *SNR* of 10 dB is equivalent to an Eb/No of 13 dB for DBPSK systems. A 13 dB Eb/No provides a bit error rate (BER) of approximately 10^{-9} when DBPSK is used. However, for a conservative estimate, a 25 dB margin is used and it is assumed that the required *SNR* is 35 dB. A receiver with a 12 dB noise figure can be built at a very reasonable cost. Using these values of *NF* and *SNR*, the received power from Equation 3.8 is found to be -75 dBm. This means that if the path loss in the elevator shaft is 2 dB/floor, then the transmitter and the receiver can be more than 35 floors apart.

3.4 Summary

The 2.4 GHz band has been proposed for the deployment of the wireless call button network. Microwave ovens in this band are not likely to cause severe signal degradation. However, even in the case of extremely low SNR, CAN's CRC is able to provide very powerful error detection.

Smart frequency hopping data radios have been proposed for the wireless call button network. The data radios will utilize a modified form of BPSK and 1-bit A/D converters. The next two chapters will analyze these features of the data radios in further detail. The proposed MCANW data rate is 42.5 kbits/s and the wireless transmission rate is expected to be less than 80 kbits/s.

Chapter 4

Modified Binary Phase Shift Keying

The data radios will utilize Binary Phase Shift Keying (BPSK). Two forms of BPSK are commonly used : plain BPSK which requires coherent detection and differential BPSK (DBPSK) which may be detected by non-coherent means. BPSK provides roughly 3 dB improvement in performance over DBPSK. However, DBPSK is often preferred over BPSK because DBPSK does not require coherent detection. The basic premise of DBPSK is that the carrier phase is changed by 180° if the current data bit is different from the previous data bit. Figure 4.1 illustrates this concept. In this figure, the DBPSK signal is plotted versus time samples. A new bit appears after every 24 time samples. DBPSK does require the transmitter to send a training sequence. In Figure 4.1 the training sequence is [0 0] and lasts until time sample 48. The phase change at time sample 49 indicates that the third bit is a binary 1. The next phase change at time sample 73 indicates that the fourth bit is a binary 0. Therefore, the sequence shown in Figure 4.1 is [0 0 1 0].

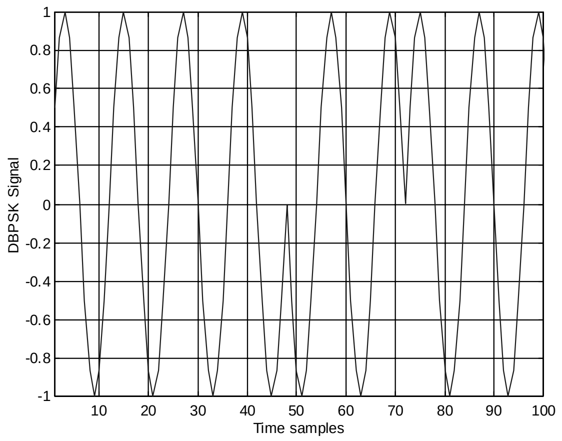


Figure 4.1 DBPSK signal versus time samples

In DBPSK, the data is encoded and decoded differentially, which means that if the receiver detects one bit incorrectly, it will also detect the next bit incorrectly. In other words, each bit error at the receiver will lead to two bit errors. A simple solution to this problem is to avoid differentially encoding the data. Instead of differential coding, a phase change of 180° can be introduced in the middle of every binary 1. If the receiver does not detect a phase change in the middle of a bit, then it knows that a binary 0 was received. Figure 4.2 illustrates this modified BPSK (MBPSK) modulation scheme. In Figure 4.2, each bit is again 24 time-samples wide. The first bit lasts from sample number 1 to sample number 24, and this bit is a binary 0 since there is no phase change in it. The second bit (from sample number 25 to 48) is also a binary 0. The third bit has a 180° phase change in the middle (at sample 60); therefore, it is a binary 1. The next bit (sample number 73 to 96) is a binary 0. The sequence of bits transmitted is again [0 0 1 0]; however, this time the bits are not coded differentially.

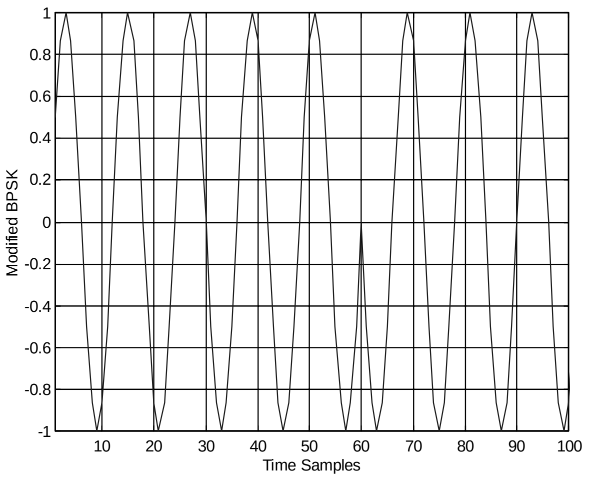


Figure 4.2 Modified BPSK for non-coherent detection

DBPSK modulation introduces a phase change between bits while MBPSK introduces a phase change in the middle of a bit. A phase change in DBPSK tells the receiver that the current bit is different from the previous bit, whereas a phase change in MBPSK tells the receiver that the current bit is a binary 1. MBPSK has two important advantages over DBPSK. First of all, an MBPSK receiver does not require a training sequence. A DBPSK receiver does require a training sequence because there is a phase ambiguity of 180°. Secondly, since the phase change in MBPSK is in the middle of a bit, a detection error only causes a single bit error. A phase change in the middle of a bit means that this phase change carries information regarding only one bit. However, in DBPSK the phase change is between two bits. If a detection error is made in DBPSK two bit errors occur. In spite of these differences, the bandwidth of an MBPSK signal and a DBPSK signal (carrying the same data) are identical.

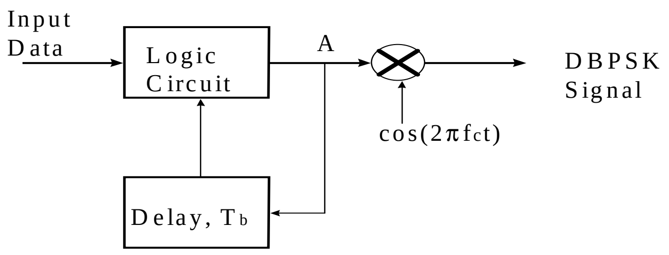
This chapter will study MBPSK modulation-demodulation in software. A tracking algorithm which uses the MBPSK detector computations to maintain synchronization at the receiver has been developed. The modulation-demodulation and the tracking algorithms are presented in Section 4.1. An MBSK transmitter and receiver were implemented using two C54 DSP evaluation boards. Section 4.2 discusses the implementation and testing of this MBPSK link.

4.1 MBPSK Implementation in Software

In this section, we will analyze how an MBPSK transmitter and receiver can be implemented in software. Three important tasks for our MBPSK link are implemented in software. Firstly, the data to be transmitted is prepared for modulation before being fed into a BPSK modulator. Secondly, the MBPSK signal is detected in software after it is down-converted. Lastly, the receiver maintains synchronization during the detection process using a tracking algorithm.

4.1.1 MBPSK Transmitter

Traditionally DBPSK modulation is implemented using a logic circuit, as shown in Figure 4.3. In this figure, T_b represents one bit period.



Source : Adapted from [Rap96]

Figure 4.3 DBPSK transmitter

The logic circuit in Figure 4.3 can change its output level only when there is a logical transition at the input. This is what requires the phase changes in DBPSK to be between bits. In MBPSK we want a phase to be in the middle of a bit. MBPSK would not be easy to implement using traditional logic gates; however, MBPSK implementation in software is a trivial task. Figure 4.4 shows the flowchart to calculate **A**, the input to the BPSK modulator. (The BPSK modulator is a mixer.) In Figure 4.4, T_b represents one bit period.

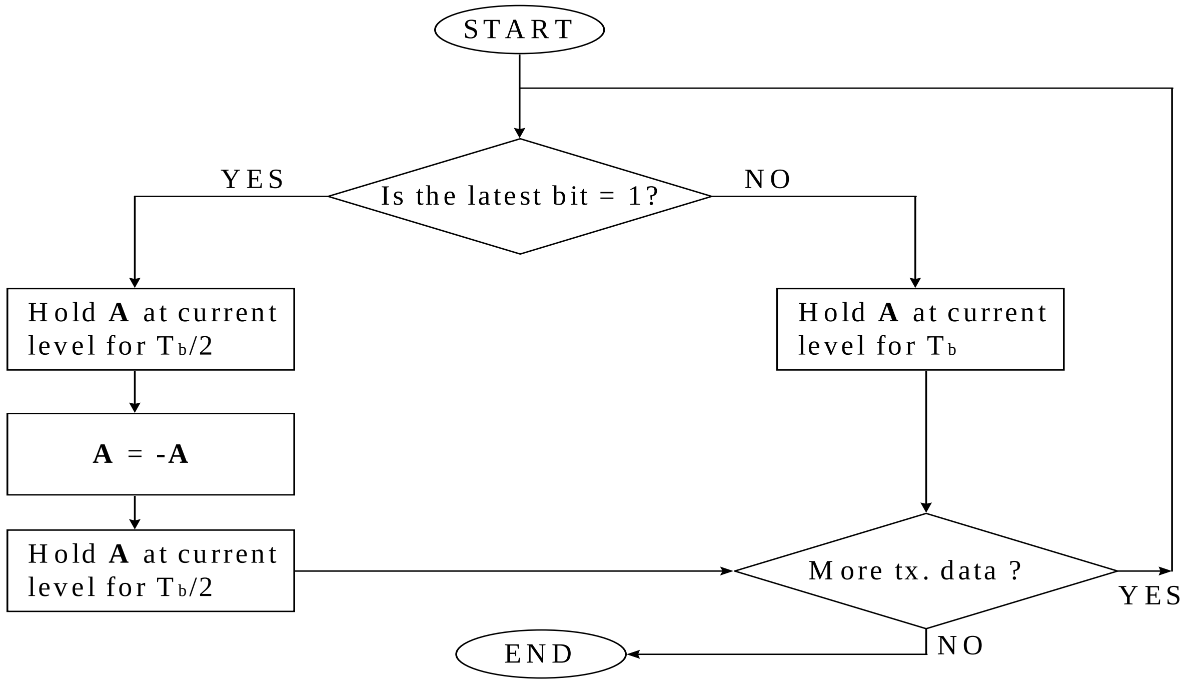
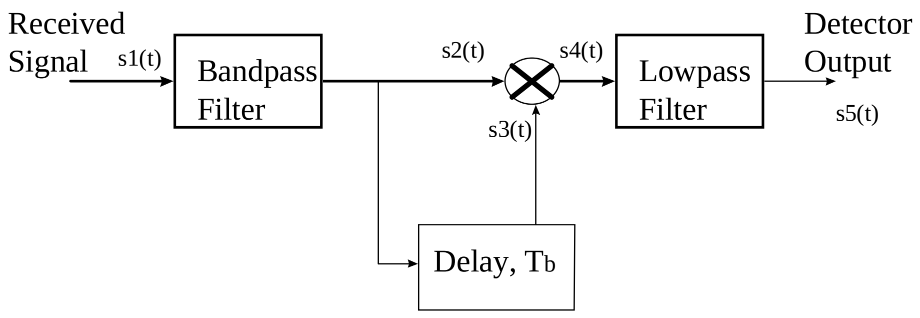

START
Is the latest bit = 1?
YES
NO
Hold A at current level for Tb/2
A = -A
Hold A at current level for Tb/2
Hold A at current level for Tb
More tx. data ?
YES
NO
END

Figure 4.4 MBPSK flowchart to calculate BPSK modulator input

4.1.2 MBPSK Receiver

The MBPSK detection process is very similar to the method utilized for DBPSK detection. First, let's take a look at the DBPSK detector shown in Figure 4.5.



Source : Adapted from [Rap96]

Figure 4.5 DBPSK receiver

The output of the bandpass filter is given by Equation 4.1. The output of the delay block is given by Equation 4.2. In this particular case, the filter output and the delay block output are 180° out of phase since they have opposite polarities. The variable *t* represents time and f_c is the carrier frequency.

$$s2(t) = A\cos(2\pi f_c t) \qquad (4.1)$$

$$s3(t) = -A\cos(2\pi f_c t) \qquad (4.2)$$

The output of the mixer is given by Equation 4.3.

$$s4(t) = s2(t)s3(t) = -A + A\cos(4\pi f_c t) \qquad (4.3)$$

The lowpass filter only allows the DC term to pass through. Therefore, in the case when s2 and s3 are out of phase by 180° the detector output is a negative voltage level. When s2 and s3 are of the same phase, the detector output is a positive voltage level. Therefore, for detection purposes we are just interested in knowing whether the DC term is positive or negative.

The DBPSK detector is slightly modified to form an MBPSK detector. In the DBPSK detector a delay of T_b was used to compare the phase of the current bit with that of the previous. The delay block for an MBPSK detector delays the signal by $T_b/2$. This allows us to compare the phase of the first half of the current bit with the second half of the current bit. An MBPSK detector algorithm has been developed with the C54 DSP assembly instruction set in mind. Before we look at this algorithm, let us take a look at one particular instruction in the C54. The multiply-accumulate (MAC) instruction is used to perform element by element multiplication of Buffer1 with Buffer2 followed by addition of all the products. Buffer1 contains the samples from the first part of the bit while Buffer2 contains the samples from the second part of the bit. Buffer1 and Buffer2 contain N/2 elements each. The MAC instruction will perform the calculation shown in Equation 4.4. Equation 4.4 shows that a total of (N/2) products are added to obtain the MAC output.

$$\text{MAC output} = \text{Buffer1}[1]\text{Buffer2}[N/2] + \text{Buffer1}[2]\text{Buffer2}[(N/2)-1] + \ldots + \text{Buffer1}[N/2]\text{Buffer2}[1] \qquad (4.4)$$

The sign of the MAC output is exactly what we need for the detection process. If the MAC output is negative then the detector output bit is a binary 1, otherwise the detector output bit is a binary 0.

The flowchart in Figure 4.6 illustrates how an MBPSK detector can be implemented in software.

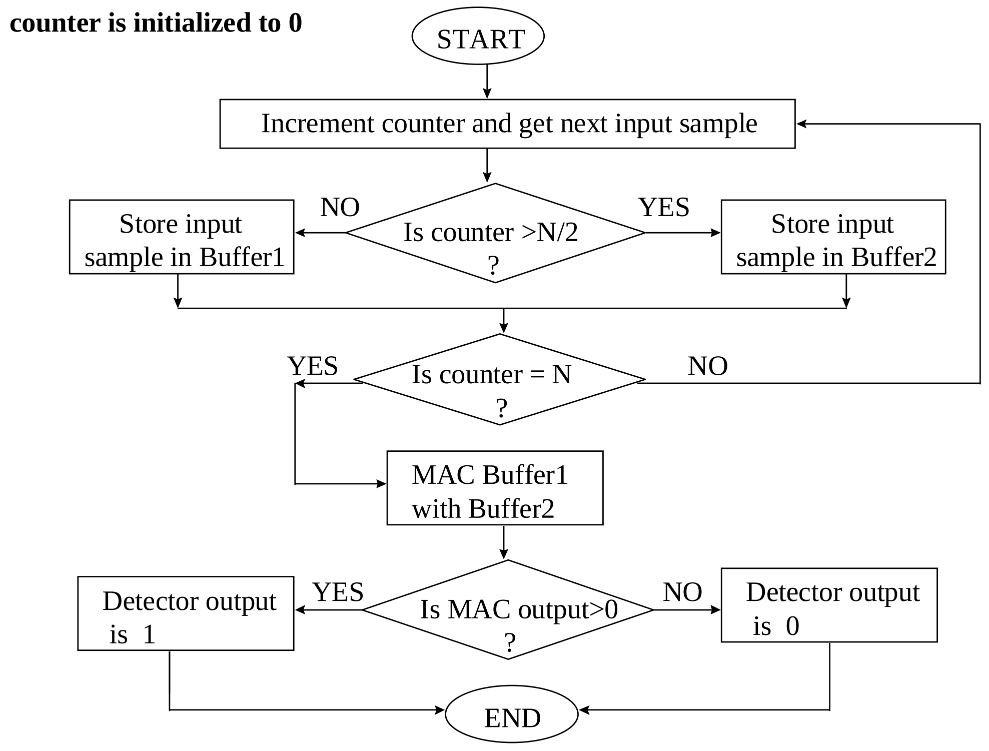


Figure 4.6 MBPSK detector flowchart for the C54

4.1.3 Maintaining Synchronization

The MBPSK receiver will need to achieve clock synchronization before the detection of data can occur. The initial synchronization will involve synchronization in time and in frequency. Once synchronization has been attained, it must be maintained during the detection process. In this subsection we present an algorithm to maintain clock synchronization. Before we present the actual algorithm, we will take a closer look at the detection process. The computations performed in the detection process will be used to maintain synchronization.

Figure 4.7 shows the received signal fed into the DSP which the receiver believes to be one entire bit. The phase change in the middle of the bit shows that this bit is a binary 1.

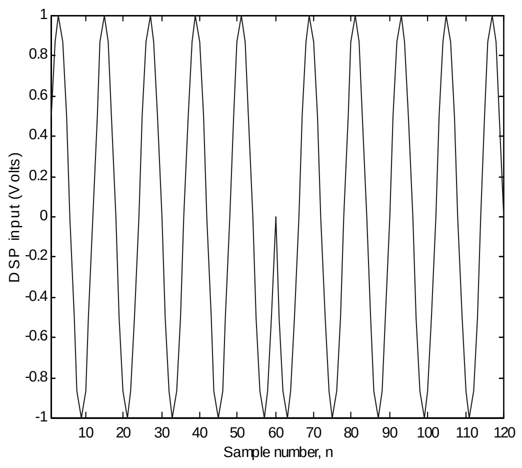


Figure 4.7 Receiver DSP input

This figure shows that the receiver is very well synchronized with the transmitter since the phase change is precisely in the middle of the bit. One way to look at this is that the received signal is divided up using different windows of the same size. Each window represents a different bit. If there is a phase shift, it should be in the middle of a window. In Figure 4.7, we are actually looking at one such window. The fact that the phase shift is in the middle indicates that the receiver is well synchronized with the transmitter. In this case the MAC output is -30. Now we will consider the situation when the receiver is not well synchronized with the transmitter. Figure 4.8 shows the received signal fed into the DSP, which the receiver again believes to be a single bit.

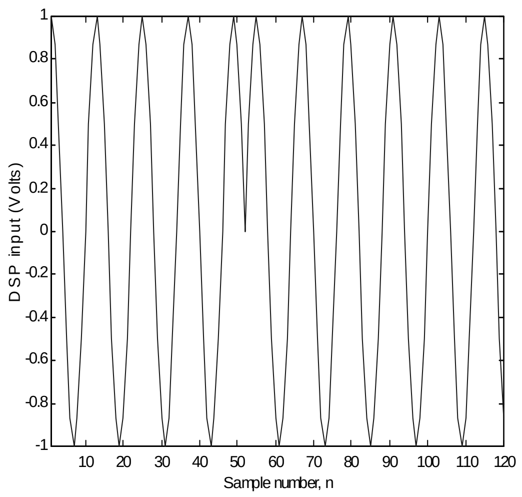


Figure 4.8 Receiver losing synchronization with the transmitter : case 1

In Figure 4.8 the phase shift is not in the middle of our window. We would want the window to be moved to the left for good synchronization. The MAC output result in this case is -22. Now consider the situation when the phase shift occurs in the second half of the window. The receiver's DSP believes the signal shown in Figure 4.9 represents one bit.

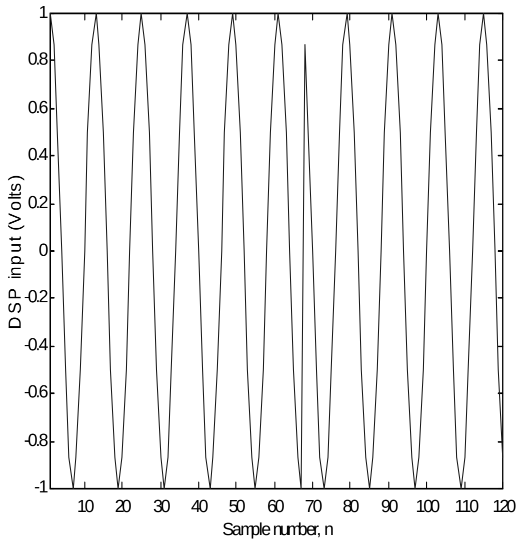


Figure 4.9 Receiver losing synchronization with the transmitter : case2

The MAC output for Figure 4.9 is -22. The MAC output calculated for Figures 4.8 and 4.9 is the same because the receiver's window center is ten samples away from the center of the received bit in both these figures.

The important point to note in the above analysis is the following : *the magnitude of the MAC output for a binary 1 is highest when the phase shift is precisely in the center of the window.* When the MAC output is a negative number it means that a phase shift was detected and that the received bit is a binary 1. The position of our window will affect the magnitude of the MAC output. For instance if the window has a phase shift precisely in the middle it will produce a MAC output larger in magnitude than when the phase shift is in the left or right half of the window. This means we can move our window by a sample or two and see which position produces the largest magnitude of MAC output. The first time a binary 1 is received we can store the MAC output in the variable NORMAL. Now after this we move our window to the left by one sample. The next time we receive a binary 1 we store the MAC output in the variable BACKWARD. We now move the window two samples to the right. The next time a binary 1 is received, we store the MAC output in the variable FORWARD. After these three binary ones we can compare the values of NORMAL, BACKWARD and FORWARD. This will tell us the best location of the window and we will move the next window accordingly. The next time a binary 1 is received we restart this process and calculate NORMAL and so on. This algorithm is presented in a flowchart form in Figure 4.10.

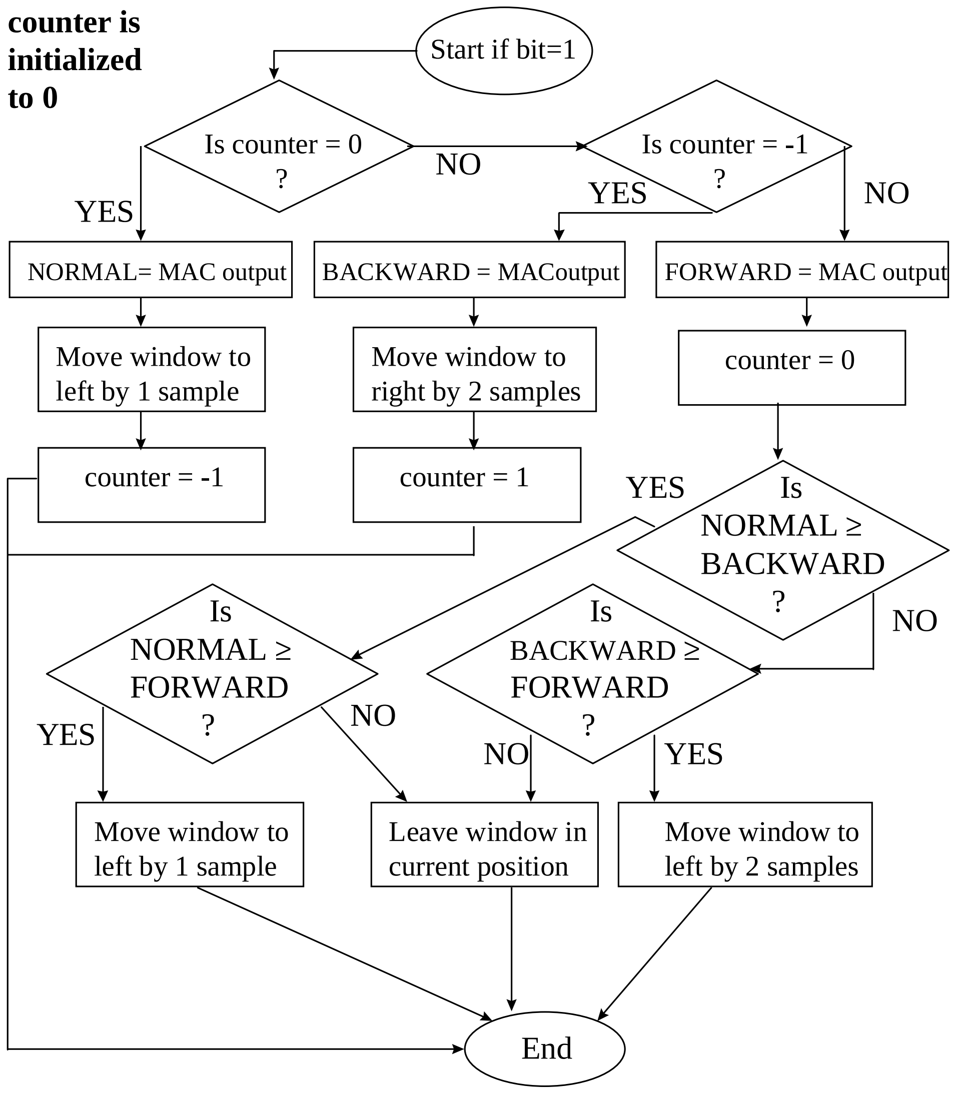


Figure 4.10 Tracking algorithm

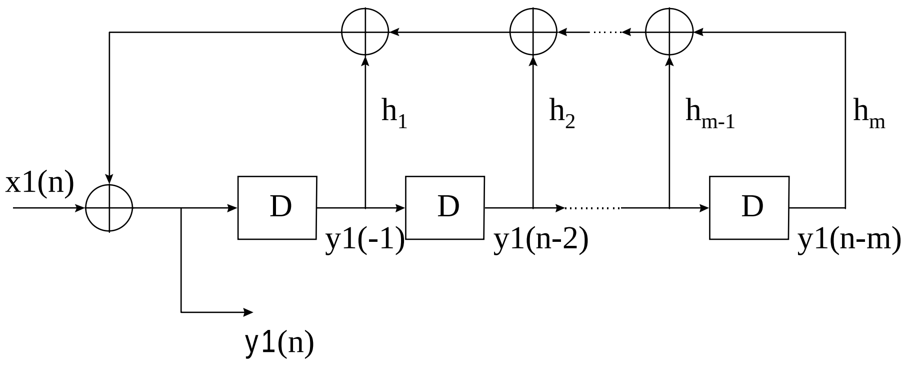
The algorithm above can be modified in several ways to fit the application at hand. For instance if the sampling rate is extremely high then each time we shift the window, we may want to move it by 2 or 3 samples instead of a single sample. We can also modify the algorithm so that the same binary 1 is used to calculate NORMAL, BACKWARD and FORWARD instead of using three different binary ones. However, the basic premise stays the same : we are looking for that window location that produces the largest magnitude MAC output when a binary 1 is received. The algorithm compares the latest values of NORMAL, BACKWARD and FORWARD with each other and not with any absolute value. This means that the algorithm is not very sensitive to change in amplitude of the received signal.

4.2 MBPSK Implementation

An MBPSK link was set up using two C54 evaluation boards operated in two personal computers. Each C54 evaluation board contains a voice codec for analog-to-digital and digital-to-analog conversion. The MBPSK transmitter uses a data scrambler to generate pseudo-random data at 200 bits/s. The transmitter DSP also generates a carrier at 2 KHz. The data at 200 bits/s is used to modulate this carrier. Using the digital-to-analog converter, the transmitter evaluation board outputs a 400 Hz wide MBPSK signal centered at 2 KHz. The transmitter's digital-to-analog converter operates at 9.6 KHz. BNC cables are used to connect the transmitter and the receiver. At the receiving end, the MBPSK receiver samples the signal at 28.8 KHz. The data is de-scrambled and the number of bits and bit errors are recorded. The previous subsections have discussed the modulation-demodulation algorithms and also the tracking algorithms. Two additional tasks are required for the testing of the MBPSK link: the generation of pseudo-random data and the addition of Gaussian noise.

4.2.1 Pseudo-random data

A common method of generating pseudo-random data is through the use of linear feedback shift register sequence generators [Tre95]. Figure 4.11 shows the block diagram of such a pseudo-random data generator. The adders perform modulo 2 addition and the D-blocks represent delay elements. Each *h* can either be a 1 or a 0, indicating a connection or no connection to the adder respectively.



Source : Ter[95] Figure 4.11 Linear feedback shift register sequence generator

If the input signal, *x1(n)*, is always 0, then the output, *y1(n)*, is given by Equation 4.5.

$$y1(n) = \sum_{k=1}^{m} y1(n-k)h_k \qquad (4.5)$$

In Equation 4.5, the summation represents modulo 2 additions. The state of the shift registers is defined by Equation 4.6.

$$s1(n) = [y1(n-1), y1(n-2),...,y1(n-m)] \qquad (4.6)$$

When the initial state is not **0**, the future states are never **0**, even if the input, *x1(n)*, is always 0. A class of polynomials called *primitive polynomials,* when used as the connection polynomial, have been found to produce pseudo-random sequences. All primitive polynomials are irreducible. The primitive connection polynomial used for the MBPSK link is shown in Equation (4.7)

$$h(D)= 1 + D^4 + D^9 \qquad (4.7)$$

The sequence generated by a primitive connection polynomial in Equation 4.7 has the following properties [Ter95]:

1) The pseudo-random sequence repeats every 2^m-1=511 bits where m is the degree of the connection polynomial.
2) The number of binary ones in one period is 2^{m-1}=256 and the number of binary zeros in one period is 2^m-1=255.
3) In one period, there is one run of m binary ones but no run of m-1 binary ones. For 1 ≤ k ≤ m-2, there are 2^{m-k-2} runs of k binary ones. There is no run of m binary zeros and one run of m-1 binary zeros. For 1 ≤ k ≤ m-2, there are 2^{m-k-2} runs of k binary zeros.
4) The periodic auto-correlation function, *R(n)*, is :

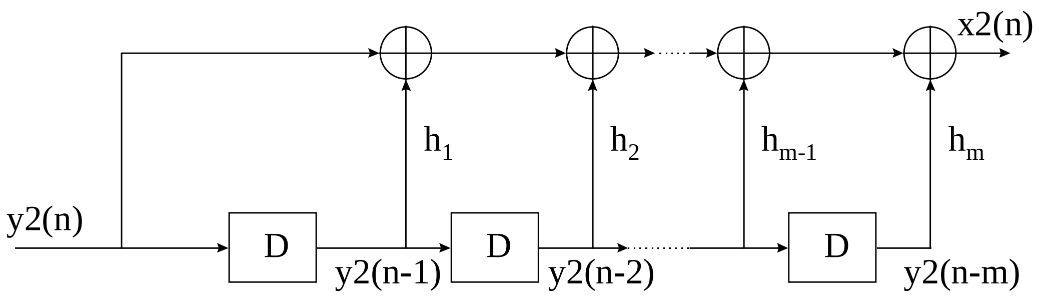
   R(n) = -1/(2^m-1) = -0.00196 when *n* is not a multiple of (2^m-1)

   R(n) = 1 when *n* is a multiple of (2^m-1)=511

The pseudo-random generator acts as a data scrambler. The input, *x1(n)*, is a constant 0, but the output, *y1(n)*, has been randomized or scrambled. y*1(n)* is transmitted via the wired link using MBPSK modulation. At the receiving end the MBPSK detector will output *y2(n)*, which is an approximation of the transmitted bit, *y1(n)*. x2*(n)* is obtained by passing *y2(n)* through a descrambler. If *x2(n)* is not 0 then the receiver records a bit error. Property 4 indicates the desired auto-correlation properties of the transmitted sequence. The probability of a bit error occurring but not being recorded by the receiver are extremely low. In fact, as long as the number of consecutive bit errors stay below (2^m-1) or 511, the receiver will always know when a bit error has occurred. This allows the receiver to keep an accurate count of the number of bit errors. Equation 4.8 describes the operation of the descrambler.

$$x2(n) = y2(n) + \sum_{k=1}^{m} y2(n-k)h_k \tag{4.8}$$

Figure 4.12 shows the block diagram of the descrambler.



Source : Ter[95] Figure 4.12 Descrambler block diagram

The descrambler is a finite impulse response (FIR) filter with *m+1* taps. When the demodulated bit, y2*(n)*, is not equal to the transmitted bit, *y1(n)*, the MBPSK link has caused an error. In such a case, the descrambled output, *x2(n)*, is different from scrambler input, *x1(n)*. In fact, for every incorrectly demodulated bit, *y2(n)*, the number of errors caused in the descrambled output is equal to the number of non-zero coefficients in the connection polynomial. The connection polynomial in Equation 4.7 has three non-zero coefficients which means that each bit error in *y2(n)* will cause three bit errors in the sequence *x2(n)*. This is because each bit of *y2(n)* is used to calculate three different bits in the sequence *x2(n)*. At the transmitting end, *x1(n)* is always 0. If *x2(n)* is not 0 at the receiving end, a bit error is recorded by incrementing the variable Bit_Errors. The number of bit errors caused by the MBPSK link is one third ofBit_Errors.

4.2.2 Addition of Gaussian Noise

In this chapter we have looked at the algorithms for implementation of the transmitter and the receiver. We have also looked at a method to generate pseudo-random data and to keep track of the number of bit errors caused by the link. Therefore, we can implement the transmitter and the receiver, and also record the number of bit errors received. To test the robustness of the detector we must analyze the performance of the receiver under various signal-to-noise ratios (SNRs). Figure 4.13 shows how Gaussian noise is added at the transmitter to the MBPSK signal. The Gaussian noise generator produces white, uncorrelated, Gaussian noise. This noise is passed through an

FIR filter and added to the MBPSK signal. Recall that our MBPSK signal is 400 Hz wide and centered at 2 KHz. A 71-tap FIR filter with a passband from 1800 Hz to 2200 Hz is used to filter the white Gaussian noise. The signal seen by the MBPSK detector consists of a 400 Hz wide MBPSK signal and in-band noise.

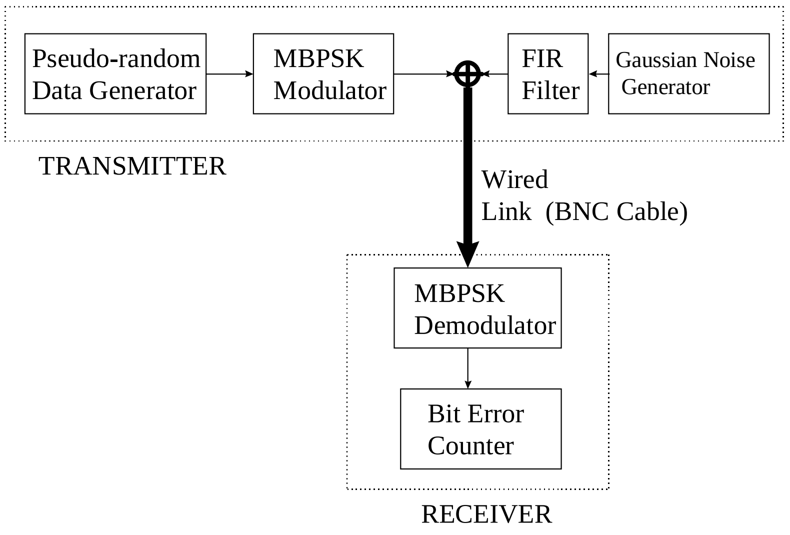


Figure 4.13 Adding noise at the transmitter for testing purposes

Generating Gaussian noise in a DSP is a complex task especially if the program is being written in assembly language. However, Appendix B in [Ter95] by Tretter is dedicated to this topic. Tretter shows how to convert a sequence of uniformly distributed random numbers into Gaussian random numbers. Texas Instruments provides an assembly language sub-routine to generate uniformly distributed random numbers using C54x DSPs. The random number sequence generated by this sub-routine is converted into a Rayleigh distributed sequence using Equation 4.9

$$R = \sqrt{-2\sigma^2 \log_e (1-V)} \qquad (4.9)$$

where V is uniformly, $0 \leq V < 1$

R follows a Rayleigh probability density

A table-look-up is used to convert the uniformly distributed random number into a Rayleigh distributed number using Equation 4.9. The next step is to convert the Rayleigh distributed number into a Gaussian random number. θ is a random number uniformly distributed and the range of θ is $0 \leq \theta < 2\pi$. (θ is independent of V where V is the random number used in Equation 4.9.) Equation 4.10 show how a Gaussian random variable, X, with zero mean and variance σ^2 can be generated.

$$X = R\cos(\theta) \qquad (4.10)$$

The cosine function is implemented by an assembly language sub-routine that uses a Taylor series expansion of cosine.

Infrequently occurring large values of noise often cause errors in communication systems. Large values of noise are seen in the tail of the Gaussian probability density function. Tretter's method of generating noise is particularly useful for testing communication systems because the tails of the Gaussian distributions are well approximated. As V approaches 1, X approaches infinity in magnitude.

The Gaussian noise generated is actually not white, but is evenly distributed from 0 to Fs/2 Hz, where Fs is the sampling frequency. The transmitter DSP generates noise samples at 9600 samples per second which means that the noise is distributed from 0 to 4.8 KHz. Tretter's equations show how to control the Gaussian noise power, σ^2. As shown in Figure 4.13, noise from the generator is passed through the FIR filter before it is added to the MBPSK signal. The bandwidth of the FIR filter is 400 Hz. Therefore, the normalized noise power added to the MBPSK signal is $\sigma^2(400/4800)$ or $\sigma^2 0.0833$. The SNR of the transmitted signal, in dB, is given by Equation 4.11.

$$SNR\ (dB) = 10\log_{10}(Signal\ Power) + 10.8 - 10\log_{10}(\sigma^2) \qquad (4.11)$$

In digital communications, the performance of the receiver is measured against Eb/No. The relationship between Eb/No and SNR for BPSK systems is given by Equation 4.12.

$$Eb/No\ (dB) = (SNR + 3\ dB) \qquad (4.12)$$

Equation 4.13 is obtained by combining Equations 4.11 and 4.12.

$$Eb/No\ (dB) = 10\log_{10}(Signal\ Power) + 13.8 - 10\log_{10}(\sigma^2) \qquad (4.13)$$

The signal power of a sinusoidal signal, $A\cos(\omega t)$, is $(A^2/2)$. Equation 4.13 can be modified to form Equation 4.14.

$$Eb/No\ (dB) = 10\log_{10}(A^2/2) + 13.8 - 10\log_{10}(\sigma^2) \qquad (4.14)$$

Using Equation 4.14 we can obtain the desired Eb/No to test our communications link.

4.2.3 MBPSK Detector Performance

Data at 200 bits per second was used to modulate a 2 KHz carrier. Gaussian noise was added at the transmitter to evaluate the MBPSK detector performance. The results of this test are shown in Figure 4.14. The continuous line shows the theoretical performance results that can be achieved using DBPSK. The 'o' curve shows the theoretical performance results that can be achieved using MBPSK. The MBPSK theoretical limit is obtained by multiplying the DBPSK limit by 0.5. The '*' indicate the measurement results for the implemented MBPSK detector.

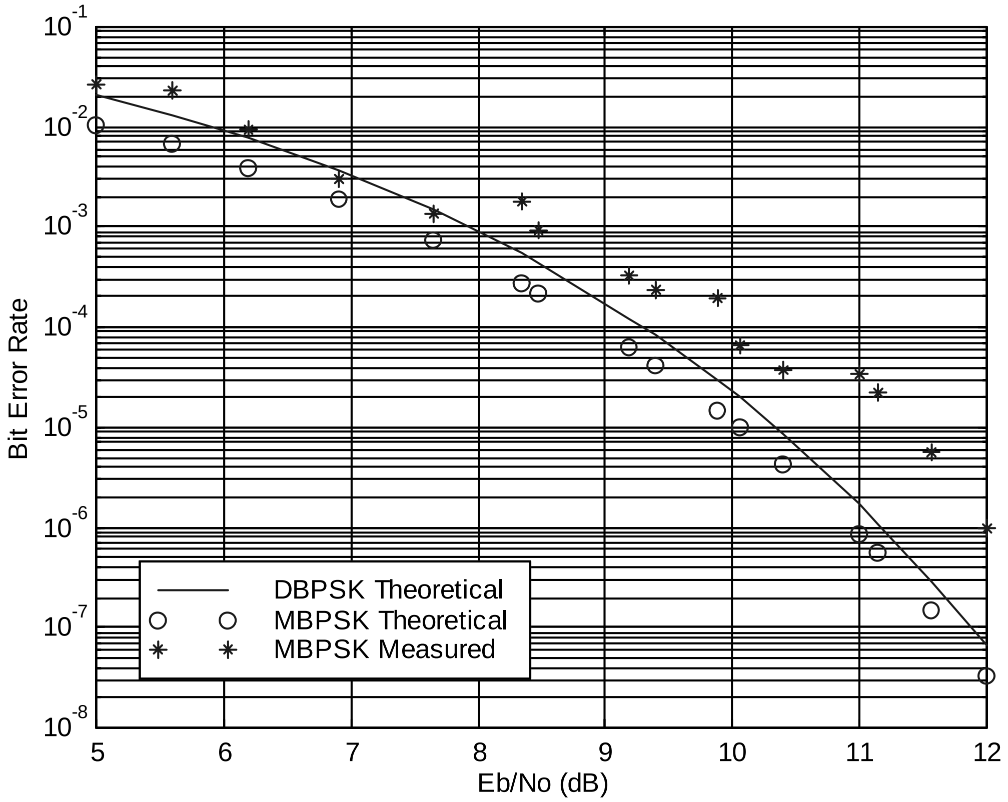


Figure 4.14 MBPSK implementation performance compared with theoretical MBPSK and DBPSK performance

Figure 4.14 shows that the theoretical performance limits of MBPSK and DBPSK are very similar. MBPSK does offer slight improvement in performance at low values of Eb/No and this improvement is obtained at no additional cost or complexity. The implementation of the MBPSK system serves as a proof of concept for the proposed modulation scheme. The receiver and the transmitter were implemented on two different DSPs, which allowed testing of the carrier tracking algorithm as well. Measured results show that MBPSK detector performance is less than 1 dB off the theoretical MBPSK performance limits.

4.3 Summary

This chapter has concentrated on MBPSK implementation and testing in software. We have discussed the algorithms needed to implement an MBPSK transmitter and receiver in a DSP. The MBPSK detector algorithm has been developed with the C54 instruction set in mind. A very simple yet effective tracking algorithm has been proposed to maintain clock synchronization at the receiver.

An MBPSK transmitter and receiver has been implemented using two C54 evaluation boards. The assembly code for the transmitter and receiver is listed in Appendix B. A wired link was set up to test the MBPSK detector performance under various SNR conditions. This test showed that the implemented MBPSK detector was less than 1 dB off the theoretical MBPSK performance limits.

Chapter 5

Analog to Digital Conversion

The received analog signal must be digitized before the DSP can demodulate it. The analog-to-digital (A/D) conversion process involves there distinct steps which are sampling, quantization, and coding [PrM96]. Sampling is the process of converting a continuous time domain signal into a discrete time domain signal. Quantizing is process of converting a discrete-time, continuous-valued signal into a discrete-time, discrete-valued signal. The difference between the unquantized sample and the quantized sample is called the quantization error. Coding is the process of representing the quantized samples in binary form.

The performance of an A/D converter is measured by the signal-to-quantization noise ratio (SQNR). This chapter looks at how oversampling can improve the performance of an A/D converter by decreasing the in-band quantization error. Two important categories of A/D converters that will be analyzed in detail are direct oversampling A/D converters and sigma-delta A/D converters.

5.1 Signal-to-Quantization Noise Ratio

An analog signal, x_a(t), acts as the input to an A/D converter. Equations 5.1 and 5.2 represent x_a(t) and the average power in x_a(t), respectively.

$$x_a(\mathrm{t}) = \frac{A}{2}\cos\Omega t \tag{5.1}$$

$$\sigma_x = \frac{1}{T}\int_0^T \{x_a(t)\}^2 dt = \frac{A^2}{8} \qquad (5.2)$$

Consider a *b*-bit A/D converter that samples this analog signal. If the analog signal has a peak-to-peak range of *A*, then the smallest voltage step, Δ*V*, using *b* bits is given by Equation 5.3.

$$\Delta V = \frac{\mathrm{A}}{(2^{\mathrm{b}} - 1)} \cong \frac{\mathrm{A}}{(2^{\mathrm{b}})} \qquad (5.3)$$

A quantized signal can differ from the analog signal by as much as ±(Δ*V* /2). The Root Mean Square (RMS) quantization noise power of the quantization error voltage, σ_e, assuming a uniform distribution of the error between -Δ*V/2* and Δ*V/2*, is given by Equation 5.4.

$$\sigma_e = \frac{(\Delta V)^2}{12} = \frac{\mathrm{A}^2}{(2^{2\mathrm{b}})12} \qquad (5.4)$$

Using Equations 5.2 and 5.4, the SQNR for our b-bit A/D converter can be calculated [PrM96].

$$\mathrm{SQNR} = \frac{\sigma_x}{\sigma_e} = \frac{3}{2}2^{2b} \qquad (5.5)$$

The SQNR in decibels is given by Equation 5.6.

$$SQNR(dB) = 10\log_{10}SQNR = 1.76 + 6.02b \qquad (5.6)$$

Equation 5.6 shows the SQNR of an A/D converter increases by approximately 6 dB with each additional bit.

5.2 Noise Reduction via Oversampling

To avoid aliasing, a signal must be sampled at twice the bandwidth of interest. This frequency is also known as the Nyquist frequency. Before an analog signal is sampled it must be lowpass filtered to prevent aliasing. If the highest frequency of interest in the analog signal is at the frequency F_m, then the Nyquist frequency is given by $2F_m$. If the sampling frequency, F_s, is the same as the Nyquist frequency then the anti-aliasing lowpass filter must have an extremely sharp cutoff. However, if the sampling is performed at a rate much above the Nyquist rate, the requirements on the anti-aliasing filter are greatly relaxed. Even a bigger advantage of this oversampling approach is that a low resolution A/D converter may be used. With oversampling, a fast but low resolution A/D converter can provide the same resolution that a slower high resolution A/D converter would have provided at the Nyquist rate. Once the signal is oversampled and digitized it can be down-sampled to the Nyquist frequency.

We will assume that the signal sampled by the A/D converter is at baseband. The RMS quantization noise power, σ_e, will have a flat spectrum in the frequency range of 0 to $F_s/2$. The noise power per unit bandwidth, i.e. the noise spectral density, is then given by Equation 5.7.

$$N_o = \frac{\sigma_e}{(F_s / 2)} = \frac{A^2}{(2^{2b})6F_s} \qquad (5.7)$$

Recall that the Nyquist frequency is 2F$_m$ which means that for a base-band signal the band of interest lies from 0 to F$_m$. The total quantization noise power in the band of interest or the in-band noise is given by Equation 5.8.

$$\text{Inband\_}\sigma_e = N_o\,(F_m) = \frac{A^2}{(2^{2b})6F_s}(F_m) = \frac{A^2}{(2^{2b})6}\left(\frac{F_m}{F_s}\right) \qquad (5.8)$$

Equation 5.8 can be analyzed to identify which parameters affect the in-band quantization noise in A/D converters. The signal amplitude, A, and Nyquist frequency, Fm, are signal dependent and the A/D converter has no control over these. However, the number of A/D bits available, b, and the sampling frequency, Fs, are controlled by the A/D converter design. Equation 5.4 shows that the total noise in the band of interest is inversely proportional to the A/D converter's sampling frequency, F$_s$, and to (2^{2b}), where b is the number of bits employed. Consider a β-bit A/D converter operating at the Nyquist rate. Furthermore, consider a b-bit A/D converter that oversamples the input signal and operates at the frequency F$_s$. Using Equation 5.8, a relationship can be derived for the oversampling ratio, M = F$_s$/2F$_m$, such that the two A/D converters provide the same in-band noise power [Mit98]. The right hand side of this equation is obtained by setting 2F$_m$ = F$_s$ in Equation 5.8.

$$\text{Inband\_}\sigma_e = \frac{A^2}{(2^{2b})6}\left(\frac{F_m}{F_s}\right) = \frac{A^2}{(2^{2\beta})12} \qquad (5.9)$$

Using Equation 5.9 we can obtain Equation 5.10.

$$(\beta - b) = +\frac{1}{2}\log_2(M) \qquad (5.10)$$

$(\beta - b)$ denotes the excess resolution bits that are in effect obtained from a b-bit converter by oversampling. Equation 5.10 shows that every doubling of the oversampling ratio increases the effective bits at the Nyquist rate by 0.5. Figure 5.1 shows a plot of the excess resolution bits obtained versus the oversampling ratio [Mit98].

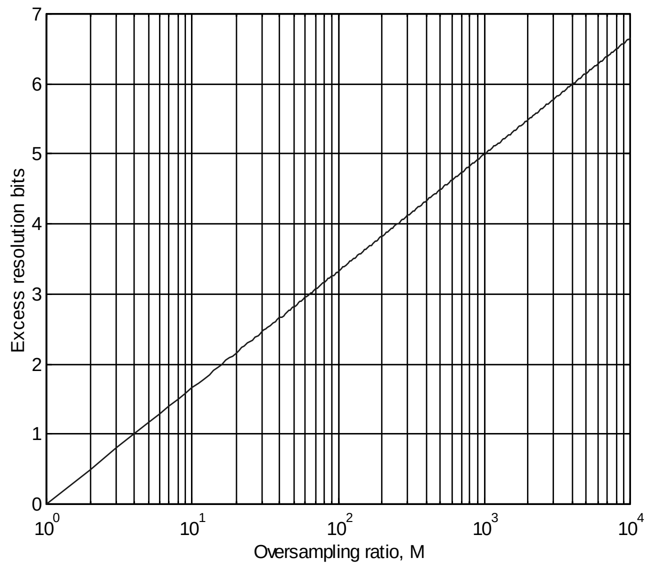


Figure 5.1 Excess resolution bits, $(\beta - b)$ versus oversampling ratio, M

From Figure 5.1, it can be seen how much oversampling is required to obtain the desired excess resolution bits. For instance, a 1-bit A/D converter that oversamples 1000 times can provide the performance of a 6-bit A/D converter operating at Nyquist frequency.

The important point to note here is that by oversampling, the quantization noise power can be spread over a wider band. Since the total quantization noise power stays constant, the noise spectral density is lowered. This means that the in-band quantization noise has been decreased. The signal of interest can now be recovered by lowpass filtering the digital samples. An A/D converter that utilizes this principle of oversampling is called a direct oversampling A/D converter. Further improvement in performance can be achieved by shaping the spectrum of noise such that there is less noise in the band of interest and the noise power is concentrated in the higher frequencies. This principle will be analyzed in the next section.

5. 3 Sigma Delta A/D Converter

A popular type of oversampling A/D converter is the sigma-delta A/D converter. The block diagram of a sigma-delta A/D converter is shown in Figure 5.2.

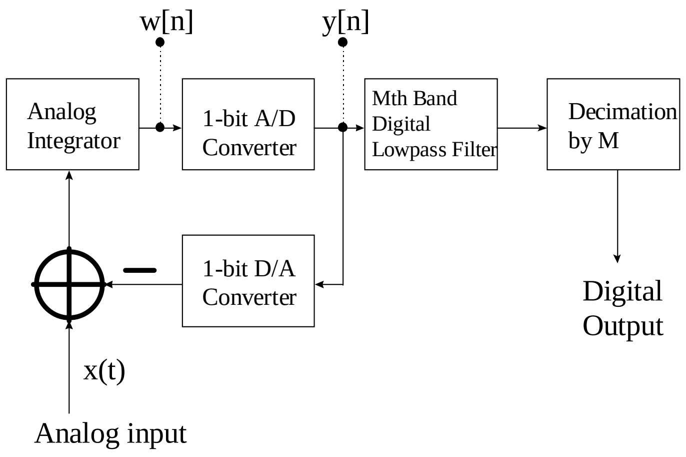


Figure 5.2 Sigma Delta A/D converter

The sigma-delta A/D converter can be broken up into two parts, the quantizer and the decimator. The quantizer consists of the analog integrator, the 1-bit A/D converter and the 1-bit Digital-to-Analog (D/A) converter. The task of the quantizer is to convert the information in the analog input into digital form. The decimator consists of an Mth band lowpass filter and a factor of M down sampler. The decimator stage is required because the signal of interest was originally oversampled by a factor of M.

The input-output relation of the sigma-delta quantizer is nonlinear; however, the low-frequency content of the analog input, x(t), can be recovered from the quantizer output, y[n]. Assume that there is an extremely low frequency analog input such that we can consider this input to be constant over a period of time. The constant input signal has a magnitude of less than 1 or in other words the input signal ranges from -1 to +1. We will model our analog integrator as a digital accumulator. The output of the accumulator is digitized to obtain y[n]. y[n] is a bounded sequence with sample values equal to -1 or +1. y[n] can only remain bounded if the accumulator output, w[n], is also bounded. Since the accumulator output is bounded, the accumulator input must have an average value of zero. Therefore, the average value of y[n] must equal the average value of the input x(t). Thus we have been able to derive the constant (or low frequency) information of x(t) from w[n].

The simplest sigma-delta quantizer uses a 1^{st} order integrator which can be modeled as an accumulator. The MATLAB program SD in Appendix C simulates the working of a 1^{st} order sigma-delta converter. Figure 5.3 shows the power spectral density (PSD) of the input signal. The input signal has been oversampled at a rate of 50 times Nyquist. In Figure 5.3, the normalized frequency is plotted on the x-axis and ranges from 0 to 1 with 1 representing (50 x Nyquist frequency).

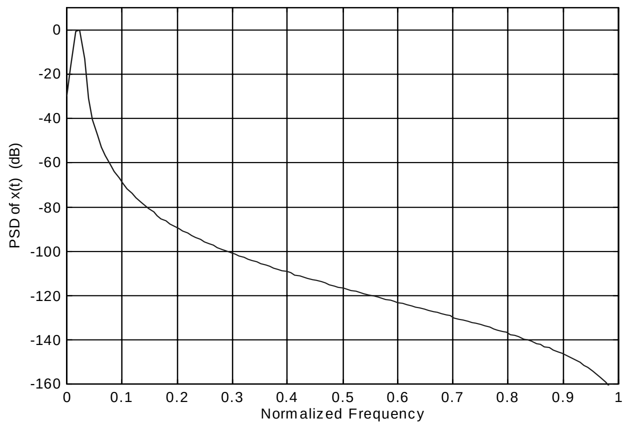


Figure 5.3 PSD of input signal after oversampling

y[n] is the digital signal represented using 1-bit. The PSD of y[n] is shown in Figure 5.4.

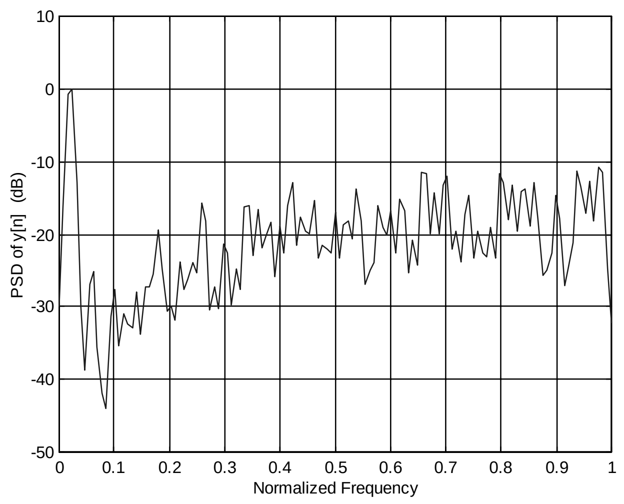


Figure 5.4 PSD of y[n]

Figure 5.4 illustrates the noise shaping abilities of the sigma-delta A/D converter. As mentioned before, a direct oversampling A/D converter is able to spread out the total quantization noise power over a larger band, thereby decreasing the in-band noise power. In addition to this, sigma-delta converters are able to perform noise shaping such that the noise power is concentrated in higher frequencies. Figure 5.4 shows that the noise floor is considerably higher at higher frequencies and relatively lower in the band of interest.

The signal y[n] is the quantizer output and is lowpass filtered using a Mth band lowpass filter, where M=50 is the oversampling ratio. The transfer function, H(z), of the Mth band lowpass filter is given by Equation 5.11.

$$\mathrm{H(z)} = \frac{1-z^{-M}}{1-z^{-1}} \tag{5.11}$$

The frequency response of this digital filter is shown in Figure 5.5.

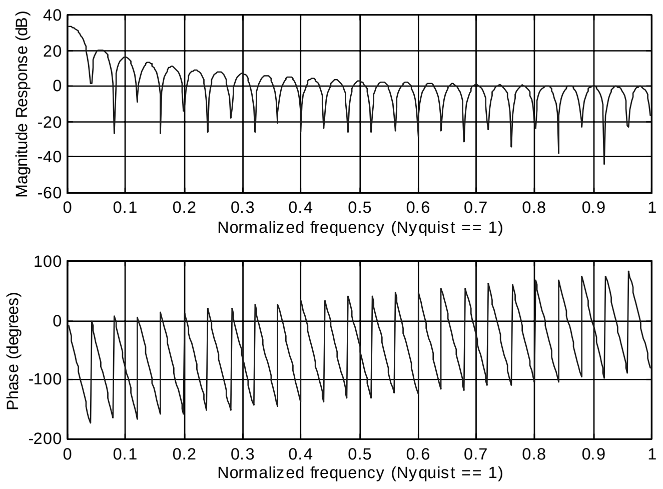


Figure 5.5 Mth band lowpass filter frequency response

The PSD of the filtered output is shown in Figure 5.6.

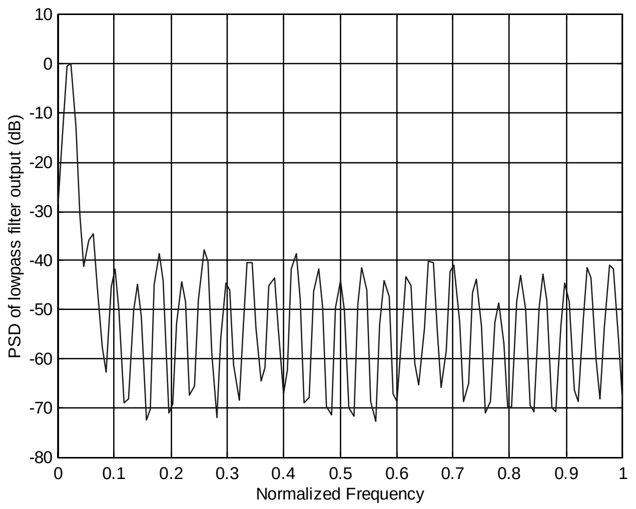


Figure 5.6 PSD of lowpass filter output

The final step is that of down-sampling. Figure 5.6 shows the PSD of the actual A/D converter output. Here it can be seen that the signal has been down-sampled back to the Nyquist rate.

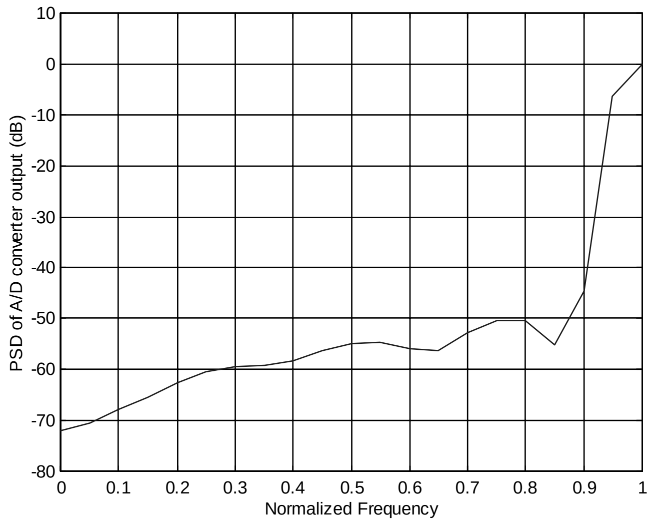


Figure 5.7 PSD of A/D converter output signal

5.4 Performance Analysis

Mitra in [Mit98] compares the performance of a sigma-delta A/D converter with that of a direct oversampling A/D converter. The improvement in performance obtained by using a sigma-delta A/D converter is illustrated in Equation 5.12.

$$\text{Improvement}(M) = -5.1718 + 20\log_{10}(M) \qquad \text{(dB)} \qquad (5.12)$$

Consider the sigma-delta converter modeled in the MATLAB program SD. The oversampling ratio, M, used was 50. Using Equation 5.12, it can be seen that a sigma-delta converter can provide an improvement of approximately 28.8 dB when compared with a direct oversampling converter that also operates at 50 times Nyquist frequency.

Figure 5.8 compares the SQNR provided by a 1-bit sigma-delta converter with the SQNR of a direct oversampling A/D converter at various oversampling ratios.

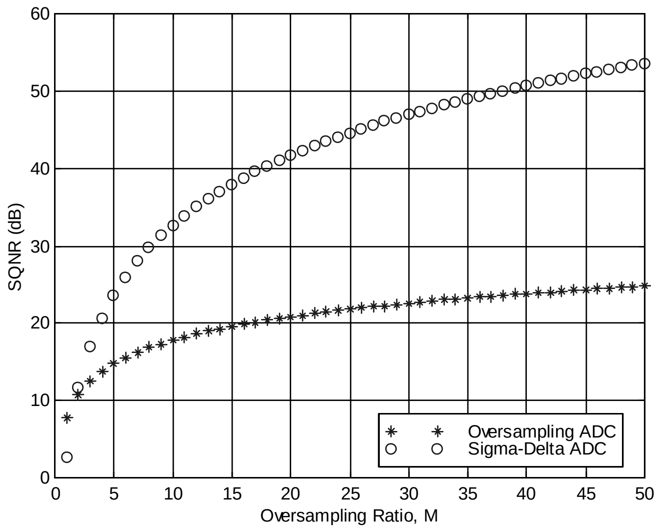


Figure 5.8 Oversampling ratio versus SQNR (dB)

A direct oversampling converter can be implemented using a lowpass analog anti-aliasing filter, a comparator and the receiver's DSP. A sigma-delta converter involves more complicated circuitry (as shown in Figure 5.2) but provides considerably improved performance when compared to a direct oversampling converter. Custom building a sigma-delta converter can be time prohibitive and buying a pre-made sigma-delta converter is likely to be cost prohibitive. This brings us back to the direct oversampling converter. The question that arises is how much oversampling would be required such that a direct oversampling converter may be used. Assume that the SNR of the analog input signal is 10 dB[2] and that a direct oversampling A/D converter is used. Given this SNR at the input, the noise floor of the digitized output will depend upon the oversampling ratio. Figure 5.9 shows the SNR of the digitized output using a 1-bit direct oversampling converter when the input signal's SNR is 10 dB. Figure 5.9 is based on the assumption that the thermal noise of the receiver is uncorrelated with the quantization noise of the A/D converter.

[2] An SNR of 10 dB corresponds to an Eb/No of 13 dB for BPSK systems.

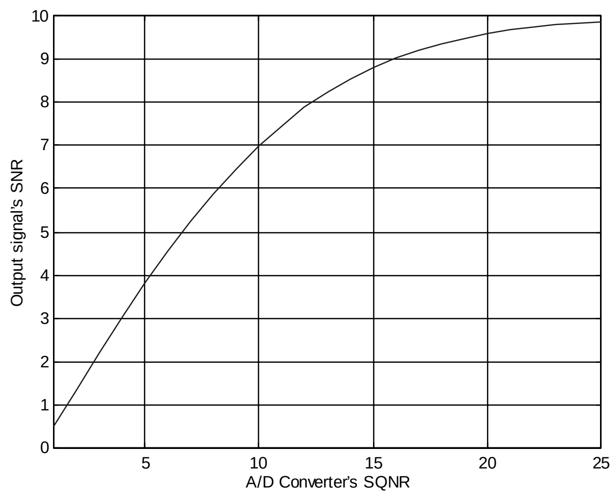


Figure 5.9 A/D converter output's SNR (assumes that input's SNR is 10 dB)

Figure 5.9 shows that for a 10 dB analog input, when the SQNR of the A/D converter is 20 dB, the digitized output will have an SNR of approximately 9.5 dB.   This means that with an SQNR of 20 dB, digitization only raises the noise floor by 0.5 dB. The noise floor is raised by even less than 0.5 dB if the analog input's SNR is less than 10 dB. Therefore, we will want our direct oversampling A/D converter to have an SQNR of 20 dB or less. From Figure 5.8 it can be seen that when the oversampling ratio is 15, the SQNR is approximately 20 dB. Consider an analog signal modulated using BPSK and carrying 80 Kbps of data.   The Nyquist frequency for this signal is 160 KHz. Oversampling by 15 would require sampling at 2.4 MHz. In other words, as long as our DSP can process samples at a rate of approximately 2.4 MHz, we will be able to implement a direct oversampling A/D converter. The C54x DSP family has several DSPs that process at rates up to 100 MIPS (Million Instructions Per Second).   With some efficient programming, it would possible to implement a direct oversampling A/D converter in the receiver's DSP.

5.5 Summary

Operating the A/D converter above the input signal's Nyquist frequency improves the performance of a low-resolution A/D converter. This is the premise behind the working of direct oversampling A/D converters. Adding noise shaping to the A/D conversion process further improves performance. Sigma-delta A/D converters utilize both noise shaping and oversampling. Sigma-delta A/D converters offer considerably better performances than direct oversampling converters. However, it has been proposed that a direct sampling converter be used because of the complexity of a sigma-delta A/D converter. It has also been shown that a direct oversampling A/D converter can be used without any serious signal degradation.

Chapter 6
Summary and Conclusion

This final chapter has been divided into two sections. Section 6.1 summarizes the work that has been presented in this thesis. Section 6.2 looks ahead at the future research that needs to be done to complete this project.

6.1 Summary of Work Done

This thesis has presented a design for a wireless call button network. A wireless call button network that utilizes modified CAN for wireless (MCANW) systems has been proposed. MCANW protocol is based on CAN protocol which will be used in the next generation call button networks. MCANW is more suitable for our wireless network for two reasons :

1) CAN uses an Acknowledgment Field in which the Slot Bit is set high by the transmitter. When a receiver sees this bit on the serial bus, it sets the bit low using the wired-AND serial bus. The transmitter is still monitoring the bus and knows that the message has been received once the Slot Bit is set low. This form of acknowledgment can not be performed in a wireless system and therefore, MCANW does not utilize the Acknowledgment Field.

2) CAN utilizes a Control Field to specify the length of the Data Field. However, for our system the data can always be carried in a 1-byte long Data Field and, therefore, a Data Field of constant length is used. MCANW does not use a Control Field, thereby utilizing the bandwidth more effectively.

Data radios are attached to the call buttons and to the central controller. The wireless network operates in a master-slave configuration with the controller radio as the lone master. The master controller polls each slave, inquiring whether the call button requires an elevator. If a call button has been depressed, it requests an elevator when it is polled. Figure 6.1 (previously shown in Figure 2.6) illustrates the wireless call button network design that has been proposed.

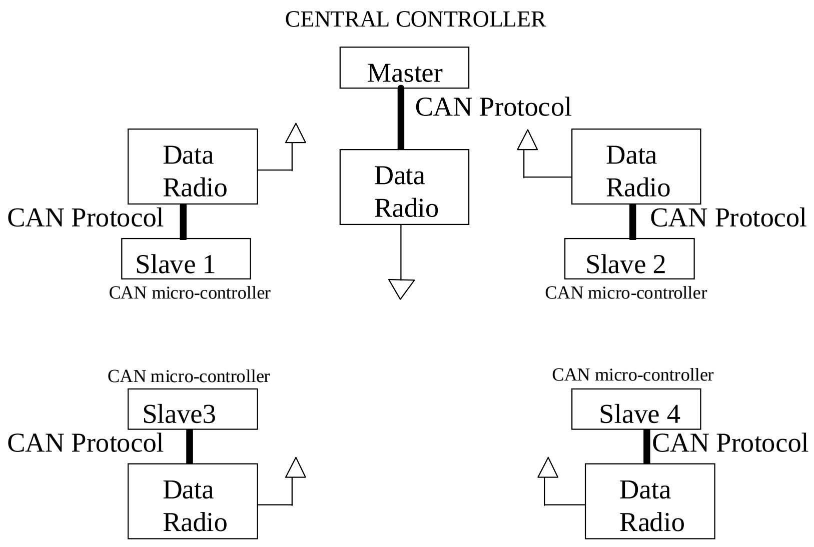


Figure 6.1 Wireless Call Button Network

The system achieves low power consumption through two design attributes. First, the slave call button data radios stay in a dormant mode until a call button is depressed. Once a call button is depressed, its slave radio awakens and starts listening to the master's polls. Secondly, the elevator shaft has been proposed as the path for RF propagation. The elevator shaft acts like a waveguide which means that very little transmitter power is required.

The proposed wireless network can be deployed in the US, Europe, and Japan. The data radios proposed for the system would operate in the 2.4 GHz band and will utilize Modified BPSK (MBPSK) modulation and demodulation. MBPSK modulation, like DBPSK, can be detected by non-coherent means. However, MBPSK detection does not require the transmitter to send a training sequence. More importantly, the bit error rate achieved through MBPSK is half that of DBPSK under the same signal-to-noise ratio. A tracking algorithm that allows the receiver to maintain synchronization with the transmitter has been developed. The real strength of this tracking algorithm is that it requires very little additional computation besides what the MBPSK detector performs. An MBPSK transmitter and receiver pair were implemented using two C54 DSP Evaluation Boards. The implemented MBPSK detector was found to perform within 1 dB of the theoretical MBPSK limits. Figure 6.2 (previously shown in Figure 3.4) illustrates the design of the proposed MBPSK data radio.

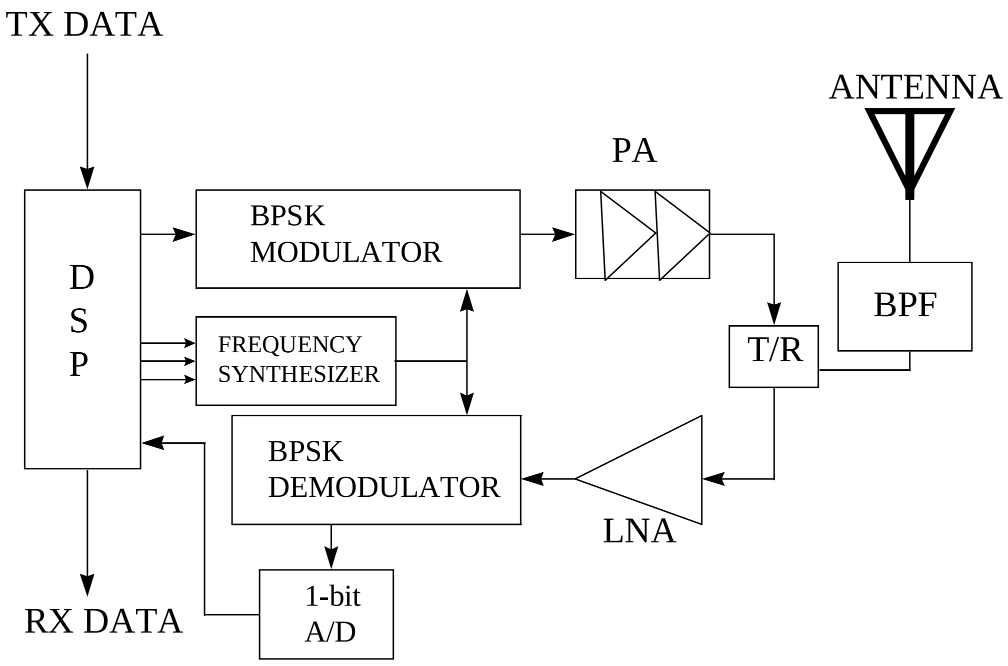


Figure 6.2 MBPSK data radio

An A/D converter contributes considerably to the cost of a data radio. Even low cost multi-bit A/D converters can cost between $5 and $10. One way to achieve cost reduction in the data radios is by using a 1-bit A/D converter instead of a multi-bit A/D converter. Two popular techniques that allow a 1-bit A/D converter to achieve multi-bit performance have been studied and compared. The two A/D converters studied are the direct oversampling converter and the sigma-delta converter. It has been suggested that direct oversampling converters be used due to the ease of their implementation.

Work Remaining

There are two main tasks that require further research. First, synchronization schemes for the smart frequency hopping data radios have to be developed. In most wireless systems, the establishment of a wireless link involves synchronization in time. However, for the proposed data radios, synchronization in both time and frequency must be achieved. Synchronization becomes an especially complex issue because the transmitters will not follow a pre-set hopping scheme. The second task which requires research is choosing the RF parts for the 2.4 GHz BPSK radio. Several companies manufacture BPSK modulator-demodulator chip-sets for the 900 MHz band, however, the choices available for the 2.4 GHz band are more limited. It might be possible to have a manufacturer custom design and produce the 2.4 GHz chip-set. Once these two tasks are performed, all the research required for implementing the wireless call button network will be complete.

APPENDIX A

C5402 FEATURES

The C5402, advertised at $5 in quantities of 50,000, is the most inexpensive processor in the C54x series. The C5402 has the following features :

- Operates at 100 MHz (up to 100 MIPS)
- 40-bit ALU with either single 32-bit or dual 16-bit configuration
- One 40-bit adder and two 40-bit accumulators
- Eight auxiliary registers
- 16 KW Data RAM, 4 KW Program ROM
- Extended addressing mode for 1M 16-bit external program space
- Six-channel DMA controller
- Two multi-channel buffered serial ports (McBPS) with 2 KW buffer
- 8-bit host port interface (HPI)
- Time division multiplex (TDM) serial port
- Two 16-bit on-chip timers
- Phase-locked loop (PLL) clock generator (internal oscillator or external clock source)
- Viterbi accelerator (compare/select/store unit)
- Single-cycle normalization and exponential (for floating-point arithmetic)
- Three power-down modes
- Low voltage operation : 1.8V
- Compact packaging

APPENDIX B
MBPSK Implementation Code

The files used to implement the MBPSK receiver are presented first followed by the transmitter files. Each file will be introduced by its FILENAME and PURPOSE. All the files except for the module that perform the modulation and detection have been supplied by Texas Instrument (TI). Some of the original TI files have been modified for the application at hand.

MBPSK RECEIVER FILES

FILENAME : RX.BAT
PURPOSE : Assembles and links the receiver modules

```
*************************START OF RX.BAT ******************************************
asm500 -ls aic_cfg.asm
asm500 -ls coeffs.asm
asm500 -ls init_54x.asm
asm500 -ls init_aic.asm
asm500 -ls init_ser.asm
asm500 -ls vectors.asm
asm500 -ls main.asm
lnk500 -v0 main.cmd
****************************END OF RX.BAT*****************************************
```

FILENAME : aic_cfg.asm
PURPOSE : Configures the AIC

```
***********************START OF aic_cfg.asm ****************************************

        .mmregs
        .include  "aic_cfg.inc"
        .def      wrt_cnfg
        .def      wait_xrdy

        .text

wrt_cnfg:
        STM     #K_0,DXR1                   ;primary data word - a jump start!
        CALL    wait_xrdy
        STM     #K_PRIM_WORD,DXR1           ;send primary word with D01-D00 = 11 to
                                            ;signify secondary communication
        CALL    wait_xrdy                   ;wait for data to be copied from DXR to XSR
        STM     #K_REG_1,DXR1                       ;change A register
        CALL    wait_xrdy
        STM     #K_PRIM_WORD,DXR1
        CALL    wait_xrdy
        STM     #K_REG_2,DXR1                       ;change B register
        CALL    wait_xrdy
        RET
;*****************************************************************************
;       MODULE: WAIT_XRDY
;       PURPOSE: Wait for data to be copied from DXR to XSR before the next
;                data is sent to DXR
;*****************************************************************************
wait_xrdy:
        BITF    SPC1,0800h                  ;test XRDY bit in SPC1
        BC      wait_xrdy,NTC               ;loop if not set
        RET
***********************END OF aic_cfg.asm ***************************************
```

FILENAME : coeffs.asm
PURPOSE :   Stores samples of one half of the incoming signal; used to calculate MAC output

```
***********************START OF coeffs.asm ****************************************

AMCOEFF         .sect       "filter"       ; filter coefficients

                .def     TA, RA, TB, RB, AIC_CTR, coeff, FILTER_LAST, FILTER

TA        .word    16        ;
RA        .word    16        ;  This set up of AIC registers give
                             ;  a sampling   freq of 10,081 Hz
                             ;
TB        .word    31        ;
RB        .word    31        ;
AIC_CTR           .word    0ch
coeff          .word    5219       ; 10438

FILTER       .word   0
FILTER2      .word   0
FILTER3      .word   0
FILTER4      .word   0
FILTER5      .word   0
FILTER6      .word   0
FILTER7      .word   0
FILTER8      .word   0
FILTER9      .word   0
FILTER10      .word   0
FILTER11      .word   0
FILTER12      .word   0
FILTER13      .word   0
FILTER14      .word   0
FILTER15      .word   0
FILTER16      .word   0
FILTER17      .word   0
FILTER18      .word   0
FILTER19      .word   0
FILTER20      .word   0
FILTER21      .word   0
FILTER22      .word   0
FILTER23      .word   0
FILTER24      .word   0
FILTER25      .word   0
FILTER26      .word   0
FILTER27      .word   0
FILTER28      .word   0
FILTER29      .word   0
FILTER30      .word   0
FILTER31      .word   0
FILTER32      .word   0
FILTER33      .word   0
FILTER34      .word   0
FILTER35      .word   0
```

```
FILTER36       .word   0
FILTER37       .word   0
FILTER38       .word   0
FILTER39       .word   0
FILTER40       .word   0
FILTER41       .word   0
FILTER42       .word   0
FILTER43       .word   0
FILTER44       .word   0
FILTER45       .word   0
FILTER46       .word   0
FILTER47       .word   0
FILTER_LAST    .word   0 ; last FILTER coefficient

MACspace    .word   0 ; MACD OVERFLOW SPACE

***********************END OF coeffs.asm ****************************************
```

FILENAME : init_54x.asm
PURPOSE :   Initializes the C54x DSP

```
********************START OF init_54x.asm ****************************************
.include "init_54x.inc"         ; contains all the initial
                                  ; values of ST0, ST1, PMST, SWWSR,
                                  ; BSCR
            . mmregs
            . def    init_54

K_INIT_DP        .set        0

            .text

init_54:
* initialize the status and control registers
            STM     #K_ST0, ST0
            STM     #K_ST1, ST1
            STM     #K_PMST,PMST

* wait states for Bank Switch
            STM     #K_BSCR, BSCR           ; 0 wait states for BANK
                                  ; SWITCH

* Init. the s/w wait state reg.for 2 wait states for I/O operations
            RETD
            STM     #K_SWWSR_IO, SWWSR       ; 2 wait states for I/O
                                  ; operations
            .end

**********************END OF init_54x.asm ****************************************
```

FILENAME : init_ser.asm
PURPOSE : Initializes the AC01

```
********************START OF init_ser.asm ****************************************
.def     serial_init

             .include " init_ser.inc"
             .include " interrpt.inc"
             . mmregs
aic_in_rst      . usect   "aic_vars",1
aic_out_of_rst  .usect   "aic_vars",1

K_TCR          .set     14h              ; Timer Control Register
K_0            .set     0h
K_8000         .set     8000h

             .text
serial_init:
             LD      # aic_in_rst,DP        ; initialize the DP for  aic_reset
             ST      #K_0,  aic_in_rst      ; bit 15 = 0 of TCR resets the AIC
             PORTW    aic_in_rst,K_TCR       ; write to address 14h (TCR)
***************************************************************************
* We need at least 12 cycles to pull the AIC out of reset. Only 6 cycles
* have been used.
***************************************************************************
             STM    #K_SERIAL_RST, SPC1      ; reset the serial port with
                                   ; 0000   0000 0000 1000
             STM    #K_SERIAL_OUT_RST, SPC1  ; bring  ser.port out of reset with
                                   ; 0000   0000 1100 1000
* Write some value to DXR1 (the AC01 on the EVM is connected to serial port 1)

             RSBX   INTM

             STM    #K_IMR,IMR             ; Enable RINT1
                                   ; 0000   0000 0100 0000
             STM    #K_IMR,IFR             ; clear RINT1

             STM    #K_0,DXR1
                                   ; 0000   0000 0100 0000
             NOP                      ; this guarantees that the AIC is held for    atleast 12 cycles in reset
* Pull the AC01 out of reset - the AC01 requires that it be held in reset for
* 1 MCLK, which is equivalent to 96.45ns (based on an MCLK of 10.368MHz)

             ST     #K_8000,  aic_out_of_rst  ; bit 15 = 1 brings AIC from reset

             RETD
             PORTW    aic_out_of_rst, K_TCR    ; AIC out of reset
             .end

********************END OF init_ser.asm ****************************************
```

FILENAME : vectors.asm
PURPOSE :   Initializes the C54x vector table

```
*********************START OF vectors.asm ****************************************

.title  "54X Vector Table Initialization"

            . ref    main_start
            . ref    receive
            . ref    transmit

            . def    STACK



            . mmregs
K_STACK_SIZE    .set    80

STACK         . usect  "stack",K_STACK_SIZE

            .sect   "vectors"
reset:        BD      main_start             ; RESET vector
            STM     #K_STACK_SIZE+STACK, SP  ; initialize stack pointer to  stack_start
nmi:          NOP                           ; ~NMI
                  NOP
                  NOP
                  RET

* software interrupts
sint17        .space  4*16
sint18        .space  4*16
sint19        .space  4*16
sint20        .space  4*16
sint21        .space  4*16
sint22        .space  4*16
sint23        .space  4*16
sint24        .space  4*16
sint25        .space  4*16
sint26        .space  4*16
sint27        .space  4*16
sint28        .space  4*16
sint29        .space  4*16
sint30        .space  4*16

int0:         RETF                          ; INT0
                  NOP
                  NOP
            NOP

int1:         RETF                          ; INT1
                  NOP
                  NOP
                  NOP
```

```
int2:      RETF                 ; INT2
                NOP
                NOP
                NOP

timer:      RETF                 ; TIMER
                NOP
                NOP
                NOP

rint0:      RETF           ; Serial Port Receive
                NOP
                NOP
                NOP


xint0:      RETF                 ; Serial Port Transmit
                NOP
                NOP
                NOP

rint1:       B      receive         ; Serial Port Receive
          NOP
          NOP
          NOP

xint1:      RETF                 ; Serial Port Transmit
                NOP
                NOP
                NOP

int3:      RETF                 ; INT3
                NOP
                NOP
                NOP

                .end
```

*********************END OF vectors.asm ***

FILENAME : main.asm
PURPOSE :   Main routine; includes the MBPSK detector

********************START OF main.asm **

```
;%%%%%%%%%%%% PUNIT MUKHIJA
;%%%%%%%%%%%% CWT, VIRGINIA TECH
;%%%%%%%%%%%% Modified Binary Phase Shift Keying - Receiver

        .title  "MBPSK Receiver"

        . def    main_start        ; reset vector
        . ref    aic_init          ; writes new  config into AC01
        . ref    serial_init       ;  intialize serial port
        . ref    init_54           ; initialize ST0,ST1, PMST
        . def    receive
        . def    transmit

;  filter coefficients  bandpass @ flcut=200hz  fhcut=1800hz
        . ref    coeff
        . ref    FILTER_LAST
        . ref    FILTER


        . mmregs

        .include " interrpt.inc"

        .data
rndnum     .word  0
XN         .word  0
samples    .word  0
ones       .word  0
zeros      .word  0
I1         .word  0
I0         .word   0
INPUT       .word   0
INPUT2      .word   0
INPUT3      .word   0
INPUT4      .word   0
INPUT5      .word   0
INPUT6      .word   0
INPUT7      .word   0
INPUT8      .word   0
INPUT9      .word   0
INPUT10      .word   0
INPUT11      .word   0
INPUT12      .word   0
INPUT13      .word   0
INPUT14      .word   0
INPUT15      .word   0
INPUT16      .word   0
INPUT17      .word   0
```

```
INPUT18      .word   0
INPUT19      .word   0
INPUT20      .word   0
INPUT21      .word   0
INPUT22      .word   0
INPUT23      .word   0
INPUT24      .word   0
INPUT25       .word   0
INPUT26      .word   0
INPUT27      .word   0
INPUT28      .word   0
INPUT29      .word   0
INPUT30      .word   0
INPUT31      .word   0
INPUT32      .word   0
INPUT33      .word   0
INPUT34      .word   0
INPUT35      .word   0
INPUT36      .word   0
INPUT37      .word   0
INPUT38      .word   0
INPUT39      .word   0
INPUT40      .word   0
INPUT41      .word   0
INPUT42      .word   0
INPUT43      .word   0
INPUT44      .word   0
INPUT45      .word   0
INPUT46      .word   0
INPUT47      .word   0
INPUT48      .word   0

normal       .word 0
ncount       .word 0
back         .word 0
bcount       .word 0
forward      .word 0
fcount       .word 0
direction    .word 0
synch        .word 0

BIT         .word   0
;h1         .word   1,0,0,1,0   ; for [2,5]
h1          .word   1,0,0,0,0,1,0,0,0 ; for [4,9]
S1          .word   0,0,0,0,0,0,0,0,0  ; for [4,9]
;S1         .word   0,0,0,0   ; for [2,5]
S1LAST      .word   1
sspace      .word   0
sspace1     .word   0
errors      .word   0
recbits     .word   0
packets     .word   0
```

```
CarDet      .word   0

;;;;;; Carrier detection circuit parameter
c          .word    13107     ;  25/16 = 0.75
cminus1     .word    3277      ;   4/16 = 0.25
;;; c and cminus1 have been left shifted by 4 bits


ad          .word    0     ; AD output
ad2         .word    0     ;
currentX    .word    0     ; y(n) for carrier detection
pastY1      .word    0
pastspace   .word    0




          .text

main_start:
          CALL     init_54               ; initialize ST0,ST1 PMST
                                   ;   and other   regsiters
          CALL      serial_init           ; initialize serial port
          CALLD    aic_init              ; initialize AIC
          LD       #0,DP
          NOP
          STM    #INPUT,AR0              ; AR0 contains INPUT address
          STM    #FILTER_LAST,AR1        ; AR1 contains Last Filter  coeff

WAIT:       B       WAIT                 ; wait for receive interrupt

receive:    LD      #XN,DP ;????????????????????????????????????

          LD CarDet,0,B ; If  CarDet is greater than zero then
          BC  Demod, BGT ; Branch to Demod
; %%%%%%%%% Carrier Detect %%%%%%%%%%%%%%
          LDM     DRR1,A                 ; LOAD ACCUMULATOR WITH WORD
          STL    A,0,currentX

          SQUR currentX,A           ; A now contains the squared
                             ; A/D sample
          STL  A,-1,ad2             ; update (ad^2)
          LD #0,A

          STM    #ad2,AR3
          STM    #cminus1,AR4
          MACD *AR3,cminus1 , A         ; A = y(n)

          STM    #pastY1,AR3
          STM    #c,AR4
          MACD *AR3, c, A         ; A = y(n)

          STH  A,-1,pastY1  ; pastY1 for next time
```

```
; pastY1 is greater than 80 then declare carrier detected
        LD pastY1,0,B    ; Load B with pastY1
        SUB #80,B        ; Subtract 20 from B
        BC  decdet, BGT  ; IF B>0 then branch and declare detect


        LD      #0,0,A            ; OUTPUT ==>Accumulator A
                               ; TWO   LSB's MUST BE ZERO FOR AIC!
        STLM    A,DXR1                ; SEND TO TRANSMIT REGISTER!

        RETE              ; Carrier Not Detected

decdet:     LD  #1,0,B
        STL  B,0,CarDet   ; Carrier Detected
; %%%%%%%%% Carrier detect calculations complete %%%%%%%%%%%%
; %%%% CarDet=1 if Carrier was detected %%%%%%

Demod:
; Increment samples
        LD     samples,0,A
        ADD    #1,A
        STL    A,0,samples

; If samples <49, then place sample into INPUT
        LD  samples,0,A
        SUB #49,0,A
        BC NEXT24, ALT ; Branch to NEXT24 if samples is less than 49

; samples is greater than 24, therefore, A/D sample should go into
; FILTER or next bit should be calculated

; If samples < 49 then place sample into Filter
        LD  samples,0,A
        SUB #97,0,A
        BC NEXT48, ANEQ
; If samples==49 then  48 samples have been collected
; time to calculate BIT, reset samples and AR0, AR1
;************* Calculate BIT %%%%%%%%%%%%%%%%%%%%%%%%
        STM #FILTER_LAST, AR1


        RPTZ A,47 ;   24 tap filter
        MACD  *AR1-,INPUT,A
        ABS   A,B;  Store  Abs(A) in B, A remains unchanged
; If A is greater than zero a phase change was detected which means a 1
; was received . If A is less than zero a 0 was received
        BC ONE, AGT
ZERO:       ST #0,BIT
        LD     zeros,0,A
        ADD    #1,A
        STL    A,0,zeros

        STM    #INPUT,AR0            ; AR0 contains INPUT address
        STM #FILTER_LAST, AR1       ; AR1 contains FILTER address
        ; PLACE SAMPLE FROM A/D INTO INPUT
        LDM    DRR1,B ; LOAD ACCUMULATOR WITH WORD FROM A/D
        STL    B,0,*AR0+ ; Store DDR1 into address pointed by AR0
                   ; Increment AR0 address pointer

        ST  #1,samples      ; Set samples, and place the first A/D

        B DESCRAM;  synch_up occurs only if bit is one

ONE:        ST #1,BIT
        LD     ones,0,A
        ADD    #1,A
        STL    A,0,ones

; Synchronize only if BI is one
synch_up:
        LD synch,0,A
        BC NOTN, ANEQ
YESN:
        STH B,-7,normal
        LD #1,0,A
        STL A,0,synch
        STM    #FILTER,AR3
        LD  *AR3,0,A
        STL  A,0,INPUT

        STM    #INPUT2,AR0             ; AR0 contains INPUT2 address
        STM #FILTER_LAST, AR1       ; AR1 contains FILTER address

        ST  #2,samples      ; Set samples, and place the first A/D
                      ; sample into INPUT2
        ; PLACE SAMPLE FROM A/D INTO INPUT2
        LDM    DRR1,B ; LOAD ACCUMULATOR WITH WORD FROM A/D
        STL    B,0,*AR0+ ; Store DDR1 into address pointed by AR0
                   ; Increment AR0 address pointer

        B DESCRAM

NOTN:
        SUB #1,0,A
        BC NOTB, ANEQ
YESB:
        STH B,-7,back
        LD #2,0,A
        STL A,0,synch

        ST #-1,samples
        STM #I1,AR0
        STM #FILTER_LAST,AR1
        ; Place sample from A/D into I1
        LDM DRR1,B
        STL B,0,*AR0+
        B DESCRAM
```

```
NOTB:   ; Time to move forward
        STH B,-7,forward ; store forward
        LD #0,0,A
        STL A,0,synch    ; reset synch

; compare normal, back and forward
        LD normal,0,A
        SUB back,0,A ; A = normal-back
        BC NGB, AGT
BGN:      ; back is greater than normal
        ; compare back and forward
        LD forward,0,A
        SUB back,0,A ; A = forward-back
        BC MOVF, AGT ; MOVF = move forward

MOVB:     ; MOVE TO BACK POSITION
        LD  bcount,0,A
        ADD #1,0,A
        STL A,0,bcount

        STM #FILTER,AR3 ; AR3 points to FILTER
        LD  *AR3+,0,A   ; A contains FILTER, AR3 points to FILTER2
        STL  A,0,INPUT2 ; INPUT2 contains FILTER
        LD  *AR3,0,A    ; A contains FILTER2
        STL A,0,INPUT   ; INPUT contains FILTER2
        STM #INPUT3,AR0 ; AR0 points to INPUT3
        STM #FILTER_LAST,AR1 ; AR1 points to FILTER_LAST
        LDM DRR1,B      ; B contains latest A/D sample
        STL B,0,*AR0+   ; Store latest sample in INPUT3
        ST #3,samples   ; samples = 3
        B DESCRAM
NGB:   ; normal is greater than back
     ; compare normal and forward
        nop

        LD forward,0,A
        SUB normal,0,A ; A = forward-normal
        BC MOVF, AGT ; MOVF = move forward
MOVN:     ; MOVE TO NORMAL POSITION
        STM #FILTER,AR3; AR3 points to FILTER
        LD *AR3,0,A   ; A contains FILTER
        STL A,0,INPUT ; INPUT contains FILTER
        STM #INPUT2, AR0; AR0 points to INPUT2
        STM #FILTER_LAST,AR1; AR1 points to FILTER_LAST
        LDM DRR1,B
        STL B,0,*AR0+; Store latest sample in INPUT2
        ; 2 samples have been placed in the INPUT array
        ST #2,samples

        LD  ncount,0,A
        ADD #1,0,A
        STL A,0,ncount
        B DESCRAM
MOVF:

        STM    #INPUT,AR0             ; AR0 contains INPUT address
        STM #FILTER_LAST, AR1       ; AR1 contains FILTER address

        ST  #1,samples      ; Set samples, and place the first A/D
                        ; sample into INPUT
        ; PLACE SAMPLE FROM A/D INTO INPUT
        LDM    DRR1,B ; LOAD ACCUMULATOR WITH WORD FROM A/D
        STL    B,0,*AR0+ ; Store DDR1 into address pointed by AR0
                    ; Increment AR0 address pointer

        LD  fcount,0,A
        ADD #1,0,A
        STL A,0,fcount
        B DESCRAM



DESCRAM:
        STM #S1LAST, AR2
        RPTZ A,#8
        MACD *AR2-,h1,A
        SFTL A,-1

        ADD BIT,0,A
        AND #1,A   ; A is the descrambled bit
        ADD errors,0,A
        STL A,0,errors
        BC  noerr, AEQ
         nop
noerr:
        ; Update the state by moving BIT into S1
        LD BIT,0,A
        STL A,0,S1

; Increment the number of bits received (recbits)
        LD recbits,0,A
        ADD #1,A
        STL A,0,recbits
        SUB #30000,0,A
        BC   NOTYET, ANEQ
        ST #0,recbits ; reset  recbits
        LD packets,0,A
        ADD #1,A
        STL A,0,packets
        SUB #3,0,A
        BC   NOTYET, ANEQ
        NOP
        NOP
        NOP
NOTYET:     B  DONE
```

```
NEXT24:
        ; PLACE SAMPLE FROM A/D INTO INPUT
        LDM    DRR1,B ; LOAD ACCUMULATOR WITH WORD FROM A/D
        STL    B,0,*AR0+ ; Store DDR1 into address pointed by AR0
                   ; Increment AR0 address pointer

        B    DONE

NEXT48:
        ; PLACE SAMPLE FROM A/D INTO FILTER
        LDM    DRR1,B ; LOAD ACCUMULATOR WITH WORD FROM A/D
        STL    B,0,*AR1- ; Store DDR1 into address pointed by AR1
                   ; Decrement AR1 address pointer


DONE:

        LD      #0,0,A            ; OUTPUT ==>Accumulator A
                          ; TWO   LSB's MUST BE ZERO FOR AIC!
        STLM    A,DXR1                ; SEND TO TRANSMIT REGISTER!

        RETE
transmit:   RETE

*********************END OF main.asm ******************************************
```

FILENAME : main.cmd
PURPOSE :   Defines memory map

```
********************START OF main.cmd *****************************************
init_aic.obj
init_ser.obj
init_54x.obj
aic_cfg.obj
vectors.obj
coeffs.obj
main.obj

-e main_start
-o main.out
-m main.map

MEMORY
        {

    PAGE 0:                        /*   Pgm.space */
        COEFF   : origin = 0x1400, length = 0x80    /* coefficients */
        PROG    : origin = 0x7000, length = 0x2000  /*   Ext.Pgm.area */
        VECS    : origin = 0xff80, length = 0x7f    /* Vector */
    PAGE 1:                        /* Data space */
              RAM0   : origin = 0x0060, length = 0x20
        RAM     : origin = 0x0080, length = 0x040   /* Ext. RAM space */

              RAM1   : origin = 0x0100, length  = 0x80
              RAM2   : origin = 0x0200, length = 0x100
              RAM3   : origin = 0x1000, length = 0x400
              RAM4   : origin = 0x2000, length = 0x400

              RAM5   : origin = 0x3000, length = 0x400

              RAM6   : origin = 0x4000, length = 0x1600
              RAM7   : origin = 0x6000, length = 0x400
              RAM8   : origin = 0x6800, length = 0x200
        REGS    : origin = 0x0000, length = 0x0060   /*   MMR's        */
        }

SECTIONS
    {
    .text          : {} > PROG    PAGE 0              /* code */
    vectors        : {} > VECS    PAGE 0              /* Vector table */
    wrt_cnfg       : {} > PROG    PAGE 0
    filter         : {} > COEFF   PAGE 0
    .bss           : {} > RAM2    PAGE 1              /* variables    */
    .data          : {} > RAM2    PAGE 1
    aic_vars       : {} > RAM2    PAGE 1
    rcv_vars       : {} > RAM2    PAGE 1
    stack          : {} > RAM8    PAGE 1
        }
********************END OF main.cmd *****************************************
```

MBPSK TRANSMITTER FILES

FILENAME : TX.BAT
PURPOSE : Assembles and links the transmitter modules

```
*************************START OF TX.BAT *****************************************
asm500 -ls aic_cfg.asm
asm500 -ls coeffs.asm
asm500 -ls init_54x.asm
asm500 -ls init_aic.asm
asm500 -ls init_ser.asm
asm500 -ls vectors.asm
asm500 -ls main.asm
lnk500 -v0 main.cmd
*************************END OF TX.BAT ****************************************
```

FILENAME : aic_cfg.asm
PURPOSE : Configures the AIC

```
************************START OF aic_cfg.asm *****************************************
        .mmregs
        .include  "aic_cfg.inc"
        .def      wrt_cnfg
        .def      wait_xrdy

        .text

wrt_cnfg:
        STM     #K_0,DXR1                  ;primary data word - a jump start!
        CALL    wait_xrdy
        STM     #K_PRIM_WORD,DXR1          ;send primary word with D01-D00 = 11 to
                                           ;signify secondary communication
        CALL    wait_xrdy                  ;wait for data to be copied from DXR to XSR
        STM     #K_REG_1,DXR1                      ;change A register
        CALL    wait_xrdy
        STM     #K_PRIM_WORD,DXR1
        CALL    wait_xrdy
        STM     #K_REG_2,DXR1                      ;change B register
        CALL    wait_xrdy
        RET
;******************************************************************************
;       MODULE: WAIT_XRDY
;       PURPOSE: Wait for data to be copied from DXR to XSR before the next
;                data is sent to DXR
;******************************************************************************
wait_xrdy:
        BITF    SPC1,0800h                 ;test XRDY bit in SPC1
        BC      wait_xrdy,NTC              ;loop if not set
        RET
************************END OF aic_cfg.asm *****************************************
```

FILENAME : coeffs.asm
PURPOSE : Stores bandpass FIR filter coefficients ; passband from 1800 to 2200 Hz

```
************************START OF coeffs.asm ******************************************
AMCOEFF         .sect       "filter"       ; filter coefficients

                . def     TA, RA, TB, RB, AIC_CTR, coeff
                . def     bp701
bp701   .word   4
bp702   .word   -13
bp703   .word   -11
bp704   .word   9
bp705   .word   17
bp706   .word   0
bp707   .word   -19
bp708   .word   -9
bp709   .word   12
bp7010   .word   12
bp7011   .word   -2
bp7012   .word   1
bp7013   .word   3
bp7014   .word   -25
bp7015   .word   -34
bp7016   .word   37
bp7017   .word   98
bp7018   .word   -1
bp7019   .word   -167
bp7020  .word   -105
bp7021   .word   185
bp7022   .word   265
bp7023   .word   -95
bp7024   .word   -412
bp7025   .word   -118
bp7026   .word   452
bp7027   .word   404
bp7028   .word   -315
bp7029   .word   -649
bp7030   .word   2
bp7031   .word   731
bp7032   .word   393
bp7033   .word   -578
bp7034   .word   -723
bp7035   .word   220
bp7036   .word   851
bp7037   .word   220
bp7038   .word   -723
bp7039   .word   -578
bp7040   .word   393
bp7041   .word   731
bp7042   .word   2
bp7043   .word   -649
bp7044   .word   -315
bp7045   .word   404
```

```
bp7046   .word   452
bp7047   .word   -118
bp7048  .word   -412
bp7049  .word   -95
bp7050   .word   265
bp7051   .word   185
bp7052   .word   -105
bp7053   .word   -167
bp7054   .word   -1
bp7055   .word   98
bp7056   .word   37
bp7057   .word   -34
bp7058   .word   -25
bp7059   .word   3
bp7060   .word   1
bp7061   .word   -2
bp7062   .word   12
bp7063  .word   12
bp7064   .word   -9
bp7065   .word   -19
bp7066   .word   0
bp7067   .word   17
bp7068   .word   9
bp7069   .word   -11
bp7070   .word   -13
bp7071   .word   4



TA         .word    16         ;
RA         .word    16         ;   This set up of AIC registers give
                          ;  a sampling   freq of 10,081 Hz
                          ;
TB         .word    31         ;
RB         .word    31         ;
AIC_CTR            .word    0ch
coeff           .word    5219       ; 10438

***********************END OF coeffs.asm ****************************************
```

FILENAME : init_54x.asm
PURPOSE : Initializes the C54x DSP

```
********************START OF init_54x.asm ****************************************
.include "init_54x.inc"          ; contains all the initial
                                   ; values of ST0, ST1, PMST, SWWSR,
                                   ; BSCR
          . mmregs
          . def    init_54

K_INIT_DP        .set        0

          .text

init_54:
* initialize the status and control registers
          STM     #K_ST0, ST0
          STM     #K_ST1, ST1
          STM     #K_PMST,PMST

* wait states for Bank Switch
          STM     #K_BSCR, BSCR          ; 0 wait states for BANK
                                   ; SWITCH

* Init. the s/w wait state reg.for 2 wait states for I/O operations
          RETD
          STM     #K_SWWSR_IO, SWWSR      ; 2 wait states for I/O
                                   ; operations
          .end

**********************END OF init_54x.asm ****************************************
```

```
 FILENAME :  init_ser.asm
PURPOSE :   Initializes the AC01

*********************START OF init_ser.asm ****************************************
.def     serial_init

             .include " init_ser.inc"
             .include " interrpt.inc"
             . mmregs
aic_in_rst      . usect   "aic_vars",1
aic_out_of_rst  .usect   "aic_vars",1

K_TCR          .set     14h                   ; Timer Control Register
K_0            .set     0h
K_8000         .set     8000h

             .text
serial_init:
             LD       # aic_in_rst,DP          ; initialize the DP for  aic_reset
             ST       #K_0,  aic_in_rst        ; bit 15 = 0 of TCR resets the AIC
             PORTW     aic_in_rst,K_TCR        ; write to address 14h (TCR)
*****************************************************************************
* We need at least 12 cycles to pull the AIC out of reset. Only 6 cycles
* have been used.
*****************************************************************************
             STM     #K_SERIAL_RST, SPC1      ; reset the serial port with
                                   ; 0000   0000 0000 1000
             STM     #K_SERIAL_OUT_RST, SPC1  ; bring  ser.port out of reset with
                                   ; 0000   0000 1100 1000
* Write some value to DXR1 (the AC01 on the EVM is connected to serial port 1)

             RSBX    INTM

             STM     #K_IMR,IMR              ; Enable RINT1
                                   ; 0000   0000 0100 0000
             STM     #K_IMR,IFR             ; clear RINT1

             STM     #K_0,DXR1
                                   ; 0000   0000 0100 0000
             NOP                        ; this guarantees that the AIC is held for    atleast 12 cycles in reset
* Pull the AC01 out of reset - the AC01 requires that it be held in reset for
* 1 MCLK, which is equivalent to 96.45ns (based on an MCLK of 10.368MHz)

             ST      #K_8000,  aic_out_of_rst  ; bit 15 = 1 brings AIC from reset

             RETD
             PORTW    aic_out_of_rst, K_TCR    ; AIC out of reset
             .end
*********************END OF init_ser.asm ****************************************
```

FILENAME : vectors.asm
PURPOSE :   Initializes the C54x vector table

```
********************START OF vectors.asm ***************************************


.title  "54X Vector Table Initialization"

          . ref    main_start
          . ref    receive
          . ref    transmit

          . def    STACK



          . mmregs
K_STACK_SIZE    .set    80

STACK        . usect  "stack",K_STACK_SIZE

          .sect   "vectors"
reset:      BD      main_start          ; RESET vector
          STM     #K_STACK_SIZE+STACK, SP  ; initialize stack pointer to  stack_start
nmi:        NOP                     ; ~NMI
                NOP
                NOP
                RET

* software interrupts
sint17       .space  4*16
sint18       .space  4*16
sint19       .space  4*16
sint20       .space  4*16
sint21       .space  4*16
sint22       .space  4*16
sint23       .space  4*16
sint24       .space  4*16
sint25       .space  4*16
sint26       .space  4*16
sint27       .space  4*16
sint28       .space  4*16
sint29       .space  4*16
sint30       .space  4*16

int0:       RETF                    ; INT0
                NOP
                NOP
                NOP
int1:       RETF                    ; INT1
                NOP
                NOP
                NOP
int2:       RETF                    ; INT2
                NOP
                NOP
                NOP
timer:      RETF                    ; TIMER
                NOP
                NOP
            NOP
rint0:      B       receive         ; Serial Port Receive
            NOP                     ; Interrupt 0
                NOP
xint0:      RETF                    ; Serial Port Transmit
            NOP                     ; Interrupt 0
                NOP
                NOP
rint1:      B       receive         ; Serial Port Receive
            NOP                     ; Interrupt 1
                NOP
xint1:      RETF                    ; Serial Port Transmit
            NOP                     ; Interrupt 1
                NOP
                NOP
int3:       RETF                    ; INT3
                NOP
                NOP
                NOP
                .end
```

*********************END OF vectors.asm ***

FILENAME : main.asm
PURPOSE : Main routine; includes the MBPSK transmitter and noise generator
Note: noise is added!!!

```
*********************START OF main.asm *****************************************
.title  "MBPSK Modulation C54x"

        . def     main_start          ; reset vector
        . ref     aic_init            ; writes new  config into AC01
        . ref     serial_init          ;  intialize serial port
        . ref     init_54              ; initialize ST0,ST1, PMST
        . def     receive
        . def     transmit

         . ref     coeff,bp701

        . mmregs

        .include " interrpt.inc"

        .data


;; DEFINE CONSTANTS for randum number generator:

RNDSEED         .set        21845                ; seed value (i.e. rndnum(1) = 21845)
RNDMULT         .set        31821                ; Multiplier value
RNDINC          .set        13849                ; Increment value

RNDSEED2 .set    1945           ; seed value (i.e.  rndnum(1) = 21845)
RNDMULT2 .set    31821             ; Multiplier value
RNDINC2  .set    13849          ; Increment value
rndnum      .word     0
rndnum2     .word     0



v_addr        .word   0
noise            .word   0


;; OTHER CONSTANTS

txsample    .word    0

sinx        .word    15h
Yminus1     .word    026fh ;
samples     .word    -1
XN          .word    0
XNspace      .word   0                   ; extra word for the bit bucket

;%%%%% variable for the sine calculator
d_coff    .word 01c7h, 030bh, 0666h, 1556h
```

```
d_x       .word 0
d_squr_x  .word 0
d_temp    .word 0
d_sinx    .word 0
C_1       .word 7fffh; 1 in fractional mode
;%%%%%%%%%%%%%

; 70 data locations for 70 stage delay line
BP1        .word    0,0,0,0,0,0,0,0,0,0
BP2        .word    0,0,0,0,0,0,0,0,0,0
BP3        .word    0,0,0,0,0,0,0,0,0,0
BP4        .word    0,0,0,0,0,0,0,0,0,0
BP5        .word    0,0,0,0,0,0,0,0,0,0
BP6        .word    0,0,0,0,0,0,0,0,0,0
BP7        .word    0,0,0,0,0,0,0,0,0,0
BPLAST      .word    0
BPspace     .word    0
BPout       .word    0


;XNPAST      .word    0,0,0,0  ; FOR [2,5]
XNPAST      .word    0,0,0,0,0,0,0,0,0  ; FOR [4,9]
XNLAST      .word    1        ; 5 data locations for 5 stage delay

YN          .word    0
OUTPUT1     .word    10; was 10
OUTPUT2     .word    -10
OUTPUT      .word    -10; -10 GIVES CHANGE AFTER 24 SAMPLES
notsure      .word    1
;h1          .word    1,0,0,1,0 ; FOR [2,5]
h1          .word     1,0,0,0,0,1,0,0,0 ; for [4,9]
            .text

noise1  .word 0
;%%%%%%%%%% v(n)
v1   .word   0
v2   .word   1876
v3   .word   2664
v4   .word   3275
v5   .word   3798
v6   .word   4263
v7   .word   4690
v8   .word   5087
v9   .word   5462
v10  .word   5819
v11  .word   6161
v12  .word   6492
v13  .word   6812
v14  .word   7123
v15  .word   7427
v16  .word   7725
v17  .word   8018
v18  .word   8306
v19  .word   8590
```

```
v20  .word   8872
v21  .word   9150
v22  .word   9427
v23  .word   9702
v24  .word   9975
v25  .word  10248
v26  .word  10520
v27  .word  10793
v28  .word  11065
v29  .word  11339
v30  .word  11613
v31  .word  11889
v32  .word  12166
v33  .word  12445
v34  .word  12727
v35  .word  13012
v36  .word  13300
v37  .word  13591
v38  .word  13887
v39  .word  14187
v40  .word  14493
v41  .word  14804
v42  .word  15122
v43  .word  15447
v44  .word  15780
v45  .word  16122
v46  .word  16473
v47  .word  16836
v48  .word  17211
v49  .word  17600
v50  .word  18005
v51  .word  18428
v52  .word  18872
v53  .word  19340
v54  .word  19837
v55  .word  20366
v56  .word  20936
v57  .word  21556
v58  .word  22237
v59  .word  22999
v60  .word  23868
v61  .word  24890
v62  .word  26150
v63  .word  27828
v64  .word  30484

main_start:
        CALL     init_54            ; initialize ST0,ST1 PMST
                                    ;   and other   regsiters
        CALL     serial_init        ; initialize serial port
        CALLD    aic_init           ; initialize AIC
        LD       #0,DP

        nop
```

```
            nop

;; Initialize Random Number Generator -      Load the SEED value
InitRand16:
            ST #RNDSEED,  rndnum


WAIT:      B      WAIT                ; wait for receive interrupt
receive:

        LD      #XN,DP ;???????????????????????????????????


;; Generate Next Random  Number:
        LD rndnum2,T
        ST #RNDMULT2, rndnum2
        MPYU rndnum2, A
        SFTL A,-1
        ADD #RNDINC2, A
        STL A, rndnum2 ; rndnum2 contains the random angle

sin_start:
                 STM #d_coff,AR5
           STM #d_x,AR2
           STM #C_1,AR4
; Load T with number from random number generator which will be between
; -1 and 1
           LD rndnum2,T
           MPY #6487h,A ; multiply by pi/4
           STH A,0,*AR2 ; store in  d_x
            nop
            nop
            nop
            nop

; for test first try pi/4
;            ST  #6487h,*AR2

           SQUR *AR2+,A ; Square  d_x, increment pointer AR2
           ST   A, *AR2 ; Store the squared term in  d_squr_x
           || LD *AR4,B ; Load B with one
           MASR  *AR2+,*AR5+,B,A
           MPYA  A

           STH   A,*AR2
           MASR  *AR2-,*AR5+,B,A

           MPYA  *AR2+
           ST    B,*AR2
           || LD *AR4,B
           MASR *AR2-,*AR5+,B,A
           MPYA *AR2+
           ST    B,*AR2
           || LD *AR4,B
           MASR  *AR2-,*AR5+,B,A
           MPYA  d_x
           STH   B, d_sinx ; d_sinx contains sin(x)

;; Now generate a random number between 0 and 63 for v(n)
        LD rndnum,T
        ST #RNDMULT, rndnum
        MPYU rndnum, A
        SFTL A,-1
        ADD #RNDINC, A
        STL A, rndnum
; Next step is scale the number between 0 and 63 for relative address
        STL A, v_addr
        ANDM  #63, v_addr
        LD   v_addr,0,A ; A contains relative address

        STM  #v1, AR3

; increment AR3 pointer if A is not zero
GETready:
        BC  AR3ready, AEQ
        SUB  *AR3+,0,B
        SUB  #1,0,A
        B GETready
AR3ready:  ; AR3 now points to the desired v(n) sample



; next step is to multiply d_sinx by *AR6
           LD  *AR3, T; T contains v(n) and n=*AR6 is incremented
           MPY d_sinx,A ; A contains the noise sample
           SFTA A,-1; undo the times 2
           STH A,0,noise; Noise sample obtained !!!!

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Count the number of samples
            LD     samples,0,A
            ADD    #1,A
            STL    A,0,samples
            SUB    #24,A  ; Check to see if 24 samples have been output

            BC     GO_ON, ANEQ; If [A NOT= 0], branch to GO_ON

;;;;;;;;;; GET NEW DATA BIT,y(n), TO TRANSMIT
;(1)y(n)=mod(x(n)h0+x(n-1)h1+x(n-2)h2+x(n-3)h3+x(n-4)h4+x(n-5)h5+y(n-1),2)
;;;; consider the input bit, x(n), to be zero. Now (1) reduces to (2) :
;(2)y(n)=mod(x(n-1)h1 + x(n-2)h2 + x(n-3)h3 + x(n-4)h4 + x(n-5)h5 +y(n-1),2)
; In other words, the input data is being sent through a scrambler. In this
; case the input data just happens to be zero all the time.

            STM    #XNLAST,AR0
            RPTZ   A, #8                 ; Repeat eight times
            MACD   *AR0-,h1,A            ; past inputs times filter   coeffs
;Here is the problem now. MACP gives 2*ans.(Fractional mode!!)
```

```
;So  A/2 i.e. logical shift right by 1
          SFTL   A,-1
          AND    #1,A
          STL    A,0,XNPAST
          BC     NOCHANGE,AEQ ; If A==0 then don't change OUTPUT
          ;  If A==1 then OUTPUT = OUTPUT*-1
CHANGE      LD    OUTPUT,0,A
          BC    PLUS, ALT ; IF OUTPUT IS LESS THAN 1 THEN GET +10
          LD    OUTPUT2,0,A
          B     LOAD
PLUS        LD    OUTPUT1,0,A
LOAD        STL    A,0,OUTPUT
NOCHANGE
          ST     #-24,samples           ; RESET samples
GO_ON
;-------- generate carrier  -----

          LD      Yminus1,16,A          ; y(n-2) ==> ACCUMULATOR A SHIFT
                                ;   LEFT BY 16 BITS! (Q31 format)
          NEG     A                   ; -y(n-2) ==> A
          MACP    sinx,coeff,A          ; (coeff)*y(n-1) - y(n-2) ==> A
          MACD    sinx,coeff,A          ; 2*(coeff)*y(n-1) - y(n-2) ==> A
          STH     A,0,sinx              ; Store the carrier generated into
                                ;   variable   sinx

          LDM     DRR1,A                ; LOAD ACCUMULATOR WITH WORD
                                ;   RECEIVED FROM AIC!

          MACD    OUTPUT,sinx,A         ; Multiply the carrier with the
                                ;   filtered input.  The extra
                                ;   sign bit is automatically
                                ;   discarded with FRCT=1
           STL  A,0,txsample      ;  txsample contains signal sample


; add noise for testing, AR3 points to the desired noise sample
          LD     noise,0,B
          SFTA   B,-2 ; negative means right shift, 0=>SNR =-3

          STL    B,0,noise  ; store the right shifted value in noise
          LD     noise,T
          MPY    #2,B ; will multiply by 2
          SFTA   B,-1
          STL    B,0,BP1  ; BP1 is the latest noise sample



          STM    #BPLAST,AR0
          RPTZ   A,#70
          MACD   *AR0-,bp701,A ; This will multiply noise by 2*(2^13)
          SFTA   A,-14; Undo the filter gain (2^13) and MACD gain (2)
          ADD    txsample,0,A
          LD     txsample,0,A


          AND     #0FFFCh,A          ; TWO  LSB's MUST BE ZERO FOR AIC!

          STLM    A,DXR1             ; SEND TO TRANSMIT REGISTER!

          RETE                    ; Enable interrupts and return
                               ;   from interrupt


transmit:   RETE                   ; Enable interrupts and return
                               ;   from interrupt

          .end
```

FILENAME : main.cmd
PURPOSE : Defines memory map

```
********************START OF main.cmd ****************************************
init_aic.obj
init_ser.obj
init_54x.obj
aic_cfg.obj
vectors.obj
coeffs.obj
main.obj


-e main_start
-o main.out
-m main.map

MEMORY
        {

    PAGE 0:                              /*  Pgm.space */
        COEFF  : origin = 0x1400, length = 0x80   /* coefficients */
        PROG   : origin = 0x7000, length = 0x2000  /*  Ext.Pgm.area */
        VECS   : origin = 0xff80, length = 0x7f   /* Vector */
    PAGE 1:                              /* Data space */
              RAM0  : origin = 0x0060, length = 0x20
        RAM    : origin = 0x0080, length = 0x040  /* Ext. RAM space */

              RAM1  : origin = 0x0100, length  = 0x80
        RAM2   : origin = 0x0200, length = 0x100
              RAM3  : origin = 0x1000, length = 0x400
              RAM4  : origin = 0x2000, length = 0x4 00

              RAM5  : origin = 0x3000, length = 0x400

              RAM6  : origin = 0x4000, length = 0x1600
              RAM7  : origin = 0x6000, length = 0x400
              RAM8  : origin = 0x6800, length = 0x200
        REGS   : origin = 0x0000, length = 0x0060  /*  MMR's      */
        }
SECTIONS
    {
    .text       : {} > PROG   PAGE 0          /* code */
    vectors     : {} > VECS   PAGE 0          /* Vector table */
    wrt_cnfg    : {} > PROG   PAGE 0
    filter      : {} > COEFF  PAGE 0
    .bss        : {} > RAM2   PAGE 1          /* variables   */
    .data       : {} > RAM2   PAGE 1
    aic_vars    : {} > RAM2   PAGE 1
    rcv_vars    : {} > RAM2   PAGE 1
    stack       : {} > RAM8   PAGE 1
        }
********************END OF main.cmd ****************************************
```

APPENDIX C SIGMA-DELTA CONVERTER SIMULATION

```
% PROGRAM NAME : SD


%%% 1st order Sigma Delta Converter %%%
clear all;
close all;
% generate the input signal %
N =2000; % number of samples
M = 50; % oversampling ratio (OSR)
fm = 1;
t1 = 0:1:N;
fs = 1/(2*fm); % Signal rate * 2
% A virtual analog signal is constructed by oversampling by M
x = sin(2*pi*fs*t1/M);

figure;
[psdx,fx]=psd(x);
px=10*log10(psdx)-max(10*log10(psdx));
plot(fx,px);
grid on;
xlabel('Normalized Frequency')
ylabel('PSD of x(t)  (dB)');
axis([0 1 -160 10]);

% Initialize variables
y(1) = 0; % A/D output
w(1) = 0;

for n = 2:1:N % time index
   % analog signal is given by x(t) or x(n)
   % A/D output is y(n)
   s(n) = x(n)- y(n-1); % s(n) is the integrator input
   w(n) = s(n) + w(n-1);
if w(n)>0
      y(n) = 1;
   else
      y(n)=-1;
   end

end

figure;
[psdy,fy]=psd(y);
py=10*log10(psdy)-max(10*log10(psdy));
plot(fy,py);
grid on;
xlabel('Normalized Frequency')
ylabel('PSD of y[n]  (dB)');
axis([0 1 -50 10]);
```

```
% y(n) is the output of the sigma-delta quantizer
% next step involves lowpass filtering and decimation
lpnum=[1,zeros(1,M-1),-1];
lpden = [1,-1];
d=filter(lpnum,lpden,y);
% number of bits available
b=16;
% possible signal levels = (2^b)
d1=((2^b)/2)*d/max(d+1);
d2=floor(d1);

figure;
[psd2,fd2]=psd(d2);
pd2=10*log10(psd2)-max(10*log10(psd2));
plot(fd2,pd2);
grid on;
xlabel('Normalized Frequency')
ylabel('PSD of lowpass filter output (dB)');
axis([0 1 -80 10]);



% decimate by M
for n=1:(N/M)
   d3(n) = d2(n*M);
end



figure;
[psd3,fd3]=psd(d3);
pd3=10*log10(psd3)-max(10*log10(psd3));
plot(fd3,pd3);
grid on;
xlabel('Normalized Frequency')
ylabel('PSD of A/D converter output (dB)');
axis([0 1 -80 10]);
```

Bibliography

[Far93] D. Farley, “How Safe is Your Oven?” *The National Food Safety Database*, http://www.foodsafety.org/sf/sf180.htm (current 1993)

[Feh95] K. Feher, *Wireless Digital Communications : Modulation and Spread Spectrum Applications*, Prentice Hall PTR, New Jersey, 1995.

[NTIA1] P. E. Gawthrop, F. H. Sanders, K. B. Nebbia, J. J. Sell, “Radio Spectrum Measurements of Individual Microwave Ovens,” NTIA Report 94-303-1.

[NTIA2] P. E. Gawthrop, F. H. Sanders, K. B. Nebbia, J. J. Sell, “Radio Spectrum Measurements of Individual Microwave Ovens,” NTIA Report 94-303-2.

[Gmb91] R.B. GmbH, *CAN Specification, Version 2.0*, tech. Report, BOSCH, D-7000 Stuttgart, Germany, 1991.

[KEP96] A. Kutlu, A., H. Ekiz, E.T. Powner, “Performance Analysis of MAC Protocols for Wireless Control Area Network,” *Second Int’l Symp on Parallel Architectures, Algorithm and Networks*, IEEE Computer Soc. Press, Los Alamitos, Calif., 1996, pp.494-499.

[Mit98] S. K. Mitra, *Digital Signal Processing : a computer-based approach*. McGraw-Hill, New York, 1998.

[Pfe95] O. O. Pfeiffer, “Controller Area Network : The Future of Industrial Microprocessor Communication?” *Hitex UK Ltd*, http://www.hitex.com/automation/docs/canintro/ (current Jan. 1995).

[PrM96] J. G. Proakis, and D. G. Manolakis. *Digital Signal Processing : principles, algorithms and applications*. Prentice-Hall, N.J., 1996.

[Rap96] T.S. Rapappport, T.S., *Wireless Communications : Principles and Practice*, Prentice Hall PTR, New Jersey, 1996.

[Sch98] M. J. Schofield, "Controller Area Network - How CAN Works" http://www.omegas.co.uk/CAN/canworks.htm (current Nov. 1998)

[Ter95] S.A. Tretter, S. A., C*ommunication System Design Using DSP Algorithms*, Plenum Press, New York, 1995.

[TI196] *Tms320C54x DSP Applications Guide,* Texas Instruments Incorporated, Reference Set Vol.4 , October 1996

[TI198] *Tms320C54x DSP Assembly Language Tools,* Texas Instruments Incorporated, User's Guide , October 1998

[TI296] *Tms320C54x DSP Mnemonic Instruction Set*, Texas Instruments Incorporated, Reference Set Vol.2 , October 1996

[TI298] "TMS320VC5402 Features", Texas Instruments Incorporated, 1998 http://www.ti.com/sc/docs/dsps/products/c5000/c54x/5402.htm (5 Nov. 1998)

[Wic95] S.B. Wicker, *Error Control Systems for Digital Communication and Storage,* Prentice-Hill, Upper Saddle River,N.J., 1995, pp. 175-186.

[Zyr98] J. Zyren, Al Petrick, "Tutorial on Basic Link Budget Analysis," *Application Note AN9804.1*, Harris Corporation, June 1998.

Vita

Punit Mukhija was born on August 2, 1974, in New Delhi, India. He received his Bachelors in Electrical Engineering in August 1997 from the University of Tennessee, Knoxville. He joined Virginia Tech and the Center for Wireless Telecommunications in Fall 1997. His areas of interest include digital signal processing and wireless communications.