*micro*ID™ 125 kHz

RFID SYSTEM DESIGN GUIDE

MICROCHIP

microID™ 125 kHz RFID System Design Guide

INCLUDES:

- Passive RFID Basics Application Note
- MCRF200 Data Sheet
- MCRF250 Data Sheet
- Contact Programming Support
- RFID Coil Design
- FSK Reader Reference Design
- PSK Reader Reference Design
- ASK Reader Reference Design
- FSK Anti-Collision Reader Reference Design
- Using the microID Programmer

DATA SHEET MARKINGS

Microchip uses various data sheet markings to designate each document phase as it relates to the product development stage. The markings appear at the bottom of the data sheet, between the copyright and document and page numbers. The definitions for each marking are provided below for your use.

Marking	Description
Advance Information	The information is on products in the design phase. Your designs should not be finalized with this information as revised information will be published when the product becomes available.
Preliminary	This is preliminary information on new products in production but not yet fully characterized. The specifications in these data sheets are subject to change without notice. Before you finalize your design, please ensure that you have the most current revision of the data sheet by contacting your Microchip sales office.
No Marking	Information contained in the data sheet is on products in full production.

Trademarks

Table of Contents

PAGE

AN680 PASSIVE RFID BASICS

MCRF200 DATA SHEET

Table of Contents

PAGE

MCRF250 DATA SHEET

TB023 CONTACT PROGRAMMING SUPPORT

Table of Contents

PAGE

AN678 RFID COIL DESIGN

FSK READER REFERENCE DESIGN

PSK READER REFERENCE DESIGN

Table of Contents

PAGE

ASK READER REFERENCE DESIGN

FSK ANTICOLLISION READER REFERENCE DESIGN

USING THE microID™ PROGRAMMER

WORLDWIDE SALES AND SERVICE

AN680

Passive RFID Basics

Author: Pete Sorrells
Microchip Technology Inc.

INTRODUCTION

Radio Frequency Identification (RFID) systems use radio frequency to identify, locate and track people, assets, and animals. Passive RFID systems are composed of three components – an interrogator (reader), a passive tag, and a host computer. The tag is composed of an antenna coil and a silicon chip that includes basic modulation circuitry and non-volatile memory. The tag is energized by a time-varying electromagnetic radio frequency (RF) wave that is transmitted by the reader. This RF signal is called a *carrier signal*. When the RF field passes through an antenna coil, there is an AC voltage generated across the coil. This voltage is rectified to supply power to the tag. The information stored in the tag is transmitted back to the reader. This is often called backscattering. By detecting the backscattering signal, the information stored in the tag can be fully identified.

DEFINITIONS

Reader

Usually a microcontroller-based unit with a wound output coil, peak detector hardware, comparators, and firmware designed to transmit energy to a tag and read information back from it by detecting the backscatter modulation.

Tag

An RFID device incorporating a silicon memory chip (usually with on-board rectification bridge and other RF front-end devices), a wound or printed input/output coil, and (at lower frequencies) a tuning capacitor.

Carrier

A Radio Frequency (RF) sine wave generated by the reader to transmit energy to the tag and retrieve data from the tag. In these examples the ISO frequencies of 125 kHz and 13.56 MHz are assumed; higher frequencies are used for RFID tagging but the communication methods are somewhat different. 2.45 GHz, for example, uses a true RF link. 125 kHz and 13.56 MHz, utilize transformer-type electromagnetic coupling.

Modulation

Periodic fluctuations in the amplitude of the carrier, used to transmit data back from the tag to the reader.

Systems incorporating passive RFID tags operate in ways that may seem unusual to anyone who already understands RF or microwave systems. There is only one transmitter – the passive tag is not a transmitter or transponder in the purest definition of the term, yet bidirectional communication is taking place. The RF field generated by a tag reader (the energy transmitter) has three purposes:

1. **Induce enough power into the tag coil to energize the tag.** Passive tags have no battery or other power source; they must derive all power for operation from the reader field. 125 kHz and 13.56 MHz tag designs must operate over a vast dynamic range of carrier input, from the very near field (in the range of 200 V_{PP}) to the maximum read distance (in the range of 5 V_{PP}).
2. **Provide a synchronized clock source to the tag.** Most RFID tags divide the carrier frequency down to generate an on-board clock for state machines, counters, etc., and to derive the data transmission bit rate for data returned to the reader. Some tags, however, employ on-board oscillators for clock generation.
3. **Act as a carrier for return data from the tag.** Backscatter modulation requires the reader to peak-detect the tag's modulation of the reader's own carrier. See Section for additional information on backscatter modulation.

SYSTEM HANDSHAKE

Typical handshake of a tag and reader is as follows:

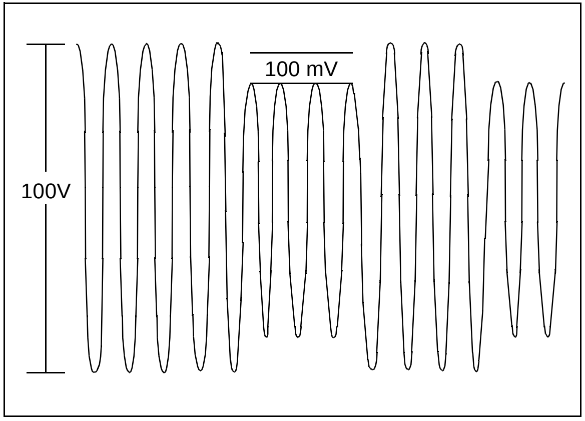
1. The reader continuously generates an RF carrier sine wave, watching always for modulation to occur. Detected modulation of the field would indicate the presence of a tag.
2. A tag enters the RF field generated by the reader. Once the tag has received sufficient energy to operate correctly, it divides down the carrier and begins clocking its data to an output transistor, which is normally connected across the coil inputs.
3. The tag's output transistor shunts the coil, sequentially corresponding to the data which is being clocked out of the memory array.
4. Shunting the coil causes a momentary fluctuation (dampening) of the carrier wave, which is seen as a slight change in amplitude of the carrier.
5. The reader peak-detects the amplitude-modulated data and processes the resulting bitstream according to the encoding and data modulation methods used.

BACKSCATTER MODULATION

This terminology refers to the communication method used by a passive RFID tag to send data back to the reader. By repeatedly shunting the tag coil through a transistor, the tag can cause slight fluctuations in the reader's RF carrier amplitude. The RF link behaves essentially as a transformer; as the secondary winding (tag coil) is momentarily shunted, the primary winding (reader coil) experiences a momentary voltage drop. The reader must peak-detect this data at about 60 dB down (about 100 mV riding on a 100V sine wave) as shown in Figure 1.

This amplitude-modulation loading of the reader's transmitted field provides a communication path back to the reader. The data bits can then be encoded or further modulated in a number of ways.

FIGURE 1: AMPLITUDE – MODULATED BACKSCATTERING SIGNAL

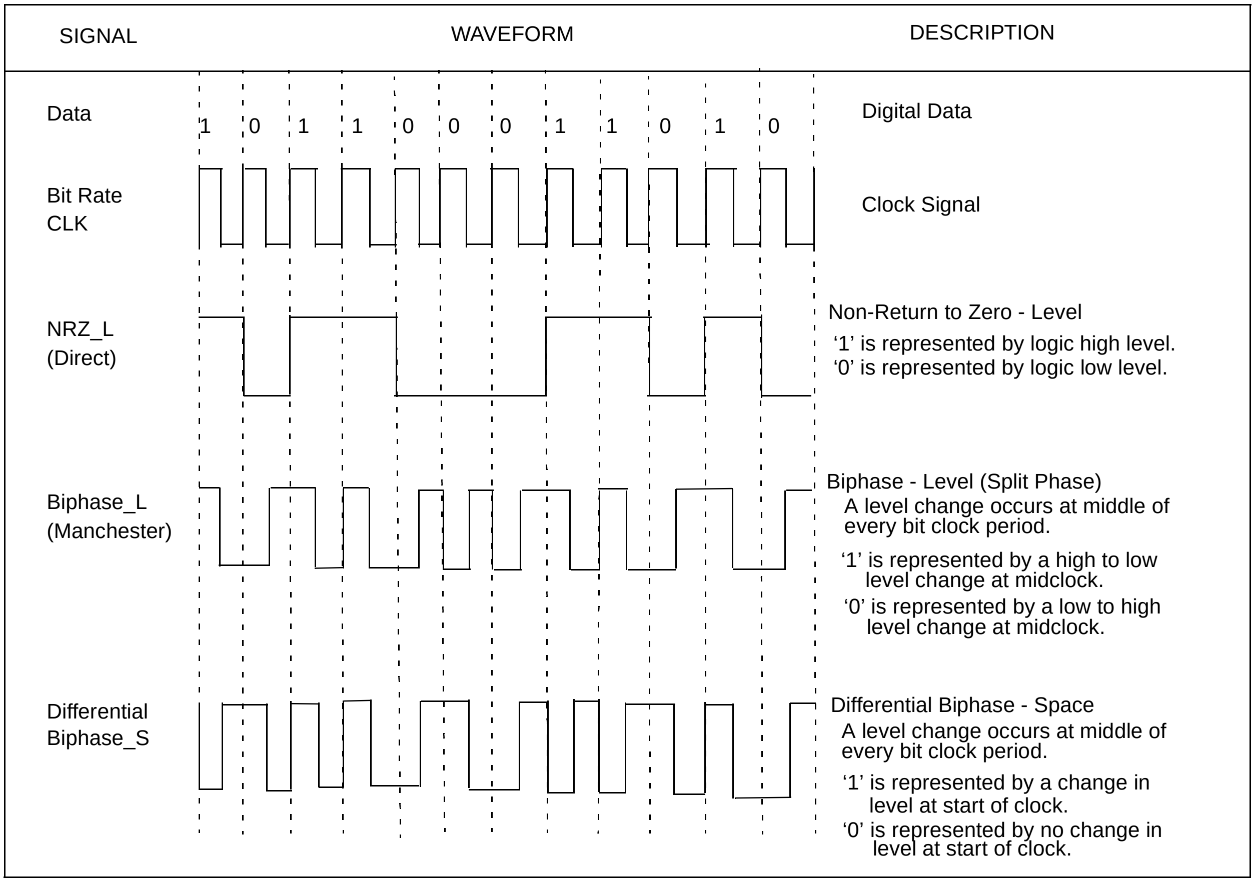


DATA ENCODING

Data encoding refers to processing or altering the data bitstream in-between the time it is retrieved from the RFID chip's data array and its transmission back to the reader. The various encoding algorithms affect error recovery, cost of implementation, bandwidth, synchronization capability, and other aspects of the system design. Entire textbooks are written on the subject, but there are several popular methods used in RFID tagging today:

1. **NRZ (Non-Return to Zero) Direct.** In this method no data encoding is done at all; the 1's and 0's are clocked from the data array directly to the output transistor. A low in the peak-detected modulation is a '0' and a high is a '1'.
2. **Differential Biphase.** Several different forms of differential biphase are used, but in general the bitstream being clocked out of the data array is modified so that a transition always occurs on every clock edge, and 1's and 0's are distinguished by the transitions within the middle of the clock period. This method is used to embed clocking information to help synchronize the reader to the bitstream; and because it always has a transition at a clock edge, it inherently provides some error correction capability. Any clock edge that does not contain a transition in the data stream is in error and can be used to reconstruct the data.
3. **Biphase_L (Manchester).** This is a variation of biphase encoding, in which there is not always a transition at the clock edge.

FIGURE 2: VARIOUS DATA CODING WAVEFORMS


SIGNAL
WAVEFORM
DESCRIPTION
Data
1 0 1 1 0 0 0 1 1 0 1 0
Digital Data
Bit Rate CLK
Clock Signal
NRZ_L (Direct)
Non-Return to Zero - Level
'1' is represented by logic high level.
'0' is represented by logic low level.
Biphase_L (Manchester)
Biphase - Level (Split Phase)
A level change occurs at middle of every bit clock period.
'1' is represented by a high to low level change at midclock.
'0' is represented by a low to high level change at midclock.
Differential Biphase_S
Differential Biphase - Space
A level change occurs at middle of every bit clock period.
'1' is represented by a change in level at start of clock.
'0' is represented by no change in level at start of clock.

DATA MODULATION

Although all the data is transferred to the host by amplitude-modulating the carrier (backscatter modulation), the actual modulation of 1's and 0's is accomplished with three additional modulation methods:

1. **Direct.** In direct modulation, the Amplitude Modulation of the backscatter approach is the only modulation used. A high in the envelope is a '1' and a low is a '0'. Direct modulation can provide a high data rate but low noise immunity.
2. **FSK (Frequency Shift Keying).** This form of modulation uses two different frequencies for data transfer; the most common FSK mode is Fc/8/10. In other words, a '0' is transmitted as an amplitude-modulated clock cycle with period corresponding to the carrier frequency divided by 8, and a '1' is transmitted as an amplitude-modulated clock cycle period corresponding to the carrier frequency divided by 10. The amplitude modulation of the carrier thus switches from Fc/8 to Fc/10 corresponding to 0's and 1's in the bitstream, and the reader has only to count cycles between the peak-detected clock edges to decode the data. FSK allows for a simple reader design, provides very strong noise immunity, but suffers from a lower data rate than some other forms of data modulation. In Figure 3, FSK data modulation is used with NRZ encoding:
3. **PSK (Phase Shift Keying).** This method of data modulation is similar to FSK, except only one frequency is used, and the shift between 1's and 0's is accomplished by shifting the phase of the backscatter clock by 180 degrees. Two common types of PSK are:
   - Change phase at any '0', or
   - Change phase at any data change (0 to 1 or 1 to 0).

   PSK provides fairly good noise immunity, a moderately simple reader design, and a faster data rate than FSK. Typical applications utilize a backscatter clock of Fc/2, as shown in Figure 4.

FIGURE 3: FSK MODULATED SIGNAL, FC/8 = 0, FC/10 = 1

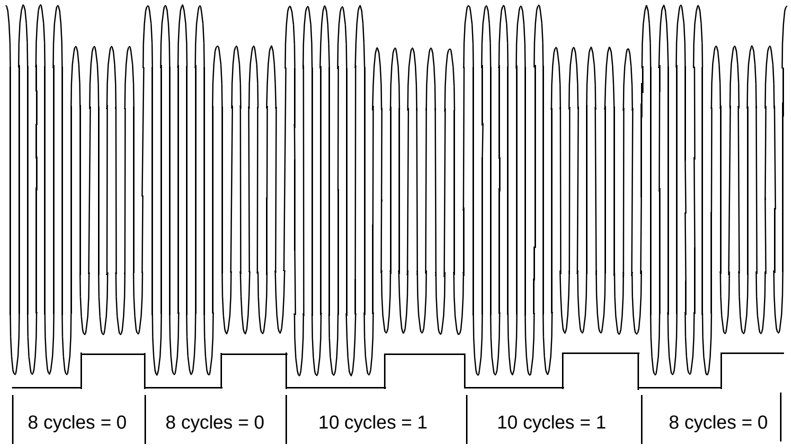
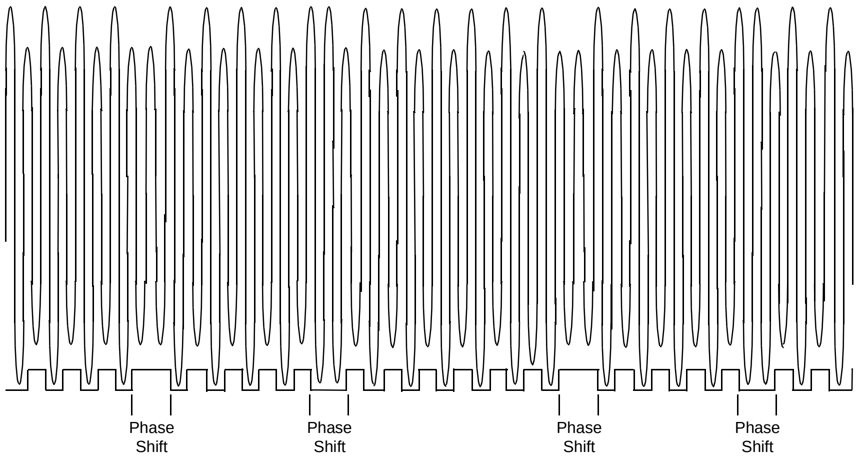


FIGURE 4: PSK MODULATED SIGNAL



ANTICOLLISION

In many existing applications, a single-read RFID tag is sufficient and even necessary: animal tagging and access control are examples. However, in a growing number of new applications, the simultaneous reading of several tags in the same RF field is absolutely critical: library books, airline baggage, garment, and retail applications are a few.

In order to read multiple tags simultaneously, the tag and reader must be designed to detect the condition that more than one tag is active. Otherwise, the tags will all backscatter the carrier at the same time, and the amplitude-modulated waveforms shown in Figures 3 and 4 would be garbled. This is referred to as a *collision*. No data would be transferred to the reader. The tag/reader interface is similar to a serial bus, even though the “bus” travels through the air. In a wired serial bus application, arbitration is necessary to prevent bus contention. The RFID interface also requires arbitration so that only one tag transmits data over the “bus” at one time.

A number of different methods are in use and in development today for preventing collisions; most are patented or patent pending, but all are related to making sure that only one tag “talks” (backscatters) at any one time. See the *MCRF250 Data Sheet* (page 15) and the *FSK Anticollision Reader Reference Design* (page 99) chapters for more information.

NOTES:

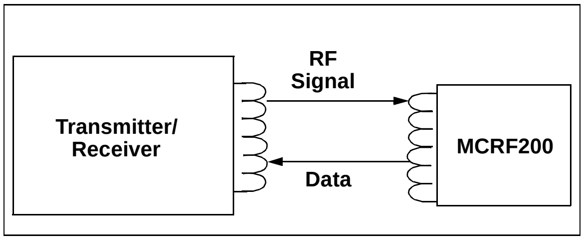
MCRF200

Contactless Programmable Passive RFID Device

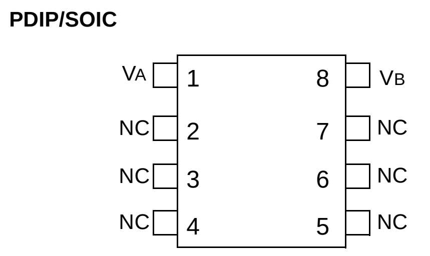
FEATURES

- Contactless programmable after encapsulation
- Read only data transmission
- 96 or 128 bits of One-Time Programmable (OTP) user memory (also supports 48 and 64-bit protocols)
- Typical operates at 125 kHz
- On chip rectifier and voltage regulator
- Ultra low power operation
- Factory programming and device serialization available
- Encoding options:
  - NRZ Direct, Differential Biphase, Manchester Biphase
- Modulation options:
  - Direct, FSK, PSK (change on data change), PSK (change at the beginning of a one)

APPLICATION



PACKAGE TYPE



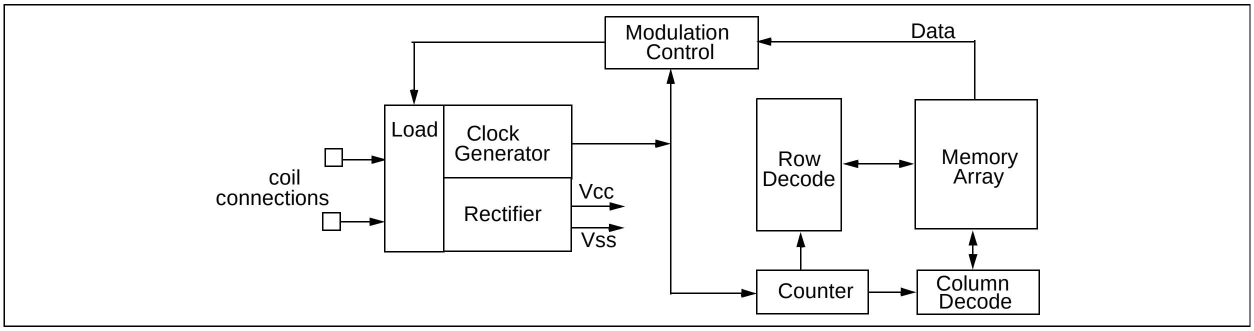
DESCRIPTION

This device is a Radio Frequency Identification (RFID) tag that provides a bidirectional interface for programming and reading the contents of the user array. The device is powered by an external RF transmitter through inductive coupling. When in read mode, the device transmits the contents of its memory array by damping (modulating) the incoming RF signal. The reader is able to detect the damping and decodes the data being transmitted. Code length, modulation option, encoding option, and bit rate are set at the factory to fit the needs of particular applications.

The user memory array of this device can be programmed contactlessly after encapsulation. This allows the user to keep encapsulated blank tags in stock for on-demand personalization. The tags can then be programmed with data as they are needed.

These devices are available in die, wafer, PDIP, SOIC, and COB module form. The encoding, modulation, frequency, and bit rate options are specified by the customer and programmed by Microchip Technology Inc. prior to shipment. Array programming and serialization (SQTP) can also be arranged upon request. See TB023 (page 23) for more information.

BLOCK DIAGRAM



1.0 ELECTRICAL CHARACTERISTICS

1.1 Maximum Ratings*

Storage temperature ....- 65°C to +150°C
Ambient temp. with power applied ....-40°C to +125°C
Maximum current into coil pads ....50 mA

***Notice:** Stresses above those listed under "Maximum ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at those or any other conditions above those indicated in the operational listings of this specification is not implied. Exposure to maximum rating conditions for extended periods may affect device reliability.

TABLE 1-1: PAD FUNCTION TABLE

Name	Function
V_A,V_B	Coil connection
NC	No connection, test pad

TABLE 1-2: AC AND DC CHARACTERISTICS

All parameters apply across the specified operating ranges unless otherwise noted.	Industrial (I): Tamb = -40°C to +85°C					
Parameter	**Symbol**	**Min.**	**Typ.**	**Max.**	**Units**	**Conditions**
Clock frequency	F_{CLK}	100	—	150	kHz	
Contactless programming time	T_{WC}	—	2	—	s	128-bit array
Data retention		200	—	—	Years	25°C
Coil current (Dynamic)	I_{CD}	—	50		µA	
Operating current	I_{DD}	—	5		µA	V_{CC} = 2V
Turn-on-voltage (Dynamic) for modulation	V_AV_B V_{CC}	10 2	— —	— —	V_{PP} V_{DC}	
Input Capacitance	C_{IN}	—	2	—	pF	Between V_A and V_B

FIGURE 1-1: DIE PLOT

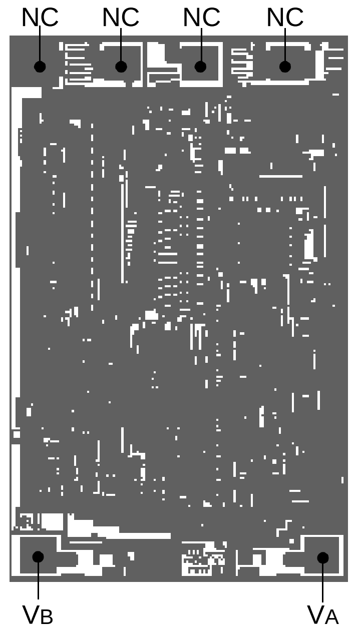


TABLE 1-3 RFID PAD COORDINATES (MICRONS)

Pad Name	Passivation Openings		Pad Center X	Pad Center Y
	Pad Width	Pad Height		
V_A	90.0	90.0	427.50	-734.17
V_B	90.0	90.0	-408.60	-734.17

Note 1: All coordinates are referenced from the center of the die.
2: Die size 1.1215 mm x 1.7384 mm.

2.0 FUNCTIONAL DESCRIPTION

The RF section generates all the analog functions needed by the transponder. These include rectification of the carrier, on-chip regulation of V_{PP} (programming voltage), and V_{DD} (operating voltage), as well as high voltage clamping to prevent excessive voltage from being applied to the transponder. This section generates a system clock from the interrogator carrier of the same frequency, detects carrier interrupts, and modulates the tuned LC antenna for transmission to the interrogator. The chip detects a power-up condition and resets the transponder when sufficient voltage develops.

2.1 Rectifier – AC Clamp

The AC voltage generated by the transponder tuned LC circuit is full wave rectified. This unregulated voltage is used as the maximum DC supply voltage for the rest of the chip. The peak voltage on the tuned circuit is clamped by the internal circuitry to a safe level to prevent damage to the IC. This voltage is adjusted during programming to allow sufficient programming voltage to the EEPROM.

2.2 Coil Load Modulation

The MCRF200 communicates to the reader by AM-modulating the coil voltage across the tuned LC circuit.

2.3 V_{DD} Regulator

The device generates a fixed supply voltage from the unregulated coil voltage.

2.4 V_{PP} Regulator

This regulates a programming voltage during the programming mode. The voltage is used for the EEPROM array to perform block erasure of the memory as well as single-bit programming during both contact and contactless programming. During reading, this voltage is level-shifted down and kept below the programming voltages to insure that the part is not inadvertently programmed.

2.5 Clock Generator

This circuit generates a clock based on the interrogator frequency. This clock is used to derive all timing in the tag, including the baud rate, modulation rate, and programming rate.

2.6 IRQ Detector

This circuitry detects an interrupt in the continuous electromagnetic field of the interrogator. An IRQ (interrupt request) is defined as the absence of the electromagnetic field for a specific number of clock cycles. This feature is used during contactless programming.

2.7 Power-On Reset

This circuit generates a power-on reset when the tag first enters the interrogator field. The reset releases when sufficient power has developed on the V_{DD} regulator to allow correct operation. The reset trip points are set such that sufficient voltage across V_{DD} has developed, which allows for correct clocking of the logic for reading of the EEPROM and configuration data, and correct modulation.

2.8 Modulation Logic

This logic acts upon the serial data being read from the EEPROM and performs two operations on the data. The logic first encodes the data according to the configuration bits CB6 and CB7. The data can be sent out direct to the modulation logic or encoded Biphase Differential, Biphase Manchester or Manchester with IDI option.

The encoded data is then either passed NRZ Direct out to modulate the coil, or FSK modulated, or PSK modulated with changes on the change of data, or PSK with changes on the bit edge of a one. Configuration bits CB8 and CB9 determine the modulation option. CB10 is used if the PSK option has been selected, and determines if the return carrier rate is $F_{CLK}/2$ or $F_{CLK}/4$.

3.0 CONFIGURATION LOGIC CONTROL BIT REGISTER

The configuration register determines the operational parameters of the device. The configuration register cannot be programmed contactlessly; it is programmed during wafer probe at the Microchip factory. CB11 is always a zero; CB12 is set when successful contact or contactless programming of the data array has been completed. Once CB12 is set, device programming and erasing is disabled. Figure 3-1 contains a description of the control register bit functions.

3.1 Organization

The configuration bit register directly controls logic blocks which generate the baud rate, memory size, encoded data, and modulated data. This register also contains bits which lock the data array.

3.2 Baud Rate Timing

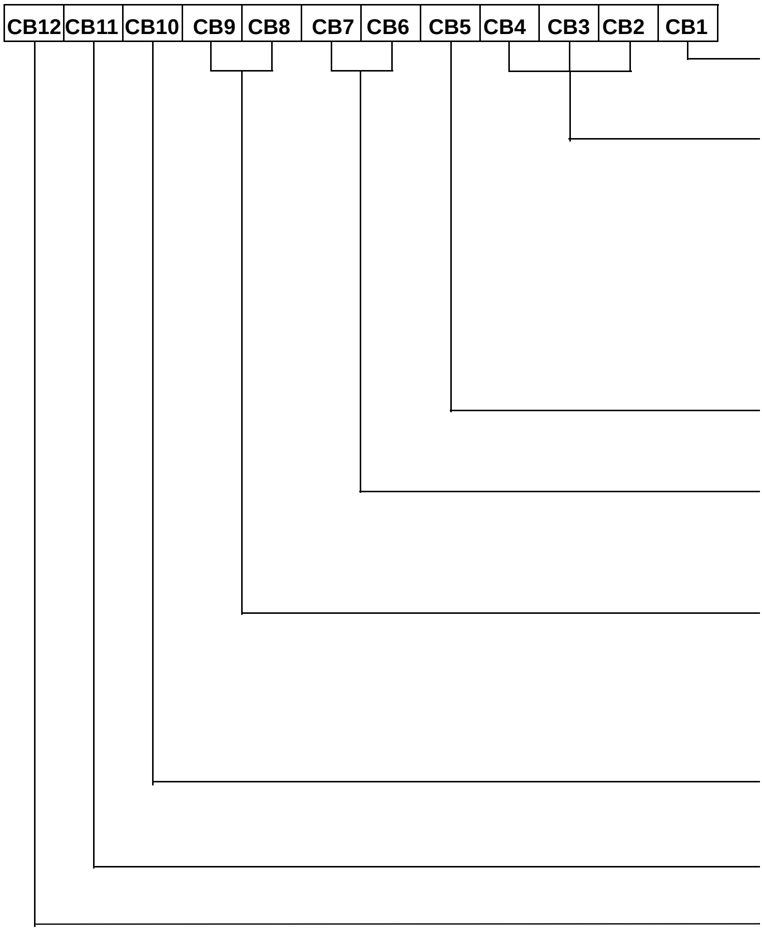
The chip will access data at a baud rate determined by bits CB2, CB3, CB4, and CB5 of the configuration register. CB2, CB3, and CB4 determine the return data rate (CACLK). The default rate of FCLK/128 is used for contact and contactless programming. Once the array is successfully programmed, the lock bit CB12 is set. When the lock bit is cleared, programming and erasing the device becomes permanently disabled. The configuration register has no effect on device timing until after the EEPROM data array is programmed. If CB2 is set to a one and CB5 is set to a one, the 1.5 bit SYNC word option is enabled.

3.3 Column and Row Decoder Logic and Bit Counter

The column and row decoders address the EEPROM array at the clock rate and generate a serial data stream for modulation. This data stream can be up to 128 bits in length. The size of the stream is user programmable with CB1 and can be set to 96 or 128 bits. Data lengths of 48 and 64 bits are available by programming the data twice in the array, end-to-end. The data is then encoded by the modulation logic. The data length during contactless programming is 128 bits.

The column and row decoders route the proper voltage to the array for programming and reading. In the programming modes, each individual bit is addressed serially from bit 1 to bit 128.

FIGURE 3-1: CONFIGURATION REGISTER



ARRAY SIZE

CB1 = 1 128-bit user array
CB1 = 0 96-bit user array

TIMING

CB2	CB3	CB4	Rate
0	0	0	MOD128
0	0	1	MOD100
0	1	0	MOD80
0	1	1	MOD32
1	0	0	MOD64
1	0	1	MOD50
1	1	0	MOD40
1	1	1	MOD16

SYNC WORD

CB5 = 1 1.5-bit sync word enable
CB5 = 0 1.5-bit sync word disable

DATA ENCODING

CB6 = 0; CB7 = 0 NRZ_L (Direct)
CB6 = 0; CB7 = 1 Biphase_S (Differential)
CB6 = 1; CB7 = 0 Biphase_L (Manchester)
CB6 = 1; CB7 = 1 (Not Used)

MODULATION OPTIONS

CB8 = 0; CB9 = 0 FSK 0 = Fc/8, 1 = Fc/10
CB8 = 0; CB9 = 1 Direct
CB8 = 1; CB9 = 0 psk_1
(phase change on change of data)
CB8 = 1; CB9 = 1 psk_2
(phase change at beginning of a one)

PSK RATE OPTION

CB10 = 1 clk/4 carrier
CB10 = 0 clk/2 carrier

(READ ONLY)

CB11 = 0

ARRAY LOCK BIT (READ ONLY)

CB12 = 0 array not locked
CB12 = 1 array is locked

4.0 MODES OF OPERATION

The device has two basic modes of operation: Native Mode and Read Mode.

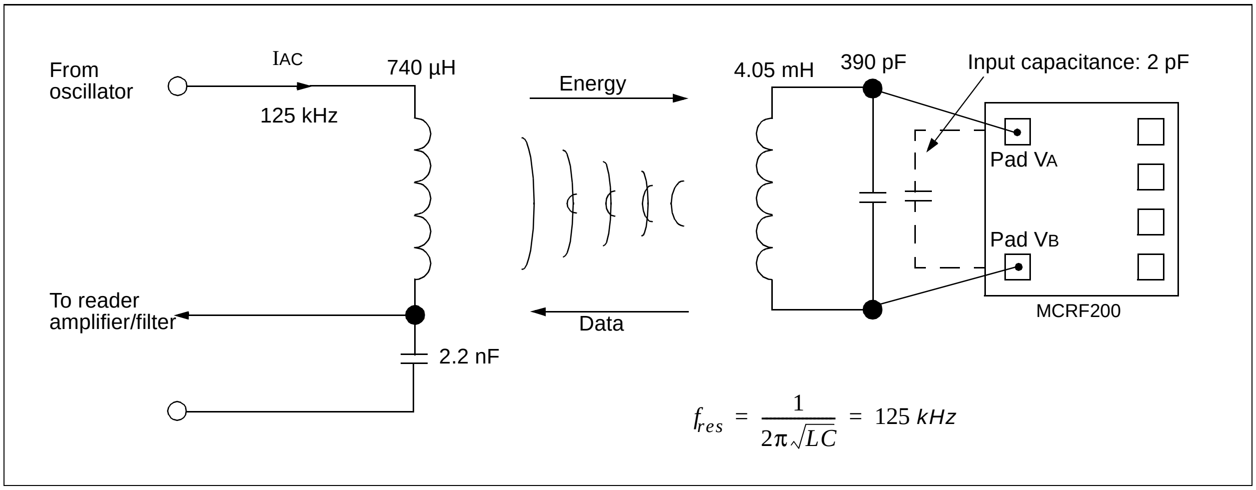
4.1 Native Mode

In native mode, a transponder will have an unprogrammed array and will be in the default mode for contactless programming (default baud rate FCLK/128, FSK, NRZ_direct).

4.2 Read Mode

The second mode is a read mode after the contactless or contact programming has been completed and for the rest of the lifetime of the device. The lock bit CB12 will be set, and when the transponder is powered, it will have the ability to transmit according to the protocol in the configuration register.

FIGURE 4-1: TYPICAL APPLICATION CIRCUIT



$$f_{res} = \frac{1}{2\pi\sqrt{LC}} = 125\ kHz$$

MCRF200 PRODUCT IDENTIFICATION SYSTEM

To order or obtain information, e.g., on pricing or delivery, refer to the factory or the listed sales office.

MCRF 200 – I /WFxxx

Configuration:	Hex Code	=	Three digit hex value to be programmed into the configuration register. Three hex characters correspond to 12 binary bits. These bits are programmed into the configuration register MSB first (CB12, CB11…CB1). See the example below.
Package:	WF	=	Sawed wafer on frame (7 mil backgrind)
	W	=	Wafer (11 mil backgrind)
	S	=	Dice in waffle pack
	SN	=	150 mil SOIC
	P	=	PDIP
	1C	=	0.45 mm COB module with 1000 pF capacitor
	3C	=	0.70 mmCOB module with 330 pF capacitor
Temperature Range:	I	=	-40°C to +85°C
Sample Part Number:			MCRF200-I /W00A

MCRF200-I/W00A = 125 kHz, industrial temperature, wafer package, contactlessly programmable, 96 bit, FSK Fc/8 Fc/10, direct encoded, Fc/50 data return rate tag. The configuration register is:

CB12	CB11	CB10	CB9	CB8	CB7	CB6	CB5	CB4	CB3	CB2	CB1
0	0	0	0	0	0	0	0	1	0	1	0

NOTES:

MCRF250

Contactless Programmable Passive RFID Device With Anti-Collision

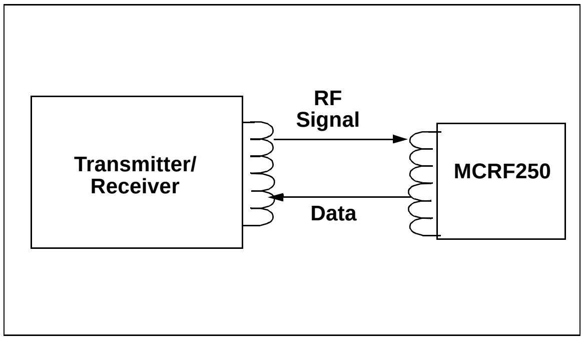
FEATURES

- Anti-collision feature to resolve multiple tags in the same RF field
- Read-only data transmission
- 96 or 128 bits of One-Time Programmable (OTP) user memory (also supports 48 and 64-bit protocols)
- Operates up to 150 kHz
- On-chip rectifier and voltage regulator
- Low power operation
- Factory programming and device serialization available
- Encoding options:
  - NRZ Direct, Differential Biphase, Manchester Biphase, Biphase IDI
- Modulation options:
  - FSK, Direct, PSK (change on data change), PSK (change at the beginning of a one)
- Contactless programmable after encapsulation

DESCRIPTION

This device is a Radio Frequency Identification (RFID) tag that provides a variety of operating modes. The device is powered by an external RF transmitter (reader) through inductive coupling. When in the reader field, the device will transmit the contents of its memory array by damping (modulating) the incoming RF signal. A reader is able to detect the damping and decodes the data being transmitted. Code length, modulation option, encoding option and bit rate are set at the factory to fit the needs of particular applications.

APPLICATION

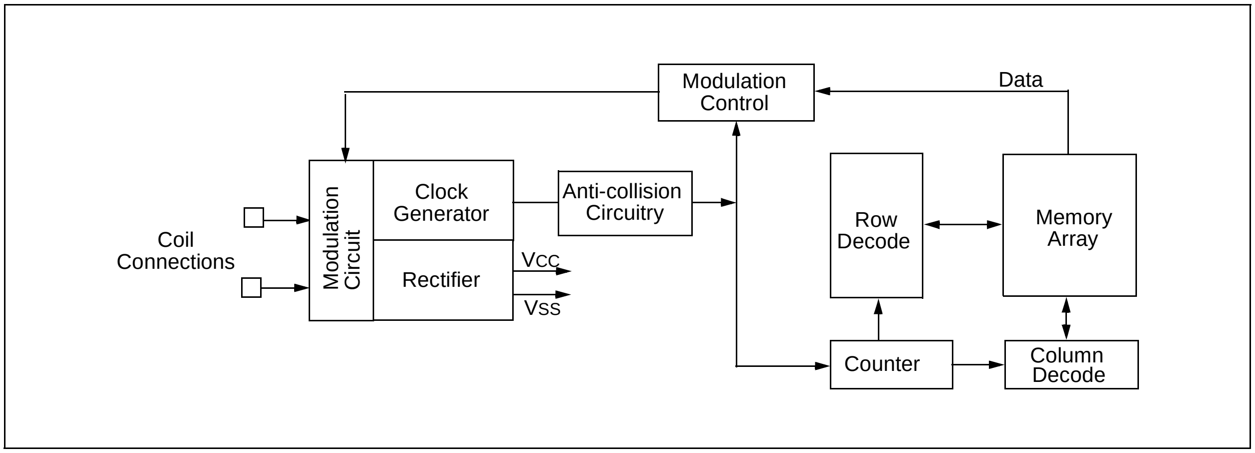


The MCRF250 is equipped with an anti-collision feature that allows multiple tags in the same field to be read simultaneously. This revolutionary feature eliminates the issue of data corruption due to simultaneous transmissions from multiple tags.

The user memory array of this device can be programmed contactlessly after encapsulation. This allows the user to keep encapsulated blank tags in stock for on-demand personalization. The tags can then be programmed with data as they are needed.

These devices are available in die form or packaged in SOIC, PDIP or COB modules. The encoding, modulation, frequency, and bit rate options are specified by the customer and programmed by Microchip Technology Inc. prior to shipment. Array programming and serialization (SQTP) can also be arranged upon request. See TB023 (page 23) for more information.

BLOCK DIAGRAM



1.0 ELECTRICAL CHARACTERISTICS

1.1 Maximum Ratings*

Storage temperature ....-65°C to +150°C
Ambient temp. with power applied ....-40°C to +125°C
Maximum current into coil pads ....50 mA

***Notice:** Stresses above those listed under “Maximum ratings” may cause permanent damage to the device. This is a stress rating only and functional operation of the device at those or any other conditions above those indicated in the operational listings of this specification is not implied. Exposure to maximum rating conditions for extended periods may affect device reliability.

TABLE 1-1: PAD FUNCTION TABLE

Name	Function
V_A,V_B	Coil connection
NC	No connection, test pad

TABLE 1-2: AC AND DC CHARACTERISTICS

All parameters apply across the specified operating ranges unless otherwise noted.	Industrial (I): Tamb = -40°C to +85°C					
Parameter	**Symbol**	**Min.**	**Typ.**	**Max.**	**Units**	**Conditions**
Clock frequency	F_{CLK}	100	—	150	kHz	
Contactless programming time	T_{WC}	—	2	—	s	128-bit array
Data retention		200	—	—	Years	25°C
Coil current (Dynamic)	I_{CD}	—	50		µA	
Operating current	I_{DD}	—	2		µA	
Turn-on-voltage (Dynamic) for modulation	V_AV_B	10	—	—	V_{PP}	
	V_{CC}	2	—	—	V_{DC}	

FIGURE 1-1: DIE PLOT

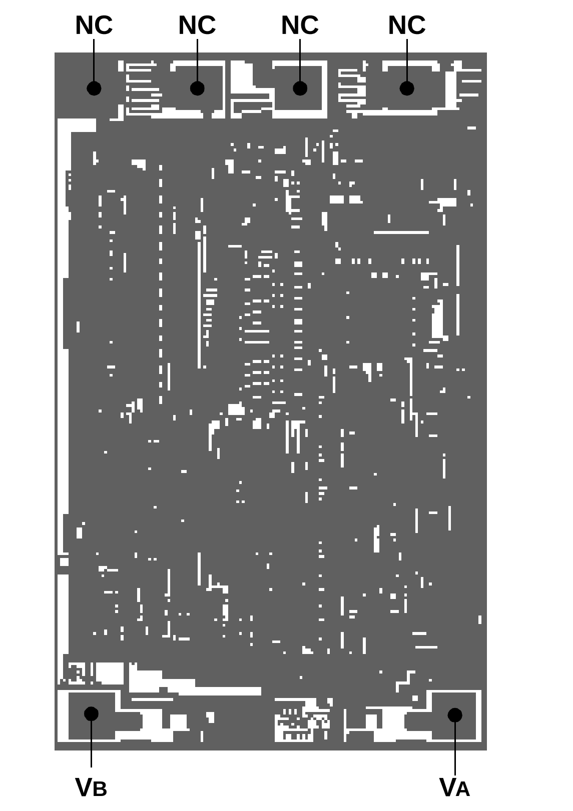


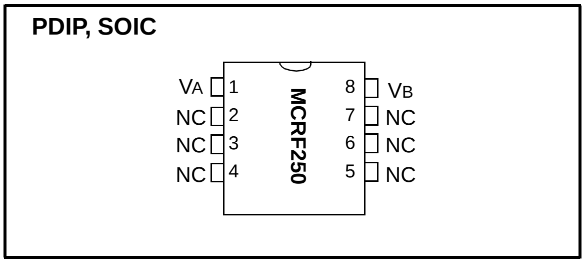
TABLE 1-3: RFID PAD COORDINATES (MICRONS)

	Passivation Openings			
Pad Name	Pad Width	Pad Height	Pad Center X	Pad Center Y
V_A	90.0	90.0	427.50	-734.17
V_B	90.0	90.0	-408.60	-734.17

Note 1: All coordinates are referenced from the center of the die.

2: Die size 1.1215 mm x 1.7384 mm.

FIGURE 1-2: PIN DIAGRAM



2.0 FUNCTIONAL DESCRIPTION

The RF section generates all the analog functions needed by the transponder. These include rectification of the carrier, on-chip regulation of VPP (programming voltage), and VDD (operating voltage), as well as high voltage clamping to prevent excessive voltage from being applied to the device. This section generates a system clock from the interrogator carrier frequency, detects carrier interrupts and modulates the tuned LC antenna for transmission to the interrogator. The chip detects a power up condition and resets the transponder when sufficient voltage develops.

2.1 Rectifier – AC Clamp

The AC voltage induced by the tuned LC circuit is full wave rectified. This unregulated voltage is used as the DC supply voltage for the rest of the chip. The peak voltage on the tuned circuit is clamped by the internal circuitry to a safe level to prevent damage to the IC. This voltage is adjusted during programming to allow sufficient programming voltage to the EEPROM.

2.2 Coil Load Modulation

The MCRF250 communicates by shunting a transistor across the tuned LC circuit, which modulates the received RF field.

2.3 VDD Regulator

The device generates a fixed supply voltage from the unregulated coil voltage.

2.4 VPP Regulator

This regulates a programming voltage during the programming mode. The voltage is switched into the EEPROM array to perform block erasure of the memory as well as single bit programming during both contact and contactless programming. During reading this voltage is level shifted down and kept below the programming voltages to insure that the part is not inadvertently programmed.

2.5 Clock Generator

This circuit generates a clock with a frequency equal to the interrogator frequency. This clock is used to derive all timing in the device, including the baud rate, modulation rate, and programming rate.

2.6 IRQ Detector

This circuitry detects an interrupt in the continuous electromagnetic field of the interrogator. An IRQ (interrupt request) is defined as the absence of the electromagnetic field for a specific number of clock cycles. Detection of an IRQ will trigger the device to enter the Anti-collision mode. This mode is discussed in detail in Section 5.0.

2.7 Power On Reset

This circuit generates a power on reset when the tag first enters the interrogator field. The reset releases when sufficient power has developed on the VDD regulator to allow correct operation. The reset trip points are set such that sufficient voltage across VDD has developed which allows for correct clocking of the logic for reading of the EEPROM and configuration data, and correct modulation.

2.8 Modulation Logic

This logic acts upon the serial data being read from the EEPROM and performs two operations on the data. The logic first encodes the data according to the configuration bits CB6 and CB7. The data can be sent out direct to the modulation logic or encoded biphase_s (differential), biphase_l (manchester) or idi (manchester).

The encoded data is then either passed NRZ Direct out to modulate the coil, or FSK modulated, or PSK modulated with phase changes on the change of data, or PSK with phase changes on the bit edge of a one. Configuration bits CB8 and CB9 determine the modulation option. CB10 is used if the PSK option has been selected and determines whether the return carrier rate is FCLK/2 or FCLK/4.

3.0 CONFIGURATIONLOGIC

3.1 Control Bit Register

The configuration register determines the operational parameters of the device. The configuration register can not be programmed contactlessly; it is programmed during wafer probe at the Microchip factory. CB11 is always a one; CB12 is set when successful contact or contactless programming of the data array has been completed. Once CB12 is set, programming and erasing of the device is disabled. Figure 3-1 contains a description of the control register bit functions.

3.2 Organization

The configuration bit register directly controls logic blocks, which generate the baud rate, memory size, encoded data, and modulated data. This register also contains bits which lock the data array.

3.3 Baud Rate Timing

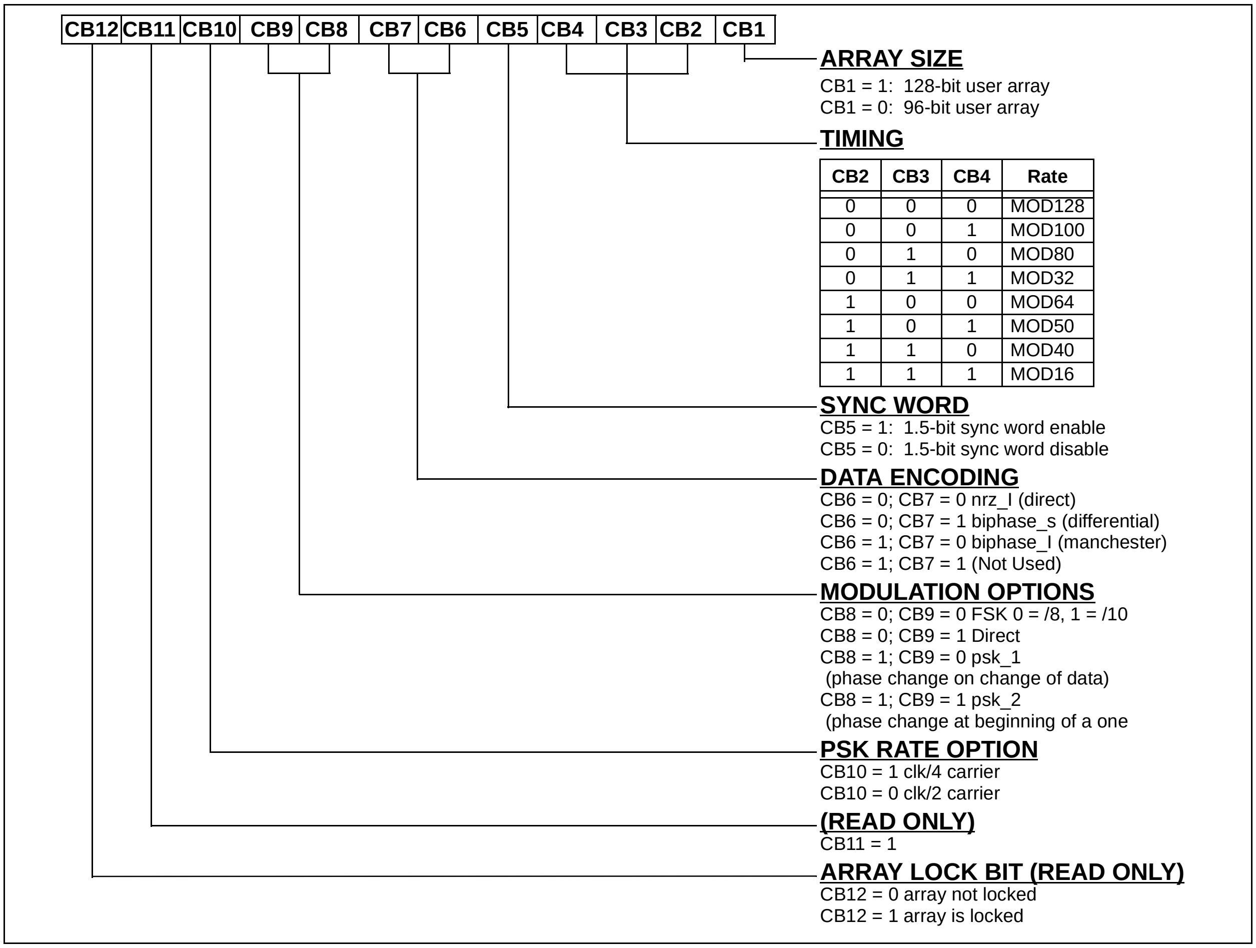
The chip will access data at a baud rate determined by bits CB2, CB3, CB4, and CB5 of the configuration register. CB2, CB3, and CB4 determine the return data rate (CACLK). The default rate of FCLK/128 is used for contact and contactless programming. Once the array is successfully programmed, the lock bit CB12 is set. When the lock bit is set, programming and erasing the device becomes permanently disabled. The configuration register has no effect on device timing or modulation until after the EEPROM data array is programmed. If CB2 is set to a one and CB5 is set to a one, the 1.5 bit SYNC word option is enabled.

3.4 Column and Row Decoder Logic and Bit Counter

The column and row decoders address the EEPROM array at the CACLK rate and generate a serial data stream for modulation. This data stream can be up to 128 bits in length. The size of the stream is user programmable with CB1, and can be set to 96 or 128 bits. Data lengths of 48 and 64 bits are available by programming the data twice in the array end to end. The data is then encoded by the modulation logic. The data length during contactless programming is 128 bits.

The column and row decoders route the proper voltage to the array for programming and reading. In the programming modes, each individual bit is addressed serially from bit 1 to bit 128.

FIGURE 3-1: CONFIGURATION REGISTER



ARRAY SIZE

CB1 = 1: 128-bit user array
CB1 = 0: 96-bit user array

TIMING

CB2	CB3	CB4	Rate
0	0	0	MOD128
0	0	1	MOD100
0	1	0	MOD80
0	1	1	MOD32
1	0	0	MOD64
1	0	1	MOD50
1	1	0	MOD40
1	1	1	MOD16

SYNC WORD

CB5 = 1: 1.5-bit sync word enable
CB5 = 0: 1.5-bit sync word disable

DATA ENCODING

CB6 = 0; CB7 = 0 nrz_I (direct)
CB6 = 0; CB7 = 1 biphase_s (differential)
CB6 = 1; CB7 = 0 biphase_I (manchester)
CB6 = 1; CB7 = 1 (Not Used)

MODULATION OPTIONS

CB8 = 0; CB9 = 0 FSK 0 = /8, 1 = /10
CB8 = 0; CB9 = 1 Direct
CB8 = 1; CB9 = 0 psk_1
(phase change on change of data)
CB8 = 1; CB9 = 1 psk_2
(phase change at beginning of a one

PSK RATE OPTION

CB10 = 1 clk/4 carrier
CB10 = 0 clk/2 carrier

(READ ONLY)

CB11 = 1

ARRAY LOCK BIT (READ ONLY)

CB12 = 0 array not locked
CB12 = 1 array is locked

4.0 MODES OF OPERATION

The device has two basic modes of operation: Native Mode and Read Mode.

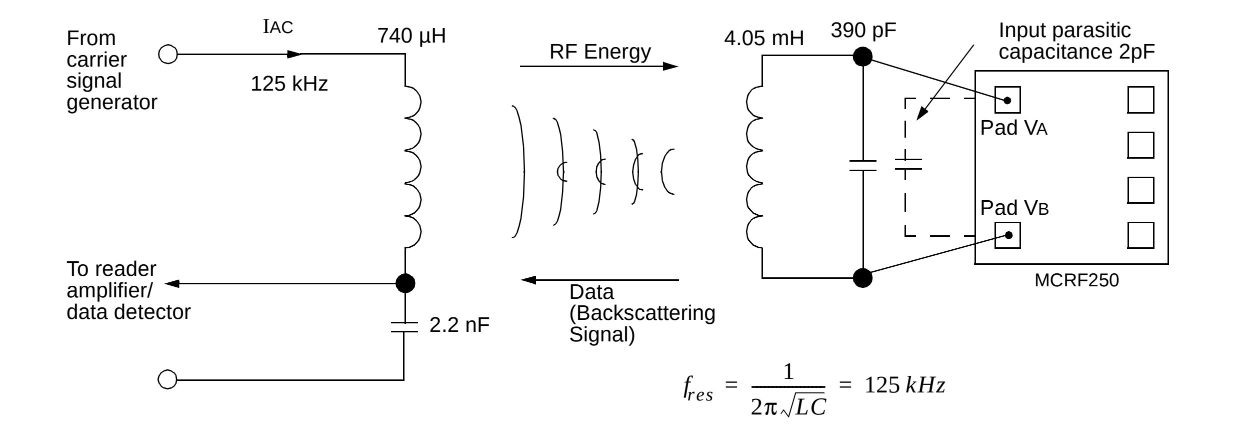
4.1 Native Mode

In native mode, the MCRF250 will have an unprogrammed array and will be in the default mode for contactless programming (default baud rate FCLK/128, FSK, NRZ_direct).

4.2 Read Mode

The second mode is a read mode after the contactless or contact programming has been completed and for the rest of the lifetime of the device. The lock bit CB12 will be set, and the transponder will have the ability to transmit when powered and enter the anti-collision algorithm.

FIGURE 4-1: TYPICAL APPLICATION CIRCUIT


From carrier signal generator
IAC
125 kHz
740 µH
To reader amplifier/ data detector
2.2 nF
RF Energy
Data (Backscattering Signal)
4.05 mH
390 pF
Input parasitic capacitance 2pF
Pad VA
Pad VB
MCRF250

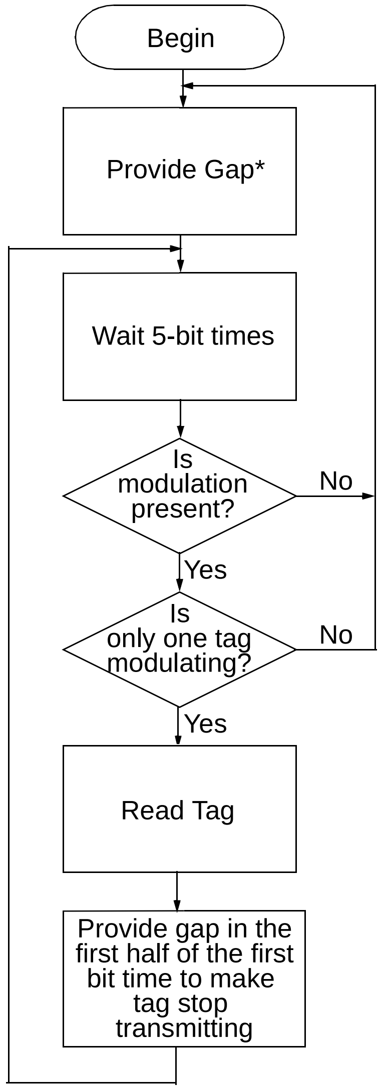
$$f_{res} = \frac{1}{2\pi\sqrt{LC}} = 125\,kHz$$

5.0 ANTICOLLISION

The anti-collision feature is enabled when the array is locked. In this mode, the MCRF250 has the ability to stop transmitting when a collision has occurred. The device will begin transmitting again when its internal anti-collision algorithm indicates that it is time to do so.

Multiple tags can enter the same reader field and be read by the reader in a short period of time. The reader must provide “gaps” (RF field off) at proper timing intervals as shown in Figure 5-1 in order to inform the MCRF250 of collisions, and to sequence from one tag to another.

FIGURE 5-1: ANTI-COLLISION FLOWCHART


Begin
Provide Gap*
Wait 5-bit times
Is modulation present?
No
Yes
Is only one tag modulating?
No
Yes
Read Tag
Provide gap in the first half of the first bit time to make tag stop transmitting

Note: *Gap = lack of RF carrier signal = 60 µs ± 20%.

MCRF250 PRODUCT IDENTIFICATION SYSTEM

To order or obtain information, e.g., on pricing or delivery, refer to the factory or the listed sales office.

MCRF250 - I /WFxxx

Configuration:

Hex Code	=	Three digit hex value to be programmed into the configuration register. Three hex characters correspond to 12 binary bits. These bits are programmed into the configuration register MSB first (CB12, CB11…CB1). See the example below.

Package:

WF	=	Sawed wafer on frame (7 mil backgrind)
W	=	Wafer
S	=	Dice in wafer pack
SN	=	150 mil SOIC
P	=	PDIP
1C	=	COB module with 1000 pF capacitor
3C	=	COB module with 330 pF capacitor

Temperature Range:

I	=	–40°C to +85°C

Device:

250	=	125 kHz

Sample Part Number: MCRF250-I /40A

MCRF250-I/W40A = 125 kHz, Industrial temperature, wafer package, anti-collision, 96 bit, FSK/8 /10,direct encoded, F/50 data return rate tag. The configuration register is:

CB12	CB11	CB10	CB9	CB8	CB7	CB6	CB5	CB4	CB3	CB2	CB1
0	1	0	0	0	0	0	0	1	0	1	0

TB023

Contact Programming Support

INTRODUCTION

The MCRF200 and MCRF250 are 125 kHz RF tags, which can be contact or contactlessly programmed. The contact programming of the device is performed by Microchip Technology, Inc. upon customer request. The customer can choose any ID code suitable to their application subject to a minimum order quantity. These devices can also be contactlessly programmed after encapsulation using the Microchip microID contactless programmer (PG103001).

DEFINITIONS

First, the customer has to define the following operation options of the MCRF200 and MCRF250 (refer to individual data sheets page 7 and page 15, respectively):

• Bit rate	Defined as clocks per bit e.g., Fc/16, Fc/32, Fc/40, Fc/50, Fc/64, Fc/80, Fc/100, and Fc/128
• Modulation	FSK, PSK1, PSK2, ASK Direct
• Encoding	NRZ_L (Direct), Biphase_L (Manchester), Differential Biphase_S
• Code length	32, 48, 64, 96, and 128 bits

Second, the ID codes and series numbers must be supplied by the customer or an algorithm can be specified by the customer. This section describes only the case in which actual serial numbers are supplied. The customer must supply the ID codes and series numbers on floppy disk or via email. The codes should conform to the SQTP format below:

FILE SPECIFICATION

SQTP codes supplied to Microchip must comply with the following format:

The ID code file is a plain ASCII text file from floppy disk or email (no headers).

The code files should be compressed. Please make self-extracting files.

The code files are used in alphabetical order of their file names (including letters and numbers).

Used (i.e., programmed) code files are discarded by Microchip after use.

Each line of the code file must contain one ID code for one IC.

The code is in hexadecimal format.

The code line is exactly as long as the selected code length (e.g., code length = 64, ID code = 16 hex characters = 64-bit number).

Each line must end with a carriage return.

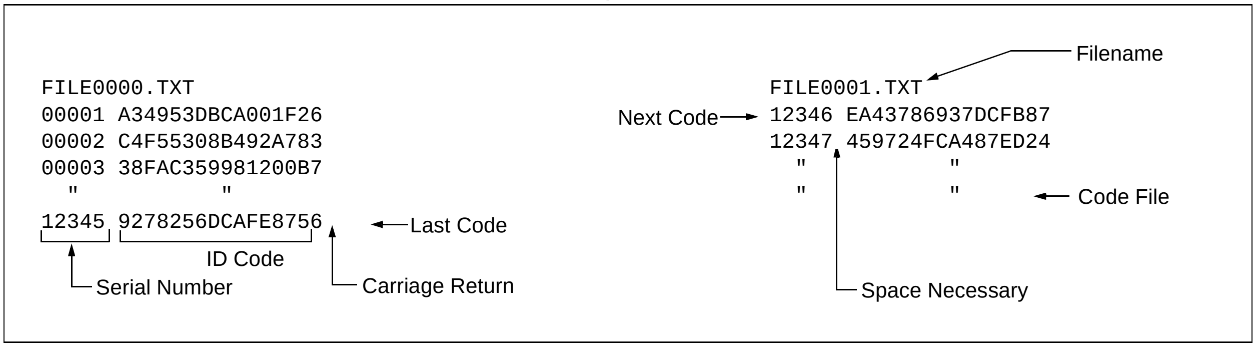
Each hexadecimal ID code must be preceded by a decimal series number.

Series number and ID code must be separated by a space.

The series number must be unique and ascending to avoid double programming.

The series numbers of two consecutive files must also count up for proper linking.

FIGURE 1: EXAMPLE OF TWO CODE FILES, CODE LENGTH = 64 BITS



```
FILE0000.TXT
00001 A34953DBCA001F26
00002 C4F55308B492A783
00003 38FAC359981200B7
  "         "
12345 9278256DCAFE8756
```

```
FILE0001.TXT
12346 EA43786937DCFB87
12347 459724FCA487ED24
  "          "
  "          "
```

NOTES:

MICROCHIP

AN678

RFID Coil Design

Author: Youbok Lee
Microchip Technology Inc.

INTRODUCTION

In a Radio Frequency Identification (RFID) application, an antenna coil is needed for two main reasons:

- To transmit the RF carrier signal to power up the tag
- To receive data signals from the tag

An RF signal can be radiated effectively if the linear dimension of the antenna is comparable with the wavelength of the operating frequency. In an RFID application utilizing the VLF (100 kHz – 500 kHz) band, the wavelength of the operating frequency is a few kilometers (λ = 2.4 Km for 125 kHz signal). Because of its long wavelength, a true antenna can never be formed in a limited space of the device. Alternatively, a small loop antenna coil that is resonating at the frequency of the interest (i.e., 125 kHz) is used. This type of antenna utilizes near field magnetic induction coupling between transmitting and receiving antenna coils.

The field produced by the small dipole loop antenna is not a propagating wave, but rather an attenuating wave. The field strength falls off with r^{-3} (where r = distance from the antenna). This near field behavior (r^{-3}) is a main limiting factor of the read range in RFID applications.

When the time-varying magnetic field is passing through a coil (antenna), it induces a voltage across the coil terminal. This voltage is utilized to activate the passive tag device. The antenna coil must be designed to maximize this induced voltage.

This application note is written as a reference guide for antenna coil designers and application engineers in the RFID industry. It reviews basic electromagnetics theories to understand the antenna coils, a procedure for coil design, calculation and measurement of inductance, an antenna-tuning method, and the relationship between read range vs. size of antenna coil.

REVIEW OF A BASIC THEORY FOR ANTENNA COIL DESIGN

Current and Magnetic Fields

Ampere's law states that current flowing on a conductor produces a magnetic field around the conductor. Figure 1 shows the magnetic field produced by a current element. The magnetic field produced by the current on a round conductor (wire) with a finite length is given by:

EQUATION 1:

$$B_\phi = \frac{\mu_o I}{4\pi r}(\cos\alpha_2 - \cos\alpha_1) \qquad (\text{Weber}/m^2)$$

where:

I = current

r = distance from the center of wire

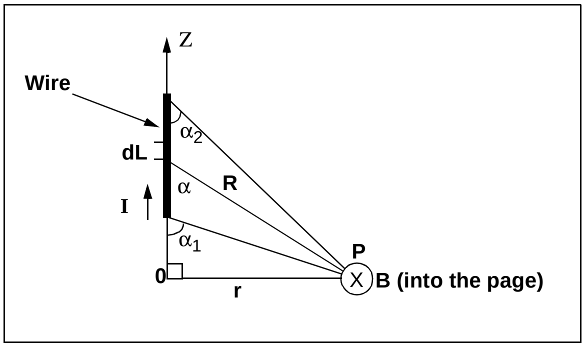
μ_o = permeability of free space and given as $\mu_o = 4\,\pi \times 10^{-7}$ (Henry/meter)

In a special case with an infinitely long wire where α_1 = -180° and α_2 = 0°, Equation 1 can be rewritten as:

EQUATION 2:

$$B_\phi = \frac{\mu_o I}{2\pi r} \qquad (\text{Weber}/m^2)$$

FIGURE 1: CALCULATION OF MAGNETIC FIELD B AT LOCATION P DUE TO CURRENT I ON A STRAIGHT CONDUCTING WIRE



The magnetic field produced by a circular loop antenna coil with N-turns as shown in Figure 2 is found by:

EQUATION 3:

$$B_z = \frac{\mu_o I N a^2}{2(a^2 + r^2)^{3/2}}$$

$$= \frac{\mu_o I N a^2}{2}\left(\frac{1}{r^3}\right) \qquad \text{for} \quad r^2 >> a^2$$

where:

a = radius of loop

Equation 3 indicates that the magnetic field produced by a loop antenna decays with $1/r^3$ as shown in Figure 3. This near-field decaying behavior of the magnetic field is the main limiting factor in the read range of the RFID device. The field strength is maximum in the plane of the loop and directly proportional to the current (I), the number of turns (N), and the surface area of the loop.

Equation 3 is frequently used to calculate the ampere-turn requirement for read range. A few examples that calculate the ampere-turns and the field intensity necessary to power the tag will be given in the following sections.

FIGURE 2: CALCULATION OF MAGNETIC FIELD B AT LOCATION P DUE TO CURRENT I ON THE LOOP

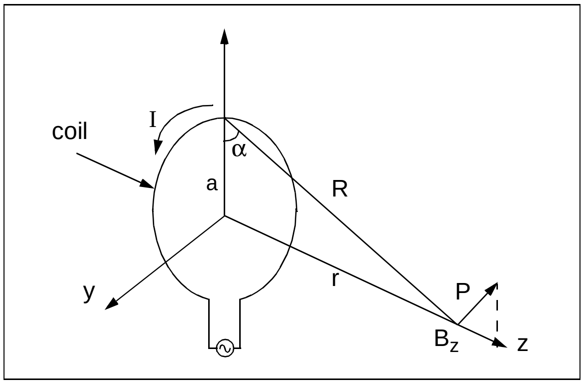
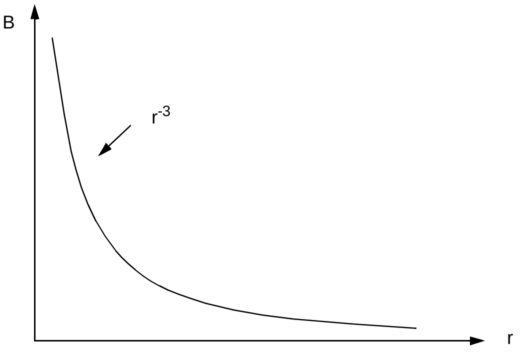


FIGURE 3: DECAYING OF THE MAGNETIC FIELD B VS. DISTANCE *r*



Note: The magnetic field produced by a loop antenna drops off with r^{-3}.

INDUCED VOLTAGE IN ANTENNA COIL

Faraday’s law states a time-varying magnetic field through a surface bounded by a closed path induces a voltage around the loop. This fundamental principle has important consequences for operation of passive RFID devices.

Figure 4 shows a simple geometry of an RFID application. When the tag and reader antennas are within a proximity distance, the time-varying magnetic field B that is produced by a reader antenna coil induces a voltage (called electromotive force or simply EMF) in the tag antenna coil. The induced voltage in the coil causes a flow of current in the coil. This is called Faraday’s law.

The induced voltage on the tag antenna coil is equal to the time rate of change of the magnetic flux Ψ.

EQUATION 4:

$$V = -N\frac{d\Psi}{dt}$$

where:

N = number of turns in the antenna coil

Ψ = magnetic flux through each turn

The negative sign shows that the induced voltage acts in such a way as to oppose the magnetic flux producing it. This is known as Lenz’s Law and it emphasizes the fact that the direction of current flow in the circuit is such that the induced magnetic field produced by the induced current will oppose the original magnetic field.

The magnetic flux Ψ in Equation 4 is the total magnetic field B that is passing through the entire surface of the antenna coil, and found by:

EQUATION 5:

$$\psi = \int B \cdot dS$$

where:

B = magnetic field given in Equation 3

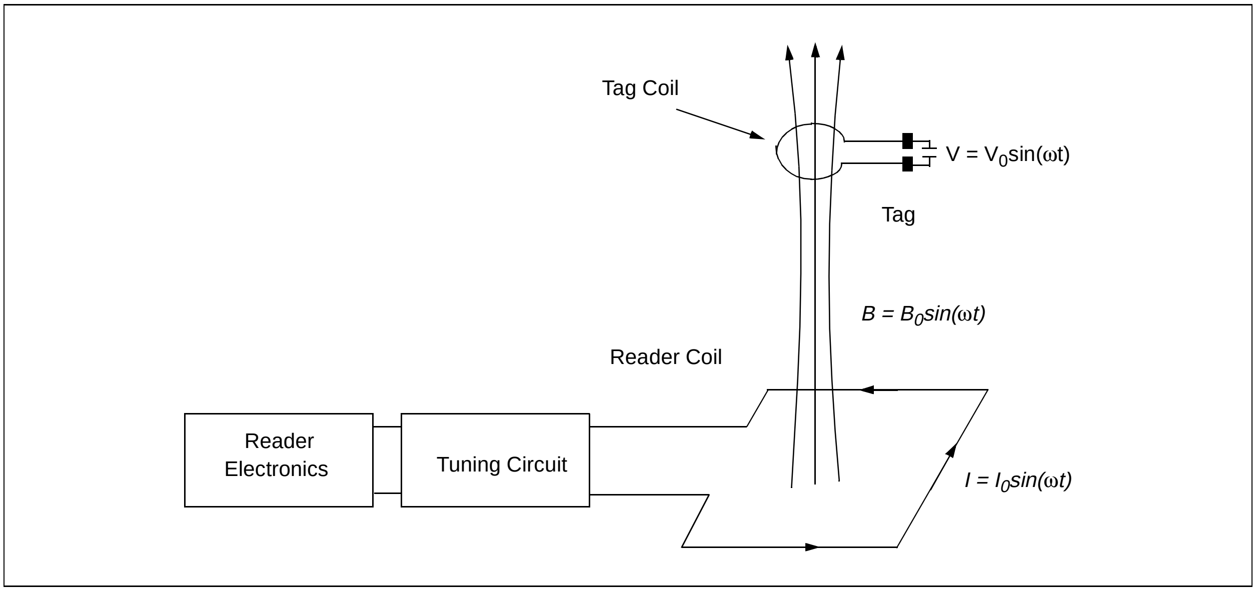
S = surface area of the coil

• = inner product (*cosine angle between two vectors*) of vectors B and surface area S

Note: Both magnetic field $\boldsymbol{B}$ and surface $\boldsymbol{S}$ are vector quantities.

The inner product presentation of two vectors in Equation 5 suggests that the total magnetic flux ψ that is passing through the antenna coil is affected by an orientation of the antenna coils. The inner product of two vectors becomes maximized when the two vectors are in the same direction. Therefore, the magnetic flux that is passing through the tag coil will become maximized when the two coils (reader coil and tag coil) are placed in parallel with respect to each other.

FIGURE 4: A BASIC CONFIGURATION OF READER AND TAG ANTENNAS IN AN RFID APPLICATION



From Equations 3, 4, and 5, the induced voltage V_0 for an untuned loop antenna is given by:

EQUATION 6:

$$V_o = 2\pi f N S B_o \cos\alpha$$

where:

f	=	frequency of the arrival signal
N	=	number of turns of coil in the loop
S	=	area of the loop in square meters (m^2)
B_o	=	strength of the arrival signal
α	=	angle of arrival of the signal

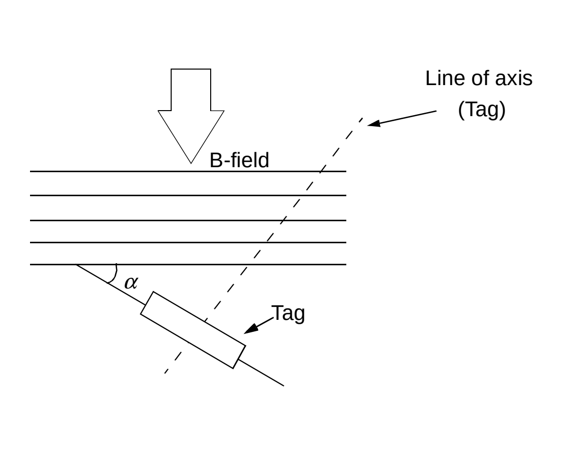
If the coil is tuned (with capacitor C) to the frequency of the arrival signal (125 kHz), the output voltage V_o will rise substantially. The output voltage found in Equation 6 is multiplied by the loaded Q (Quality Factor) of the tuned circuit, which can be varied from 5 to 50 in typical low-frequency RFID applications:

EQUATION 7:

$$V_o = 2\pi f_o N Q S B_o \cos\alpha$$

where the loaded Q is a measure of the selectivity of the frequency of the interest. The Q will be defined in Equations 30, 31, and 37 for general, parallel, and serial resonant circuit, respectively.

FIGURE 5: ORIENTATION DEPENDENCY OF THE TAG ANTENNA.



The induced voltage developed across the loop antenna coil is a function of the angle of the arrival signal. The induced voltage is maximized when the antenna coil is placed perpendicular to the direction of the incoming signal where α = 0.

EXAMPLE 1: B-FIELD REQUIREMENT

The strength of the B-field that is needed to turn on the tag can be calculated from Equation 7:

EQUATION 8:

$$B_o = \frac{Vo}{2\pi f_o N Q S \cos\alpha}$$

$$= \frac{7(2.4)}{(2\pi)(125\ \text{kHz})(100)(15)(38.71\text{cm}^2)}$$

$$\approx 1.5 \qquad \mu\text{Wb/m}^2$$

where the following parameters are used in the above calculation:

tag coil size	=	2 x 3 inches = 38.71 cm^2: (credit card size)
frequency	=	125 kHz
number of turns	=	100
Q of antenna coil	=	15
AC coil voltage to turn on the tag	=	7 V
cos α	=	1 (normal direction, α = 0).

EXAMPLE 2: NUMBER OF TURNS AND CURRENT (AMPERE-TURNS) OF READER COIL

Assuming that the reader should provide a read range of 10 inches (25.4 cm) with a tag given in Example 1, the requirement for the current and number of turns (Ampere-turns) of a reader coil that has an 8 cm radius can be calculated from Equation 3:

EQUATION 9:

$$(NI) = \frac{2B_z(a^2 + r^2)^{3/2}}{\mu a^2}$$

$$= \frac{2(1.5\times10^{-6})(0.08^2 + 0.254^2)^{3/2}}{(4\pi\times10^{-7})(0.08)}$$

$$= 7.04 \ (\text{ampere - turns})$$

This is an attainable number. If, however, we wish to have a read range of 20 inches (50.8 cm), it can be found that NI increases to 48.5 ampere-turns. At 25.2 inches (64 cm), it exceeds 100 ampere-turns.

For a longer read range, it is instructive to consider increasing the radius of the coil. For example, by doubling the radius (16 cm) of the loop, the ampere-turns requirement for the same read range (10 inches: 25.4 cm) becomes:

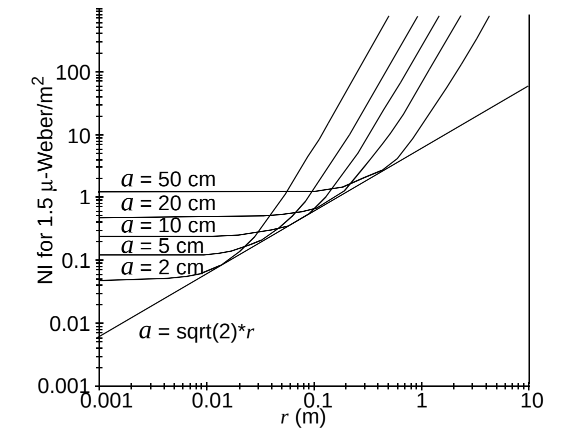
EQUATION 10:

$$NI = \frac{2(1.5 \times 10^{-6})(0.16^2 + 0.25^2)^{3/2}}{(4\pi \times 10^{-7})(0.16^2)}$$

$$= 2.44 \text{ (ampere-turns)}$$

At a read range of 20 inches (50.8 cm), the ampere-turns becomes 13.5 and at 25.2 inches (64 cm), 26.8. Therefore, for a longer read range, increasing the tag size is often more effective than increasing the coil current. Figure 6 shows the relationship between the read range and the ampere-turns (IN).

FIGURE 6: AMPERE-TURNS VS. READ RANGE FOR AN ACCESS CONTROL CARD (CREDIT CARD SIZE)



Note: B_O = 1.5 µWb/m² is used.

The optimum radius of loop that requires the minimum number of ampere-turns for a particular read range can be found from Equation 3 such as:

EQUATION 11:

$$NI = K\frac{(a^2 + r^2)^{\frac{3}{2}}}{a^2}$$

where:

$$K = \frac{2B_z}{\mu_o}$$

By taking derivative with respect to the radius a,

$$\frac{d(NI)}{da} = K\frac{3/2(a^2 + r^2)^{1/2}(2a^3) - 2a(a^2 + r^2)^{3/2}}{a^4}$$

$$= K\frac{(a^2 - 2r^2)(a^2 + r^2)^{1/2}}{a^3}$$

The above equation becomes minimized when:

$$a^2 - 2r^2 = 0$$

The above result shows a relationship between the read range vs. tag size. The optimum radius is found as:

$$a = \sqrt{2}r$$

where:

a	=	radius of coil
r	=	read range

The above result indicates that the optimum radius of loop for a reader antenna is 1.414 times the read range r.

WIRE TYPES AND OHMIC LOSSES

Wire Size and DC Resistance

The diameter of electrical wire is expressed as the American Wire Gauge (AWG) number. The gauge number is inversely proportional to diameter and the diameter is roughly doubled every six wire gauges. The wire with a smaller diameter has higher DC resistance. The DC resistance for a conductor with a uniform cross-sectional area is found by:

EQUATION 12:

$$R_{DC} = \frac{l}{\sigma S} \qquad (\Omega)$$

where:

l	=	total length of the wire
σ	=	conductivity
S	=	cross-sectional area

Table 1 shows the diameter for bare and enamel-coated wires, and DC resistance.

AC Resistance of Wire

At DC, charge carriers are evenly distributed through the entire cross section of a wire. As the frequency increases, the reactance near the center of the wire increases. This results in higher impedance to the current density in the region. Therefore, the charge moves away from the center of the wire and towards the edge of the wire. As a result, the current density decreases in the center of the wire and increases near the edge of the wire. This is called a *skin effect*. The depth into the conductor at which the current density falls to 1/e, or 37% of its value along the surface, is known as the *skin depth* and is a function of the frequency and the permeability and conductivity of the medium. The skin depth is given by:

EQUATION 13:

$$\delta = \frac{1}{\sqrt{\pi f \mu \sigma}}$$

where:

f	=	frequency
μ	=	permeability of material
σ	=	conductivity of the material

EXAMPLE 3:

The skin depth for a copper wire at 125 kHz can be calculated as:

EQUATION 14:

$$\delta = \frac{1}{\sqrt{\pi f (4\pi \times 10^{-7})(5.8 \times 10^{-7})}}$$

$$= \frac{0.06608}{\sqrt{f}} \qquad (m)$$

$$= 0.187 \qquad (mm)$$

The wire resistance increases with frequency, and the resistance due to the skin depth is called an AC resistance. An approximated formula for the ac resistance is given by:

EQUATION 15:

$$R_{ac} \approx \frac{1}{2\sigma\pi\delta} = (R_{DC})\frac{a}{2\delta} \qquad (\Omega)$$

where:

a	=	coil radius

For copper wire, the loss is approximated by the DC resistance of the coil, if the wire radius is greater than $0.066 / \sqrt{f}$ cm. At 125 kHz, the critical radius is 0.019 cm. This is equivalent to #26 gauge wire. Therefore, for minimal loss, wire gauge numbers of greater than #26 should be avoided if coil Q is to be maximized.

TABLE 1: AWG WIRE CHART

Wire Size (AWG)	Dia. in Mils (bare)	Dia. in Mils (coated)	Ohms/ 1000 ft.	Cross Section (mils)
1	289.3	—	0.126	83690
2	287.6	—	0.156	66360
3	229.4	—	0.197	52620
4	204.3	—	0.249	41740
5	181.9	—	0.313	33090
6	162.0	—	0.395	26240
7	166.3	—	0.498	20820
8	128.5	131.6	0.628	16510
9	114.4	116.3	0.793	13090
10	101.9	106.2	0.999	10380
11	90.7	93.5	1.26	8230
12	80.8	83.3	1.59	6530
13	72.0	74.1	2.00	5180
14	64.1	66.7	2.52	4110
15	57.1	59.5	3.18	3260
16	50.8	52.9	4.02	2580
17	45.3	47.2	5.05	2060
18	40.3	42.4	6.39	1620
19	35.9	37.9	8.05	1290
20	32.0	34.0	10.1	1020
21	28.5	30.2	12.8	812
22	25.3	28.0	16.2	640
23	22.6	24.2	20.3	511
24	20.1	21.6	25.7	404
25	17.9	19.3	32.4	320
26	15.9	17.2	41.0	253
27	14.2	15.4	51.4	202
28	12.6	13.8	65.3	159
29	11.3	12.3	81.2	123
30	10.0	11.0	106.0	100
31	8.9	9.9	131	79.2
32	8.0	8.8	162	64.0
33	7.1	7.9	206	50.4
34	6.3	7.0	261	39.7
35	5.6	6.3	331	31.4
36	5.0	5.7	415	25.0
37	4.5	5.1	512	20.2
38	4.0	4.5	648	16.0
39	3.5	4.0	847	12.2
40	3.1	3.5	1080	9.61
41	2.8	3.1	1320	7.84
42	2.5	2.8	1660	6.25
43	2.2	2.5	2140	4.84
44	2.0	2.3	2590	4.00
45	1.76	1.9	3350	3.10
46	1.57	1.7	4210	2.46
47	1.40	1.6	5290	1.96
48	1.24	1.4	6750	1.54
49	1.11	1.3	8420	1.23
50	0.99	1.1	10600	0.98

Note: 1 mil = 2.54 x 10^{-3} cm

Note: 1 mil = 2.54 x 10^{-3} cm

INDUCTANCE OF VARIOUS ANTENNA COILS

The electrical current flowing through a conductor produces a magnetic field. This time-varying magnetic field is capable of producing a flow of current through another conductor. This is called inductance. The inductance L depends on the physical characteristics of the conductor. A coil has more inductance than a straight wire of the same material, and a coil with more turns has more inductance than a coil with fewer turns. The inductance L of inductor is defined as the ratio of the total magnetic flux linkage to the current I through the inductor: i.e.,

EQUATION 16:

$$L = \frac{N\psi}{I} \qquad \text{(Henry)}$$

where:

N = number of turns

I = current

Ψ = magnetic flux

In a typical RFID antenna coil for 125 kHz, the inductance is often chosen as a few (mH) for a tag and from a few hundred to a few thousand (μH) for a reader. For a coil antenna with multiple turns, greater inductance results with closer turns. Therefore, the tag antenna coil that has to be formed in a limited space often needs a multi-layer winding to reduce the number of turns.

The design of the inductor would seem to be a relatively simple matter. However, it is almost impossible to construct an ideal inductor because:

a) The coil has a finite conductivity that results in losses, and

b) The distributed capacitance exists between turns of a coil and between the conductor and surrounding objects.

The actual inductance is always a combination of resistance, inductance, and capacitance. The apparent inductance is the effective inductance at any frequency, i.e., inductive minus the capacitive effect. Various formulas are available in literatures for the calculation of inductance for wires and coils[1, 2].

The parameters in the inductor can be measured. For example, an HP 4285 Precision LCR Meter can measure the inductance, resistance, and Q of the coil.

Inductance of a Straight Wire

The inductance of a straight wound wire shown in Figure 1 is given by:

EQUATION 17:

$$L = 0.002l\left[\log_e\frac{2l}{a} - \frac{3}{4}\right] \qquad (\mu H)$$

where:

l and a = length and radius of wire in cm, respectively.

EXAMPLE 4: CALCULATION OF INDUCTANCE FOR A STRAIGHT WIRE

The inductance of a wire with 10 feet (304.8 cm) long and 2 mm diameter is calculated as follows:

EQUATION 18:

$$L = 0.002(304.8)\left[\ln\left(\frac{2(304.8)}{0.1}\right) - \frac{3}{4}\right]$$

$$= 0.60967(7.965)$$

$$= 4.855(\mu H)$$

Inductance of a Single Layer Coil

The inductance of a single layer coil shown in Figure 7 can be calculated by:

EQUATION 19:

$$L = \frac{(aN)^2}{22.9l + 25.4a} \qquad (\mu H)$$

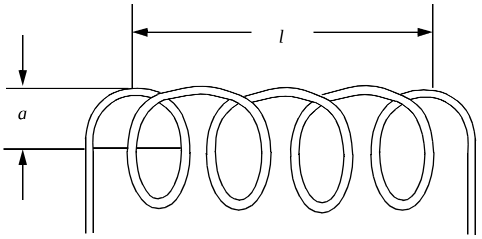
where:

a = coil radius (cm)

l = coil length (cm)

N = number of turns

FIGURE 7: A SINGLE LAYER COIL



Note: For best Q of the coil, the length should be roughly the same as the diameter of the coil.

Inductance of a Circular Loop Antenna Coil with Multilayer

To form a big inductance coil in a limited space, it is more efficient to use multilayer coils. For this reason, a typical RFID antenna coil is formed in a planar multi-turn structure. Figure 8 shows a cross section of the coil. The inductance of a circular ring antenna coil is calculated by an empirical formula[2]:

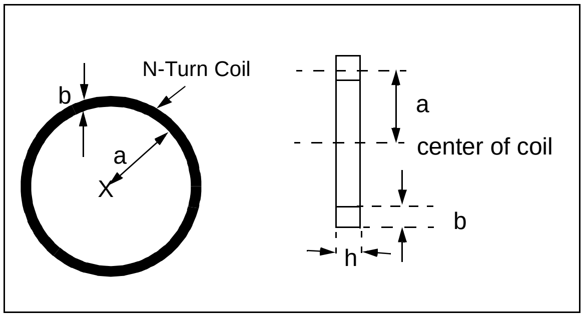
EQUATION 20:

$$L = \frac{0.31(aN)^2}{6a + 9h + 10b} \qquad (\mu H)$$

where:

a = average radius of the coil in cm
N = number of turns
b = winding thickness in cm
h = winding height in cm

FIGURE 8: A CIRCULAR LOOP AIR CORE ANTENNA COIL WITH N-TURNS



The number of turns needed for a certain inductance value is simply obtained from Equation 20 such that:

EQUATION 21:

$$N = \sqrt{\frac{L_{\mu H}(6a + 9h + 10b)}{(0.31)a^2}}$$

EXAMPLE 5: EXAMPLE ON NUMBER OF TURNS

Equation 21 results in N = 200 turns for L = 3.87 mH with the following coil geometry:

a = 1 inch (2.54 cm)
h = 0.05 cm
b = 0.5 cm

To form a resonant circuit for 125 kHz, it needs a capacitor across the inductor. The resonant capacitor can be calculated as:

EQUATION 22:

$$C = \frac{1}{(2\pi f)^2 L} = \frac{1}{(4\pi^2)(125 \times 10^3)(3.87 \times 10^{-3})}$$

$$= 419 \qquad (pF)$$

Inductance of a Square Loop Coil with Multilayer

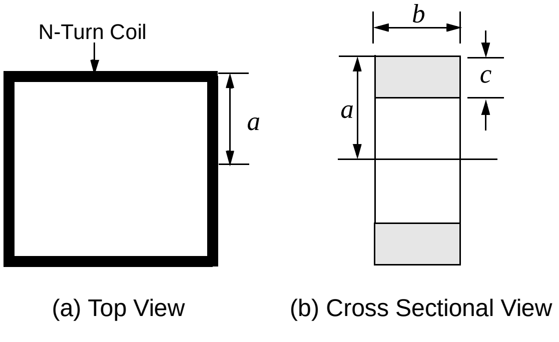
If N is the number of turns and a is the side of the square measured to the center of the rectangular cross section that has length b and depth c as shown in Figure 9, then[2]:

EQUATION 23:

$$L = 0.008aN^2\left(2.303\log_{10}\left(\frac{a}{b+c}\right) + 0.2235\frac{b+c}{a} + 0.726\right) \quad (\mu H)$$

The formulas for inductance are widely published and provide a reasonable approximation for the relationship between inductance and number of turns for a given physical size[1]-[4]. When building prototype coils, it is wise to exceed the number of calculated turns by about 10%, and then remove turns to achieve resonance. For production coils, it is best to specify an inductance and tolerance rather than a specific number of turns.

FIGURE 9: A SQUARE LOOP ANTENNA COIL WITH MULTILAYER



(a) Top View

(b) Cross Sectional View

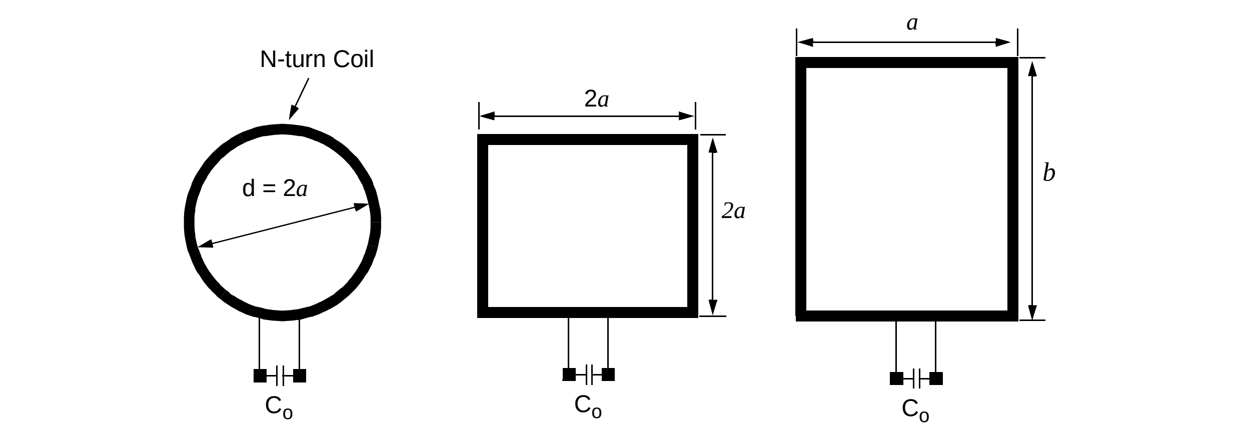
CONFIGURATION OF ANTENNA COILS

Tag Antenna Coil

An antenna coil for an RFID tag can be configured in many different ways, depending on the purpose of the application and the dimensional constraints. A typical inductance L for the tag coil is a few (mH) for 125 kHz devices. Figure 10 shows various configurations of tag antenna coils. The coil is typically made of a thin wire. The inductance and the number of turns of the coil can be calculated by the formulas given in the previous section. An Inductance Meter is often used to measure the inductance of the coil. A typical number of turns of the coil is in the range of 100 turns for 125 kHz and 3~5 turns for 13.56 MHz devices.

For a longer read range, the antenna coil must be tuned properly to the frequency of interest (i.e., 125 kHz). Voltage drop across the coil is maximized by forming a parallel resonant circuit. The tuning is accomplished with a resonant capacitor that is connected in parallel to the coil as shown in Figure 10. The formula for the resonant capacitor value is given in Equation 22.

FIGURE 10: VARIOUS CONFIGURATIONS OF TAG ANTENNA COIL



Reader Antenna Coil

The inductance for the reader antenna coil is typically in the range of a few hundred to a few thousand micro-Henries (µH) for low frequency applications. The reader antenna can be made of either a single coil that is typically forming a series resonant circuit or a double loop (transformer) antenna coil that forms a parallel resonant circuit.

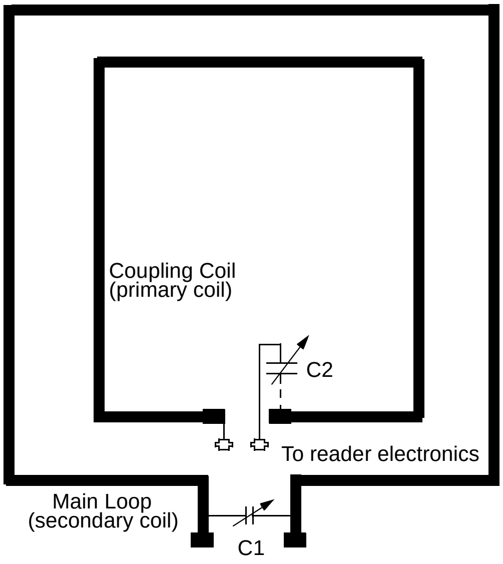
The series resonant circuit results in minimum impedance at the resonance frequency. Therefore, it draws a maximum current at the resonance frequency. On the other hand, the parallel resonant circuit results in maximum impedance at the resonance frequency. Therefore, the current becomes minimized at the resonance frequency. Since the voltage can be stepped up by forming a double loop (parallel) coil, the parallel resonant circuit is often used for a system where a higher voltage signal is required.

Figure 11 shows an example of the transformer loop antenna. The main loop (secondary) is formed with several turns of wire on a large frame, with a tuning capacitor to resonate it to the resonance frequency (125 kHz). The other loop is called a coupling loop (primary), and it is formed with less than two or three turns of coil. This loop is placed in a very close proximity to the main loop, usually (but not necessarily) on the inside edge and not more than a couple of centimeters away from the main loop. The purpose of this loop is to couple signals induced from the main loop to the reader (or vise versa) at a more reasonable matching impedance.

The coupling (primary) loop provides an impedance match to the input/output impedance of the reader. The coil is connected to the input/output signal driver in the reader electronics. The main loop (secondary) must be tuned to resonate at the resonance frequency and is not physically connected to the reader electronics.

The coupling loop is usually untuned, but in some designs, a tuning capacitor C2 is placed in series with the coupling loop. Because there are far fewer turns on the coupling loop than the main loop, its inductance is considerably smaller. As a result, the capacitance to resonate is usually much larger.

FIGURE 11: A TRANSFORMER LOOP ANTENNA FOR READER



RESONANCE CIRCUITS, QUALITY FACTOR Q, AND BANDWIDTH

In RFID applications, the antenna coil is an element of resonant circuit and the read range of the device is greatly affected by the performance of the resonant circuit.

Figures 12 and 13 show typical examples of resonant circuits formed by an antenna coil and a tuning capacitor. The resonance frequency (f_o) of the circuit is determined by:

EQUATION 24:

$$f_o = \frac{1}{2\pi\sqrt{LC}}$$

where:

L = inductance of antenna coil

C = tuning capacitance

The resonant circuit can be formed either series or parallel.

The series resonant circuit has a minimum impedance at the resonance frequency. As a result, maximum current is available in the circuit. This series resonant circuit is typically used for the reader antenna.

On the other hand, the parallel resonant circuit has maximum impedance at the resonance frequency. It offers minimum current and maximum voltage at the resonance frequency. This parallel resonant circuit is used for the tag antenna.

Parallel Resonant Circuit

Figure 12 shows a simple parallel resonant circuit. The total impedance of the circuit is given by:

EQUATION 25:

$$Z(j\omega) = \frac{j\omega L}{(1-\omega^2 LC) + j\frac{\omega L}{R}} \qquad (\Omega)$$

where:

ω = angular frequency = $2\pi f$

R = load resistor

The ohmic resistance r of the coil is ignored. The maximum impedance occurs when the denominator in the above equation minimized such as:

EQUATION 26:

$$\omega^2 LC = 1$$

This is called a resonance condition and the resonance frequency is given by:

EQUATION 27:

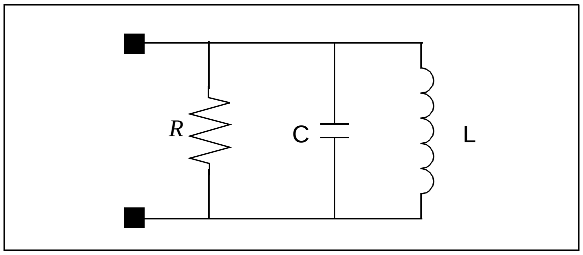
$$f_o = \frac{1}{2\pi\sqrt{LC}}$$

By applying Equation 26 into Equation 25, the impedance at the resonance frequency becomes:

EQUATION 28:

$$Z = R$$

FIGURE 12: PARALLEL RESONANT CIRCUIT



The R and C in the parallel resonant circuit determine the bandwidth, B, of the circuit.

EQUATION 29:

$$B = \frac{1}{2\pi RC} \qquad (Hz)$$

The quality factor, Q, is defined by various ways such as:

EQUATION 30:

$$Q = \frac{\text{Energy Stored in the System per One Cycle}}{\text{Energy Dissipated in the System per One Cycle}}$$

$$= \frac{f_o}{B}$$

where:

f_o = resonant frequency

B = bandwidth

By applying Equation 27 and Equation 29 into Equation 30, the loaded Q in the parallel resonant circuit is:

EQUATION 31:

$$Q = R\sqrt{\frac{C}{L}}$$

The Q in parallel resonant circuit is directly proportional to the load resistor R and also to the square root of the ratio of capacitance and inductance in the circuit.

When this parallel resonant circuit is used for the tag antenna circuit, the voltage drop across the circuit can be obtained by combining Equations 7 and 31,

EQUATION 32:

$$V_o = 2\pi f_o NQSB_o \cos\alpha$$

$$= 2\pi f_o N\left(R\sqrt{\frac{C}{L}}\right)SB_o \cos\alpha$$

The above equation indicates that the induced voltage in the tag coil is inversely proportional to the square root of the coil inductance, but proportional to the number of turns and surface area of the coil.

The parallel resonant circuit can be used in the transformer loop antenna for a long-range reader as discussed in "Reader Antenna Coil" (Figure 11). The voltage in the secondary loop is proportional to the turn ratio (n_2/n_1) of the transformer loop. However, this high voltage signal can corrupt the receiving signals. For this reason, a separate antenna is needed for receiving the signal. This receiving antenna circuit should be tuned to the modulating signal of the tag and detunned to the carrier signal frequency for maximum read range.

Series Resonant Circuit

A simple series resonant circuit is shown in Figure 13. The expression for the impedance of the circuit is:

EQUATION 33:

$$Z(j\omega) = r + j(X_L - X_C) \qquad (\Omega)$$

where:

r = ohmic resistance of the circuit

EQUATION 34:

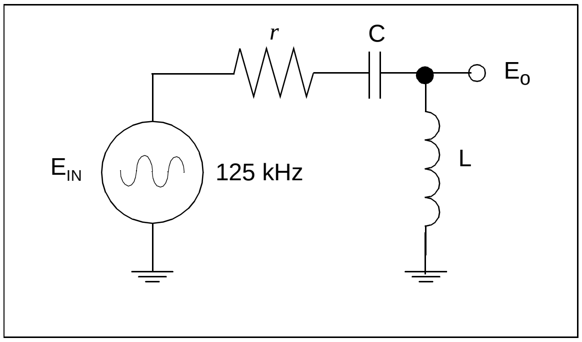
$$X_L = 2\pi f_o L \qquad (\Omega)$$

EQUATION 35:

$$X_c = \frac{1}{2\pi f_o C} \qquad (\Omega)$$

The impedance in Equation 33 becomes minimized when the reactance component cancelled out each other such that $X_L = X_C$. This is called a resonance condition. The resonance frequency is same as the parallel resonant frequency given in Equation 27.

FIGURE 13: SERIES RESONANCE CIRCUIT



The half power frequency bandwidth is determined by r and L, and given by:

EQUATION 36:

$$\boldsymbol{B} = \frac{r}{2\pi L} \qquad (Hz)$$

The quality factor, Q, in the series resonant circuit is given by:

EQUATION 37:

$$Q = \frac{f_o}{B} = \begin{cases} \dfrac{\omega L}{r} = \dfrac{1}{\omega C r} & \text{; for unloaded circuit} \\ \dfrac{1}{r}\sqrt{\dfrac{L}{C}} & \text{; for loaded circuit} \end{cases}$$

The series circuit forms a voltage divider; the voltage drops in the coil is given by:

EQUATION 38:

$$V_o = \frac{jX_L}{r + jX_L - jX_c} V_{in}$$

or

EQUATION 39:

$$\left|\frac{V_o}{V_{in}}\right| = \frac{X_L}{\sqrt{r^2 + (X_L - X_c)^2}} = \frac{X_L}{r\sqrt{1 + \left(\dfrac{X_L - X_c}{r}\right)^2}} = \frac{Q}{\sqrt{1 + \left(\dfrac{X_L - X_c}{r}\right)^2}}$$

EXAMPLE 6: CIRCUIT PARAMETERS.

If the series resistance of the circuit is 15 Ω, then the L and C values form a 125 kHz resonant circuit with Q = 8 are:

EQUATION 40:

$$X_L = Qr_s = 120\Omega$$

$$L = \frac{X_L}{2\pi f} = \frac{120}{2\pi(125\text{ kHz})} = 153 \qquad (\mu H)$$

$$C = \frac{1}{2\pi f X_L} = \frac{1}{2\pi(125\text{ kHz})(120)} = 10.6 \qquad (nF)$$

EXAMPLE 7: CALCULATION OF READ RANGE

Let us consider designing a reader antenna coil with L = 153 µH, diameter = 10 cm, and winding thickness and height are small compared to the diameter.

The number of turns for the inductance can be calculated from Equation 21, resulting in 24 turns.

If the current flow through the coil is 0.5 amperes, the ampere-turns becomes 12. Therefore, the read range for this coil will be about 20 cm with a credit card size tag.

Q and Bandwidth

Figure 14 shows the approximate frequency bands for common forms of Amplitude Shift Keying (ASK), Frequency Shift Keying (FSK), and Phase Shift Keying (PSK) modulation. For a full recovery of data signal from the tag, the reader circuit needs a bandwidth that is at least twice the data rate. Therefore, if the data rate is 8 kHz for an ASK signal, the bandwidth must be at least 16 kHz for a full recovery of the information that is coming from the tag.

The data rate for FSK (÷ 10) signal is 12.5 kHz. Therefore, a bandwidth of 25 kHz is needed for a full data recovery.

The Q for this FSK (÷ 10) signal can be obtained from Equation 30.

EQUATION 41:

$$Q = \frac{f_o}{B} = \frac{125 \text{ kHz}}{25 \text{ kHz}}$$

$$= 5$$

For a PSK (÷ 2) signal, the data rate is 62.5 kHz (if the carrier frequency is 125 kHz) therefore, the reader circuit needs 125 kHz of bandwidth. The Q in this case is 1, and consequently the circuit becomes Q-independent.

This problem may be solved by separating the transmitting and receiving coils. The transmitting coil can be designed with higher Q and the receiving coil with lower Q.

Limitation on *Q*

When designing a reader antenna circuit, the temptation is to design a coil with very high Q. There are three important limitations to this approach.

a) Very high voltages can cause insulation breakdown in either the coil or resonant capacitor.

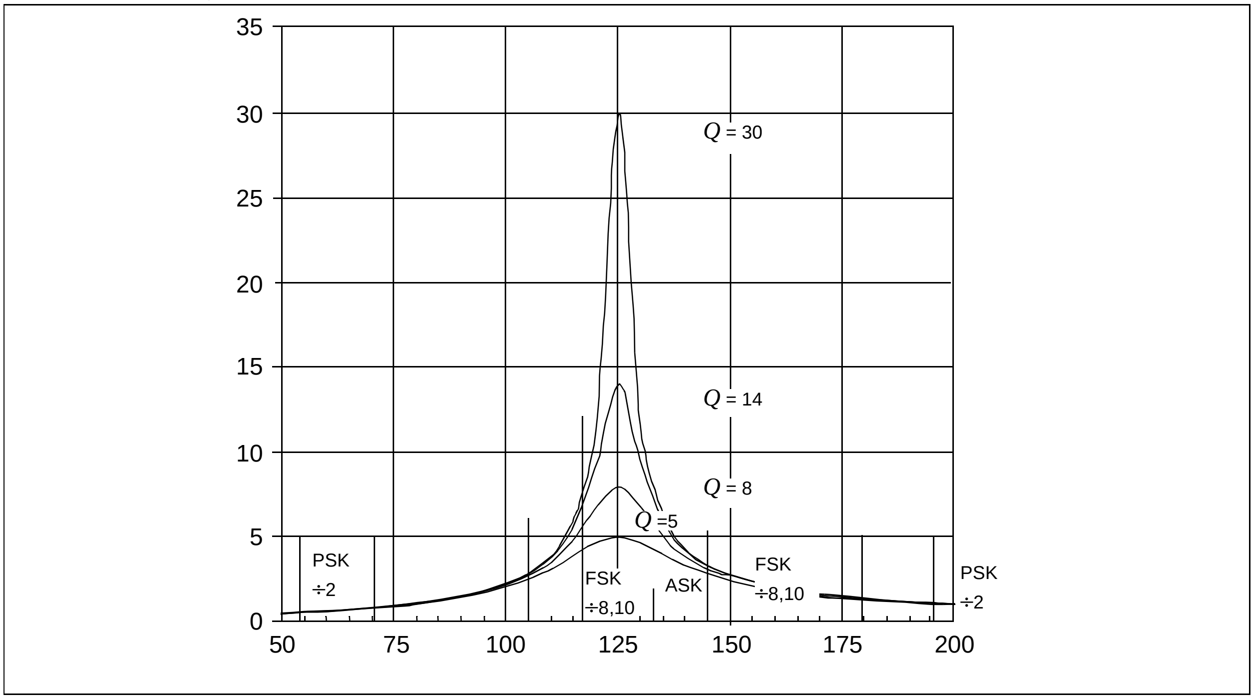
   For example, a 1 ampere of current flow in a 2 mH coil will produce a voltage drop of 1500 VPP. Such voltages are easy to obtain but difficult to isolate. In addition, in the case of single coil reader designs, recovery of the return signal from the tag must be accomplished in the presence of these high voltages.

b) Tuning becomes critical.

   To implement a high Q antenna circuit, high voltage components with a close tolerance and high stability would have to be used. Such parts are generally expensive and difficult to obtain.

c) As the Q of the circuit gets higher, the amplitude of the return signal relative to the power of the carrier gets proportionally smaller complicating its recovery by the reader circuit.

FIGURE 14: *Q* FACTOR VS. MODULATION SIGNALS



Tuning Method

The circuit must be tuned to the resonance frequency for a maximum performance (read range) of the device. Two examples of tuning the circuit are as follows:

- **Voltage Measurement Method:**
  - a) Set up a voltage signal source at the resonance frequency (125 kHz)
  - b) Connect a voltage signal source across the resonant circuit.
  - c) Connect an Oscilloscope across the resonant circuit.
  - d) Tune the capacitor or the coil while observing the signal amplitude on the Oscilloscope.
  - e) Stop the tuning at the maximum voltage.
- **S-parameter or Impedance Measurement Method using Network Analyzer:**
  - a) Set up an S-Parameter Test Set (Network Analyzer) for S11 measurement, and do a calibration.
  - b) Measure the S11 for the resonant circuit.
  - c) Reflection impedance or reflection admittance can be measured instead of the S11.
  - d) Tune the capacitor or the coil until a maximum null (S11) occurs at the resonance frequency, f_o. For the impedance measurement, the maximum peak will occur for the parallel resonant circuit, and minimum peak for the series resonant circuit.

FIGURE 15: VOLTAGE VS. FREQUENCY FOR RESONANT CIRCUIT

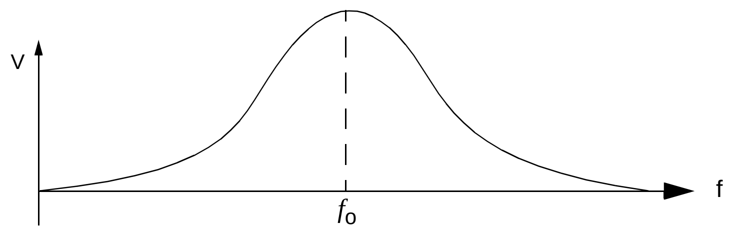
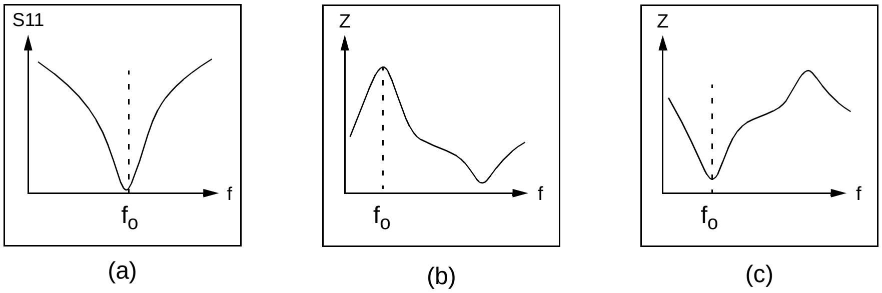


FIGURE 16: FREQUENCY RESPONSES FOR RESONANT CIRCUIT



Note 1: (a) S11 Response, (b) Impedance Response for a Parallel Resonant Circuit, and (c) Impedance Response for a Series Resonant Circuit.

2: In (a), the null at the resonance frequency represents a minimum input reflection at the resonance frequency. This means the circuit absorbs the signal at the frequency while other frequencies are reflected back. In (b), the impedance curve has a peak at the resonance frequency. This is because the parallel resonant circuit has a maximum impedance at the resonance frequency. (c) shows a response for the series resonant circuit. Since the series resonant circuit has a minimum impedance at the resonance frequency, a minimum peak occurs at the resonance frequency.

READ RANGE OF RFID DEVICES

Read range is defined as a maximum communication distance between the reader and tag. The read range of typical passive RFID products varies from about 1 inch to 1 meter, depending on system configuration. The read range of an RFID device is, in general, affected by the following parameters:

a) Operating frequency and performance of antenna coils

b) *Q* of antenna and tuning circuit

c) Antenna orientation

d) Excitation current and voltage

e) Sensitivity of receiver

f) Coding (or modulation) and decoding (or demodulation) algorithm

g) Number of data bits and detection (interpretation) algorithm

h) Condition of operating environment (metallic, electrical noise), etc.

With a given operating frequency, the above conditions (a – c) are related to the antenna configuration and tuning circuit. The conditions (d – e) are determined by a circuit topology of the reader. The condition (f) is called the communication protocol of the device, and (g) is related to a firmware program for data interpretation.

Assuming the device is operating under a given condition, the read range of the device is largely affected by the performance of the antenna coil. It is always true that a longer read range is expected with the larger size of the antenna. Figures 17 and 18 show typical examples of the read range of various passive RFID devices.

FIGURE 17: READ RANGE VS. TAG SIZE FOR PROXIMITY APPLICATIONS

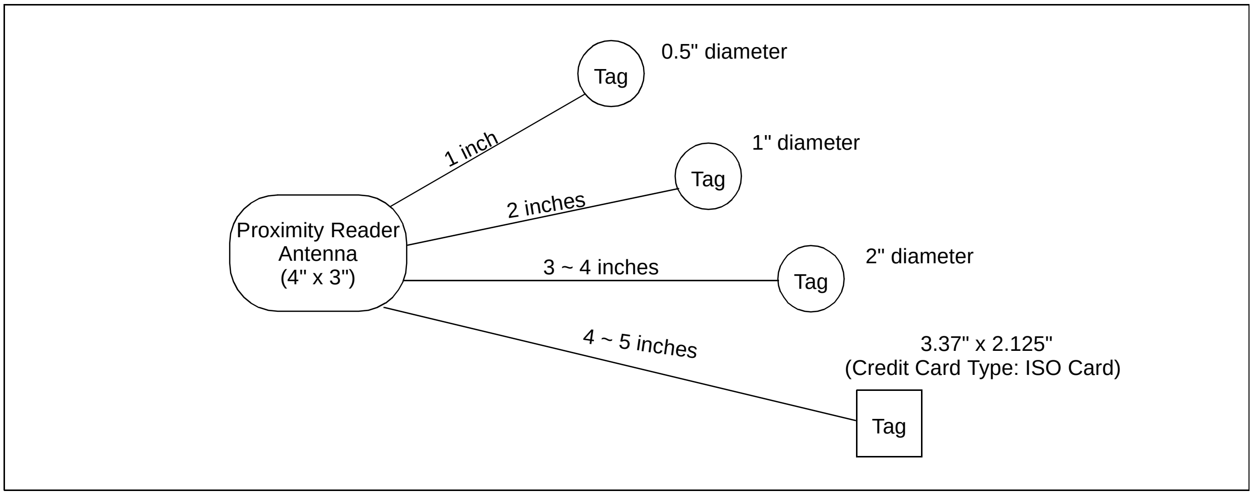
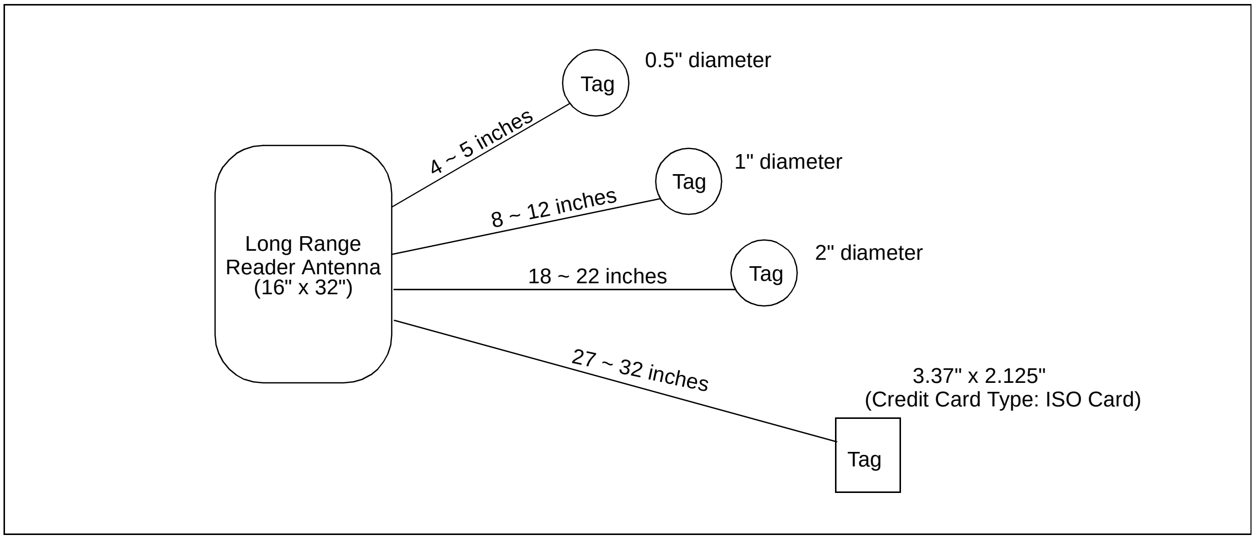


FIGURE 18: READ RANGE VS. TAG SIZE FOR LONG RANGE APPLICATIONS



REFERENCES

1. Frederick W. Grover, Inductance Calculations: Working Formulas and Tables, Dover Publications, Inc., New York, NY., 1946.
2. Keith Henry, Editor, Radio Engineering Handbook, McGraw-Hill Book Company, New York, NY., 1963.
3. V. G. Welsby, The Theory and Design of Inductance Coils, John Wiley and Sons, Inc., 1960.
4. James K. Hardy, High Frequency Circuit Design, Reston Publishing Company, Inc., Reston, Virginia, 1975.

FSK Reader Reference Design

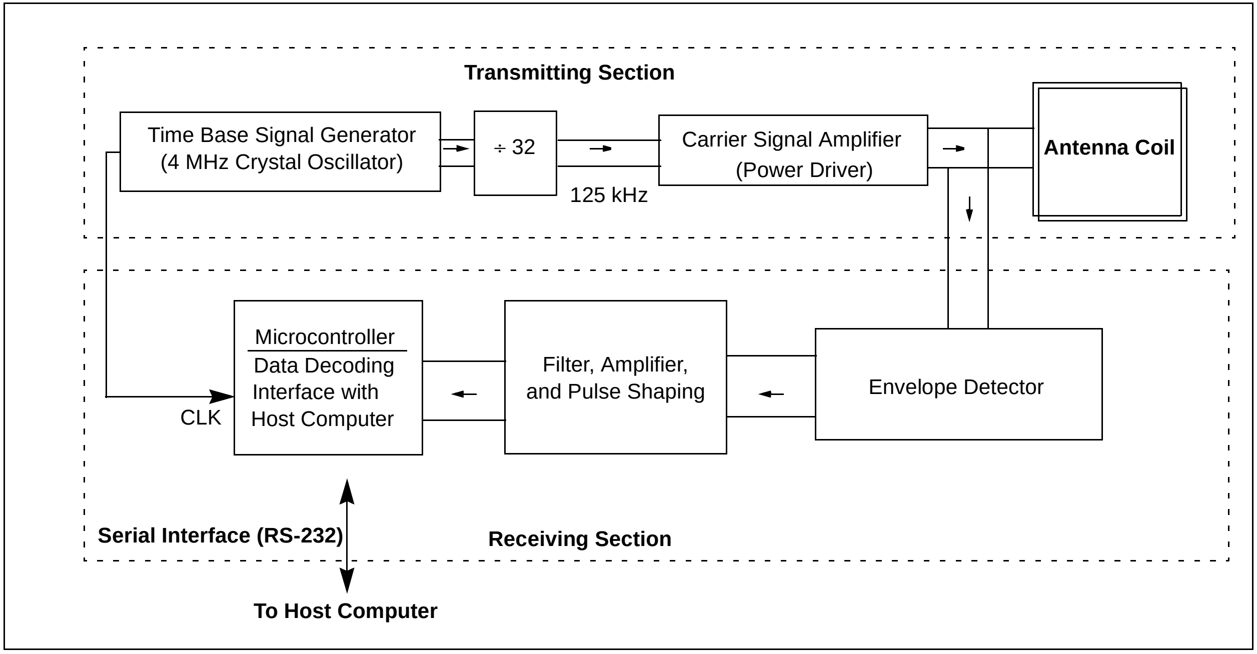
1.0 INTRODUCTION

This application note is written as a reference guide for FSK reader designers. Microchip Technology Inc. provides basic reader electronics circuitry for the MCRF200 customers as a part of this design guide. The circuit is designed for a read range of 3 ~ 5 inches with an access control card. The microID FSK Reader (demo unit), which is built based on the FSK reference design, is available in the microID Designers Kit (DV103001). The circuit can be modified for longer read range or other applications with the MCRF200. An electronic copy of the FSK microID PICmicro® source code is available upon request.

2.0 READER CIRCUITS

The RFID reader consists of transmitting and receiving sections. It transmits a carrier signal, receives the backscattering signal, and performs data processing. The reader also communicates with an external host computer. A basic block diagram of the typical RFID reader is shown in Figure 2-1.

FIGURE 2-1: BLOCK DIAGRAM OF TYPICAL RFID READER FOR FSK SIGNAL (125 kHz)



PICmicro is a registered trademark of Microchip Technology Inc.

2.1 Transmitting Section

The transmitting section contains circuitry for a carrier signal (125 kHz), power amplifiers, and a tuned antenna coil.

The 125 kHz carrier signal is typically generated by dividing a 4 MHz (4 MHz/32 = 125 kHz) crystal oscillator signal. The signal is amplified before it is fed into the antenna tuning circuit. A complementary power amplifier circuit is typically used to boost the transmitting signal level.

An antenna impedance tuning circuit consisting of capacitors is used to maximize the signal level at the carrier frequency. This tuning circuit is also needed to form an exact LC resonant circuit for the carrier signal. The tuning compensates the variations in the component values and the perturbation of coil inductance due to environment effect. A design guide for the antenna coil is given in *AN678, RFID Coil Design,* page 25.

2.1.1 LIMITS ON TRANSMITTING SIGNAL LEVEL (FCC PART 15) IN THE USA

Each country limits the signal strength of the RF wave that is intentionally radiated by a device. In the USA, the signal strength of the carrier signal (125 kHz) radiating from the antenna coil must comply with the FCC (Federal Communications Commission) part 15 regulation. The signal level is specified by the 47 CFR Part 15.209a of the federal regulation. For a 125 kHz signal, the FCC limits the signal level to 19.2 µv per meter, or 25.66 dBµV (i.e., 20 log(19.2) = 25.66 dBµV), at 300 meters away from the antenna. For a close distance measurement, an extrapolation rule (40 dB per decade) is applied (Part 15.31.f.2). For example, the signal level at 30 meters away from the device must not exceed:

$$25.66 \text{ dB}\mu\text{V} + 40 \text{ dB}\mu\text{V} = 65.66 \text{ dB}\mu\text{V}$$

2.2 Receiving Section

The receiving section consists of an antenna coil, demodulator, filters, amplifiers, and microcontroller. In applications for close proximity read range, a single coil is often used for both transmitting and receiving. For long read-range applications, however, separated antennas may be used. More details on the antenna coil are given in *AN678, RFID Coil Design,* page 25.

In the FSK communication protocol, a ‘0’ and a ‘1’ are represented by two different frequencies. In the MCRF200, a ‘0’ and a ‘1’ are represented by Fc/8 and Fc/10, respectively. Fc is the carrier frequency. The MCRF200 sends this FSK signal to the reader by an amplitude modulation of the carrier signal.

The FSK reader needs two steps for a full recovery of the data. The first step is demodulating the backscattering signal, and the second step is detecting the frequency (or period) of the demodulation signal.

The demodulation is accomplished by detecting the envelope of the carrier signal. A half-wave capacitor-filtered rectifier circuit is used for the demodulation process. A diode detects the peak voltage of the backscattering signal. The voltage is then fed into an RC charging/discharging circuit. The RC time constant must be small enough to allow the voltage across C to fall fast enough to keep in step with the envelope. However, the time constant must not be so small as to introduce excessive ripple. The demodulated signal must then pass through a filter and signal shaping circuit before it is fed to the microcontroller. The microcontroller performs data decoding and communicates with the host computer through an RS-232 or other serial interface protocols.

3.0 microID FSK READER

The electronic circuitry for an FSK reader is shown in Figure 3-1. The reader needs +9 VDC power supply. The 125 kHz carrier signal is generated by dividing the 4 MHz time base signal that is generated by a crystal oscillator. A 16-stage binary ripple counter (74HC4060) is used for this purpose. The 74HC4060 also provides a clock signal for the PIC16C84 microcontroller. The 125 kHz signal is passed to an RF choke (L1) and filter before it is fed into a power amplifier that is formed by a pair of complementary bipolar transistors (Q2 and Q3).

For long read-range applications, this power amplifier circuit can be modified. Power MOSFETs may be used instead of the bipolar transistors (2N2222). These power MOSFETs can be driven by +24 VDC power supply. A push-pull predriver can be added at the front of the complementary circuit. This modification will enhance the signal level of the carrier signal.

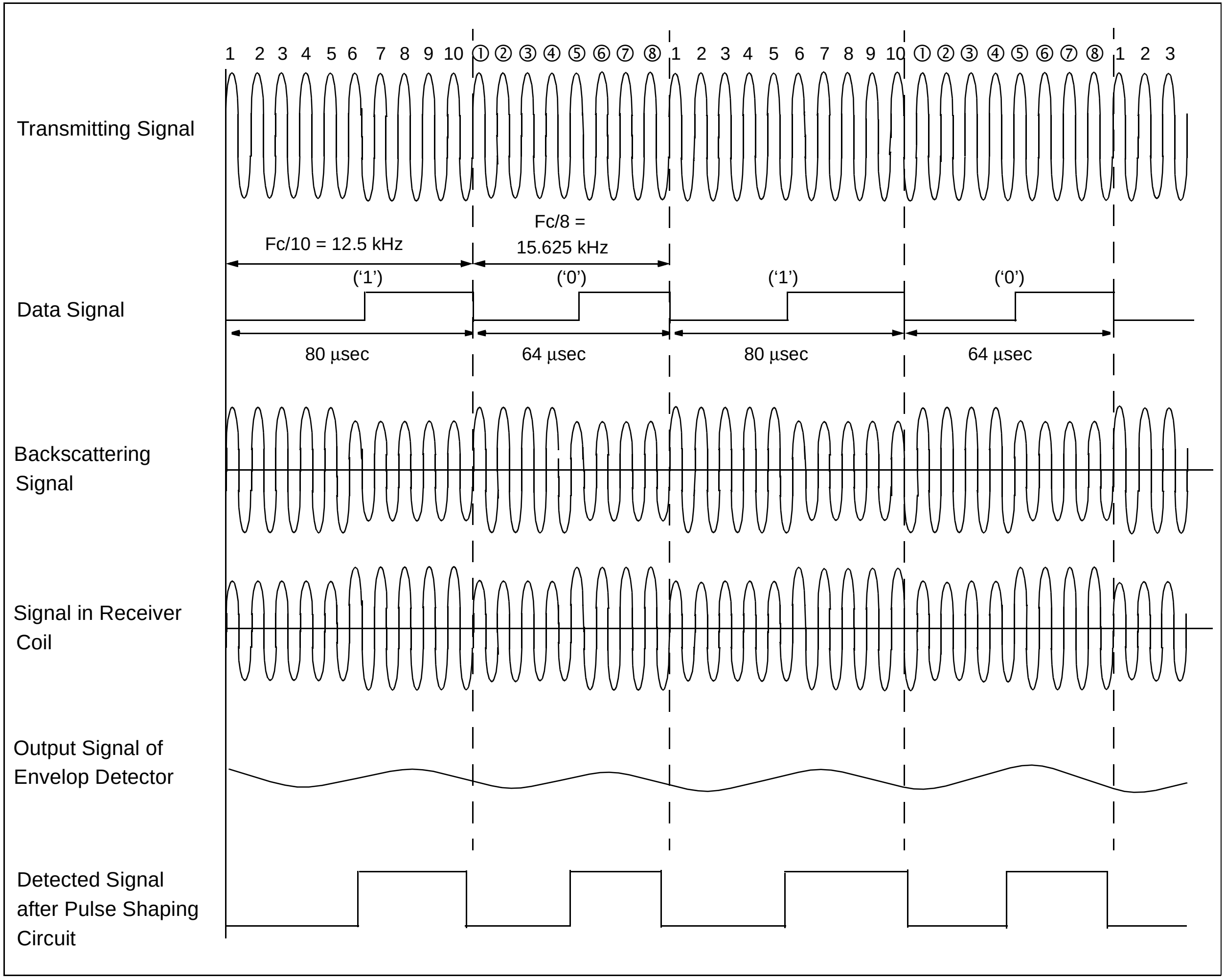
The reader circuit uses a single coil for both transmitting and receiving signals. An antenna coil (L2: 1.62 mH) and a resonant capacitor (C2: 1000 pF) forms a series resonant circuit for a 125 kHz resonance frequency. Since the C2 is grounded, the carrier signal (125 kHz) is filtered out to ground after passing the antenna coil. The circuit provides a minimum impedance at the resonance frequency. This results in maximizing the antenna current, and therefore, the magnetic field strength is maximized.

L2, C15, D7, and the other bottom parts in the circuit form a signal receiving section. The voltage drop in the antenna coil is a summation (superposition) of transmitting signal and backscattering signal. The D7 is a demodulator which detects the envelope of the backscattering signal. The FSK signal waveforms are shown in Figure 3-1.

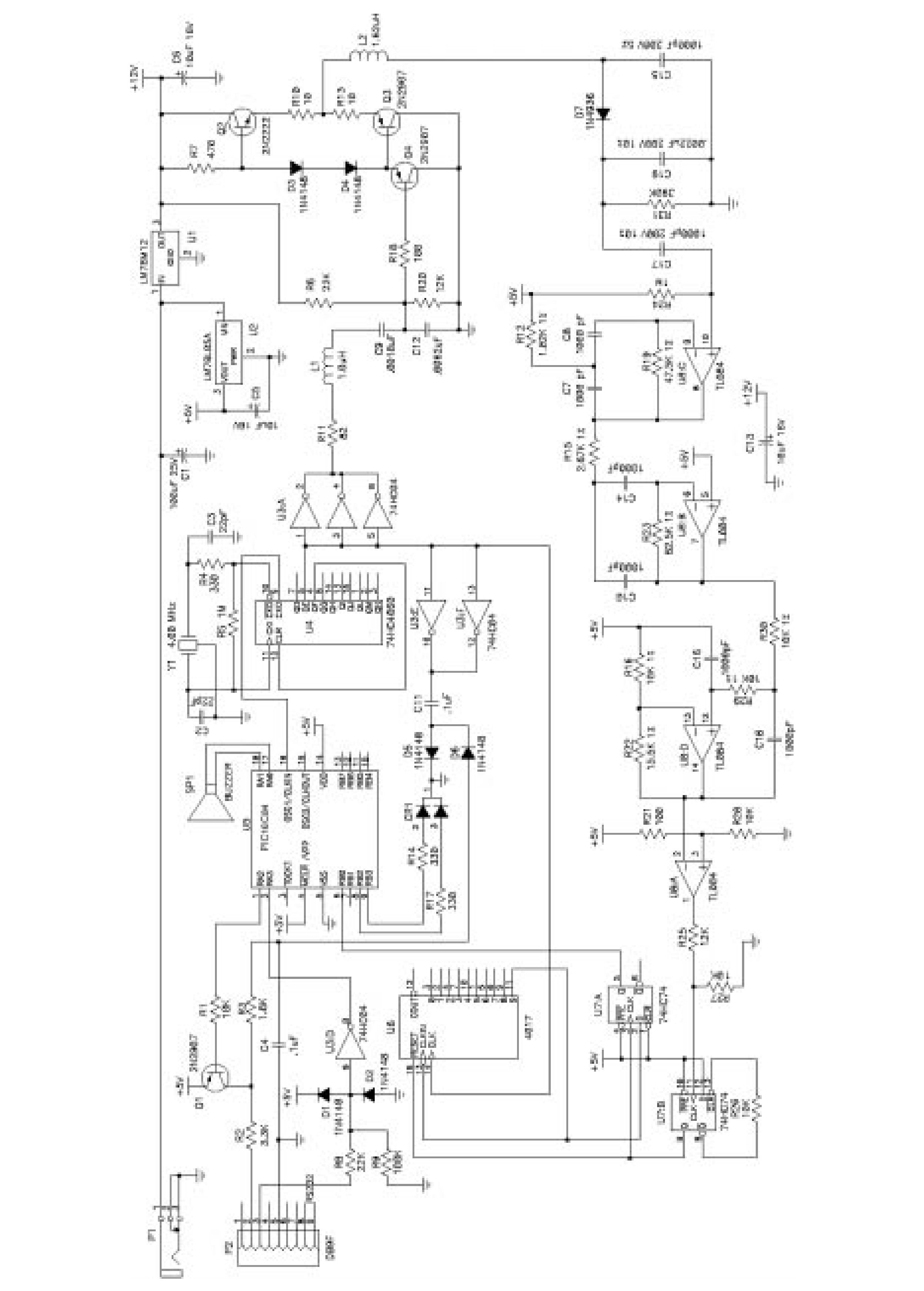
D7 and C19 form a half-wave capacitor-filtered rectifier circuit. The detected envelope signal is charged into the C19. R21 provides a discharge path for the voltage charged in the C19. This voltage passes active filters (U8) and the pulse shaping circuitry (U8) before it is fed into the PIC16C84 for data processing.

The PIC16C84 microcontroller performs data decoding and communicates with the host computer via an RS-232 serial interface.

FIGURE 3-1: SIGNAL WAVEFORM FOR FSK PROTOCOL (Fc = 125 KHZ)



4.0 FSK READER SCHEMATIC



5.0 FSK READER BILL OF MATERIALS

Item #	Qty	Part #	Reference Designator	Part Description	Manufacturer	Vendor	Vendor Part #
1	1	110-93-318-41-001	xU5	SOCKET, 18P OPEN FRAME COLLET (0.300)	MILL-MAX	DIGIKEY	ED3318-ND
2	1	DE9S-FRS	P2	CONN, D-SUB 9P RECPT RT ANGLE	SPC TECHNOLOGY		
3	1	DJ005B	P1	JACK, POWER, 2.5 mm DC PC MOUNT	LZR ELECTRONICS		
4	1	PKM22EPP-4001	SP1	BUZZER, PIEZO, 4kHz, 3-20V	MURATA		
5	2	D220J20COGHAAAC	C2, C3	CAP, 22 pF CER DISK RAD COG 100V	PHILIPS	DIGIKEY	1330PH-ND
6	6	ECQ-P6102JU	C7, C8, C10, C14, C16, C18	CAP, 0.001 uF POLYPROPYLENE 630V	PHILIPS	DIGIKEY	P3497-ND
7	2	2222 370 52102	C15, C17	CAP, 0.001 uF METAL FILM, 5%, RAD, 400V	PHILIPS	DIGIKEY	3001PH-ND
8	1	ECU-S2A182JCB	C9	CAP, 1800 pF MONOLITH CERM, 5%, RAD, 100V	PHILIPS	DIGIKEY	P4864-ND
9	1	2222 370 52222	C19	CAP, 0.0022 UF 400V 5% MF BOX	PHILIPS	DIGIKEY	3003PH-ND
10	1	ECU-S1H682JCB	C12	CAP, 6800 pF 50V CERAMIC MONO 5%	PANASONIC	DIGIKEY	P4946-ND
11	2	ECQ-E1104KF	C4, C11	CAP, 0.1UF 100VDC 10% RAD METAL POLY CAP	PANASONIC	DIGIKEY	EF1104-ND
12	3	ECS-F1CE106K	C5, C6, C13	CAP, TANT, 10uF, 16V	PANASONIC	DIGIKEY	P2038-ND
13	1	ECS-F1AE107	C1	CAP, 100 UFD @ 10VDC 20% TANTALUM CAP	PANASONIC	DIGIKEY	P2032-ND
14	6	1N4148	D1-D6	DIODE, GENERAL PURPOSE, 1N4148 (DO-35)	DIODES INC.	DIGIKEY	1N4148DITR-ND
15	1	1N4936	D7	DIODE, 1A 400V FAST-RECOVERY RECTIFIER	DIODES INC	DIGIKEY	1N4936CT-ND
16	1	-SPARE-	LED1	-SPARE- LOCATION DO NOT INSTALL			
17	1	78F102J	L1	INDUCTOR, 1000 µH, COATED	JW MILLER	DIGIKEY	M7849-ND
18	1	MCT0003-001	L2	INDUCTOR, 1.62 mH	CORNELL DUBILIER		
19	3	2N2907A	Q1, Q3, Q4	TRANSISTOR, PNP, 2N2907A, TO-92	MOTOROLA		
20	1	2N2222A	Q2	TRANSISTOR, NPN, 2N2222A, TO-92	MOTOROLA	ALLIED	2N2222A
21	2	5043CX10R0J	R10, R13	RES, CF 10 OHM 1/4W 5%	PHILLIPS		
22	1	82E CR-1/4W-B 5%	R11	RES, CF 82 OHM 1/4W 5%	YAGEO	DIGIKEY	82QBK-ND
23	2	5043CX100R0J	R18, R21	RES, CF 100 OHM 1/4W 5%	PHILLIPS		
24	3	5043CX330R0J	R4, R14, R17	RES, CF 330 OHM 1/4W 5%	PHILLIPS		

Item #	Qty	Part #	Reference Designator	Part Description	Manufacturer	Vendor	Vendor Part #
25	1	5043CX470R0J	R7	RES, CF 470 OHM 5% 1/4W	PHILLIPS		
26	1	1K8 CR-1/4W-B 5%	R3	RES, CF 1.8K OHM 1/4W 5%	YAGEO	DIGIKEY	1.8KQBK-ND
27	1	1K82 MF-1/4W-B 1%	R12	RES, MF 1.82K OHM 1/4W 1%	YAGEO	DIGIKEY	1.82KXBK-ND
28	1	2K67 MF-1/4W-B 1%	R15	RES, 2.67K OHM 1/4W 1% MF	YAGEO	DIGIKEY	2.67KXBK-ND
29	1	3K3 CR-1/4W-B 5%	R2	RES, CF 3.3K OHM 1/4W 5%	YAGEO	DIGIKEY	3.3KQBK-ND
30	4	10K CR-1/4W-B 5%	R1, R26, R27, R28	RES, CF 10K OHM 1/4W 5%	YAGEO	DIGIKEY	10KQBK-ND
31	3	5043ED10K00F	R16, R29, R30	RES, MF 10K 1/4W 1%	PHILLIPS		
32	2	12K CR-1/4W-B 5%	R20, R25	RES, CF 12K OHM 1/4W 5%	YAGEO	DIGIKEY	12KQBK-ND
33	1	16K5 MF-1/4W-B 1%	R22	RES, MF 16.5K OHM 1/4W 1%	YAGEO	DIGIKEY	16.5KXBK-ND
34	1	22K CR-1/4W-B 5%	R6	RES, CF 22K OHM 1/4W 5%	YAGEO	DIGIKEY	22KQBK-ND
35	1	47K5 MF-1/4W-B 1%	R19	RES, MF 47.5K OHM 1/4W 1%	YAGEO	DIGIKEY	47.5KXBK-ND
36	1	82K5 MF-1/4W-B 1%	R23	RES, 82.5K OHM 1/4W 1% MF	YAGEO	DIGIKEY	82.5KXBK-ND
37	1	5043CX100K0J	R9	RES, CF 100K 5% 1/4W	PHILLIPS		
38	2	1M0 CR-1/4W-B 5%	R5, R24	RES, CF 1.0M OHM 1/4W 5%	YAGEO	DIGIKEY	1.0MQBK-ND
39	1	390K CR-1/4W-B 5%	R31	RES, 390K OHM 1/4W 5% CF	YAGEO	DIGIKEY	390KQBK-ND
40	1	LM78M12Ct	U1	IC, REG 12V 3 TERM POS (TO-220)	NATIONAL	DIGIKEY	LM78M12CT-ND
41	1	LM78L05ACZ	U2	IC, REG, +5V 0.1A TO-92	NATIONAL	DIGIKEY	LM78L05ACZ-ND
42	1	MM74HC04N	U3	IC, HEX INVERTER 14P DIP	FAIRCHILD SEMICONDUCTOR	DIGIKEY	MM74HC04N-ND
43	1	MM74HC4060N	U4	IC, 14 STAGE BINARY COUNTER, 16P DIP	FAIRCHILD SEMICONDUCTOR	DIGIKEY	MM74HC4060N-ND
44	1	PIC16C84/P	U5	IC, PIC16C84 PLASTIC, 14P DIP	MICROCHIP		
45	1	CD4017BCN	U6	IC, DECADE COUNTER	FAIRCHILD	DIGIKEY	CD4017BCN-ND
46	1	MM74HC74AN	U7	IC, DUAL D TYPE FLIP FLOP 14P DIP	FAIRCHILD	DIGIKEY	MM74HC74AN-ND
47	1	TL084CN	U8	IC, QUAD OP AMP, 14P DIP	SGS THOMPSON	MOUSER	511-TL084CN
48	1	EFO-EC4004A4	Y1	RESONATOR, 4.00MHZ CERAMIC W/CAP	PANASONIC	DIGIKEY	PX400-ND

6.0 FSK SOURCE CODE FOR THE PICmicro® MCU

The following source code is for the PIC16C84 microcontroller used in the FSK reader electronics.

```
; #=#=#=#=#=#=#=#=#=#=#= PROJECT  Microchip FSK Reader =#=#=#=#=#=#=#=#=#=#=#
;
; PIC16C84 running at 4MHz, Ti=1us

; ///////////////////////////////////////////////////////////////////////////
; Revision history
; ///////////////////////////////////////////////////////////////////////////
;
; Ver    Date        Comment
;
; 0.01   29 Dec 97   Copied from MChip\Reader\FSK
; 0.03   28 Jan 98   TRANSMIT TAB (h’09’) REGULARLY
;        20 Aug 98   Modified to correct FSK comments
;
;       Tbit=50Tcy=400Ti
;       Ttag=96Tbit
;       Header=h’802A’
;

    processor pic16c84
    #include “p16c84.inc”
        __config b’11111111101001’
        ; Code Protect on, power-up timer on, WDT off, XT oscillator

#define _CARRY          STATUS,0
#define _ZERO           STATUS,2
#define _TO             STATUS,4
#define _RP0            STATUS,5

#define _BUZZ1          PORTA,0
#define _BUZZ2          PORTA,1
#define _RS232TX        PORTA,2
#define _RS232RX        PORTA,3
#define _T0CKI          PORTA,4
StartPORTA      = b’01100’
StartTRISA      = b’11000’
BeepPort        = PORTA
Beep0           = StartPORTA
Beep1           = StartPORTA | b’00001’
Beep2           = StartPORTA | b’00010’

#define _DATA_IN        PORTB,0
#define _UNUSED1        PORTB,1
#define _LED1           PORTB,2
#define _LED2           PORTB,3
#define _UNUSED2        PORTB,4
#define _UNUSED3        PORTB,5
#define _UNUSED4        PORTB,6
#define _UNUSED5        PORTB,7
StartPORTB      = b’00000000’
StartTRISB      = b’00000001’

StartOPTION     = b’00001111’   ; TMR0 internal, prescaler off

BO3             = h’0C’
DelayReg        = h’0C’
BitCtr          = h’0D’
BeepCtrHi       = h’0D’
TxByte          = h’0E’
BeepCtrLo       = h’0E’

Buffer0         = h’10’ ; --- IMMOBILE --- IMMOBILE --- IMMOBILE --- IMMOBILE
```

```
Buffer1         = h’11’ ; |
Buffer2         = h’12’ ; |
Buffer3         = h’13’ ; |
Buffer4         = h’14’ ; |
Buffer5         = h’15’ ; |
Buffer6         = h’16’ ; |
Buffer7         = h’17’ ; |
Buffer8         = h’18’ ; |
Buffer9         = h’19’ ; |
BufferA         = h’1A’ ; |
BufferB         = h’1B’ ; |
;BufferC         = h’1C’ ; |
;BufferD         = h’1D’ ; |
;BufferE         = h’1E’ ; |
;BufferF         = h’1F’ ; |
Old0            = h’20’ ; |
Old1            = h’21’ ; |
Old2            = h’22’ ; |
Old3            = h’23’ ; |
Old4            = h’24’ ; |
Old5            = h’25’ ; |
Old6            = h’26’ ; |
Old7            = h’27’ ; |
Old8            = h’28’ ; |
Old9            = h’29’ ; |
OldA            = h’2A’ ; |
OldB            = h’2B’ ; |
;OldC            = h’2C’ ; |
;OldD            = h’2D’ ; |
;OldE            = h’2E’ ; |
;OldF            = h’2F’ ; |

SKIP macro
        BTFSC   PCLATH,7
 endm

        org h’0000’             ; *#*#*#* RESET VECTOR *#*#*#*
        CLRF    PCLATH
        CLRF    INTCON
        CLRF    STATUS
        GOTO    RESET_A

        org h’0004’             ; *#*#*#* INTERRUPT VECTOR *#*#*#*
        CLRF    PCLATH
        CLRF    INTCON
        CLRF    STATUS
        GOTO    RESET_A

; ***** Subroutines, Page 0

Delay07                         ;[0] Delay 7Ti
        NOP                     ; |
Delay06                         ;[0] Delay 6Ti
        NOP                     ; |
Delay05                         ;[0] Delay 5Ti
        NOP                     ; |
Delay04                         ;[0] Delay 4Ti
        RETLW   0               ; |

RS232CR                         ;[1] Transmit CR on RS232
        MOVLW   d’13’           ; |
        GOTO    RS232TxW        ; |
RS232TxDigit                    ;[1] Transmit LSnybble of W on RS232
        ANDLW   h’0F’           ; |
        MOVWF   TxByte          ; |
        MOVLW   h’0A’           ; |
        SUBWF   TxByte,W        ; |
        BTFSS   _CARRY          ; |
        GOTO    DigitLT10       ; |
DigitGE10                       ; |
        MOVLW   ‘A’-’0’-h’0A’   ; |
        ADDWF   TxByte,f        ; |
DigitLT10                       ; |
        MOVLW   ‘0’             ; |
        ADDWF   TxByte,W        ; |
RS232TxW                        ;[1] Transmit W on RS232 at 9615 baud
        MOVWF   TxByte          ; | TxByte=W
RS232Tx                         ;[1] Transmit TxByte - 104us = 9615.4 baud
        BSF     _RS232TX        ; | Stop bit
        MOVLW   d’35’           ; | |
        MOVLW   DelayReg        ; | |
RS232TxD1                       ; | |
        DECFSZ  DelayReg,f      ; | |
        GOTO    RS232TxD1       ; | |
        BCF     _RS232TX        ; | Start bit
        NOP                     ; | |
        MOVLW   d’32’           ; | |
        MOVWF   DelayReg        ; | |
RS232TxD2                       ; | |
        DECFSZ  DelayReg,f      ; | |
        GOTO    RS232TxD2       ; | |
        CLRF    BitCtr          ; | BitCtr=#8
        BSF     BitCtr,3        ; | |
RS232TxL1                       ; | {% -4Ti
        BTFSC   TxByte,0        ; |   Transmit TxByte.0, RR TxByte
        GOTO    RS232TxBit1     ; |   |
        NOP                     ; |   |
RS232TxBit0                     ; |   |
        BCF     _RS232TX        ; |   |
        BCF     _CARRY          ; |   |
        GOTO    RS232TxBitDone  ; |   |
RS232TxBit1                     ; |   |
        BSF     _RS232TX        ; |   |
        BSF     _CARRY          ; |   |
        GOTO    RS232TxBitDone  ; |   |
RS232TxBitDone                  ; |   |
        RRF     TxByte,f        ; |   |% 4Ti
        MOVLW   d’30’           ; |   delay 1 bit
        MOVWF   DelayReg        ; |   |
        GOTO    RS232TxD3       ; |   |
RS232TxD3                       ; |   |
        DECFSZ  DelayReg,f      ; |   |
        GOTO    RS232TxD3       ; |   |
        DECFSZ  BitCtr,f        ; |   DEC BitCtr
        GOTO    RS232TxL1       ; | } until (BitCtr==#0)
        CALL    Delay04         ; | delay
        BSF     _RS232TX        ; | stop bit
        RETLW   0               ; end

; ***** End of subroutines, Page 0

RESET_A
        CLRWDT
                                ; Initialise registers
        CLRF    STATUS          ; | Access register page 0
        CLRF    FSR             ; | FSR=#0
        MOVLW   StartPORTA      ; | Initialise PORT and TRIS registers
        MOVWF   PORTA           ; | |
        MOVLW   StartPORTB      ; | |
        MOVWF   PORTB           ; | |
        BSF     _RP0            ;^| |
        MOVLW   StartTRISA      ;^| |
        MOVWF   TRISA           ;^| |
        MOVLW   StartTRISB      ;^| |
        MOVWF   TRISB           ;^| |
        MOVLW   StartOPTION     ;^| Initialise OPTION register
        MOVWF   OPTION_REG      ;^| |
        BCF     _RP0            ; | |
        CLRF    Old0            ; | Clear Old buffer
        CLRF    Old1            ; | |
        CLRF    Old2            ; | |
        CLRF    Old3            ; | |
        CLRF    Old4            ; | |
        CLRF    Old5            ; | |
        CLRF    Old6            ; | |
        CLRF    Old7            ; | |
        CLRF    Old8            ; | |
        CLRF    Old9            ; | |
        CLRF    OldA            ; | |
        CLRF    OldB            ; | |

BigLoop1
;303-581-1041
        BSF     _LED1           ; LEDs “reading”
        CALL    Delay07         ; |
        BCF     _LED2           ; |
        MOVLW   h’09’           ; Transmit TAB regularly
        CALL    RS232TxW        ; |
        MOVLW   d’96’           ; set BitCtr
        MOVWF   BitCtr          ; |
GetEdge                         ; Get an edge on _DATA_IN
        BTFSC   _DATA_IN        ; |
        GOTO    PreSync_H       ; |
        NOP                     ; |
PreSync_L                       ; |
        BTFSC   _DATA_IN        ; |
        GOTO    PreSync_H       ; |
        BTFSC   _DATA_IN        ; |
        GOTO    PreSync_H       ; |
DoSync_L                        ; |
        CLRWDT                  ; |
        BTFSS   _DATA_IN        ; |
        GOTO    DoSync_L        ; |
        BTFSS   _DATA_IN        ; |
        GOTO    DoSync_L        ; |
        GOTO    Sync_Done       ; |
                                ; |
PreSync_H                       ; |
        BTFSS   _DATA_IN        ; |
        GOTO    PreSync_L       ; |
        BTFSS   _DATA_IN        ; |
        GOTO    PreSync_L       ; |
DoSync_H                        ; |
        CLRWDT                  ; |
        BTFSC   _DATA_IN        ; |
        GOTO    DoSync_H        ; |
        BTFSC   _DATA_IN        ; |
        GOTO    DoSync_H        ; |
        GOTO    Sync_Done       ; |
Sync_Done                       ; |% 6 to (+4) from edge, say 8 from edge
        ;% -192Ti from sample
        MOVLW   d’62’
        MOVWF   DelayReg
        ;% -190Ti from sample
ReadBit                         ; {% -4-DelayReg*3 Ti from sample
        GOTO    ReadBitD1       ;   delay
ReadBitD1                       ;   |
        DECFSZ  DelayReg,f      ;   |
        GOTO    ReadBitD1       ;   |
        CLRF    BO3             ;   BO3.1=_DATA_IN
        BTFSC   _DATA_IN        ;   |
        INCF    BO3,f           ;   |% effective sample time
        BTFSC   _DATA_IN        ;   |
        INCF    BO3,f           ;   |
        BTFSC   _DATA_IN        ;   |
        INCF    BO3,f           ;   |
        BCF     _CARRY          ;   _CARRY=BO3.1
        BTFSC   BO3,1           ;   |
        BSF     _CARRY          ;   |
        RLF     Buffer0,f       ;   roll in _CARRY
        RLF     Buffer1,f       ;   |
        RLF     Buffer2,f       ;   |
        RLF     Buffer3,f       ;   |
        RLF     Buffer4,f       ;   |
        RLF     Buffer5,f       ;   |
        RLF     Buffer6,f       ;   |
        RLF     Buffer7,f       ;   |
        RLF     Buffer8,f       ;   |
        RLF     Buffer9,f       ;   |
        RLF     BufferA,f       ;   |
        RLF     BufferB,f       ;   |
                                ;  % 19Ti from sample = -381Ti from sample
        MOVLW   d’124’          ;   set bit delay
        MOVWF   DelayReg        ;   |% -379Ti from sample
        ;% -7-DelayReg*3 Ti from sample
        DECFSZ  BitCtr,f        ;   DEC BitCtr
        GOTO    ReadBit         ; } until (BitCtr==#0)

HeadSearch
        MOVLW   d’96’           ; set BitCtr
        MOVWF   BitCtr          ; |
HeadSearchL1                    ; {
        MOVLW   h’80’           ;   if (header found)
        XORWF   BufferB,W       ;   |
        BTFSS   _ZERO           ;   |
        GOTO    NotHead0        ;   |
        MOVLW   h’2A’           ;   |
        XORWF   BufferA,W       ;   |
        BTFSS   _ZERO           ;   |
        GOTO    NotHead0        ;   {
        GOTO    HeadFound       ;     goto HeadFound
NotHead0                        ;   }
        RLF     Buffer0,f       ;   ROL Buffer
        RLF     Buffer1,f       ;   |
        RLF     Buffer2,f       ;   |
        RLF     Buffer3,f       ;   |
        RLF     Buffer4,f       ;   |
        RLF     Buffer5,f       ;   |
        RLF     Buffer6,f       ;   |
        RLF     Buffer7,f       ;   |
        RLF     Buffer8,f       ;   |
        RLF     Buffer9,f       ;   |
        RLF     BufferA,f       ;   |
        RLF     BufferB,f       ;   |
        BCF     Buffer0,0       ;   |
        BTFSC   _CARRY          ;   |
        BSF     Buffer0,0       ;   |
        DECFSZ  BitCtr,f        ;   DEC BitCtr
        GOTO    HeadSearchL1    ; } until (BitCtr==#0)
        GOTO    BigLoop1        ; goto BigLoop1

HeadFound

CheckSame
```

```
        MOVF    Buffer0,W
        XORWF   Old0,W
        BTFSS   _ZERO
        GOTO    NotSame
        MOVF    Buffer1,W
        XORWF   Old1,W
        BTFSS   _ZERO
        GOTO    NotSame
        MOVF    Buffer2,W
        XORWF   Old2,W
        BTFSS   _ZERO
        GOTO    NotSame
        MOVF    Buffer3,W
        XORWF   Old3,W
        BTFSS   _ZERO
        GOTO    NotSame
        MOVF    Buffer4,W
        XORWF   Old4,W
        BTFSS   _ZERO
        GOTO    NotSame
        MOVF    Buffer5,W
        XORWF   Old5,W
        BTFSS   _ZERO
        GOTO    NotSame
        MOVF    Buffer6,W
        XORWF   Old6,W
        BTFSS   _ZERO
        GOTO    NotSame
        MOVF    Buffer7,W
        XORWF   Old7,W
        BTFSS   _ZERO
        GOTO    NotSame
        MOVF    Buffer8,W
        XORWF   Old8,W
        BTFSS   _ZERO
        GOTO    NotSame
        MOVF    Buffer9,W
        XORWF   Old9,W
        BTFSS   _ZERO
        GOTO    NotSame
        MOVF    BufferA,W
        XORWF   OldA,W
        BTFSS   _ZERO
        GOTO    NotSame
        MOVF    BufferB,W
        XORWF   OldB,W
        BTFSS   _ZERO
        GOTO    NotSame
        GOTO    Same
NotSame
        MOVF    Buffer0,W
        MOVWF   Old0
        MOVF    Buffer1,W
        MOVWF   Old1
        MOVF    Buffer2,W
        MOVWF   Old2
        MOVF    Buffer3,W
        MOVWF   Old3
        MOVF    Buffer4,W
        MOVWF   Old4
        MOVF    Buffer5,W
        MOVWF   Old5
        MOVF    Buffer6,W
        MOVWF   Old6
        MOVF    Buffer7,W
        MOVWF   Old7
        MOVF    Buffer8,W
        MOVWF   Old8
        MOVF    Buffer9,W
        MOVWF   Old9
        MOVF    BufferA,W
        MOVWF   OldA
        MOVF    BufferB,W
        MOVWF   OldB
        GOTO    BigLoop1
Same

TxTag                           ;- Transmit tag
        BSF     _LED2           ; LEDs “Found tag”
        CALL    Delay07         ; |
        BCF     _LED1           ; |
        MOVLW   d’4’            ; Beep at 3597Hz for 1024 cycles
        MOVWF   BeepCtrHi       ; |
        MOVLW   d’0’            ; |
        MOVWF   BeepCtrLo       ; |
BeepLoopJ1                      ; |
        GOTO    BeepLoopJ2      ; |
BeepLoopJ2                      ; |
        MOVLW   Beep1           ; |
        MOVWF   BeepPort        ; |
        MOVLW   d’34’           ; |
        MOVWF   DelayReg        ; |
BeepD1                          ; |
        CLRWDT                  ; |
        DECFSZ  DelayReg,f      ; |
        GOTO    BeepD1          ; |
        MOVLW   Beep2           ; |
        MOVWF   BeepPort        ; |
        MOVLW   d’32’           ; |
        MOVWF   DelayReg        ; |
        NOP                     ; |
        GOTO    BeepD2          ; |
BeepD2                          ; |
        CLRWDT                  ; |
        DECFSZ  DelayReg,f      ; |
        GOTO    BeepD2          ; |
        DECFSZ  BeepCtrLo,f     ; |
        GOTO    BeepLoopJ1      ; |
        DECFSZ  BeepCtrHi,f     ; |
        GOTO    BeepLoopJ2      ; |
        NOP                     ; |
        MOVLW   Beep0           ; |
        MOVWF   BeepPort        ; |

        CALL    RS232CR         ; Transmit tag info
        MOVLW   ‘F’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘S’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘K’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘ ‘             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘/’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘8’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘-’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘/’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘1’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘0’             ; |
        CALL    RS232TxW        ; |
        CALL    RS232CR         ; |
        MOVLW   ‘T’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘b’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘i’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘t’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘=’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘5’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘0’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘T’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘c’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘y’             ; |
        CALL    RS232TxW        ; |
        CALL    RS232CR         ; |
        MOVLW   ‘C’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘o’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘n’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘s’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘t’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘a’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘n’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘t’             ; |
        CALL    RS232TxW        ; |
        CALL    RS232CR         ; |
        MOVLW   ‘T’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘t’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘a’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘g’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘=’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘9’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘6’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘T’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘b’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘i’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘t’             ; |
        CALL    RS232TxW        ; |
        CALL    RS232CR         ; |
        MOVLW   ‘P’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘o’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘l’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘a’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘r’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘i’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘t’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘y’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘ ‘             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘0’             ; |
        CALL    RS232TxW        ; |
        CALL    RS232CR         ; |

        MOVLW   BufferB         ; Transmit tag ID
        MOVWF   FSR             ; |
TxLoop1                         ; |
        SWAPF   INDF,W          ; |
        CALL    RS232TxDigit    ; |
        MOVF    INDF,W          ; |
        CALL    RS232TxDigit    ; |
        DECF    FSR,f           ; |
        BTFSC   FSR,4           ; |
        GOTO    TxLoop1         ; |
        CALL    RS232CR         ; |

        GOTO    BigLoop1        ; goto BigLoop1

        end
```

NOTES:

PSK Reader Reference Design

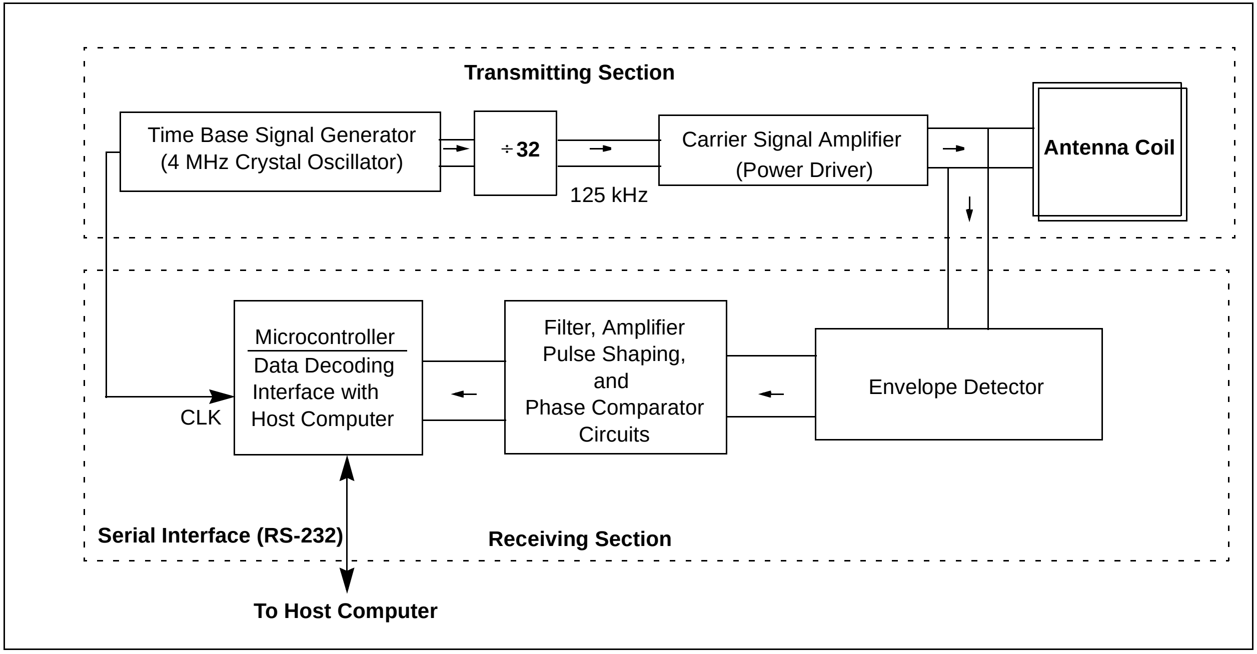
1.0 INTRODUCTION

This application note is written as a reference guide for PSK reader designers. Microchip Technology Inc. provides basic reader schematic for the MCRF200 customers as a part of this design guide. The circuit is designed for a read range of 3 ~ 5 inches with an access control card. The microID PSK Reader (demo unit), which is built based on the PSK reference design, is available in the microID Designers Kit (DV103001). The circuit can be modified for longer read range or other applications with the MCRF200. An electronic copy of the PSK microID PICmicro® source code is available upon request.

2.0 READER CIRCUITS

The RFID reader consists of transmitting and receiving sections. It transmits a carrier signal, receives the backscattering signal, and performs data processing. The reader also communicates with an external host computer. A basic block diagram of the typical RFID reader is shown in Figure 2-1.

FIGURE 2-1: BLOCK DIAGRAM OF TYPICAL RFID READER FOR PSK SIGNAL (125 kHz)


Transmitting Section
Time Base Signal Generator (4 MHz Crystal Oscillator)
÷32
125 kHz
Carrier Signal Amplifier (Power Driver)
Antenna Coil
Microcontroller
Data Decoding Interface with Host Computer
CLK
Filter, Amplifier Pulse Shaping, and Phase Comparator Circuits
Envelope Detector
Serial Interface (RS-232)
Receiving Section
To Host Computer

PICmicro is a registered trademark of Microchip Technology Inc.

2.1 Transmitting Section

The transmitting section contains circuitry for a carrier signal (125 kHz), power amplifiers, and a tuned antenna coil.

The 125 kHz carrier signal is typically generated by dividing a 4 MHz (4 MHz/32 = 125 kHz) crystal oscillator signal. The signal is amplified before it is fed into the antenna tuning circuit. A complementary power amplifier circuit is typically used to boost the transmitting signal level.

An antenna impedance tuning circuit consisting of capacitors is used to maximize the signal level at the carrier frequency. This tuning circuit is needed to form an exact LC resonant circuit for the carrier signal. The tuning compensates the variations in the component values and the perturbation of coil inductance due to environment effect. A design guide for the antenna coil is given in *AN678, RFID Coil Design,* page 25.

2.1.1 LIMITS ON TRANSMITTING SIGNAL LEVEL (FCC PART 15) IN THE USA

Each country limits the signal strength of the RF wave that is intentionally radiated by a device. In the USA, the signal strength of the carrier signal (125 kHz) radiating from the antenna coil must comply with the FCC (Federal Communications Commission) part 15 regulation. The signal level is specified by the 47 CFR Part 15.209a of the federal regulation. For a 125 kHz signal, the FCC limits the signal level to 19.2 µV per meter, or 25.66 dBµV (i.e., 20 log(19.2) = 25.66 dBµV), at 300 meters away from the antenna. For a close distance measurement, an extrapolation rule (40 dB per decade) is applied (Part 15.31.f.2). For example, the signal level at 30 meters away from the device must not exceed:

$$25.66\ \text{dB}\mu\text{V} + 40\ \text{dB}\mu\text{V} = 65.66\ \text{dB}\mu\text{V}$$

2.2 Receiving Section

The receiving section consists of an antenna coil, demodulator, filter, amplifier, pulse shaping, phase comparator, and microcontroller. In applications for proximity read-range, a single coil is often used for both transmitting and receiving. For long read range application, however, separated antennas may be used. More details on the antenna coil are given in *AN678, RFID Coil Design,* page 25.

In the PSK communication protocol, the phase of the modulation signal changes with the data. Two most common types of phase encoding method are: (a) change phase at any data change (‘0’ to ‘1’ or ‘1’ to ‘0’), and (b) change phase at ‘1’. A typical data rate for PSK applications is one half of the carrier frequency, and it is faster than FSK. However, it requires a wider bandwidth than FSK.

The PSK reader needs two steps for a full recovery of the data. The first step is demodulating the backscattering signal, and the second step is detecting the phase changes in the demodulation signal.

The demodulation is accomplished by detecting the envelope of the carrier signal. A full-wave capacitor-filtered rectifier circuit is used for the demodulation process. A diode detects the peak voltage of the backscattering signal. The voltage is then fed into an RC charging/discharging circuit. The RC time constant must be small enough to allow the voltage across C to fall fast enough to keep in step with the envelope. However, the time constant must not be so small as to introduce excessive ripple. The demodulated signal must then pass through a filter, an amplifier, signal shaping, and phase comparator circuits before it is fed to the microcontroller. The microcontroller performs data decoding and communicates with the host computer through an RS-232 or other serial interface protocols.

3.0 microID PSK READER

The MCRF200 can be configured with either PSK_1 or PSK_2 modulation. The PSK_1 changes the phase of the modulation signal on any change of the data (i.e., 0 to 1 or 1 to 0). The PSK_2 changes the phase of the modulation signal on the first clock edge of a data '1'. Figure 3-1 shows the optional PSK encoding protocols. The PSK encoded data is amplitude modulating the carrier signal. A typical PSK modulated signal is shown in Figure 3 in AN680, *Passive RFID Basics* page 15.

This reference reader was designed for use with an MCRF200 with 08Dh in its configuration register, which represents PSK_1, NRZ Direct, Fc/32, data rate, and 128 bits.

The electronic circuitry for the PSK reader is shown in Figure 3-1. The reader needs +9 to +15 VDC power supply. The 125 kHz carrier signal is generated by dividing the 4 MHz time-base signal that is generated by a crystal oscillator. A 16-stage binary ripple counter (74HC4060) is used for this purpose. The 74HC4060 also provides a clock signal for the PIC16C84 microcontroller. Signal from the U8 is also used as a phase reference for receiving signals.

The 125 kHz signal is passed to an RF choke (L1) and filter before it is fed into a power amplifier that is formed by a pair of complementary bipolar transistors (Q2 and Q3).

For long read-range applications, this power amplifier circuit can be modified. Power MOSFETs may be used instead of bipolar transistors (2N2222). These power MOSFETs can be driven by +24 VDC power supply. A push-pull predriver can be added at the front of the complementary circuit. This modification will enhance the signal level of the carrier signal.

The reader circuit uses a single coil for both transmitting and receiving signals. An antenna coil (L2: 1.62 mH) and a resonant capacitor (C21: 1000 pF) forms a series resonant circuit for 125 kHz resonance frequency. Since the C21 is grounded, the carrier signal (125 kHz) is filtered out to the ground after passing the antenna coil. The circuit provides minimum impedance at the resonance frequency. This results in maximizing the antenna current, and therefore, the magnetic field strength is maximized.

In the circuit, D7 and D8 are amplitude demodulators that are detecting the envelope of the backscattering signal. D7 provides a current path during a positive half cycle and the D8 during the negative half cycle. The detected envelope signal is charged into the C27. A discharge path for the voltage charged in the C27 is provided by R33. This voltage passes active filters (U11:C) and the pulse shaping circuitry (U11:A).

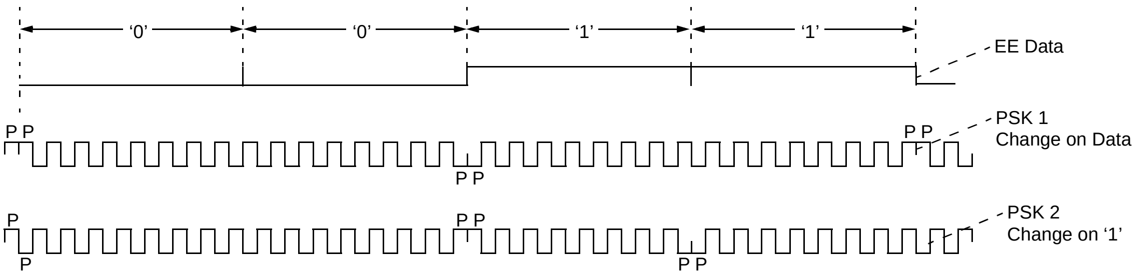
The output from the U11 is a square wave at 62.5 kHz, which exhibits 180 degree phase-shifts in accordance with changes in the data stream from the tag. This signal is used as a clock for D flip-flop (U6:A) for which the D input is a reference 62.5 kHz square wave derived from the 125 kHz transmitting signal. As the phase of the received signal changes, the output of the flip-flop changes, based on whether the clocking occurs during the high or low portions of the reference signal. The recovered data signal is fed to the input I/O pin of the PICmicro MCU (U7) for decoding.

One of the major problems encountered with the PSK reader is that the phase of the returned signal with respect to a reference signal is, for several reasons, indeterminate. If the transitions of the incoming signal and the reference are occurring at the same time, the output of the D flip-flop will be unpredictable. To guarantee that this does not happen, additional circuits have been added.

The received 62.5 kHz signal is buffered by U9:D and a pulse is generated upon every transition of the received signal by U4:C. Likewise, U4:B provides a string of pulses on every transition of the reference 62.5 kHz signal. Note that these pulse strings are at 125 kHz and are independent of the phase state of the received signal.

These pulses are fed to the set and reset lines of U5:A and result in a 125 kHz output at $\overline{Q}$ whose duty cycle is proportional to the phase difference between the two pulse signals. If the duty cycle is near 50%, then the transitions of the 62.5 kHz signals are approximately 90 degrees different which is ideal for PSK demodulation.

FIGURE 3-1: PSK DATA MODULATION



R6 and C10 filter the output of U5:A resulting in a DC level proportional to the phase shift. This level is the input to a window detector consisting of U10 and U4:A. If the DC level is near the midpoint, the output of comparator U10:B would be high and the output of comparator U10:A would be low. Therefore, the output of U4:A would be high. If the DC level is higher than the reference level set by R21, R26, and R30 then the outputs of both comparators would be high, resulting in a low output from U4:A. Similarly, if the DC level is low, both outputs would be low, which would also result in a low output at U4:A.

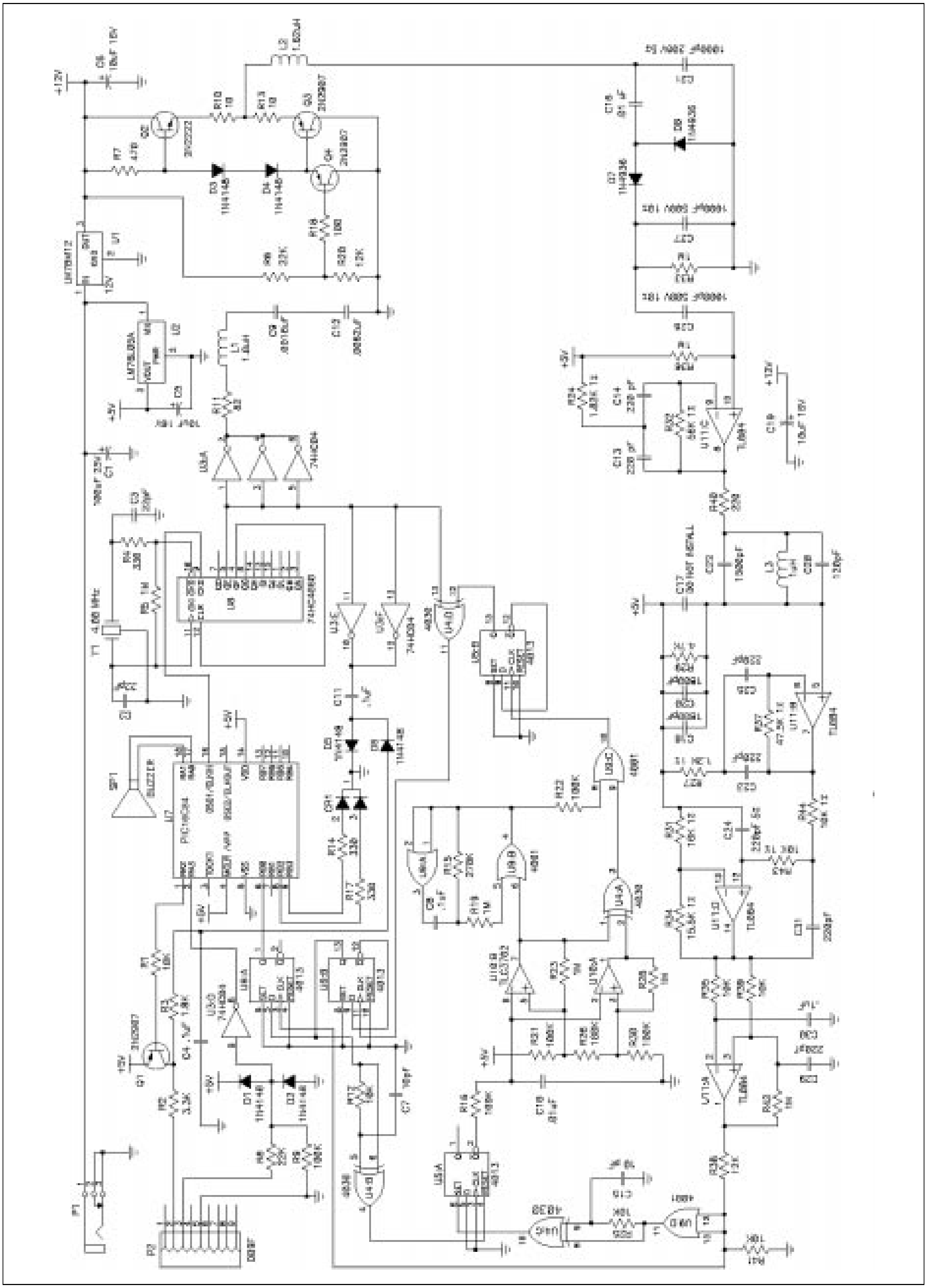
Note that the 125 kHz signal from which the 62.5 kHz reference is obtained passes through gate U4:D. A change of the state on the control output to this gate allows the 125 kHz signal to be 180 degree phase-shifted. This results in a phase-shift in the 62.5 kHz reference of 90 degrees. If the output of the U9:C is low, the flip-flop U5:B will maintain its current state.

If the output of U4:A goes low, which would signify an undesirable phase relationship between the 62.5 kHz signals, then the output of U9:C would have a transition to high, causing U5:B to change state. This would change the reference phase 90 degrees, thus bringing the phases of the 62.5 kHz signals back into a desirable relationship and return the output of U4:A to a high state.

In the event that no tag is present, $\overline{Q}$ of U5:A is always high which makes the output of U10:B low. This turns on an oscillator consisting of U9:A, U9:B, C8, R15, and R19. This oscillator toggles U5:B at about 200 Hz, allowing the reader to be looking for a tag signal with both reference signal phases. When a good tag signal appears, the circuit locks on in a good phase relationship and demodulates the incoming 62.5 kHz signal. As the tag comes closer to the reader, the phase will be shift for a number of reasons. If the shift is sufficient, the reference signal will shift as necessary to maintain good demodulation.

The PIC16C84 microcontroller performs data decoding and communicates with host computer via an RS-232 serial interface.

4.0 PSK READER SCHEMATIC



5.0 PSK READER BILL OF MATERIALS

Item #	Qty	Part #	Reference Designator	Part Description	Manufacturer	Vendor	Vendor Part #
1	1	110-93-314-41-001	xU6	SOCKET, 14P COLLET OPEN FRAME (0.300W)	MILL-MAX	DIGIKEY	ED3314-ND
2	1	DE9S-FRS	P2	CONN, D-SUB 9P RECPT RT ANGLE	SPC TECHNOLOGY		
3	1	DJ005B	P1	JACK, POWER, 2.5mm DC PC MOUNT	LZR ELECTRONICS		
4	1	PKM22EPP-4001	SP1	BUZZER, PIEZO, 4KHz, 3-20V	MURATA		
5	2	D100D20U2MHAAAC	C7, C15	CAP, 10 pF CER DISK RAD, 100V	PHILIPS	DIGIKEY	1301PH-ND
6	2	D220J20COGHAAAC	C2, C3	CAP, 22 pF CER DISK RAD COG 100V	PHILIPS	DIGIKEY	1330PH-ND
7	7	ECU-S1H221JCA	C13, C14, C23-C25, C29, C31	CAP, 220pF, CER MONO, RAD, 50V, 5%	PANASONIC	DIGIKEY	P4929-ND
8	1	ECQ-P6102JU	C21	CAP, 0.001 µF POLYPROPYLENE 630V	PHILIPS	DIGIKEY	P3497-ND
9	2	2222 370 52102	C26, C27	CAP, 0.001 µF METAL FILM, 5%, RAD, 400V	PHILIPS	DIGIKEY	3001PH-ND
10	1	ECU-S2A152JCB	C22	CAP, 1500 pF MONOLITH CERM, 5%, RAD, 100V	PHILIPS	DIGIKEY	P4863-ND
11	3	ECU-S2A182JCB	C9, C18, C20	CAP, 1800 pF MONOLITH CERM, 5%, RAD, 100V	PHILIPS	DIGIKEY	P4864-ND
12	1	ECU-S1H682JCB	C12	CAP, 6800 pF 50V CERAMIC MONO 5%	PANASONIC	DIGIKEY	P4946-ND
13	2	ECK-F1H103ZF	C8, C10	CAP, 0.01 µF CERM DISK, +80/-20%, RAD, 50V	PHILIPS	DIGIKEY	P4066A-ND
14	1	ECQ-V1103JM	C16	CAP, 0.01 µF 100V STACK METAL FILM	PANASONIC	DIGIKEY	P4713-ND
15	3	ECQ-E1104KF	C4, C11, C30	CAP, 0.1 µUF 100VDC 10% RAD METAL POLY CAP	PANASONIC	DIGIKEY	EF1104-ND
16	1	ECU-S1H121JCA	C28	CAP, 120 pF, CER MONO, RAD, 50V, 5%	PANASONIC	DIGIKEY	P4926-ND
17	3	ECE-A16Z10	C5, C6, C19	CAP, 10 µF, ELECTRO, RAD, 16V, 20%	PANASONIC	DIGIKEY	P6616-ND
18	1	ECE-A25Z100	C1	CAP, 100 µF, ELECTRO, RAD, 25V, 20%	PANASONIC	DIGIKEY	P6616-ND
19	6	1N4148	D1-D6	DIODE, GENERAL PURPOSE, 1N4148 (DO-35)	DIODES INC.	DIGIKEY	1N4148DITR-ND
20	2	1N4936	D7, D8	DIODE, 1A 400V FAST-RECOVERY RECTIFIER	DIODES INC	DIGIKEY	1N4936CT-ND
21	1	-SPARE-	LED1, C17	-SPARE- LOCATION DO NOT INSTALL			
22	2	78F102J	L1, L3	INDUCTOR, 1000 µH, COATED	JW MILLER	DIGIKEY	M7849-ND
23	1	MCT0003-001	L2	INDUCTOR, 1.62 mH	CORNELL DUBILIER		

Item #	Qty	Part #	Reference Designator	Part Description	Manufacturer	Vendor	Vendor Part #
24	3	2N2907A	Q1, Q3, Q4	TRANSISTOR, PNP, 2N2907A, TO-92	MOTOROLA		
25	1	2N2222A	Q2	TRANSISTOR, NPN, 2N2222A, TO-92	MOTOROLA	ALLIED	2N2222A
26	2	5043CX10R0J	R10, R13	RES, CF 10 OHM 1/4W 5%	PHILLIPS		
27	1	82E CR-1/4W-B 5%	R11	RES, CF 82 OHM 1/4W 5%	YAGEO	DIGIKEY	82QBK-ND
28	1	5043CX100R0J	R18	RES, CF 100 OHM 1/4W 5%	PHILLIPS		
29	1	5043CX220R0J	R40	RES, CF 220 OHM 5% 1/4W	PHILLIPS		
30	3	5043CX330R0J	R4, R14, R17	RES, CF 330 OHM 1/4W 5%	PHILLIPS		
31	1	5043CX470R0J	R7	RES, CF 470 OHM 5% 1/4W	PHILLIPS		
32	1	1K21 MF-1/4W-B 1%	R27	RES, MF 1.21K OHM 1/4W 1%	YAGEO	DIGIKEY	1.21KXBK-ND
33	1	1K8 CR-1/4W-B 5%	R3	RES, CF 1.8K OHM 1/4W 5%	YAGEO	DIGIKEY	1.8KQBK-ND
34	1	1K82 MF-1/4W-B 1%	R24	RES, MF 1.82K OHM 1/4W 1%	YAGEO	DIGIKEY	1.82KXBK-ND
35	1	3K3 CR-1/4W-B 5%	R2	RES, CF 3.3K OHM 1/4W 5%	YAGEO	DIGIKEY	3.3KQBK-ND
36	1	5043CX4K700J	R29	RES, CF 4.7K 5% 1/4W, AXIAL	PHILLIPS		
37	6	10K CR-1/4W-B 5%	R1, R12, R25, R35, R39, R41	RES, CF 10K OHM 1/4W 5%	YAGEO	DIGIKEY	10KQBK-ND
38	3	5043ED10K00F	R31, R43, R44	RES, MF 10K 1/4W 1%	PHILLIPS		
39	2	12K CR-1/4W-B 5%	R20, R38	RES, CF 12K OHM 1/4W 5%	YAGEO	DIGIKEY	12KQBK-ND
40	1	16K5 MF-1/4W-B 1%	R34	RES, MF 16.5K OHM 1/4W 1%	YAGEO	DIGIKEY	16.5KXBK-ND
41	2	22K CR-1/4W-B 5%	R6, R8	RES, CF 22K OHM 1/4W 5%	YAGEO	DIGIKEY	22KQBK-ND
42	1	47K5 MF-1/4W-B 1%	R37	RES, MF 47.5K OHM 1/4W 1%	YAGEO	DIGIKEY	47.5KXBK-ND
43	1	56K CR-1/4W-B 5%	R32	RES, CF 56K OHM 1/4W 5%	YAGEO	DIGIKEY	56KQBK-ND
44	5	5043CX100K0J	R9, R16, R21, R22, R30	RES, CF 100K 5% 1/4W	PHILLIPS		
45	1	180K CR-1/4W-B 5%	R26	RES, CF 180K OHM 1/4W 5%	YAGEO	DIGIKEY	180KQBK-ND
46	1	270K CR-1/4W-B 5%	R15	RES, CF 270K OHM 1/4W 5%	YAGEO	DIGIKEY	270KQBK-ND
47	7	1M0 CR-1/4W-B 5%	R5, R19, R23, R28, R33, R36, R42	RES, CF 1.0M OHM 1/4W 5%	YAGEO	DIGIKEY	1.0MQBK-ND
48	1	LM78M12CT	U1	IC, REG 12V 3 TERM POS (TO-220)	NATIONAL	DIGIKEY	LM78M12CT-ND
49	1	LM78L05ACZ	U2	IC, REG, +5V 0.1 A TO-92	NATIONAL	DIGIKEY	LM78L05ACZ-ND

Item #	Qty	Part #	Reference Designator	Part Description	Manufacturer	Vendor	Vendor Part #
50	1	MM74HC04N	U3	IC, HEX INVERTER 14P DIP	FAIRCHILD SEMICONDUCTOR	DIGIKEY	MM74HC04N-ND
51	1	CD4030CN	U4	IC, QUAD EXCLUSIVE OR GATE, 14P DIP	FAIRCHILD SEMICONDUCTOR	DIGIKEY	CD4030CN-ND
52	2	CD4013BCN	U5, U6	IC, DUAL D FLIP FLOP, 14P DIP	FAIRCHILD SEMICONDUCTOR	DIGIKEY	CD4013BCN-ND
53	1	PIC16C84/P	U7	IC, PIC16C84 PLASTIC, 14P DIP	MICROCHIP		
54	1	MM74HC4060N	U8	IC, 14 STAGE BINARY COUNTER, 16P DIP	FAIRCHILD SEMICONDUCTOR	DIGIKEY	MM74HC4060N-ND
55	1	CD4001BCN	U9	IC, QUAD 2-IN NOR GATE, 14P DIP	FAIRCHILD SEMICONDUCTOR	DIGIKEY	CD4001BCN-ND
56	1	TLC3702CP	U10	IC, DUAL VOLTAGE COMPARATORS, 1000mW, 8P DIP	TEXAS INSTRUMENTS	MOUSER	TLC3702CP
57	1	TL084CN	U11	IC, QUAD OP AMP, 14P DIP	SGS THOMPSON	MOUSER	511-TL084CN
58	1	EFO-EC4004A4	Y1	RESONATOR, 4.00MHZ CERAMIC W/CAP	PANASONIC	DIGIKEY	PX400-ND

6.0 PSK SOURCE CODE FOR THE PICMICRO® MCU

The following source code is for the PIC16C84 microcontroller used in the PSK reader electronics.

```
; #=#=#=#=#=#=#=#=#=#=#= PROJECT Microchip PSK Reader =#=#=#=#=#=#=#=#=#=#=#
;
; PIC16C84 running at 4MHz, Ti=1us

; ///////////////////////////////////////////////////////////////////////////
; Revision history
; ///////////////////////////////////////////////////////////////////////////
;
; Ver    Date        Comment
;
; 0.01   29 Dec 97   Copied from MChip\Reader\PSK
; 0.03   28 Jan 98   TRANSMIT TAB (h’09’) REGULARLY
;        20 Aug 98   Modified to correct PSK comments
;
;       Tbit=32Tcy=256Ti
;       Ttag=128Tbit
;       Header=h’802A’
;
;


    processor pic16c84
    #include “p16c84.inc”
        __config b’11111111101001’
        ; Code Protect on, power-up timer on, WDT off, XT oscillator

#define _CARRY          STATUS,0
#define _ZERO           STATUS,2
#define _TO             STATUS,4
#define _RP0            STATUS,5

#define _BUZZ1          PORTA,0
#define _BUZZ2          PORTA,1
#define _RS232TX        PORTA,2
#define _RS232RX        PORTA,3
#define _T0CKI          PORTA,4
StartPORTA      = b’01100’
StartTRISA      = b’11000’
BeepPort        = PORTA
Beep0           = StartPORTA
Beep1           = StartPORTA | b’00001’
Beep2           = StartPORTA | b’00010’

#define _DATA_IN        PORTB,0
#define _UNUSED1        PORTB,1
#define _LED1           PORTB,2
#define _LED2           PORTB,3
#define _UNUSED2        PORTB,4
#define _UNUSED3        PORTB,5
#define _UNUSED4        PORTB,6
#define _UNUSED5        PORTB,7
StartPORTB      = b’00000000’
StartTRISB      = b’00000001’

StartOPTION     = b’00001111’   ; TMR0 internal, prescaler off

BO3             = h’0C’
DelayReg        = h’0C’
BitCtr          = h’0D’
BeepCtrHi       = h’0D’
TxByte          = h’0E’
BeepCtrLo       = h’0E’

Buffer0         = h’10’ ; --- IMMOBILE --- IMMOBILE --- IMMOBILE --- IMMOBILE
```

```
Buffer1         = h’11’ ; |
Buffer2         = h’12’ ; |
Buffer3         = h’13’ ; |
Buffer4         = h’14’ ; |
Buffer5         = h’15’ ; |
Buffer6         = h’16’ ; |
Buffer7         = h’17’ ; |
Buffer8         = h’18’ ; |
Buffer9         = h’19’ ; |
BufferA         = h’1A’ ; |
BufferB         = h’1B’ ; |
BufferC         = h’1C’ ; |
BufferD         = h’1D’ ; |
BufferE         = h’1E’ ; |
BufferF         = h’1F’ ; |
Old0            = h’20’ ; |
Old1            = h’21’ ; |
Old2            = h’22’ ; |
Old3            = h’23’ ; |
Old4            = h’24’ ; |
Old5            = h’25’ ; |
Old6            = h’26’ ; |
Old7            = h’27’ ; |
Old8            = h’28’ ; |
Old9            = h’29’ ; |
OldA            = h’2A’ ; |
OldB            = h’2B’ ; |
OldC            = h’2C’ ; |
OldD            = h’2D’ ; |
OldE            = h’2E’ ; |
OldF            = h’2F’ ; |

SKIP macro
        BTFSC   PCLATH,7
 endm

        org h’0000’             ; *#*#*# RESET VECTOR *#*#*#
        CLRF    PCLATH
        CLRF    INTCON
        CLRF    STATUS
        GOTO    RESET_A

        org h’0004’             ; *#*#*# INTERRUPT VECTOR *#*#*#
        CLRF    PCLATH
        CLRF    INTCON
        CLRF    STATUS
        GOTO    RESET_A

; ***** Subroutines, Page 0

Delay07                         ;[0] Delay 7Ti
        NOP                     ; |
Delay06                         ;[0] Delay 6Ti
        NOP                     ; |
Delay05                         ;[0] Delay 5Ti
        NOP                     ; |
Delay04                         ;[0] Delay 4Ti
        RETLW   0               ; |

RS232CR                         ;[1] Transmit CR on RS232
        MOVLW   d’13’           ; |
        GOTO    RS232TxW        ; |
RS232TxDigit                    ;[1] Transmit LSnybble of W on RS232
        ANDLW   h’0F’           ; |
        MOVWF   TxByte          ; |
        MOVLW   h’0A’           ; |
        SUBWF   TxByte,W        ; |
        BTFSS   _CARRY          ; |
        GOTO    DigitLT10       ; |
DigitGE10                       ; |
        MOVLW   ‘A’-’0’-h’0A’   ; |
        ADDWF   TxByte,f        ; |
DigitLT10                       ; |
        MOVLW   ‘0’             ; |
        ADDWF   TxByte,W        ; |
RS232TxW                        ;[1] Transmit W on RS232 at 9615 baud
        MOVWF   TxByte          ; | TxByte=W
RS232Tx                         ;[1] Transmit TxByte - 104us = 9615.4 baud
        BSF     _RS232TX        ; | Stop bit
        MOVLW   d’35’           ; | |
        MOVLW   DelayReg        ; | |
RS232TxD1                       ; | |
        DECFSZ  DelayReg,f      ; | |
        GOTO    RS232TxD1       ; | |
        BCF     _RS232TX        ; | Start bit
        NOP                     ; | |
        MOVLW   d’32’           ; | |
        MOVWF   DelayReg        ; | |
RS232TxD2                       ; | |
        DECFSZ  DelayReg,f      ; | |
        GOTO    RS232TxD2       ; | |
        CLRF    BitCtr          ; | BitCtr=#8
        BSF     BitCtr,3        ; | |
RS232TxL1                       ; | {% -4Ti
        BTFSC   TxByte,0        ; |   Transmit TxByte.0, RR TxByte
        GOTO    RS232TxBit1     ; |   |
        NOP                     ; |   |
RS232TxBit0                     ; |   |
        BCF     _RS232TX        ; |   |
        BCF     _CARRY          ; |   |
        GOTO    RS232TxBitDone  ; |   |
RS232TxBit1                     ; |   |
        BSF     _RS232TX        ; |   |
        BSF     _CARRY          ; |   |
        GOTO    RS232TxBitDone  ; |   |
RS232TxBitDone                  ; |   |
        RRF     TxByte,f        ; |   |% 4Ti
        MOVLW   d’30’           ; |   delay 1 bit
        MOVWF   DelayReg        ; |   |
        GOTO    RS232TxD3       ; |   |
RS232TxD3                       ; |   |
        DECFSZ  DelayReg,f      ; |   |
        GOTO    RS232TxD3       ; |   |
        DECFSZ  BitCtr,f        ; |   DEC BitCtr
        GOTO    RS232TxL1       ; | } until (BitCtr==#0)
        CALL    Delay04         ; | delay
        BSF     _RS232TX        ; | stop bit
        RETLW   0               ; end

; ***** End of subroutines, Page 0

RESET_A
        CLRWDT
                                ; Initialise registers
        CLRF    STATUS          ; | Access register page 0
        CLRF    FSR             ; | FSR=#0
        MOVLW   StartPORTA      ; | Initialise PORT and TRIS registers
        MOVWF   PORTA           ; | |
        MOVLW   StartPORTB      ; | |
        MOVWF   PORTB           ; | |
        BSF     _RP0            ;^| |
        MOVLW   StartTRISA      ;^| |
        MOVWF   TRISA           ;^| |
        MOVLW   StartTRISB      ;^| |
        MOVWF   TRISB           ;^| |
        MOVLW   StartOPTION     ;^| Initialise OPTION register
        MOVWF   OPTION_REG      ;^| |
        BCF     _RP0            ; | |
        CLRF    Old0            ; | Clear Old buffer
        CLRF    Old1            ; | |
        CLRF    Old2            ; | |
        CLRF    Old3            ; | |
        CLRF    Old4            ; | |
        CLRF    Old5            ; | |
        CLRF    Old6            ; | |
        CLRF    Old7            ; | |
        CLRF    Old8            ; | |
        CLRF    Old9            ; | |
        CLRF    OldA            ; | |
        CLRF    OldB            ; | |
        CLRF    OldC            ; | |
        CLRF    OldD            ; | |
        CLRF    OldE            ; | |
        CLRF    OldF            ; | |

BigLoop1
        BSF     _LED1           ; LEDs “reading”
        CALL    Delay07         ; |
        BCF     _LED2           ; |
        MOVLW   h’09’           ; Transmit TAB regularly
        CALL    RS232TxW        ; |
        MOVLW   d’128’          ; set BitCtr
        MOVWF   BitCtr          ; |
GetEdge                         ; Get an edge on _DATA_IN
        BTFSC   _DATA_IN        ; |
        GOTO    PreSync_H       ; |
        NOP                     ; |
PreSync_L                       ; |
        BTFSC   _DATA_IN        ; |
        GOTO    PreSync_H       ; |
        BTFSC   _DATA_IN        ; |
        GOTO    PreSync_H       ; |
DoSync_L                        ; |
        CLRWDT                  ; |
        BTFSS   _DATA_IN        ; |
        GOTO    DoSync_L        ; |
        BTFSS   _DATA_IN        ; |
        GOTO    DoSync_L        ; |
        GOTO    Sync_Done       ; |
                                ; |
PreSync_H                       ; |
        BTFSS   _DATA_IN        ; |
        GOTO    PreSync_L       ; |
        BTFSS   _DATA_IN        ; |
        GOTO    PreSync_L       ; |
DoSync_H                        ; |
        CLRWDT                  ; |
        BTFSC   _DATA_IN        ; |
        GOTO    DoSync_H        ; |
        BTFSC   _DATA_IN        ; |
        GOTO    DoSync_H        ; |
        GOTO    Sync_Done       ; |
Sync_Done                       ; |% 6 to (+4) from edge, say 8 from edge
        ;% -120Ti from sample
        NOP
        MOVLW   d’38’
        MOVWF   DelayReg
        ;% -117Ti from sample
```

```
ReadBit                         ; {% -3-DelayReg*3 Ti from sample
        NOP                     ;   delay
ReadBitD1                       ;   |
        DECFSZ  DelayReg,f      ;   |
        GOTO    ReadBitD1       ;   |
        CLRF    BO3             ;   BO3.1=_DATA_IN
        BTFSC   _DATA_IN        ;   |
        INCF    BO3,f           ;   |% effective sample time
        BTFSC   _DATA_IN        ;   |
        INCF    BO3,f           ;   |
        BTFSC   _DATA_IN        ;   |
        INCF    BO3,f           ;   |
        BCF     _CARRY          ;   _CARRY=BO3.1
        BTFSC   BO3,1           ;   |
        BSF     _CARRY          ;   |
        RLF     Buffer0,f       ;   roll in _CARRY
        RLF     Buffer1,f       ;   |
        RLF     Buffer2,f       ;   |
        RLF     Buffer3,f       ;   |
        RLF     Buffer4,f       ;   |
        RLF     Buffer5,f       ;   |
        RLF     Buffer6,f       ;   |
        RLF     Buffer7,f       ;   |
        RLF     Buffer8,f       ;   |
        RLF     Buffer9,f       ;   |
        RLF     BufferA,f       ;   |
        RLF     BufferB,f       ;   |
        RLF     BufferC,f       ;   |
        RLF     BufferD,f       ;   |
        RLF     BufferE,f       ;   |
        RLF     BufferF,f       ;   |
                                ;  % 23Ti from sample = -233Ti from sample
        MOVLW   d’75’           ;   set bit delay
        MOVWF   DelayReg        ;   |% -231Ti from sample
        ;% -6-DelayReg*3 Ti from sample
        DECFSZ  BitCtr,f        ;   DEC BitCtr
        GOTO    ReadBit         ; } until (BitCtr==#0)

HeadSearch
        MOVLW   d’128’          ; set BitCtr
        MOVWF   BitCtr          ; |
HeadSearchL1                    ; {
        MOVLW   h’80’           ;   if (header found)
        XORWF   BufferF,W       ;   |
        BTFSS   _ZERO           ;   |
        GOTO    NotHead0        ;   |
        MOVLW   h’2A’           ;   |
        XORWF   BufferE,W       ;   |
        BTFSS   _ZERO           ;   |
        GOTO    NotHead0        ;   {
        GOTO    HeadPolarity0   ;     goto HeadPolarity0
NotHead0                        ;   }
        MOVLW   h’7F’           ;   if (inverse header found)
        XORWF   BufferF,W       ;   |
        BTFSS   _ZERO           ;   |
        GOTO    NotHead1        ;   |
        MOVLW   h’D5’           ;   |
        XORWF   BufferE,W       ;   |
        BTFSS   _ZERO           ;   |
        GOTO    NotHead1        ;   {
        GOTO    HeadPolarity1   ;     goto HeadPolarity1
NotHead1                        ;   }
        RLF     Buffer0,f       ;   ROL Buffer
        RLF     Buffer1,f       ;   |
        RLF     Buffer2,f       ;   |
        RLF     Buffer3,f       ;   |
        RLF     Buffer4,f       ;   |
        RLF     Buffer5,f       ;   |
        RLF     Buffer6,f       ;   |
        RLF     Buffer7,f       ;   |
        RLF     Buffer8,f       ;   |
        RLF     Buffer9,f       ;   |
        RLF     BufferA,f       ;   |
        RLF     BufferB,f       ;   |
        RLF     BufferC,f       ;   |
        RLF     BufferD,f       ;   |
        RLF     BufferE,f       ;   |
        RLF     BufferF,f       ;   |
        BCF     Buffer0,0       ;   |
        BTFSC   _CARRY          ;   |
        BSF     Buffer0,0       ;   |
        DECFSZ  BitCtr,f        ;   DEC BitCtr
        GOTO    HeadSearchL1    ; } until (BitCtr==#0)
        GOTO    BigLoop1        ; goto BigLoop1

HeadPolarity1
        COMF    Buffer0,f
        COMF    Buffer1,f
        COMF    Buffer2,f
        COMF    Buffer3,f
        COMF    Buffer4,f
        COMF    Buffer5,f
        COMF    Buffer6,f
        COMF    Buffer7,f
        COMF    Buffer8,f
        COMF    Buffer9,f
        COMF    BufferA,f
        COMF    BufferB,f
        COMF    BufferC,f
        COMF    BufferD,f
        COMF    BufferE,f
        COMF    BufferF,f
HeadPolarity0
HeadFound

CheckSame
        MOVF    Buffer0,W
        XORWF   Old0,W
        BTFSS   _ZERO
        GOTO    NotSame
        MOVF    Buffer1,W
        XORWF   Old1,W
        BTFSS   _ZERO
        GOTO    NotSame
        MOVF    Buffer2,W
        XORWF   Old2,W
        BTFSS   _ZERO
        GOTO    NotSame
        MOVF    Buffer3,W
        XORWF   Old3,W
        BTFSS   _ZERO
        GOTO    NotSame
        MOVF    Buffer4,W
        XORWF   Old4,W
        BTFSS   _ZERO
        GOTO    NotSame
        MOVF    Buffer5,W
        XORWF   Old5,W
        BTFSS   _ZERO
        GOTO    NotSame
        MOVF    Buffer6,W
        XORWF   Old6,W
        BTFSS   _ZERO
        GOTO    NotSame
        MOVF    Buffer7,W
        XORWF   Old7,W
        BTFSS   _ZERO
        GOTO    NotSame
        MOVF    Buffer8,W
        XORWF   Old8,W
        BTFSS   _ZERO
        GOTO    NotSame
        MOVF    Buffer9,W
        XORWF   Old9,W
        BTFSS   _ZERO
        GOTO    NotSame
        MOVF    BufferA,W
        XORWF   OldA,W
        BTFSS   _ZERO
        GOTO    NotSame
        MOVF    BufferB,W
        XORWF   OldB,W
        BTFSS   _ZERO
        GOTO    NotSame
        MOVF    BufferC,W
        XORWF   OldC,W
        BTFSS   _ZERO
        GOTO    NotSame
        MOVF    BufferD,W
        XORWF   OldD,W
        BTFSS   _ZERO
        GOTO    NotSame
        MOVF    BufferE,W
        XORWF   OldE,W
        BTFSS   _ZERO
        GOTO    NotSame
        MOVF    BufferF,W
        XORWF   OldF,W
        BTFSS   _ZERO
        GOTO    NotSame
        GOTO    Same
NotSame
        MOVF    Buffer0,W
        MOVWF   Old0
        MOVF    Buffer1,W
        MOVWF   Old1
        MOVF    Buffer2,W
        MOVWF   Old2
        MOVF    Buffer3,W
        MOVWF   Old3
        MOVF    Buffer4,W
        MOVWF   Old4
        MOVF    Buffer5,W
        MOVWF   Old5
        MOVF    Buffer6,W
        MOVWF   Old6
        MOVF    Buffer7,W
        MOVWF   Old7
        MOVF    Buffer8,W
        MOVWF   Old8
        MOVF    Buffer9,W
        MOVWF   Old9
        MOVF    BufferA,W
        MOVWF   OldA
        MOVF    BufferB,W
        MOVWF   OldB
        MOVF    BufferC,W
        MOVWF   OldC
        MOVF    BufferD,W
        MOVWF   OldD
        MOVF    BufferE,W
        MOVWF   OldE
        MOVF    BufferF,W
        MOVWF   OldF
        GOTO    BigLoop1
Same

TxTag                           ;- Transmit tag
        BSF     _LED2           ; LEDs “Found tag”
        CALL    Delay07         ; |
        BCF     _LED1           ; |
        MOVLW   d’4’            ; Beep at 3597Hz for 1024 cycles
        MOVWF   BeepCtrHi       ; |
        MOVLW   d’0’            ; |
        MOVWF   BeepCtrLo       ; |
BeepLoopJ1                      ; |
        GOTO    BeepLoopJ2      ; |
BeepLoopJ2                      ; |
        MOVLW   Beep1           ; |
        MOVWF   BeepPort        ; |
        MOVLW   d’34’           ; |
        MOVWF   DelayReg        ; |
BeepD1                          ; |
        CLRWDT                  ; |
        DECFSZ  DelayReg,f      ; |
        GOTO    BeepD1          ; |
        MOVLW   Beep2           ; |
        MOVWF   BeepPort        ; |
        MOVLW   d’32’           ; |
        MOVWF   DelayReg        ; |
        NOP                     ; |
        GOTO    BeepD2          ; |
BeepD2                          ; |
        CLRWDT                  ; |
        DECFSZ  DelayReg,f      ; |
        GOTO    BeepD2          ; |
        DECFSZ  BeepCtrLo,f     ; |
        GOTO    BeepLoopJ1      ; |
        DECFSZ  BeepCtrHi,f     ; |
        GOTO    BeepLoopJ2      ; |
        NOP                     ; |
        MOVLW   Beep0           ; |
        MOVWF   BeepPort        ; |

        CALL    RS232CR         ; Transmit tag info
        MOVLW   ‘P’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘S’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘K’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘/’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘2’             ; |
        CALL    RS232TxW        ; |
        CALL    RS232CR         ; |
        MOVLW   ‘T’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘b’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘i’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘t’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘=’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘3’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘2’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘T’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘c’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘y’             ; |
        CALL    RS232TxW        ; |
        CALL    RS232CR         ; |
        MOVLW   ‘C’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘o’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘n’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘s’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘t’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘a’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘n’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘t’             ; |
        CALL    RS232TxW        ; |
        CALL    RS232CR         ; |
        MOVLW   ‘T’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘t’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘a’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘g’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘=’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘1’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘2’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘8’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘T’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘b’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘i’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘t’             ; |
        CALL    RS232TxW        ; |
        CALL    RS232CR         ; |
        MOVLW   ‘P’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘o’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘l’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘a’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘r’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘i’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘t’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘y’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘ ‘             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘0’             ; |
        CALL    RS232TxW        ; |
        CALL    RS232CR         ; |

        MOVLW   BufferF         ; Transmit tag ID
        MOVWF   FSR             ; |
TxLoop1                         ; |
        SWAPF   INDF,W          ; |
        CALL    RS232TxDigit    ; |
        MOVF    INDF,W          ; |
        CALL    RS232TxDigit    ; |
        DECF    FSR,f           ; |
        BTFSC   FSR,4           ; |
        GOTO    TxLoop1         ; |
        CALL    RS232CR         ; |

        GOTO    BigLoop1        ; goto BigLoop1

        end
```

ASK Reader Reference Design

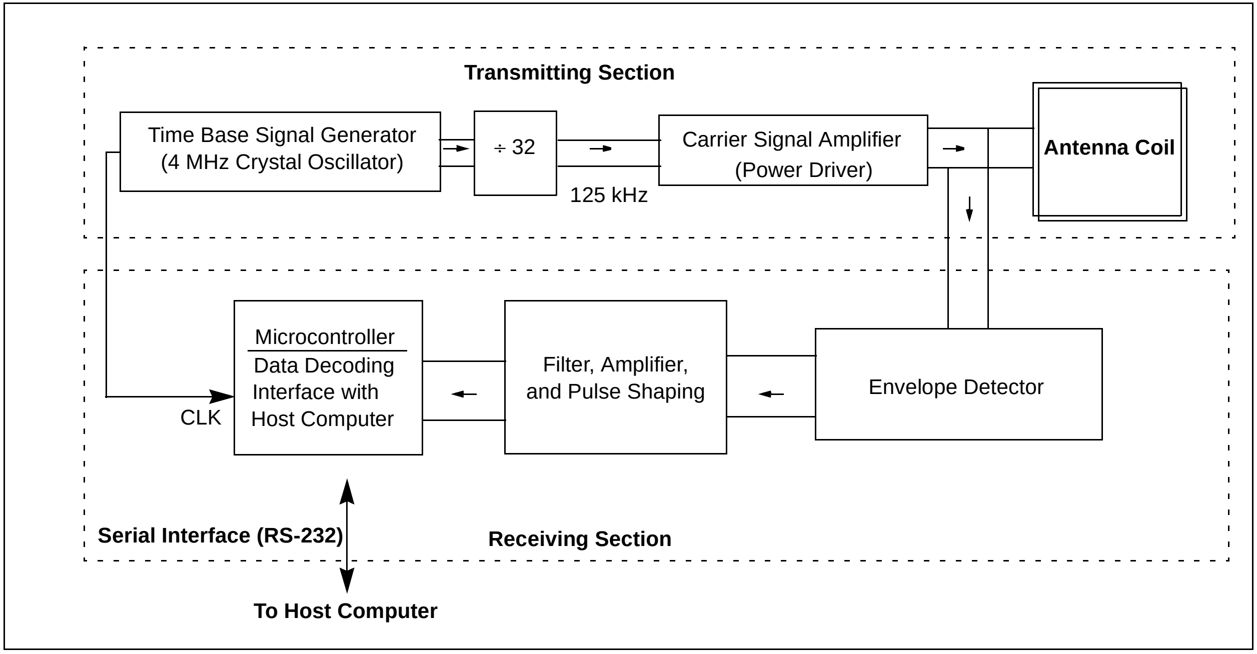
1.0 INTRODUCTION

This application note is written as a reference guide for ASK reader designers. Microchip Technology Inc. provides basic reader electronics circuitry for the MCRF200 customers as a part of this design guide. The circuit is designed for a read range of 3 ~ 5 inches with an access control card. The microID ASK Reader (demo unit), which is built based on the ASK reference design, is available in the microID Designers Kit (DV103001). The circuit can be modified for longer read range or other applications with the MCRF200. An electronic copy of the ASK microID PICmicro® source code is available upon request.

2.0 READER CIRCUITS

The RFID reader consists of transmitting and receiving sections. It transmits a carrier signal, receives the backscattering signal, and performs data processing. The reader also communicates with an external host computer. A basic block diagram of the typical ASK RFID reader is shown in Figure 2-1.

FIGURE 2-1: BLOCK DIAGRAM OF TYPICAL RFID READER FOR ASK SIGNAL (125 khz)


Transmitting Section
Time Base Signal Generator (4 MHz Crystal Oscillator)
÷ 32
125 kHz
Carrier Signal Amplifier (Power Driver)
Antenna Coil
Microcontroller
Data Decoding Interface with Host Computer
CLK
Filter, Amplifier, and Pulse Shaping
Envelope Detector
Serial Interface (RS-232)
Receiving Section
To Host Computer

PICmicro is a registered trademark of Microchip Technology Inc.

2.1 Transmitting Section

The transmitting section contains circuitry for a carrier signal (125 kHz), power amplifiers, and a tuned antenna coil.

The 125 kHz carrier signal is typically generated by dividing a 4 MHz (4 MHz/32 = 125 kHz) crystal oscillator signal. The signal is amplified before it is fed into the antenna tuning circuit. A complementary power amplifier circuit is typically used to boost the ransmitting signal level.

An antenna impedance tuning circuit consisting of capacitors is used to maximize the signal level at the carrier frequency. The tuning compensates the variations in the component values and the perturbation of coil inductance due to environment effect. A design guide for the antenna coil is given in *AN678, RFID Coil Design*.

2.1.1 LIMITS ON TRANSMITTING SIGNAL LEVEL (FCC PART 15) IN THE USA

Each country limits the signal strength of the RF wave that is intentionally radiated by a device. In the USA, the signal strength of the carrier signal (125 kHz) radiating from the antenna coil must comply with the FCC (Federal Communications Commission) part 15 regulation. The signal level is specified by the 47 CFR Part 15.209a of the federal regulation. For a 125 kHz signal, the FCC limits the signal level to 19.2 µv per meter, or 25.66 dBµV (i.e., 20 log(19.2) = 25.66 dBµV), at 300 meters away from the antenna. For a close distance measurement, an extrapolation rule (40 dB per decade) is applied (Part 15.31.f.2). For example, the signal level at 30 meters away from the device must not exceed:

$$25.66 \text{ dB}\mu\text{V} + 40 \text{ dB}\mu\text{V} = 65.66 \text{ dB}\mu\text{V}$$

2.2 Receiving Section

The receiving section consists of an antenna coil, demodulator, filters, amplifiers, and microcontroller. In applications for close proximity read range, a single coil is often used for both transmitting and receiving. For long read-range applications, however, separated antennas may be used. More details on the antenna coil are given in *AN678, RFID Coil Design* page 25.

In the ASK communication protocol, a ‘0’ and a ‘1’ are represented by an amplitude status of receiving signal. Various data coding waveforms that are available by MCRF200 are shown in Figure 1 in *AN680 Passive RFID Basics*, page 1.

The demodulation of the ASK signal is accomplished by detecting the envelope of the carrier signal. A half-wave capacitor-filtered rectifier circuit is used for the demodulation process. The peak voltage of the back-scattering signal is detected by a diode, and this voltage is then fed into an RC charging/discharging circuit. The RC time constant must be small enough to allow the voltage across C to fall fast enough to keep in step with the envelope. However, the time constant must not be so small as to introduce excessive ripple. The charging capacitor and load R has the following relationship for a full recovery of the data signal.

$$\frac{1}{\omega_s} > R > \frac{1}{\omega_o C}$$

where ω_S and ω_O are the angular frequencies of the modulation (data) and carrier (125 kHz), respectively. R is the load (discharging) resistor.

The demodulated signal must then pass through a filter and signal shaping circuit before it is fed to the microcontroller. The microcontroller performs data decoding and communicates with the host computer through an RS-232 or other serial interface protocols.

3.0 microID ASK READER

The electronic circuitry for an ASK reader is shown in Section 4.0. The reader needs +9 VDC power supply. The 125 kHz carrier signal is generated by dividing the 4 MHz time base signal that is generated by a crystal oscillator. A 16-stage binary ripple counter (74HC4060) is used for this purpose. The 74HC4060 also provides a clock signal for the PIC16C84 microcontroller. The 125 kHz signal is passed to an RF choke (L1) and filter before it is fed into a power amplifier that is formed by a pair of complementary bipolar transistors (Q2 and Q3).

For long read-range applications, this power amplifier circuit can be modified. Power MOSFETs may be used instead of the bipolar transistors (2N2222). These power MOSFETs can be driven by +24 VDC power supply. A push-pull predriver can be added at the front of the complementary circuit. This modification will enhance the signal level of the carrier signal and the read range of the ASK Reader.

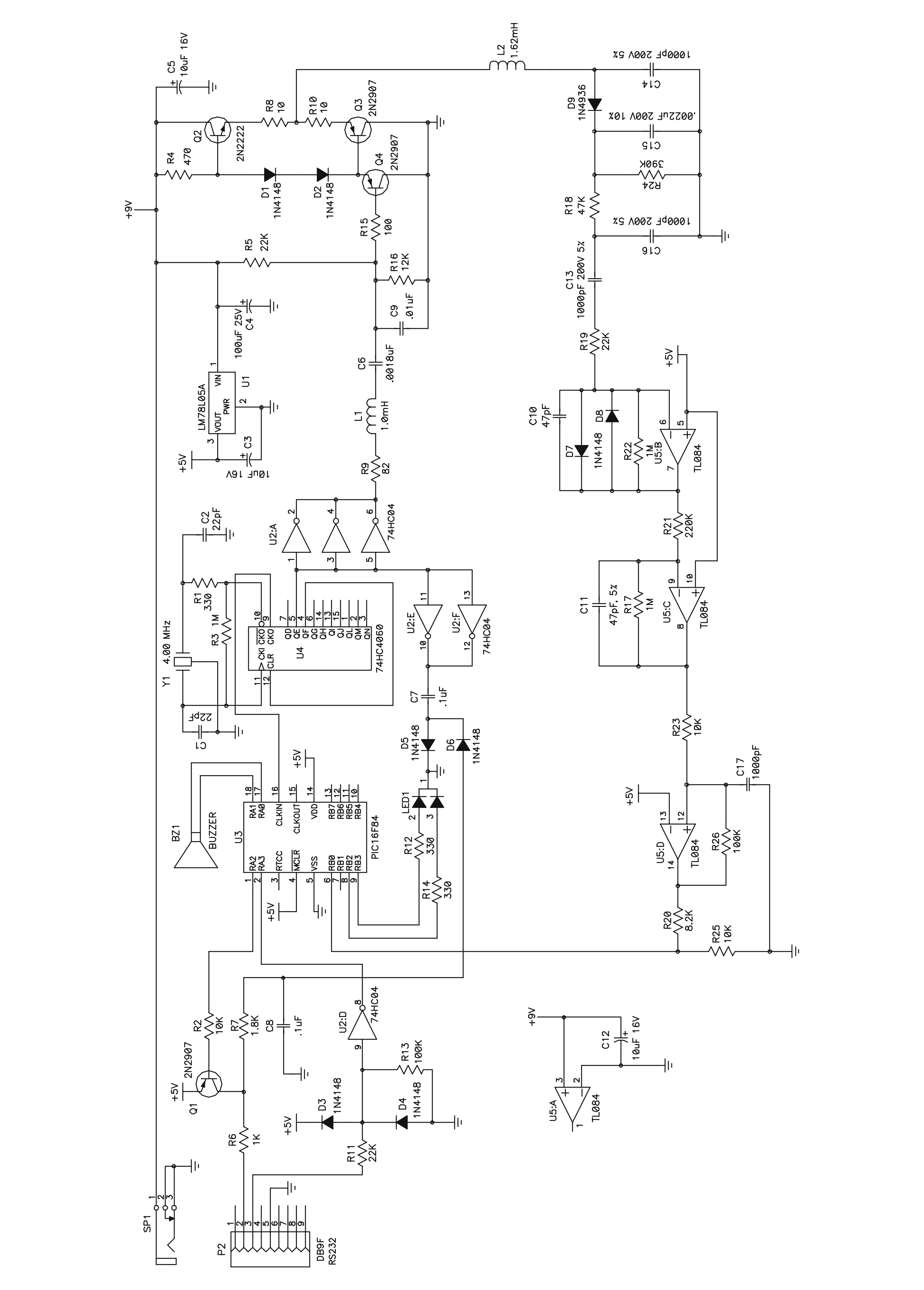
The reader circuit uses a single coil for both transmitting and receiving signals. An antenna coil (L2: 1.62 mH) and a resonant capacitor (C14: 1000 pF) forms a series resonant circuit for a 125 kHz resonance frequency. Since the C14 is grounded, the carrier signal (125 kHz) is filtered out to ground after passing the antenna coil. The circuit provides a minimum impedance at the resonance frequency. This results in maximizing the antenna current, and therefore, the magnetic field strength is maximized.

L2, C14, D7, C15, R24, and the other components in the bottom part of the circuit form a signal receiving section. D9 is a demodulator which detects the envelope of the backscattering signal.

D9 and C15 form a half-wave capacitor-filtered rectifier. The detected envelope signal is charged into C15. R24 provides a discharge path for the voltage charged in C15. This voltage passes active filters (U5:B and C) and the pulse shaping circuitry (U5:A) before it is fed into the PIC16C84 for data processing.

The PIC16C84 microcontroller performs data decoding and communicates with the host computer via an RS-232 serial interface.

4.0 ASK READER SCHEMATIC



5.0 ASK READER BILL OF MATERIALS

Quantity:	Part Number	Part Description	Reference Design
1	02-01518-D	PCB ASSEMBLY DWG, microID ASK READER	
1	03-01518	SCHEMATIC, microID ASK READER	
1	04-01518	PCB FAB, microID ASK READER	
1	08-00161	LABEL, microID ASK READER,U3,CHKS:C1AAh, v1.0, ASK1.HEX	@U3
1	110-93-318-41-001	SOCKET, 18P OPEN FRAME COLLET (0.300)	xU3
1	DE9S-FRS	CONN, D-SUB 9P RECPT RT ANGLE	P2
1	DJ005B	JACK, POWER, 2.5 mm DC	PC MOUNT SP1
1	PKM22EPP-4001	BUZZER, PIEZO, 4 kHz, 3-20V	BZ1
2	D470J25COGHAAAC CAP, 47PF 100V CERAMIC DISC C0G C10,C11 2	D220J20COGHAAAC CAP, 22 pF CER DISK RAD COG 100V	C1, C2
1	ECU-S1H221JCA	CAP, 220pF, CER MONO, RAD, 50V, 5%	C15
1	ECQ-P1102JZ	CAP, 0.001uF POLYPROPYLENE 100V	C17
3	ECQ-P6102JU	CAP, 0.001uF POLYPROPYLENE 630V	C13, C14, C16
1	ECU-S2A182JCB	CAP, 1800pF MONOLITH CERM, 5%, RAD, 100V	C6
1	ECQ-V1103JM	CAP, 0.01uF 100V STACK METAL FILM	C9
2	ECQ-E1104KF	CAP, 0.1UF 100VDC 10% RAD METAL POLY CAP	C7, C8
3	ECE-A16Z10	CAP, 10uF, ELECTRO, RAD, 16V, 20%	C3, C5, C12
1	ECE-A25Z100	CAP, 100uF, ELECTRO, RAD, 25V, 20%	C4
8	1N4148	DIODE, GENERAL PURPOSE, 1N4148 (DO-35)	D1-D8
1	1N4936	DIODE, 1A 400V FAST-RECOVERY RECTIFIER	D9
1	-SPARE- -SPARE- LOCATION DO NOT INSTALL LED1,		
1	78F102J INDUCTOR, 1000uH, COATED		L1
1	MCT0003-001	INDUCTOR, 1.62 µH,	L2
3	2N2907A-TO18	TRANSISTOR, 2N2907A PNP, GEN PURPOUS TO-18	Q1, Q3, Q4
1	2N2222A-TO18	TRANSISTOR, 2N2222A NPN, GEN PURPOUS TO-18	Q2
2	5043CX10R0J	RES, CF 10 OHM 1/4W 5%	R10,R8
1	82E CR-1/4W-B 5%	RES, CF 82 OHM 1/4W 5%	R9
1	5043CX100R0J	RES, CF 100 OHM 1/4W 5%	R15
1	5043CX1K000J	RES, CF 1K 1/4W 5%	R6
3	5043CX330R0J	RES, CF 330 OHM 1/4W 5%	R1, R12, R14
1	5043CX470R0J	RES, CF 470 OHM 5% 1/4W	R4
1	1K8 CR-1/4W-B 5%	RES, CF 1.8K OHM 1/4W 5%	R7
1	390K CR-1/4W-T 5%	RES, CF 390K-OHM,5%,1/4W	R24
1	220K CR-1/4W-T 5%	RES, CF 220K OHM 1/4W 5%	R21
1	8K2 CR-1/4W-T 5%	RES, 8.2K OHM 1/4W 5% CF	R20
3	10K CR-1/4W-B 5%	RES, CF 10K OHM 1/4W 5%	R2, R23, R25

Quantity:	Part Number	Part Description	Reference Design
1	5043CX47K00J	RES, CF 47K 5% 1/4W	R18
1	12K CR-1/4W-B 5%	RES, CF 12K OHM 1/4W 5%	R16
3	22K CR-1/4W-B 5%	RES, CF 22K OHM 1/4W 5%	R5, R11, R19
2	5043CX100K0J	RES, CF 100K 5% 1/4W	R13,R26
3	1M0 CR-1/4W-B 5%	RES, CF 1.0M OHM 1/4W 5%	R3, R17, R22
1	LM78L05ACZ	IC, REG, +5V 0.1A TO-92	U1
1	MM74HC04N	IC, HEX INVERTER 14P DIP	U2
1	PIC16F84-10/P	IC, PIC16F84 PLASTIC, 18P DIP	U3
1	MM74HC4060N	IC, 14 STAGE BINARY COUNTER, 16P DIP	U4
1	TL084CN IC, QUAD OP AMP, 14P DIP		U5
1	EFO-EC4004A4	RESONATOR, 4.00MHZ CERAMIC W/CAP	Y1
2	JS-01	SCREW, JACKSCREW, #4-40x0.416"	P2

6.0 ASK READER SOURCE CODE FOR THE PICmicro® MCU

The following source code is for the PIC16C84 microcontroller used in the ASK reader electronics.

```
; #=#=#=#=#=#=#=#=#=#=#= PROJECT  Microchip ASK Reader =#=#=#=#=#=#=#=#=#=#=#
; v002.asm
; PIC16C84 running at 4MHz, Ti=1us

; ///////////////////////////////////////////////////////////////////////////
; Revision history
; ///////////////////////////////////////////////////////////////////////////
;
; Ver    Date        Comment
;
; 0.01   01 Jul 98   Copied from MCHIP\READER\FSK
; 0.02   29 Jul 98   MICROCHIP TAG HAS 128 BITS
;
;       Tbit=64Tcy=512Ti
;       Manchester encoded
;       Microchip - Header=h’802A’  Ttag=128Tbit
;       - OR -
;       EM ASK - Header=b’111111111’ trailer=b’0’  Ttag=64Tbit
;

    processor pic16c84
    #include “p16c84.inc”
        __config b’11111111101001’
        ; Code Protect on, power-up timer on, WDT off, XT oscillator

#define bit_CARRY       STATUS,0
#define bit_ZERO        STATUS,2
#define bit_RP0         STATUS,5

#define _BUZZ1          PORTA,0
#define _BUZZ2          PORTA,1
#define _RS232TX        PORTA,2
#define _RS232RX        PORTA,3
#define _T0CKI          PORTA,4
StartPORTA      = b’01100’
StartTRISA      = b’11000’
BeepPort        = PORTA
Beep0           = StartPORTA
Beep1           = StartPORTA | b’00001’
Beep2           = StartPORTA | b’00010’

#define _DATA_IN        PORTB,0
#define _UNUSED1        PORTB,1
#define _LED2           PORTB,2
#define _LED1           PORTB,3
#define _UNUSED2        PORTB,4
#define _UNUSED3        PORTB,5
#define _UNUSED4        PORTB,6
#define _UNUSED5        PORTB,7
StartPORTB      = b’00000000’
StartTRISB      = b’00000001’

StartOPTION     = b’10001111’   ; TMR0 internal, prescaler off
                                ; PORTB pullups off

BO3             = h’0C’
DelayReg1       = h’0C’
Mask            = h’0C’
BitCtr          = h’0D’
BeepCtrHi       = h’0D’
TxByte          = h’0E’
BeepCtrLo       = h’0E’
ParityReg1      = h’0E’
```

```
Period          = h’0F’
ParityReg2      = h’0F’

Buffer0         = h’10’ ; --- IMMOBILE --- IMMOBILE --- IMMOBILE --- IMMOBILE
Buffer1         = h’11’ ; |
Buffer2         = h’12’ ; |
Buffer3         = h’13’ ; |
Buffer4         = h’14’ ; |
Buffer5         = h’15’ ; |
Buffer6         = h’16’ ; |
Buffer7         = h’17’ ; |
Buffer8         = h’18’ ; |
Buffer9         = h’19’ ; |
BufferA         = h’1A’ ; |
BufferB         = h’1B’ ; |
BufferC         = h’1C’ ; |
BufferD         = h’1D’ ; |
BufferE         = h’1E’ ; |
BufferF         = h’1F’ ; |
Old0            = h’20’ ; |
Old1            = h’21’ ; |
Old2            = h’22’ ; |
Old3            = h’23’ ; |
Old4            = h’24’ ; |
Old5            = h’25’ ; |
Old6            = h’26’ ; |
Old7            = h’27’ ; |
Old8            = h’28’ ; |
Old9            = h’29’ ; |
OldA            = h’2A’ ; |
OldB            = h’2B’ ; |
OldC            = h’2C’ ; |
OldD            = h’2D’ ; |
OldE            = h’2E’ ; |
OldF            = h’2F’ ; |

SKIP macro
        BTFSC   PCLATH,7
 endm

        org h’0000’             ; *#*#*#* RESET VECTOR *#*#*#*
        CLRF    PCLATH
        CLRF    INTCON
        CLRF    STATUS
        GOTO    RESET_A

        org h’0004’             ; *#*#*#* INTERRUPT VECTOR *#*#*#*
        CLRF    PCLATH
        CLRF    INTCON
        CLRF    STATUS
        GOTO    RESET_A

; ***** Subroutines, Page 0

Delay07:                        ;[0] Delay 7Ti
        NOP                     ; |
Delay06:                        ;[0] Delay 6Ti
        NOP                     ; |
Delay05:                        ;[0] Delay 5Ti
        NOP                     ; |
Delay04:                        ;[0] Delay 4Ti
        RETLW   0               ; |

RS232CR:                        ;[1] Transmit CR on RS232
        MOVLW   d’13’           ; |
        GOTO    RS232TxW        ; |
RS232TxDigit:                   ;[1] Transmit LSnybble of W on RS232
        ANDLW   h’0F’           ; |
        MOVWF   TxByte          ; |
        MOVLW   h’0A’           ; |
        SUBWF   TxByte,W        ; |
        BTFSS   bit_CARRY       ; |
        GOTO    DigitLT10       ; |
DigitGE10:                      ; |
        MOVLW   ‘A’-’0’-h’0A’   ; |
        ADDWF   TxByte,f        ; |
DigitLT10:                      ; |
        MOVLW   ‘0’             ; |
        ADDWF   TxByte,W        ; |
RS232TxW:                       ;[1] Transmit W on RS232 at 9615 baud
        MOVWF   TxByte          ; | TxByte=W
RS232Tx:                        ;[1] Transmit TxByte - 104us = 9615.4 baud
        BSF     _RS232TX        ; | Stop bit
        MOVLW   d’35’           ; | |
        MOVLW   DelayReg1       ; | |
RS232TxD1:                      ; | |
        DECFSZ  DelayReg1,f     ; | |
        GOTO    RS232TxD1       ; | |
        BCF     _RS232TX        ; | Start bit
        NOP                     ; | |
        MOVLW   d’32’           ; | |
        MOVWF   DelayReg1       ; | |
RS232TxD2:                      ; | |
        DECFSZ  DelayReg1,f     ; | |
        GOTO    RS232TxD2       ; | |
        CLRF    BitCtr          ; | BitCtr=#8
        BSF     BitCtr,3        ; | |
RS232TxL1:                      ; | {% -4Ti
        BTFSC   TxByte,0        ; |   Transmit TxByte.0, RR TxByte
        GOTO    RS232TxBit1     ; |   |
        NOP                     ; |   |
RS232TxBit0:                    ; |   |
        BCF     _RS232TX        ; |   |
        BCF     bit_CARRY       ; |   |
        GOTO    RS232TxBitDone  ; |   |
RS232TxBit1:                    ; |   |
        BSF     _RS232TX        ; |   |
        BSF     bit_CARRY       ; |   |
        GOTO    RS232TxBitDone  ; |   |
RS232TxBitDone:                 ; |   |
        RRF     TxByte,f        ; |   |% 4Ti
        MOVLW   d’30’           ; |   delay 1 bit
        MOVWF   DelayReg1       ; |   |
        GOTO    RS232TxD3       ; |   |
RS232TxD3:                      ; |   |
        DECFSZ  DelayReg1,f     ; |   |
        GOTO    RS232TxD3       ; |   |
        DECFSZ  BitCtr,f        ; |   DEC BitCtr
        GOTO    RS232TxL1       ; | } until (BitCtr==#0)
        CALL    Delay04         ; | delay
        BSF     _RS232TX        ; | stop bit
        RETLW   0               ; end

ParityCheck:                    ;[0] Check parity
        CLRF    ParityReg1      ; | ParityReg1=0
        MOVLW   d’10’           ; | BitCtr=10
        MOVWF   BitCtr          ; | |
ParityL1:                       ; | {
        CLRF    ParityReg2      ; |   ParityReg2=0
        MOVLW   h’10’           ; |   Mask=h’10’
        MOVWF   Mask            ; |   |
ParityL2:                       ; |   {
        BCF     bit_CARRY       ; |     LSL Buffer0-7
        RLF     Buffer0,f       ; |     |
        RLF     Buffer1,f       ; |     |
        RLF     Buffer2,f       ; |     |
        RLF     Buffer3,f       ; |     |
        RLF     Buffer4,f       ; |     |
        RLF     Buffer5,f       ; |     |
        RLF     Buffer6,f       ; |     |
        RLF     Buffer7,f       ; |     |
        BTFSC   Buffer6,7       ; |     if (Buffer6.7==1)
        INCF    ParityReg2,f    ; |     { INC ParityReg2 }
        MOVF    Mask,W          ; |     W=Mask
        BTFSC   Buffer6,7       ; |     if (Buffer6.7==1)
        XORWF   ParityReg1,f    ; |     { ParityReg1=ParityReg1 XOR W }
        BCF     bit_CARRY       ; |     LSR Mask
        RRF     Mask,f          ; |     |
        BTFSS   bit_CARRY       ; |   } until (bit_CARRY==1)
        GOTO    ParityL2        ; |   |
        BTFSC   ParityReg2,0    ; |   if (ParityReg2.0==1)
        GOTO    ParityBad       ; |   { goto ParityBad }
        DECFSZ  BitCtr,f        ; |   DEC BitCtr
        GOTO    ParityL1        ; | } until (BitCtr==0)
        MOVLW   h’10’           ; | Mask=h’10’
        MOVWF   Mask            ; | |
ParityL3:                       ; | {
        BCF     bit_CARRY       ; |   LSL Buffer0-7
        RLF     Buffer0,f       ; |   |
        RLF     Buffer1,f       ; |   |
        RLF     Buffer2,f       ; |   |
        RLF     Buffer3,f       ; |   |
        RLF     Buffer4,f       ; |   |
        RLF     Buffer5,f       ; |   |
        RLF     Buffer6,f       ; |   |
        RLF     Buffer7,f       ; |   |
        MOVF    Mask,W          ; |   W=Mask
        BTFSC   Buffer6,7       ; |   if (Buffer6.7==1)
        XORWF   ParityReg1,f    ; |   { ParityReg1=ParityReg1 XOR W }
        BCF     bit_CARRY       ; |   LSR Mask
        RRF     Mask,f          ; |   |
        BTFSS   Mask,0          ; | } until (Mask.0==1)
        GOTO    ParityL3        ; | |
        MOVF    ParityReg1,W    ; | if ((ParityReg1 AND h’1E’)!=0)
        ANDLW   h’1E’           ; | |
        BTFSS   bit_ZERO        ; | |
        GOTO    ParityBad       ; | { goto ParityBad }
ParityGood:                     ; |
        MOVF    BufferF,W       ; | Buffer0-7=Buffer8-F
        MOVWF   Buffer7         ; | |
        MOVF    BufferE,W       ; | |
        MOVWF   Buffer6         ; | |
        MOVF    BufferD,W       ; | |
        MOVWF   Buffer5         ; | |
        MOVF    BufferC,W       ; | |
        MOVWF   Buffer4         ; | |
        MOVF    BufferB,W       ; | |
        MOVWF   Buffer3         ; | |
        MOVF    BufferA,W       ; | |
        MOVWF   Buffer2         ; | |
        MOVF    Buffer9,W       ; | |
        MOVWF   Buffer1         ; | |
        MOVF    Buffer8,W       ; | |
        MOVWF   Buffer0         ; | |
        BCF     bit_CARRY       ; | bit_CARRY=0
        RETLW   0               ; |
ParityBad:                      ; |
        MOVF    BufferF,W       ; | Buffer0-7=Buffer8-F
        MOVWF   Buffer7         ; | |
        MOVF    BufferE,W       ; | |
        MOVWF   Buffer6         ; | |
        MOVF    BufferD,W       ; | |
        MOVWF   Buffer5         ; | |
        MOVF    BufferC,W       ; | |
        MOVWF   Buffer4         ; | |
        MOVF    BufferB,W       ; | |
        MOVWF   Buffer3         ; | |
        MOVF    BufferA,W       ; | |
        MOVWF   Buffer2         ; | |
        MOVF    Buffer9,W       ; | |
        MOVWF   Buffer1         ; | |
        MOVF    Buffer8,W       ; | |
        MOVWF   Buffer0         ; | |
        BSF     bit_CARRY       ; | bit_CARRY=1
        RETLW   0               ; |

; ***** End of subroutines, Page 0

RESET_A:
        CLRWDT
                                ; Initialise registers
        CLRF    STATUS          ; | Access register page 0
        CLRF    FSR             ; | FSR=#0
        MOVLW   StartPORTA      ; | Initialise PORT and TRIS registers
        MOVWF   PORTA           ; | |
        MOVLW   StartPORTB      ; | |
        MOVWF   PORTB           ; | |
        BSF     bit_RP0         ;^| |
        MOVLW   StartTRISA      ;^| |
        MOVWF   TRISA           ;^| |
        MOVLW   StartTRISB      ;^| |
        MOVWF   TRISB           ;^| |
        MOVLW   StartOPTION     ;^| Initialise OPTION register
        MOVWF   OPTION_REG      ;^| |
        BCF     bit_RP0         ; | |
        CLRF    Old0            ; | Clear Old buffer
        CLRF    Old1            ; | |
        CLRF    Old2            ; | |
        CLRF    Old3            ; | |
        CLRF    Old4            ; | |
        CLRF    Old5            ; | |
        CLRF    Old6            ; | |
        CLRF    Old7            ; | |

BigLoop1:
        BSF     _LED1           ; LEDs “reading”
        CALL    Delay07         ; |
        BCF     _LED2           ; |
        MOVLW   h’09’           ; Transmit TAB regularly
        CALL    RS232TxW        ; |
        MOVLW   d’128’          ; set BitCtr
        MOVWF   BitCtr          ; |

GetEdge:                        ; Get an edge on _DATA_IN
        BTFSC   _DATA_IN        ; |
        GOTO    PreSync_H0      ; |
        NOP                     ; |
PreSync_L0:                     ; |% 3 from low sample
        NOP                     ; |
        BTFSC   _DATA_IN        ; |
        GOTO    PreSync_H0      ; |
        CLRF    Period          ; | Period=0
PreSync_L1:                     ; | { % 7+Period*8 from low sample
        INCF    Period,f        ; |   INC Period
        BTFSC   Period,6        ; |   if ((Period*8Ti)>=Tbit*1.25=512Ti*1.25=640Ti)
        BTFSS   Period,4        ; |   |
        SKIP                    ; |   |
        GOTO    BigLoop1        ; |   { goto BigLoop1 }
        BTFSS   _DATA_IN        ; | } until (_DATA_IN==1)
        GOTO    PreSync_L1      ; | |
                                ; |% 6+Period*8 from low sample
                                ; |% 6 from rise
        MOVLW   d’48’           ; | if ((Period*8)>=Tbit*0.75=512Ti*0.75=384Ti)
        SUBWF   Period,W        ; | |
        BTFSC   bit_CARRY       ; | |
        GOTO    Sync_Done       ; | { goto Sync_Done }
                                ; |% 10 from rise
        CALL    Delay05         ; | delay
DoSync_H:                       ; |% 15 from rise
        MOVLW   d’2’            ; | Period=2
        MOVWF   Period          ; | |
        CALL    Delay04         ; | delay
        GOTO    DoSync_HL       ; | |
DoSync_HL:                      ; | {% 7+Period*8 from rise
        INCF    Period,f        ; |   INC Period
        BTFSC   Period,6        ; |   if ((Period*8Ti)>=Tbit*1.25=512Ti*1.25=640Ti)
        BTFSS   Period,4        ; |   |
        SKIP                    ; |   |
        GOTO    BigLoop1        ; |   { goto BigLoop1 }
        BTFSC   _DATA_IN        ; | } until (_DATA_IN==0)
        GOTO    DoSync_HL       ; | |
                                ; |% 6+Period*8 from rise
                                ; |% 6 from fall
        MOVLW   d’16’           ; | if ((Period*8Ti)<Tbit*0.25=512Ti*0.25=128Ti)
        SUBWF   Period,W        ; | |
        BTFSS   bit_CARRY       ; | |
        GOTO    BigLoop1        ; | { goto BigLoop1 }
                                ; |% 10 from fall
        MOVLW   d’48’           ; | if ((Period*8Ti)<Tbit*0.75=512Ti*0.75=384Ti)
        SUBWF   Period,W        ; | |
        BTFSS   bit_CARRY       ; | |
        GOTO    DoSync_L        ; | { goto DoSync_L }
        GOTO    Sync_Done       ; | goto Sync_Done

PreSync_H0:                     ; |% 3 from high sample
        NOP                     ; |
        BTFSS   _DATA_IN        ; |
        GOTO    PreSync_L0      ; |
        CLRF    Period          ; | Period=0
PreSync_H1:                     ; | {% 7+Period*8 from high sample
        INCF    Period,f        ; |   INC Period
        BTFSC   Period,6        ; |   if ((Period*8Ti)>=Tbit*1.25=512Ti*1.25=640Ti)
        BTFSS   Period,4        ; |   |
        SKIP                    ; |   |
        GOTO    BigLoop1        ; |   { goto BigLoop1 }
        BTFSC   _DATA_IN        ; | } until (_DATA_IN==0)
        GOTO    PreSync_H1      ; | |
                                ; |% 6+Period*8 from high sample
                                ; |% 6 from fall
        MOVLW   d’48’           ; | if ((Period*8Ti)>=Tbit*0.75=512Ti*0.75=384Ti)
        SUBWF   Period,W        ; | |
        BTFSC   bit_CARRY       ; | |
        GOTO    Sync_Done       ; | { goto Sync_Done }
                                ; |% 10 from fall
        CALL    Delay05         ; | delay
DoSync_L:                       ; |% 15 from fall
        MOVLW   d’2’            ; | Period=2
        MOVWF   Period          ; | |
        CALL    Delay04         ; | delay
        GOTO    DoSync_LL       ; | |
DoSync_LL:                      ; | {% 7+Period*8 from fall
        INCF    Period,f        ; |   INC Period
        BTFSC   Period,6        ; |   if ((Period*8Ti)>=Tbit*1.25=512Ti*1.25=640Ti)
        BTFSS   Period,4        ; |   |
        SKIP                    ; |   |
        GOTO    BigLoop1        ; |   { goto BigLoop1 }
        BTFSS   _DATA_IN        ; | } until (_DATA_IN==1)
        GOTO    DoSync_LL       ; | |
                                ; |% 6+Period*8 from fall
                                ; |% 6 from rise
        MOVLW   d’16’           ; | if ((Period*8Ti)<Tbit*0.25=512Ti*0.25=128Ti)
        SUBWF   Period,W        ; | |
        BTFSS   bit_CARRY       ; | |
        GOTO    BigLoop1        ; | { goto BigLoop1 }
                                ; |% 10 from rise
        MOVLW   d’48’           ; | if ((Period*8Ti)<Tbit*0.75=512Ti*0.75=384Ti)
        SUBWF   Period,W        ; | |
        BTFSS   bit_CARRY       ; | |
        GOTO    DoSync_H        ; | { goto DoSync_H }
        GOTO    Sync_Done       ; goto Sync_Done

Sync_Done:                      ; |% 16 from edge
                                ; |% -368 from sample
        MOVLW   d’121’          ; | DelayReg1=121
        MOVWF   DelayReg1       ; | |
        NOP                     ; | delay
ReadBit:                        ; {% -2-DelayReg1*3 Ti from sample
ReadBitD1:                      ;   delay
        DECFSZ  DelayReg1,f     ;   |
        GOTO    ReadBitD1       ;   |
        CLRF    BO3             ;   BO3.1=_DATA_IN
        BTFSC   _DATA_IN        ;   |
        INCF    BO3,f           ;   |% effective sample time
        BTFSC   _DATA_IN        ;   |
        INCF    BO3,f           ;   |
        BTFSC   _DATA_IN        ;   |
        INCF    BO3,f           ;   |
        BCF     bit_CARRY       ;   bit_CARRY=BO3.1
        BTFSC   BO3,1           ;   |
        BSF     bit_CARRY       ;   |
        RLF     Buffer0,f       ;   roll in bit_CARRY
        RLF     Buffer1,f       ;   |
        RLF     Buffer2,f       ;   |
        RLF     Buffer3,f       ;   |
        RLF     Buffer4,f       ;   |
        RLF     Buffer5,f       ;   |
        RLF     Buffer6,f       ;   |
        RLF     Buffer7,f       ;   |
        RLF     Buffer8,f       ;   |
        RLF     Buffer9,f       ;   |
        RLF     BufferA,f       ;   |
        RLF     BufferB,f       ;   |
        RLF     BufferC,f       ;   |
        RLF     BufferD,f       ;   |
        RLF     BufferE,f       ;   |
        RLF     BufferF,f       ;   |% 23 from sample
                                ;   |% -233 from sample
        MOVLW   d’76’           ;   delay 230Ti
        MOVWF   DelayReg1       ;   |
        NOP                     ;   |
ReadBitD2:                      ;   |
        DECFSZ  DelayReg1,f     ;   |
        GOTO    ReadBitD2       ;   |
                                ;   |% -3 from sample
        CLRF    BO3             ;   BO3.1=_DATA_IN
        BTFSC   _DATA_IN        ;   |
        INCF    BO3,f           ;   |% effective sample time
        BTFSC   _DATA_IN        ;   |
        INCF    BO3,f           ;   |
        BTFSC   _DATA_IN        ;   |
        INCF    BO3,f           ;   |
        BTFSC   Buffer0,0       ;   BO3.1=BO3.1 XOR Buffer0.0
        COMF    BO3,f           ;   |
        BTFSS   BO3,1           ;   if (BO3.1==0)
        GOTO    BigLoop1        ;   { goto BigLoop1 }
                                ;  % 8 from sample
                                ;  % -248 from sample
        MOVLW   d’80’           ;   DelayReg1=80
        MOVWF   DelayReg1       ;   |
        NOP                     ;   delay
                                ;  % -5-DelayReg1*3 Ti from sample
        DECFSZ  BitCtr,f        ;   DEC BitCtr
        GOTO    ReadBit         ; } until (BitCtr==#0)

HeadSearch1:
        MOVLW   d’128’          ; set BitCtr
        MOVWF   BitCtr          ; |
HeadSearch1L1:                  ; {
        MOVF    BufferF,W       ;   if (header found)
        XORLW   h’80’           ;   |
        BTFSS   bit_ZERO        ;   |
        GOTO    NotHead1A       ;   |
        MOVF    BufferE,W       ;   |
        XORLW   h’2A’           ;   |
        BTFSS   bit_ZERO        ;   |
        GOTO    NotHead1A       ;   {
        GOTO    HeadFound0      ;     goto HeadFound0
NotHead1A:                      ;   }
        MOVF    BufferF,W       ;   if (inverse header found)
        XORLW   h’7F’           ;   |
        BTFSS   bit_ZERO        ;   |
        GOTO    NotHead1B       ;   |
        MOVF    BufferE,W       ;   |
        XORLW   h’D5’           ;   |
        BTFSS   bit_ZERO        ;   |
        GOTO    NotHead1B       ;   {
        GOTO    HeadFound1      ;     goto HeadFound1
NotHead1B:                      ;   }
        RLF     Buffer0,f       ;   ROL Buffer
        RLF     Buffer1,f       ;   |
        RLF     Buffer2,f       ;   |
        RLF     Buffer3,f       ;   |
        RLF     Buffer4,f       ;   |
        RLF     Buffer5,f       ;   |
        RLF     Buffer6,f       ;   |
        RLF     Buffer7,f       ;   |
        RLF     Buffer8,f       ;   |
        RLF     Buffer9,f       ;   |
        RLF     BufferA,f       ;   |
        RLF     BufferB,f       ;   |
        RLF     BufferC,f       ;   |
        RLF     BufferD,f       ;   |
        RLF     BufferE,f       ;   |
        RLF     BufferF,f       ;   |
        BCF     Buffer0,0       ;   |
        BTFSC   bit_CARRY       ;   |
        BSF     Buffer0,0       ;   |
        DECFSZ  BitCtr,f        ;   DEC BitCtr
        GOTO    HeadSearch1L1   ; } until (BitCtr==#0)

        MOVF    Buffer0,W       ; if ((Buffer0-7)!=(Buffer8-F)) { goto BigLoop1 }
        XORWF   Buffer8,W       ; |
        BTFSS   bit_ZERO        ; |
        GOTO    BigLoop1        ; |
        MOVF    Buffer1,W       ; |
        XORWF   Buffer9,W       ; |
        BTFSS   bit_ZERO        ; |
        GOTO    BigLoop1        ; |
        MOVF    Buffer2,W       ; |
        XORWF   BufferA,W       ; |
        BTFSS   bit_ZERO        ; |
        GOTO    BigLoop1        ; |
        MOVF    Buffer3,W       ; |
        XORWF   BufferB,W       ; |
        BTFSS   bit_ZERO        ; |
        GOTO    BigLoop1        ; |
        MOVF    Buffer4,W       ; |
        XORWF   BufferC,W       ; |
        BTFSS   bit_ZERO        ; |
        GOTO    BigLoop1        ; |
        MOVF    Buffer5,W       ; |
        XORWF   BufferD,W       ; |
        BTFSS   bit_ZERO        ; |
        GOTO    BigLoop1        ; |
        MOVF    Buffer6,W       ; |
        XORWF   BufferE,W       ; |
        BTFSS   bit_ZERO        ; |
        GOTO    BigLoop1        ; |
        MOVF    Buffer7,W       ; |
        XORWF   BufferF,W       ; |
        BTFSS   bit_ZERO        ; |
        GOTO    BigLoop1        ; |

HeadSearch2:
        MOVLW   d’64’           ; set BitCtr
        MOVWF   BitCtr          ; |
HeadSearch2L1:                  ; {
        MOVF    BufferF,W       ;   if (header found)
        XORLW   h’FF’           ;   |
        BTFSS   bit_ZERO        ;   |
        GOTO    NotHead2A       ;   |
        BTFSS   BufferE,7       ;   |
        GOTO    NotHead2A       ;   |
        BTFSC   Buffer8,0       ;   |
        GOTO    NotHead2A       ;   {
        GOTO    HeadFound2      ;     goto HeadFound2
NotHead2A:                      ;   }
        RLF     Buffer0,f       ;   ROL Buffer
        RLF     Buffer1,f       ;   |
        RLF     Buffer2,f       ;   |
        RLF     Buffer3,f       ;   |
        RLF     Buffer4,f       ;   |
        RLF     Buffer5,f       ;   |
        RLF     Buffer6,f       ;   |
        RLF     Buffer7,f       ;   |
        RLF     Buffer8,f       ;   |
        RLF     Buffer9,f       ;   |
        RLF     BufferA,f       ;   |
        RLF     BufferB,f       ;   |
        RLF     BufferC,f       ;   |
        RLF     BufferD,f       ;   |
        RLF     BufferE,f       ;   |
        RLF     BufferF,f       ;   |
        BCF     Buffer0,0       ;   |
        BTFSC   bit_CARRY       ;   |
        BSF     Buffer0,0       ;   |
        DECFSZ  BitCtr,f        ;   DEC BitCtr
        GOTO    HeadSearch2L1   ; } until (BitCtr==#0)
```

```
HeadSearch3:
        MOVLW   d’64’           ; set BitCtr
        MOVWF   BitCtr          ; |
HeadSearch3L1:                  ; {
        MOVF    BufferF,W       ;   if (header found)
        XORLW   h’00’           ;   |
        BTFSS   bit_ZERO        ;   |
        GOTO    NotHead3A       ;   |
        BTFSC   BufferE,7       ;   |
        GOTO    NotHead3A       ;   |
        BTFSS   Buffer8,0       ;   |
        GOTO    NotHead3A       ;   {
        GOTO    HeadFound3      ;     goto HeadFound3
NotHead3A:                      ;   }
        RLF     Buffer0,f       ;   ROL Buffer
        RLF     Buffer1,f       ;   |
        RLF     Buffer2,f       ;   |
        RLF     Buffer3,f       ;   |
        RLF     Buffer4,f       ;   |
        RLF     Buffer5,f       ;   |
        RLF     Buffer6,f       ;   |
        RLF     Buffer7,f       ;   |
        RLF     Buffer8,f       ;   |
        RLF     Buffer9,f       ;   |
        RLF     BufferA,f       ;   |
        RLF     BufferB,f       ;   |
        RLF     BufferC,f       ;   |
        RLF     BufferD,f       ;   |
        RLF     BufferE,f       ;   |
        RLF     BufferF,f       ;   |
        BCF     Buffer0,0       ;   |
        BTFSC   bit_CARRY       ;   |
        BSF     Buffer0,0       ;   |
        DECFSZ  BitCtr,f        ;   DEC BitCtr
        GOTO    HeadSearch3L1   ; } until (BitCtr==#0)

        GOTO    BigLoop1        ; goto BigLoop1

HeadFound3:
        COMF    BufferF,f
        COMF    BufferE,f
        COMF    BufferD,f
        COMF    BufferC,f
        COMF    BufferB,f
        COMF    BufferA,f
        COMF    Buffer9,f
        COMF    Buffer8,f
        COMF    Buffer7,f
        COMF    Buffer6,f
        COMF    Buffer5,f
        COMF    Buffer4,f
        COMF    Buffer3,f
        COMF    Buffer2,f
        COMF    Buffer1,f
        COMF    Buffer0,f
        CALL    ParityCheck
        BTFSC   bit_CARRY
        GOTO    BigLoop1
        GOTO    CheckSame

HeadFound2:
        CALL    ParityCheck
        BTFSC   bit_CARRY
        GOTO    HeadSearch3
        GOTO    CheckSame
```

```
HeadFound1:
        COMF    BufferF,f
        COMF    BufferE,f
        COMF    BufferD,f
        COMF    BufferC,f
        COMF    BufferB,f
        COMF    BufferA,f
        COMF    Buffer9,f
        COMF    Buffer8,f
        COMF    Buffer7,f
        COMF    Buffer6,f
        COMF    Buffer5,f
        COMF    Buffer4,f
        COMF    Buffer3,f
        COMF    Buffer2,f
        COMF    Buffer1,f
        COMF    Buffer0,f
HeadFound0:

CheckSame:                      ; if (Buffer!=Old) { goto NotSame }
        MOVF    Buffer0,W       ; |
        XORWF   Old0,W          ; |
        BTFSS   bit_ZERO        ; |
        GOTO    NotSame         ; |
        MOVF    Buffer1,W       ; |
        XORWF   Old1,W          ; |
        BTFSS   bit_ZERO        ; |
        GOTO    NotSame         ; |
        MOVF    Buffer2,W       ; |
        XORWF   Old2,W          ; |
        BTFSS   bit_ZERO        ; |
        GOTO    NotSame         ; |
        MOVF    Buffer3,W       ; |
        XORWF   Old3,W          ; |
        BTFSS   bit_ZERO        ; |
        GOTO    NotSame         ; |
        MOVF    Buffer4,W       ; |
        XORWF   Old4,W          ; |
        BTFSS   bit_ZERO        ; |
        GOTO    NotSame         ; |
        MOVF    Buffer5,W       ; |
        XORWF   Old5,W          ; |
        BTFSS   bit_ZERO        ; |
        GOTO    NotSame         ; |
        MOVF    Buffer6,W       ; |
        XORWF   Old6,W          ; |
        BTFSS   bit_ZERO        ; |
        GOTO    NotSame         ; |
        MOVF    Buffer7,W       ; |
        XORWF   Old7,W          ; |
        BTFSS   bit_ZERO        ; |
        GOTO    NotSame         ; |
        MOVF    Buffer8,W       ; |
        XORWF   Old8,W          ; |
        BTFSS   bit_ZERO        ; |
        GOTO    NotSame         ; |
        MOVF    Buffer9,W       ; |
        XORWF   Old9,W          ; |
        BTFSS   bit_ZERO        ; |
        GOTO    NotSame         ; |
        MOVF    BufferA,W       ; |
        XORWF   OldA,W          ; |
        BTFSS   bit_ZERO        ; |
        GOTO    NotSame         ; |
        MOVF    BufferB,W       ; |
        XORWF   OldB,W          ; |
        BTFSS   bit_ZERO        ; |
        GOTO    NotSame         ; |
        MOVF    BufferC,W       ; |
        XORWF   OldC,W          ; |
        BTFSS   bit_ZERO        ; |
        GOTO    NotSame         ; |
        MOVF    BufferD,W       ; |
        XORWF   OldD,W          ; |
        BTFSS   bit_ZERO        ; |
        GOTO    NotSame         ; |
        MOVF    BufferE,W       ; |
        XORWF   OldE,W          ; |
        BTFSS   bit_ZERO        ; |
        GOTO    NotSame         ; |
        MOVF    BufferF,W       ; |
        XORWF   OldF,W          ; |
        BTFSS   bit_ZERO        ; |
        GOTO    NotSame         ; |

Same:

TxTag:                          ;- Transmit tag
        BSF     _LED2           ; LEDs “Found tag”
        CALL    Delay07         ; |
        BCF     _LED1           ; |
        MOVLW   d’4’            ; Beep at 3597Hz for 1024 cycles
        MOVWF   BeepCtrHi       ; |
        MOVLW   d’0’            ; |
        MOVWF   BeepCtrLo       ; |
BeepLoopJ1:                     ; |
        GOTO    BeepLoopJ2      ; |
BeepLoopJ2:                     ; |
        MOVLW   Beep1           ; |
        MOVWF   BeepPort        ; |
        MOVLW   d’34’           ; |
        MOVWF   DelayReg1       ; |
BeepD1:                         ; |
        CLRWDT                  ; |
        DECFSZ  DelayReg1,f     ; |
        GOTO    BeepD1          ; |
        MOVLW   Beep2           ; |
        MOVWF   BeepPort        ; |
        MOVLW   d’32’           ; |
        MOVWF   DelayReg1       ; |
        NOP                     ; |
        GOTO    BeepD2          ; |
BeepD2:                         ; |
        CLRWDT                  ; |
        DECFSZ  DelayReg1,f     ; |
        GOTO    BeepD2          ; |
        DECFSZ  BeepCtrLo,f     ; |
        GOTO    BeepLoopJ1      ; |
        DECFSZ  BeepCtrHi,f     ; |
        GOTO    BeepLoopJ2      ; |
        NOP                     ; |
        MOVLW   Beep0           ; |
        MOVWF   BeepPort        ; |

        MOVF    OldF,W
        MOVWF   BufferF
        MOVF    OldE,W
        MOVWF   BufferE
        MOVF    OldD,W
        MOVWF   BufferD
        MOVF    OldC,W
        MOVWF   BufferC
        MOVF    OldB,W
        MOVWF   BufferB
        MOVF    OldA,W
        MOVWF   BufferA
        MOVF    Old9,W
        MOVWF   Buffer9
        MOVF    Old8,W
        MOVWF   Buffer8
        MOVF    Old7,W
        MOVWF   Buffer7
        MOVF    Old6,W
        MOVWF   Buffer6
        MOVF    Old5,W
        MOVWF   Buffer5
        MOVF    Old4,W
        MOVWF   Buffer4
        MOVF    Old3,W
        MOVWF   Buffer3
        MOVF    Old2,W
        MOVWF   Buffer2
        MOVF    Old1,W
        MOVWF   Buffer1
        MOVF    Old0,W
        MOVWF   Buffer0

        CALL    RS232CR         ; Transmit tag info
        MOVLW   ‘A’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘S’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘K’             ; |
        CALL    RS232TxW        ; |
        CALL    RS232CR         ; |
        MOVLW   ‘T’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘b’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘i’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘t’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘=’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘6’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘4’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘T’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘c’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘y’             ; |
        CALL    RS232TxW        ; |
        CALL    RS232CR         ; |
        MOVLW   ‘C’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘o’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘n’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘s’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘t’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘a’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘n’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘t’             ; |
        CALL    RS232TxW        ; |
        CALL    RS232CR         ; |
        MOVLW   ‘T’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘t’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘a’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘g’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘=’             ; |
        CALL    RS232TxW        ; |
        MOVF    BufferF,W       ; |
        XORLW   h’80’           ; |
        BTFSS   bit_ZERO        ; |
        GOTO    Ttag64          ; |
Ttag128:                        ; |
        MOVLW   ‘1’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘2’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘8’             ; |
        CALL    RS232TxW        ; |
        GOTO    TtagJ1          ; |
Ttag64:                         ; |
        MOVLW   ‘6’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘4’             ; |
        CALL    RS232TxW        ; |
        GOTO    TtagJ1          ; |
TtagJ1:                         ; |
        MOVLW   ‘T’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘b’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘i’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘t’             ; |
        CALL    RS232TxW        ; |
        CALL    RS232CR         ; |
        MOVLW   ‘P’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘o’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘l’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘a’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘r’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘i’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘t’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘y’             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘ ‘             ; |
        CALL    RS232TxW        ; |
        MOVLW   ‘0’             ; |
        CALL    RS232TxW        ; |
        CALL    RS232CR         ; |
        MOVLW   BufferF         ; Transmit tag ID
        MOVWF   FSR             ; |
        MOVF    BufferF,W       ; |
        XORLW   h’80’           ; |
        BTFSC   bit_ZERO        ; |
        GOTO    TxLoop1         ; |
        MOVLW   Buffer7         ; |
        MOVWF   FSR             ; |
TxLoop1:                        ; |
        SWAPF   INDF,W          ; |
        CALL    RS232TxDigit    ; |
        MOVF    INDF,W          ; |
        CALL    RS232TxDigit    ; |
        DECF    FSR,f           ; |
        BTFSC   FSR,4           ; |
        GOTO    TxLoop1         ; |
        CALL    RS232CR         ; |

        GOTO    BigLoop1        ; goto BigLoop1

NotSame:                        ; Old=Data
        MOVF    Buffer0,W       ; |
        MOVWF   Old0            ; |
        MOVF    Buffer1,W       ; |
        MOVWF   Old1            ; |
        MOVF    Buffer2,W       ; |
        MOVWF   Old2            ; |
        MOVF    Buffer3,W       ; |
        MOVWF   Old3            ; |
        MOVF    Buffer4,W       ; |
        MOVWF   Old4            ; |
        MOVF    Buffer5,W       ; |
        MOVWF   Old5            ; |
        MOVF    Buffer6,W       ; |
        MOVWF   Old6            ; |
        MOVF    Buffer7,W       ; |
        MOVWF   Old7            ; |
        MOVF    Buffer8,W       ; |
        MOVWF   Old8            ; |
        MOVF    Buffer9,W       ; |
        MOVWF   Old9            ; |
        MOVF    BufferA,W       ; |
        MOVWF   OldA            ; |
        MOVF    BufferB,W       ; |
        MOVWF   OldB            ; |
        MOVF    BufferC,W       ; |
        MOVWF   OldC            ; |
        MOVF    BufferD,W       ; |
        MOVWF   OldD            ; |
        MOVF    BufferE,W       ; |
        MOVWF   OldE            ; |
        MOVF    BufferF,W       ; |
        MOVWF   OldF            ; |
        GOTO    BigLoop1        ; goto BigLoop1

        end
```

NOTES:

FSK Anticollision Reader Reference Design

1.0 INTRODUCTION

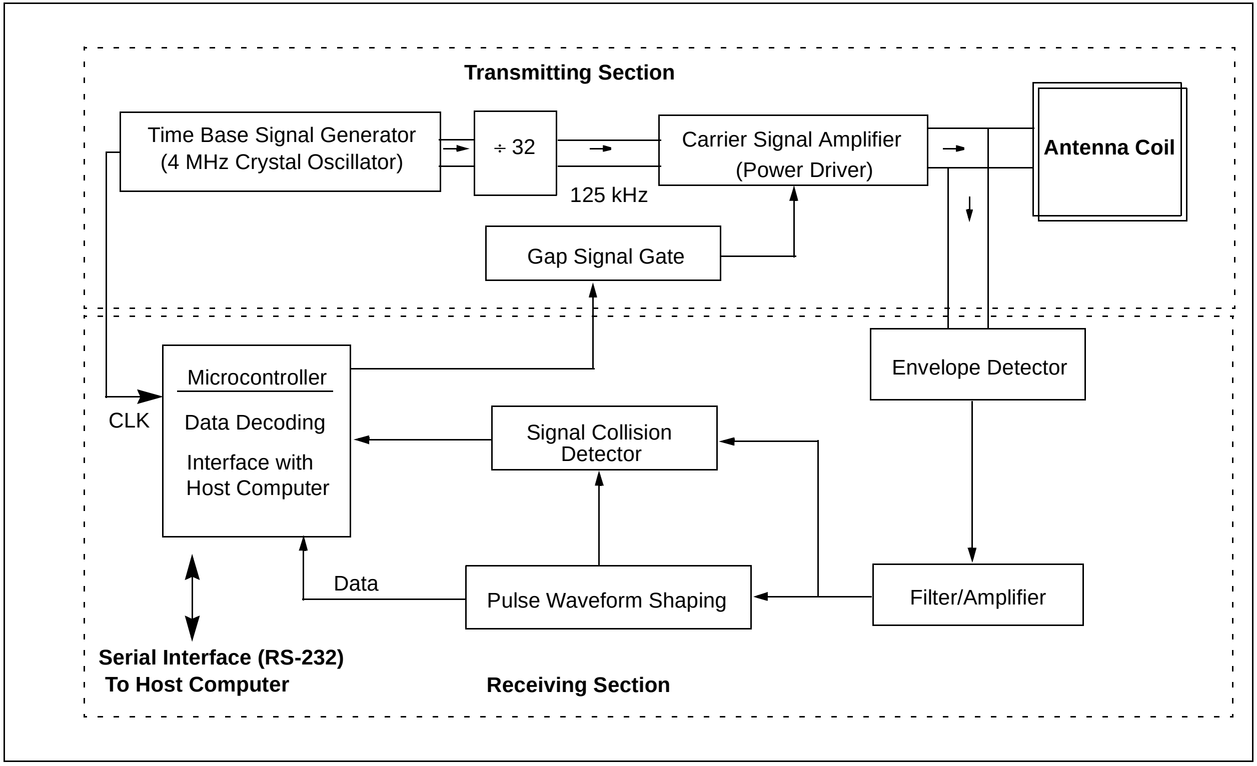
When more than one tag is in the same RF field of a reader, each tag will transmit data at the same time. This results in data collision at the receiving end of the reader. No correct decision can be made based on this data. The reader must receive data from a tag at a time for correct data processing.

The anti-collision device (MCRF250) is designed to send FSK data to reader without data collision, and it must be read by an anticollision reader. This type of device can be effectively used in inventory and asset control applications where multiple tags are read in the same RF field. The anti-collision algorithm of the device is explained in the *MCRF250 Data Sheet*, page 15.

This application note is written as a reference guide for anti-collision reader designers. The anticollision reader is designed to provide correct signals to the anticollision device (MCRF250) to perform an anticollision action during operation.

Microchip Technology Inc. provides basic anti-collision FSK reader electronic circuitry for the MCRF250 customers as a part of this design guide. The microID Anticollision Reader (demo unit), that can read 10 tags or more in the same RF field, is available in the microID Developers Kit (DV103002). An electronic copy of the microID PICmicro® source code is also available upon request.

FIGURE 1-1: BLOCK DIAGRAM OF TYPICAL RFID READER FOR FSK SIGNAL (125 KHZ)


Transmitting Section
Time Base Signal Generator (4 MHz Crystal Oscillator)
÷ 32
125 kHz
Carrier Signal Amplifier (Power Driver)
Antenna Coil
Gap Signal Gate
Envelope Detector
Microcontroller
Data Decoding
Interface with Host Computer
CLK
Signal Collision Detector
Data
Pulse Waveform Shaping
Filter/Amplifier
Serial Interface (RS-232) To Host Computer
Receiving Section

PICmicro is a registered tradmark of Microchip Technology Inc.

2.0 READER CIRCUITS

The anti-collision RFID reader consists of a transmitting and a receiving section. The transmitting section includes a carrier frequency generator, gap signal gate, and an antenna circuit. The receiving section includes peak detector, signal amplifier/filter, signal collision detector, and the microcontroller for data processing.

The reader also communicates with an external host computer. A basic block diagram of the typical RFID reader is shown in Figure 1-1.

The electronic circuitry for an anti-collision FSK reader is shown in Section 3.0. The reader needs a +9 VDC power supply.

The 125 kHz carrier signal is generated by dividing the 4 MHz time base signal that is generated by a crystal oscillator. A 16-stage binary ripple counter (74HC4060) is used for this purpose. The 74HC4060 also provides a clock signal for the PIC16C84 microcontroller. The 125 kHz signal from Pin no. 5 of U5 is fed into U1 (Nor gate) and two stage power amplifiers that are formed by U4, Q1, and Q2.

The 125 kHz signal from Q1 and Q2 is fed into the antenna circuit formed by L1(1.62 mH) and C22 (1000 pF). L1 and C22 form a series resonant circuit for a 125 kHz resonance frequency. Since the C22 is grounded, the carrier signal (125 kHz) is filtered out to ground after passing the antenna coil. The circuit provides a minimum impedance at the resonance frequency. This results in maximizing the antenna current, and therefore, the magnetic field strength is maximized.

The gap signal from Pin no. 7 of U6 (Microcontroller) controls the 125 KHz antenna driver circuit (Q1 and Q2). Q1 and Q2 are turned off during the gap signal “high”. There is no RF signal at the antenna coil during this gap period.

The reader circuit uses a single coil for both transmitting and receiving signals. L1, C22, D8, and the other components in the bottom parts of the circuit form a signal receiving section.

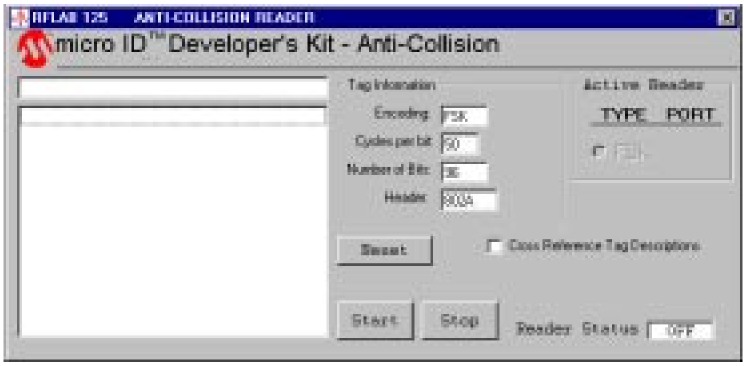
In the FSK communication protocol, a ‘0’ and a ‘1’ are represented by two different frequencies. In the MCRF250, a ‘0’ and a ‘1’ are represented by Fc/8 and Fc/10, respectively. Fc is the carrier frequency. The MCRF250 sends this FSK signal to the reader by an amplitude modulation of the carrier signal.

The demodulation is accomplished by detecting the envelope of the carrier signal. A half-wave capacitor-filtered rectifier circuit (D8 and C26) is used for the demodulation process. The detected envelope signal is charged into the C26. R37 provides a discharge path for the voltage charged in the C26. This voltage passes active filters (U11:B,D,C) and the pulse shaping circuitry (U11:A) before it is fed into the PIC16C84 for data processing.

When more than one tag are transmitting data at same time, there will be a waggle of data signals in the receiver. This waggle is detected in U7. If the waggle occurs, the Microcontroller turns on the gap control gate (U3:A) to send a gap signal to the tags.

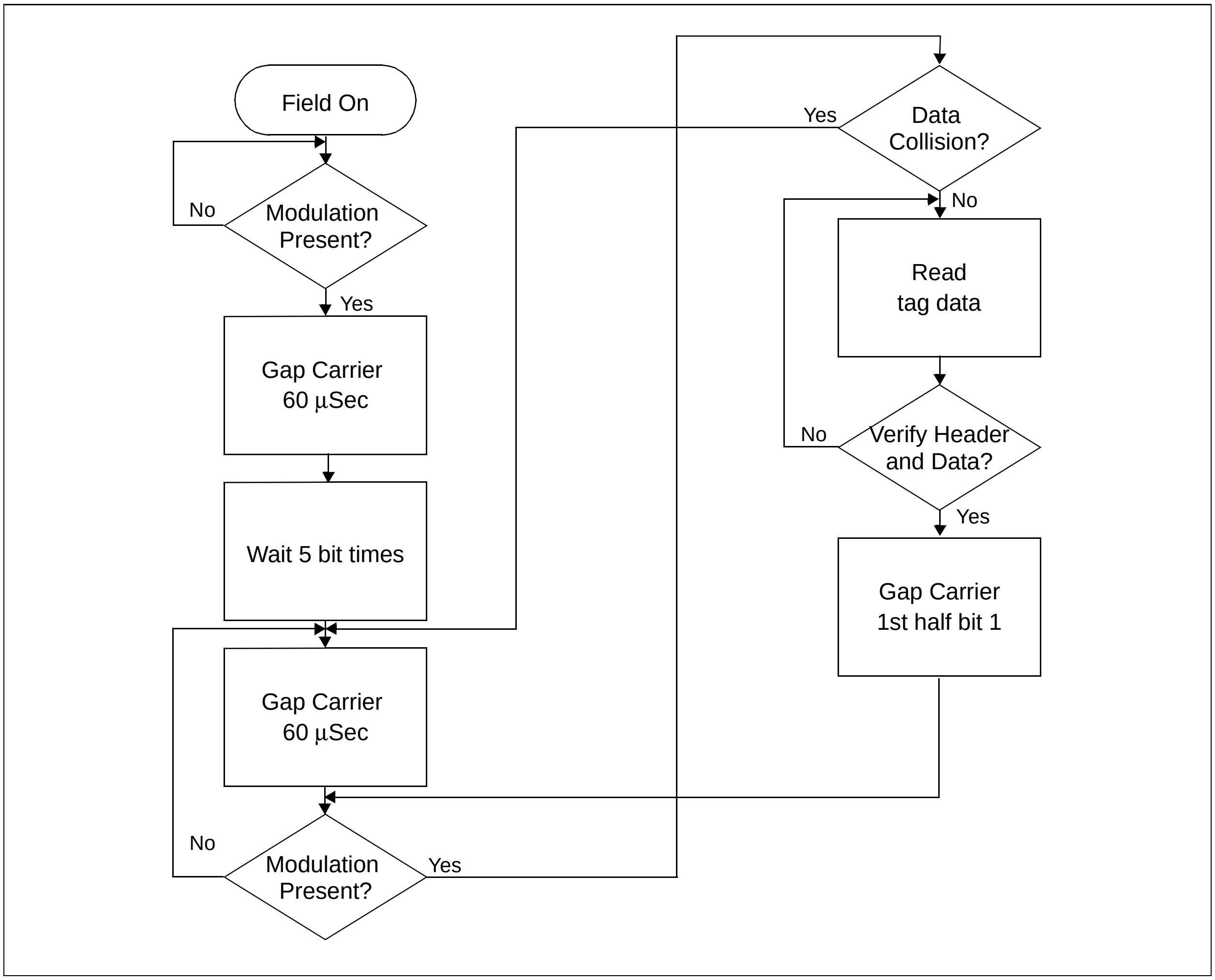
The PIC16C84 microcontroller performs data decoding, provides gap timing signals, and communicates with the host computer via an RS-232 serial interface.

FIGURE 2-1: RFID FSK ANTICOLLISION WINDOW

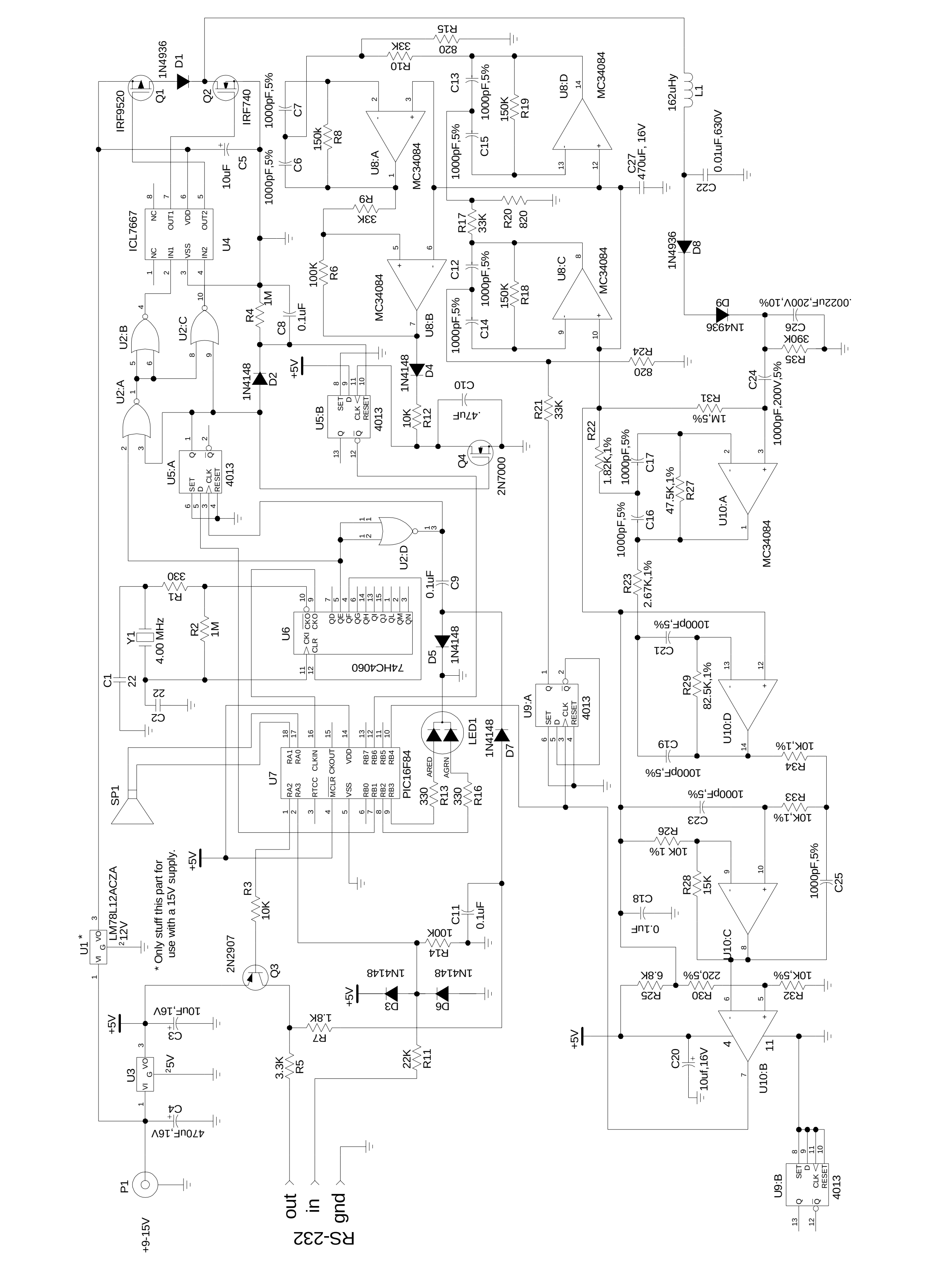

RFLAB 125 ANTI-COLLISION READER
micro ID™ Developer's Kit - Anti-Collision
Tag Information
Active Reader
TYPE PORT
Reset
Start
Stop

Encoding: FSK
Cycles per bit: 50
Number of Bits: 96
Header: 802A
☐ Cross Reference Tag Descriptions
Reader Status: OFF

FIGURE 2-2: ANTICOLLISION ALGORITHM FOR A MCRF250 READER



3.0 ANTICOLLISION READER SCHEMATIC



4.0 ANTICOLLISION READER BILL OF MATERIALS

Quantity	Type	Value	Reference Designator	Part Number
1	PIEZO Buzzer	PKM17EPP-4001	BZ1	MURATA PART #
2	Capacitor	22 pF	C1, C2	1330PH-ND
12	Capacitor	lOOO pF, 5%	C6,C7,C12,C13,C14C15,C16, C17,C19,C21,C23,C25	P4937-ND
1	Capacitor	1000 pF, 200V, 5%	C24	(P3497-ND)
1	Capacitor	.0022 µF, 200V, 10%	C26	P3S01-ND
1	Capacitor	0.01µF, 630V	c22	P3509-ND
4	Capacitor	0.1 µF	C8, C9, C11, C18	P4539-ND
1	Capacitor	0.47 µF	C10	P4967-ND
3	Capacitor	10 µF, 16V	C3, C5, C20	P6616-ND
1	Capacitor	l00 µF, 25V	C4	P10269-ND
1	Capacitor	470 µF, 16V	C27	P10247-ND
6	Diode	1N4148	D2, D3, D4, D5, D6, D7	1N4148DICT-ND
3	Diode	1N4936	D1, DB, D9	1N4936CT-ND
1	Bicolor LED	P392	LED 1	p392-ND
1	Coil Antenna	162 uBy	L1	Custom Wound
1	P-Chain MOSFET	IRF9520	Q1	IRF9520 FUTURE
1	N-Chain MOSFET	IRF740	Q2	IRF740-ND
1	PNP Transistor	2N2907	Q3	PN2907A-ND
1	N-Chain MOSFET	2N7000	Q4	2N7000DICT-ND
1	Resistor	220, 5%	R30	5043CX220ROJ
3	Resistor	330, 5%	R1, R13, R16	5043CX330ROJ
3	Resistor	820, 5%	R15, R20, R24	5043CX830ROJ
	Resistor	1.8K, 5%	R7	1K8CR-1/4W-B 5%
1	Resistor	1.82K, 1%	R22	1K82MF-1/4W-B 5%
1	Resistor	2.67K, 15	R23	2K67MF-1/4W-B 1%
1	Resistor	3.3K, 5%	R5	3K3CR-1/4W-B 5%
1	Resistor	6.8K, 5%	R25	6K8CR-1/4W-B 5%
3	Resistor	10R,1’	R26, R33, R34	5043ED10KOOF
3	Resistor	10K, 5%	R3, R12, R32	10KCR-1/4W-B 5%
1	Resistor	15K, 5%	R28	15KCR-1/4W-B 5%
1	Resistor	22K, 5%	R11	22KCR-1/4W-B 5%
4	Resistor	33K, 5%	R9,R10, R17, R21	33JCR1/4-B 5%
1	Resistor	47.5K, 1%	R27	47K5MF-1/4W-B 1%
1	Resistor	82.5K, 1%	R29	82.5KMF-1/4W-B 1%
2	Resistor	100K, 5%	R6, R14	5043CX100KOJ
3	Resistor	150K, 5%	R8, R18, R19	150KCR-1/4W-B 5%
1	Resistor	390K, 5%	R35	390KCR-1/4-B, 5%
3	Resistor	1M, 5%	R2, R4, R31	1MOCR-1/4W-B 5%
1	QUAD NOR GATE	74HC02	U2	MM74HC02N-ND
1	5V Regulator	LM78L05	U3	NJM78L05A-ND
1	MOSFET Driver	ICL7667	U4	ICL7667CPA-ND
2	DUAL FLIP-FLOP	4013	U5, U9	CD4013BCN-ND

Note: All resistors are 5% 1/4 watt carbon film resistors unless otherwise noted. DIGI-KEY part numbers follow some parts where applicable (these part numbers are only intended as a reference).

Quantity	Type	Value	Reference Designator	Part Number
1	Binary Counter	74HC4060	U6	MM74HC4060N-ND
1	Microprocessor	PIC16F84	U7	PIC16F84-04/P
2	OP-AMP	MC3407	U8, U10	FUTURE PART #
1	Crystal	4.00 MHz	Y1	X405-ND

Note: All resistors are 5% 1/4 watt carbon film resistors unless otherwise noted. DIGI-KEY part numbers follow some parts where applicable (these part numbers are only intended as a reference).

5.0 FSK ANTICOLLISION SOURCE CODE FOR THE PICmicro® MCU

The following source code is for the PIC16C84 microcontroller used in the FSK reader electronics.

```
; #=#=#=#=#=#=#=#=# PROJECT Microchip FSK anticollision Reader #=#=#=#=#=#=#=#=#
; v008.asm
; PIC16F84 running at 4MHz, Ti=1us

; ///////////////////////////////////////////////////////////////////////////
; Revision history
; ///////////////////////////////////////////////////////////////////////////
;
; Ver    Date        Comment
;
; 0.01   29 Dec 97   Copied from MChip\Reader\FSK
; 0.02   27 Feb 98   Gap during first half of first bit
; 0.05   28 Apr 98   Change from PIC16C84 to PIC16F84
; 0.06   29 Apr 98   Count to 256 instead of to 512
; 0.07   30 Apr 98   Make PORTB.0 low output (previously demodulated data input)
; 0.07a  08 May 98   Make gaps 80us wide
; 0.08   13 Aug 98   TAKE OUT CODE INTENDED FOR LAB USE ONLY
;
;       Tbit=50Tcy=400Ti
;       Ttag=96Tbit
;       Header=h’802A’
;

    processor pic16f84
    #include “p16f84.inc”
        __config b’00000000000001’
        ; Code Protect on, power-up timer on, WDT off, XT oscillator

#define _CARRY          STATUS,0
#define _ZERO           STATUS,2
#define _TO             STATUS,4
#define _RP0            STATUS,5
#define _PAGE0          PCLATH,3

#define _BUZZ1          PORTA,0
#define _BUZZ2          PORTA,1
#define _RS232TX        PORTA,2
#define _RS232RX        PORTA,3
#define _SDA            PORTA,4
StartPORTA      = b’11100’
StartTRISA      = b’01000’
BeepPort        = PORTA
Beep0           = StartPORTA
Beep1           = StartPORTA | b’00001’
Beep2           = StartPORTA | b’00010’

#define _UNUSED1        PORTB,0
#define _COIL_PWR       PORTB,1
#define _LED1           PORTB,2
#define _LED2           PORTB,3
#define _RAW_DATA       PORTB,4
#define _UNUSED2        PORTB,5
#define _COLLISION      PORTB,6 ; <  Goes low when a collision occurs
#define _SCL            PORTB,7
StartPORTB      = b’10000010’ ; Coil_Off
StartTRISB      = b’01010000’

StartOPTION     = b’10001111’
; PORTB pullups disabled, TMR0 internal, prescaler off, WDT/256

BO3             = h’0C’ ; Could be doubled-up with DelayReg1
DelayReg1       = h’0D’ ; Could be doubled-up with BO3
BitCtr          = h’0E’ ; Could be doubled-up with BeepCtrHi
```

```
TxByte          = h’0F’ ; Could be doubled-up with BeepCtrLo
TagsDetected    = h’10’
GapCountLo      = h’11’
Counter1        = h’12’
Counter2        = h’13’
Flags           = h’14’
#define _GotHeader      Flags,0
#define _FirstTime      Flags,1
Period          = h’15’ ; Used to read FSK
GapCountHi      = h’16’

Buffer00        = h’18’ ; --- IMMOBILE --- IMMOBILE --- IMMOBILE --- IMMOBILE
Buffer01        = h’19’ ; |
Buffer02        = h’1A’ ; |
Buffer03        = h’1B’ ; |
Buffer04        = h’1C’ ; |
Buffer05        = h’1D’ ; |
Buffer06        = h’1E’ ; |
Buffer07        = h’1F’ ; |
Buffer08        = h’20’ ; |
Buffer09        = h’21’ ; |
Buffer0A        = h’22’ ; |
Buffer0B        = h’23’ ; |
Buffer0C        = h’24’ ; |
Buffer0D        = h’25’ ; |
Buffer0E        = h’26’ ; |
Buffer0F        = h’27’ ; |
Buffer10        = h’28’ ; |
Buffer11        = h’29’ ; |
Buffer12        = h’2A’ ; |
Buffer13        = h’2B’ ; |
Buffer14        = h’2C’ ; |
Buffer15        = h’2D’ ; |
Buffer16        = h’2E’ ; |
Buffer17        = h’2F’ ; |

BeepCtrHi       = h’30’ ; Could be doubled-up with BitCtr
BeepCtrLo       = h’31’ ; Could be doubled-up with TxByte

SKIP macro
        BTFSC   PCLATH,7
 endm

Coil_On macro
        BCF     _COIL_PWR
 endm

Coil_Off macro
        BSF     _COIL_PWR
 endm

        org h’0000’             ; *#*#*#* RESET VECTOR *#*#*#*
        CLRF    PCLATH
        CLRF    INTCON
        CLRF    STATUS
        GOTO    RESET_A

        org h’0004’             ; *#*#*#* INTERRUPT VECTOR *#*#*#*
        CLRF    PCLATH
        CLRF    INTCON
        CLRF    STATUS
        GOTO    RESET_A

; ***** Subroutines, Page 0

Delay10:                        ;[0] Delay 10Ti
```

```
        GOTO    Delay08         ; |
Delay08:                        ;[0] Delay 8Ti
        GOTO    Delay06         ; |
Delay06:                        ;[0] Delay 6Ti
        NOP                     ; |
Delay05:                        ;[0] Delay 5Ti
        NOP                     ; |
Delay04:                        ;[0] Delay 4Ti
        RETLW   0               ; |

;%% CALL RS232CR takes 1052Ti
;%% CALL RS232TxDigit takes 1057Ti
;%% CALL RS232TxW takes 1049Ti
RS232CR:                        ;[1] Transmit CR on RS232
        MOVLW   d’13’           ; |
        GOTO    RS232TxW        ; |
RS232TxDigit:                   ;[1] Transmit LSnybble of W on RS232
        ANDLW   h’0F’           ; |
        MOVWF   TxByte          ; |
        MOVLW   h’A’            ; |
        SUBWF   TxByte,W        ; |
        MOVLW   ‘0’             ; |
        BTFSC   _CARRY          ; |
        MOVLW   ‘A’-h’A’        ; |
        ADDWF   TxByte,W        ; |
RS232TxW:                       ;[1] Transmit W on RS232 at 9615 baud
        MOVWF   TxByte          ; | TxByte=W
RS232Tx:                        ;[1] Transmit TxByte - 104us = 9615.4 baud
        BSF     _RS232TX        ; | Stop bit
        MOVLW   d’35’           ; | | Delay 106Ti
        MOVWF   DelayReg1       ; | | |
RS232TxD1:                      ; | | |
        DECFSZ  DelayReg1,f     ; | | |
        GOTO    RS232TxD1       ; | | |
        BCF     _RS232TX        ; | Start bit
        NOP                     ; | | Delay 98Ti
        MOVLW   d’32’           ; | | |
        MOVWF   DelayReg1       ; | | |
RS232TxD2:                      ; | | |
        DECFSZ  DelayReg1,f     ; | | |
        GOTO    RS232TxD2       ; | | |
        CLRF    BitCtr          ; | BitCtr=#8
        BSF     BitCtr,3        ; | |
RS232TxL1:                      ; | {% -4Ti
        BTFSC   TxByte,0        ; |   Transmit TxByte.0, RR TxByte
        GOTO    RS232TxBit1     ; |   |
        NOP                     ; |   |
RS232TxBit0:                    ; |   |
        BCF     _RS232TX        ; |   |
        BCF     _CARRY          ; |   |
        GOTO    RS232TxBitDone  ; |   |
RS232TxBit1:                    ; |   |
        BSF     _RS232TX        ; |   |
        BSF     _CARRY          ; |   |
        GOTO    RS232TxBitDone  ; |   |
RS232TxBitDone:                 ; |   |
        RRF     TxByte,f        ; |   |% 4Ti
        MOVLW   d’30’           ; |   Delay 93Ti
        MOVWF   DelayReg1       ; |   |
        GOTO    RS232TxD3       ; |   |
RS232TxD3:                      ; |   |
        DECFSZ  DelayReg1,f     ; |   |
        GOTO    RS232TxD3       ; |   |
        DECFSZ  BitCtr,f        ; |   DEC BitCtr
        GOTO    RS232TxL1       ; | } until (BitCtr==#0)
        CALL    Delay04         ; | Delay 4Ti
        BSF     _RS232TX        ; | stop bit
        RETLW   0               ; end

DelayTtag:                      ;[?] Delay Ttag-3Ti=38400-3Ti=38397Ti
        BSF     _PAGE0
        GOTO    P1DelayTtag

; ***** End of subroutines, Page 0

RESET_A:
        CLRWDT
                                ; Initialise registers
        CLRF    STATUS          ; | Access register page 0
        CLRF    FSR             ; | FSR=#0
        MOVLW   StartPORTA      ; | Initialise PORT and TRIS registers
        MOVWF   PORTA           ; | |
        MOVLW   StartPORTB      ; | |
        MOVWF   PORTB           ; | |
        BSF     _RP0            ;^| |
        MOVLW   StartTRISA      ;^| |
        MOVWF   TRISA           ;^| |
        MOVLW   StartTRISB      ;^| |
        MOVWF   TRISB           ;^| |
        BCF     _RP0            ; | Initialise OPTION register
        MOVLW   StartOPTION     ; | |
        CLRF    TMR0            ; | |
        BSF     _RP0            ;^| |
        MOVWF   OPTION_REG      ;^| |
        BCF     _RP0            ; | |

BigLoop1:
        CALL    Delay08         ; LEDs “reading”
        BSF     _LED1           ; |
        CALL    Delay08         ; |
        BCF     _LED2           ; |
        CALL    Delay08         ; |
        Coil_Off                ; Turn coil off

        BSF     _PAGE0
        GOTO    ResetDelay
ResetDelayDone:

        CLRF    TagsDetected    ; TagsDetected=#0
        CLRF    GapCountHi      ; GapCount=#0
        CLRF    GapCountLo      ; |
GapLoop:                        ; {
        Coil_Off                ;   Turn coil off
        CALL    Delay08         ;   LEDs “reading”
        BSF     _LED1           ;   |
        CALL    Delay08         ;   |
        BCF     _LED2           ;   |
        CALL    Delay10         ;   Wait 80us
        CALL    Delay10         ;   |
        CALL    Delay10         ;   |
        CALL    Delay10         ;   |
        CALL    Delay10         ;   |
        CALL    Delay10         ;   |
        NOP                     ;   |
        Coil_On                 ;   Turn coil on
        ;% 0 Ti from 1st bit


;(Ttag=38400Ti)
; If it’s the first gap since reset, delay Ttag

        BTFSC   _FirstTime
        CALL    DelayTtag
        BCF     _FirstTime


        CLRF    DelayReg1       ;   Delay 2047Ti
GapD1:                          ;   |
        CLRWDT                  ;   |
        DECFSZ  DelayReg1,f     ;   |
        GOTO    GapD1           ;   |
GapD2:                          ;   |
        CLRWDT                  ;   |
        DECFSZ  DelayReg1,f     ;   |
        GOTO    GapD2           ;   |
        ;% 2050Ti from 1st bit
        MOVLW   d’8’            ;   DelayReg1=#8
        MOVWF   DelayReg1       ;   |
        ;% 2052Ti from 1st bit
        ;% 2076-3*DelayReg1 from 1st bit
        ;% 5*400+76-3*DelayReg1 from 1st bit
        ;% 76-3*DelayReg1 Ti from 6th bit
                                ;   Read tag, with timeouts everywhere
        MOVLW   d’2’            ;   | Counter2=#2
        MOVWF   Counter2        ;   | |
ReadBit_L1:                     ;   | {% 78-3*DelayReg1 Ti from bit
        MOVLW   d’96’           ;   |   BitCtr=#96
        MOVWF   BitCtr          ;   |   |
ReadBit_L2:                     ;   |   {% 80-3*DelayReg1 Ti from bit
ReadBit_D1:                     ;   |     delay
        DECFSZ  DelayReg1,f     ;   |     |
        GOTO    ReadBit_D1      ;   |     |
                                ;   |    % 79Ti from bit
        CLRF    Counter1        ;   |     Counter1=#0
                                ;   |    % 80Ti=10Tcy from bit, time to start frequency sample
ReadBit_Hi0:                    ;   |     {% 80+(Counter1*8)Ti from bit
        INCF    Counter1,f      ;   |       ++Counter1
                                ;   |      % 73+(Counter1*8)Ti from bit
        BTFSC   Counter1,6      ;   |       if (timeout)
        GOTO    GapX            ;   |       { goto GapX } // could be at 1st half of 1st bit!!!
        NOP                     ;   |       |
        BTFSC   _RAW_DATA       ;   |     } until (_RAW_DATA==#1)
        BTFSS   _RAW_DATA       ;   |     |
        GOTO    ReadBit_Hi0     ;   |     |
        NOP                     ;   |     |
ReadBit_Lo0:                    ;   |     {% 80+(Counter1*8)Ti from bit
        INCF    Counter1,f      ;   |       ++Counter1
                                ;   |      % 73+(Counter1*8)Ti from bit
        BTFSC   Counter1,6      ;   |       if (timeout)
        GOTO    GapX            ;   |       { goto GapX } // could be at 1st half of 1st bit!!!
        NOP                     ;   |       |
        BTFSS   _RAW_DATA       ;   |     } until (_RAW_DATA==#0)
        BTFSC   _RAW_DATA       ;   |     |
        GOTO    ReadBit_Lo0     ;   |     |
        NOP                     ;   |     |
                                ;   |    % 80+(Counter1*8)Ti from bit
        MOVF    Counter1,W      ;   |     Period=Counter1
        MOVWF   Period          ;   |     |
        INCF    Counter1,f      ;   |     |
        CALL    Delay05         ;   |     |
ReadBit_Hi1:                    ;   |     {% 80+(Counter1*8)Ti from bit
        INCF    Counter1,f      ;   |       ++Counter1;
                                ;   |      % 73+(Counter1*8)Ti from bit
        BTFSC   Counter1,6      ;   |       if (timeout)
        GOTO    GapX            ;   |       { goto GapX } // could be at 1st half of 1st bit!!!
        NOP                     ;   |       |
        BTFSC   _RAW_DATA       ;   |     } until (_RAW_DATA==#1)
        BTFSS   _RAW_DATA       ;   |     |
        GOTO    ReadBit_Hi1     ;   |     |
        NOP                     ;   |     |
ReadBit_Lo1:                    ;   |     {% 80+(Counter1*8)Ti from bit
        INCF    Counter1,f      ;   |       ++Counter1;
                                ;   |      % 73+(Counter1*8)Ti from bit
        BTFSC   Counter1,6      ;   |       if (timeout)
        GOTO    GapX            ;   |       { goto GapX } // could be at 1st half of 1st bit!!!
        NOP                     ;   |       |
        BTFSS   _RAW_DATA       ;   |     } until (_RAW_DATA==#0)
        BTFSC   _RAW_DATA       ;   |     |
        GOTO    ReadBit_Lo1     ;   |     |
        NOP                     ;   |     |
ReadBit_Hi2:                    ;   |     {% 80+(Counter1*8)Ti from bit
        INCF    Counter1,f      ;   |       ++Counter1;
                                ;   |    % 73+(Counter1*8)Ti from bit
        BTFSC   Counter1,6      ;   |       if (timeout)
        GOTO    GapX            ;   |       { goto GapX } // could be at 1st half of 1st bit!!!
        NOP                     ;   |       |
        BTFSC   _RAW_DATA       ;   |     } until (_RAW_DATA==#1)
        BTFSS   _RAW_DATA       ;   |     |
        GOTO    ReadBit_Hi2     ;   |     |
        NOP                     ;   |     |
ReadBit_Lo2:                    ;   |     {% 80+(Counter1*8)Ti from bit
        INCF    Counter1,f      ;   |       ++Counter1;
                                ;   |      % 73+(Counter1*8)Ti from bit
        BTFSC   Counter1,6      ;   |       if (timeout)
        GOTO    GapX            ;   |       { goto GapX } // could be at 1st half of 1st bit!!!
        NOP                     ;   |       |
        BTFSS   _RAW_DATA       ;   |     } until (_RAW_DATA==#0)
        BTFSC   _RAW_DATA       ;   |     |
        GOTO    ReadBit_Lo2     ;   |     |
        NOP                     ;   |     |
                                ;   |    % 80+(Counter1*8)Ti from bit
        MOVF    Period,W        ;   |     Period=Counter1-Period
        SUBWF   Counter1,W      ;   |     |
        MOVWF   Period          ;   |     |
                                ;   |    % 83+(Counter1*8)Ti from bit
        COMF    Counter1,W      ;   |     W=32-Counter1
        ADDLW   d’1’            ;   |     |
        ADDLW   d’32’           ;   |     |
                                ;   |    % 86+(32-W)*8Ti from bit
                                ;   |    % 86+(Counter1*8)Ti from bit
        INCF    Counter1,f      ;   |     ++Counter1
        INCF    Counter1,f      ;   |     ++Counter1
        NOP                     ;   |     |
                                ;   |    % 89+(32-W)*8Ti from bit
                                ;   |    % 73+(Counter1*8)Ti from bit
        BTFSS   _CARRY          ;   |     if (W<0)
        GOTO    GapX            ;   |     { goto GapX } // could occur in 1st half of 1st bit!!!
                                ;   |    % 91+(32-W)*8Ti from bit
        MOVWF   Counter1        ;   |     Counter1=W
                                ;   |    % 92+(32-Counter1)*8 Ti from bit
ReadBit_D2:                     ;   |     Delay 4+Counter1*8 Ti
        MOVF    Counter1,f      ;   |     |
        BTFSC   _ZERO           ;   |     |
        GOTO    ReadBit_D2_done ;   |     |
        NOP                     ;   |     |
        NOP                     ;   |     |
        DECF    Counter1,f      ;   |     |
        GOTO    ReadBit_D2      ;   |     |
ReadBit_D2_done:                ;   |     |
                                ;   |    % 92+32*8-(oldCounter1)*8+4+(oldCounter1)*8 Ti from bit
                                ;   |    % 352Ti from bit
        BTFSS   _COLLISION      ;   |     if (collision occurred)
        GOTO    Gap1            ;   |     { goto Gap1 } // after 1st half of bit
        MOVF    Period,W        ;   |     if (Period<#14)
        ADDLW   low(0-d’14’)    ;   |     |
        BTFSS   _CARRY          ;   |     |
        GOTO    Gap0            ;   |     { goto Gap0 } // after 1st half of bit
        ADDLW   low(d’14’-d’18’);   |     if (Period<#18)
        BTFSS   _CARRY          ;   |     |
        GOTO    ReadBit_Got0    ;   |     { goto ReadBit_Got0 }
        ADDLW   low(d’18’-d’22’);   |     if (Period>=#22)
        BTFSC   _CARRY          ;   |     |
        GOTO    Gap0            ;   |     { goto Gap0 } // after 1st half of bit

ReadBit_Got1:                   ;   |    % 364Ti from bit
        BSF     _CARRY          ;   |     _CARRY=#1
        GOTO    ReadBit_GotBit  ;   |     goto ReadBit_GotBit

ReadBit_Got0:                   ;   |    % 362Ti from bit
        NOP                     ;   |     |
        NOP                     ;   |     |
        BCF     _CARRY          ;   |     |
        GOTO    ReadBit_GotBit  ;   |     |

ReadBit_GotBit:                 ;   |    % 367Ti from bit
        RLF     Buffer00,f      ;   |     roll in _CARRY
        RLF     Buffer01,f      ;   |     |
        RLF     Buffer02,f      ;   |     |
        RLF     Buffer03,f      ;   |     |
        RLF     Buffer04,f      ;   |     |
        RLF     Buffer05,f      ;   |     |
        RLF     Buffer06,f      ;   |     |
        RLF     Buffer07,f      ;   |     |
        RLF     Buffer08,f      ;   |     |
        RLF     Buffer09,f      ;   |     |
        RLF     Buffer0A,f      ;   |     |
        RLF     Buffer0B,f      ;   |     |
        RLF     Buffer0C,f      ;   |     |
        RLF     Buffer0D,f      ;   |     |
        RLF     Buffer0E,f      ;   |     |
        RLF     Buffer0F,f      ;   |     |
        RLF     Buffer10,f      ;   |     |
        RLF     Buffer11,f      ;   |     |
        RLF     Buffer12,f      ;   |     |
        RLF     Buffer13,f      ;   |     |
        RLF     Buffer14,f      ;   |     |
        RLF     Buffer15,f      ;   |     |
        RLF     Buffer16,f      ;   |     |
        RLF     Buffer17,f      ;   |     |
                                ;   |    % 391Ti from bit
                                ;   |    % -9Ti from bit (Tbit=400Ti)
        MOVLW   d’28’           ;   |     DelayReg1=#28
        MOVWF   DelayReg1       ;   |     |
                                ;   |    % -7Ti from bit
                                ;   |    % 77-3*DelayReg1 Ti from bit
        DECFSZ  BitCtr,f        ;   |     DEC BitCtr
        GOTO    ReadBit_L2      ;   |   } until (BitCtr==#0)
                                ;   |  % -5Ti from bit
        MOVLW   d’26’           ;   |   DelayReg=#26
        MOVWF   DelayReg1       ;   |   |
                                ;   |  % -3Ti from bit
                                ;   |  % 75-3*DelayReg1 Ti from bit
        DECFSZ  Counter2,f      ;   |   DEC Counter2
        GOTO    ReadBit_L1      ;   | } until (Counter2==#0)

                                ;  % -1Ti from first bit
        BSF     _PAGE0          ;   Delay 1568Ti
        GOTO    BigDelay        ;   |
BigDelayDone:                   ;   |% 1567Ti from first bit
```

```
CheckTtag:                      ;   if (tag is not 96 bits long) { goto Gap2 }
        MOVLW   Buffer00        ;   | FSR=#Buffer00
        MOVWF   FSR             ;   | |
        MOVLW   h’0C’           ;   | Counter1=h’0C’
        MOVWF   Counter1        ;   | |
CheckTTagLoop:                  ;   | {% 1571+(12-Counter1)*15Ti from first bit
        BTFSS   _COLLISION      ;   |   if (collision occurred)
        GOTO    Gap1            ;   |   { goto Gap1 } // never happens during first bit
        MOVF    INDF,W          ;   |   Counter2=INDF
        MOVWF   Counter2        ;   |   |
        MOVLW   h’0C’           ;   |   FSR=FSR+h’0C’
        ADDWF   FSR,f           ;   |   |
        MOVF    INDF,W          ;   |   if (Couter2!=INDF)
        XORWF   Counter2,W      ;   |   |
        BTFSS   _ZERO           ;   |   |
        GOTO    Gap2            ;   |   { goto Gap2 } // never happens during first bit
        MOVLW   low(0-h’0C’+1)  ;   |   FSR=FSR-h’0C’+1
        ADDWF   FSR,f           ;   |   |
        DECFSZ  Counter1,f      ;   |   DEC Counter1
        GOTO    CheckTTagLoop   ;   | } until (Counter1==#0)
                                ;  % 1570+12*15Ti = 1752Ti from first bit
HeadSearch:                     ;   if (no header in Buffer) { goto Gap2 }
        MOVLW   d’96’           ;   | set BitCtr
        MOVWF   BitCtr          ;   | |
HeadSearchL1:                   ;   | {% 1752+(96-BitCtr)*31 Ti from first bit
        BTFSS   _COLLISION      ;   |   if (collision occurred)
        GOTO    Gap1            ;   |   { goto Gap1 } // never happens during 1st bit
        BSF     _GotHeader      ;   |   if (header found) { goto HeadFound }
        MOVF    Buffer0B,W      ;   |   |
        XORLW   h’80’           ;   |   |
        BTFSS   _ZERO           ;   |   |
        BCF     _GotHeader      ;   |   |
        MOVF    Buffer0A,W      ;   |   |
        XORLW   h’2A’           ;   |   |
        BTFSS   _ZERO           ;   |   |
        BCF     _GotHeader      ;   |   |
        BTFSC   _GotHeader      ;   |   |
        GOTO    HeadFound       ;   |   |
        RLF     Buffer00,f      ;   |   ROL Buffer
        RLF     Buffer01,f      ;   |   |
        RLF     Buffer02,f      ;   |   |
        RLF     Buffer03,f      ;   |   |
        RLF     Buffer04,f      ;   |   |
        RLF     Buffer05,f      ;   |   |
        RLF     Buffer06,f      ;   |   |
        RLF     Buffer07,f      ;   |   |
        RLF     Buffer08,f      ;   |   |
        RLF     Buffer09,f      ;   |   |
        RLF     Buffer0A,f      ;   |   |
        RLF     Buffer0B,f      ;   |   |
        BCF     Buffer00,0      ;   |   |
        BTFSC   _CARRY          ;   |   |
        BSF     Buffer00,0      ;   |   |
        DECFSZ  BitCtr,f        ;   |   DEC BitCtr
        GOTO    HeadSearchL1    ;   | } until (BitCtr==#0)
                                ;   |% 1751+96*31 Ti = 4727Ti from first bit
        GOTO    Gap2            ;   | goto Gap2 // never happens during first bit
HeadFound:                      ;  % 1766+(96-BitCtr)*29 Ti from first bit
                                ;   Delay to fixed time
HeadDelay:                      ;   | {% 1766+(96-BitCtr)*31 Ti from first bit
        BTFSS   _COLLISION      ;   |   if (collision occurred)
        GOTO    Gap1            ;   |   { goto Gap1 } // never happens during 1st bit
        CALL    Delay08         ;   |   Delay 26Ti
        CALL    Delay08         ;   |   |
        CALL    Delay06         ;   |   |
        CALL    Delay04         ;   |   |
        DECFSZ  BitCtr,f        ;   |   DEC BitCtr
        GOTO    HeadDelay       ;   | } until (BitCtr==#0)
                                ;  % 1765+96*31 = 4741Ti from first bit

        BTFSS   _COLLISION      ;   if (collision occurred)
        GOTO    Gap1            ;   { goto Gap1 } // never happens during 1st bit

                                ;  % 4743Ti from first bit

        BSF     _LED2           ;   LEDs “Found tag”
        CALL    Delay08         ;   |
        BCF     _LED1           ;   |
                                ;  % 4753Ti from first bit

        SWAPF   Buffer0B,W      ;   Transmit tag ID
        CALL    RS232TxDigit    ;   |%% CALL RS232TxDigit takes 1057Ti
        MOVF    Buffer0B,W      ;   |
        CALL    RS232TxDigit    ;   |%% CALL RS232TxDigit takes 1057Ti
        SWAPF   Buffer0A,W      ;   |
        CALL    RS232TxDigit    ;   |%% CALL RS232TxDigit takes 1057Ti
        MOVF    Buffer0A,W      ;   |
        CALL    RS232TxDigit    ;   |%% CALL RS232TxDigit takes 1057Ti
        SWAPF   Buffer09,W      ;   |
        CALL    RS232TxDigit    ;   |%% CALL RS232TxDigit takes 1057Ti
        MOVF    Buffer09,W      ;   |
        CALL    RS232TxDigit    ;   |%% CALL RS232TxDigit takes 1057Ti
        SWAPF   Buffer08,W      ;   |
        CALL    RS232TxDigit    ;   |%% CALL RS232TxDigit takes 1057Ti
        MOVF    Buffer08,W      ;   |
        CALL    RS232TxDigit    ;   |%% CALL RS232TxDigit takes 1057Ti
        SWAPF   Buffer07,W      ;   |
        CALL    RS232TxDigit    ;   |%% CALL RS232TxDigit takes 1057Ti
        MOVF    Buffer07,W      ;   |
        CALL    RS232TxDigit    ;   |%% CALL RS232TxDigit takes 1057Ti
        SWAPF   Buffer06,W      ;   |
        CALL    RS232TxDigit    ;   |%% CALL RS232TxDigit takes 1057Ti
        MOVF    Buffer06,W      ;   |
        CALL    RS232TxDigit    ;   |%% CALL RS232TxDigit takes 1057Ti
        SWAPF   Buffer05,W      ;   |
        CALL    RS232TxDigit    ;   |%% CALL RS232TxDigit takes 1057Ti
        MOVF    Buffer05,W      ;   |
        CALL    RS232TxDigit    ;   |%% CALL RS232TxDigit takes 1057Ti
        SWAPF   Buffer04,W      ;   |
        CALL    RS232TxDigit    ;   |%% CALL RS232TxDigit takes 1057Ti
        MOVF    Buffer04,W      ;   |
        CALL    RS232TxDigit    ;   |%% CALL RS232TxDigit takes 1057Ti
        SWAPF   Buffer03,W      ;   |
        CALL    RS232TxDigit    ;   |%% CALL RS232TxDigit takes 1057Ti
        MOVF    Buffer03,W      ;   |
        CALL    RS232TxDigit    ;   |%% CALL RS232TxDigit takes 1057Ti
        SWAPF   Buffer02,W      ;   |
        CALL    RS232TxDigit    ;   |%% CALL RS232TxDigit takes 1057Ti
        MOVF    Buffer02,W      ;   |
        CALL    RS232TxDigit    ;   |%% CALL RS232TxDigit takes 1057Ti
        SWAPF   Buffer01,W      ;   |
        CALL    RS232TxDigit    ;   |%% CALL RS232TxDigit takes 1057Ti
        MOVF    Buffer01,W      ;   |
        CALL    RS232TxDigit    ;   |%% CALL RS232TxDigit takes 1057Ti
        SWAPF   Buffer00,W      ;   |
        CALL    RS232TxDigit    ;   |%% CALL RS232TxDigit takes 1057Ti
        MOVF    Buffer00,W      ;   |
        CALL    RS232TxDigit    ;   |%% CALL RS232TxDigit takes 1057Ti
;% 30145Ti from first bit

        CALL    RS232CR         ;%% CALL RS232CR takes 1052Ti
;% 31197Ti from first bit
```

```
        MOVLW   d’255’          ;   Delay 7396Ti
        MOVWF   DelayReg1       ;   |
WaitingL1:                      ;   |
        CLRWDT                  ;   |
        CALL    Delay10         ;   |
        CALL    Delay10         ;   |
        CALL    Delay05         ;   |
        DECFSZ  DelayReg1,f     ;   |
        GOTO    WaitingL1       ;   |

;% 38593Ti from first bit
;% 38400+193 = 193Ti from first bit, -7Ti from gap
        INCFSZ  GapCountLo,f    ;   INC GapCount
        SKIP                    ;   |
        INCF    GapCountHi,f    ;   |
        BTFSC   GapCountHi,0    ; } until (GapCount>#257)
        BTFSS   GapCountLo,1    ; |
        GOTO    GapLoop         ; |
        GOTO    BigLoop1

Gap1: ; !!!!! goto here after collision

                                ;  % -4Ti from gap
        CLRF    GapCountHi
        CLRF    GapCountLo
        GOTO    GapLoop

GapX:                           ;% 76+(Counter1*8)Ti from bit
GapXDelay:                      ; Delay 3+(128-Counter1)*8Ti
        BTFSC   Counter1,7      ; |
        GOTO    GapXDelayDone   ; |
        INCF    Counter1,f      ; |
        NOP                     ; |
        GOTO    GapXDelayJ1     ; |
GapXDelayJ1:                    ; |
        GOTO    GapXDelay       ; |
GapXDelayDone:                  ; |
                                ;% 76+(oldCounter1)*8+3+128*8-(oldCounter1)*8Ti from bit
                                ;% 1103Ti from bit = (400*2)+303Ti from bit
                                ;// Not in first half of bit
Gap0: ; !!!!! goto here for gap which does NOT occur in first half of first bit
                                ;  % -7Ti from gap
        INCFSZ  GapCountLo,f    ;   INC GapCount
        SKIP                    ;   |
        INCF    GapCountHi,f    ;   |

        BTFSC   GapCountHi,0    ; } until (GapCount>#257)
        BTFSS   GapCountLo,1    ; |
        GOTO    GapLoop         ; |
        GOTO    BigLoop1

Gap2: ; !!!!! goto here for valid FSK but invalid code
        INCFSZ  GapCountLo,f    ;   INC GapCount
        SKIP                    ;   |
        INCF    GapCountHi,f    ;   |
        BTFSC   GapCountHi,0    ; } until (GapCount>#257)
        BTFSS   GapCountLo,1    ; |
        GOTO    GapLoop         ; |
        GOTO    BigLoop1

        org h’0200’
P1Delay20:
        GOTO    P1Delay18
P1Delay18:
        NOP
```

```
P1Delay17:
        NOP
P1Delay16:
        GOTO    P1Delay14
P1Delay14:
        NOP
P1Delay13:
        NOP
P1Delay12:
        GOTO    P1Delay10
P1Delay10:
        GOTO    P1Delay08
P1Delay08:
        GOTO    P1Delay06
P1Delay06:
        GOTO    P1Delay04
P1Delay04:
        RETLW   0

BigDelay:
;!!!!! delay (1568-6)Ti = 1562Ti

        MOVLW   d'15'           ; Delay 1501Ti
        MOVWF   DelayReg1       ; |
BigDelayL1:                     ; |
        CALL    P1Delay20       ; |
        CALL    P1Delay20       ; |
        CALL    P1Delay20       ; |
        CALL    P1Delay20       ; |
        CALL    P1Delay17       ; |
        DECFSZ  DelayReg1,f     ; |
        GOTO    BigDelayL1      ; |
        CALL    P1Delay20       ; Delay 61Ti
        CALL    P1Delay20       ; |
        CALL    P1Delay20       ; |
        NOP                     ; |

        BCF     _PAGE0
        GOTO    BigDelayDone


P1DelayTtag:                     ; Delay 38393Ti
        CLRF    DelayReg1       ; | Delay 38144Ti
P1DelayTtagL1:                  ; | |
        CALL    P1Delay20       ; | |
        CALL    P1Delay20       ; | |
        CALL    P1Delay20       ; | |
        CALL    P1Delay20       ; | |
        CALL    P1Delay20       ; | |
        CALL    P1Delay20       ; | |
        CALL    P1Delay20       ; | |
        CALL    P1Delay06       ; | |
        DECFSZ  DelayReg1,f     ; | |
        GOTO    P1DelayTtagL1   ; | |
        MOVLW   d'19'           ; | Delay 248Ti
        MOVLW   DelayReg1       ; | |
P1DelayTtagL2:                  ; | |
        CALL    P1Delay10       ; | |
        DECFSZ  DelayReg1,f     ; | |
        GOTO    P1DelayTtagL2   ; | |
        NOP                     ; | Delay 1Ti
        BCF     _PAGE0
        RETLW   0

ResetDelay:
        CALL    RS232CR         ; Transmit CR regularly
        MOVLW   d’4’            ; Beep at 3597Hz for 1024 cycles
        MOVWF   BeepCtrHi       ; |
        MOVLW   d’0’            ; |
        MOVWF   BeepCtrLo       ; |% 27277Ti from first bit
BeepLoopJ1:                     ; |
        GOTO    BeepLoopJ2      ; |
BeepLoopJ2:                     ; |
        MOVLW   Beep1           ; |
        MOVWF   BeepPort        ; |
        MOVLW   d’34’           ; | Delay 137Ti
        MOVWF   DelayReg1       ; | |
BeepD1:                         ; | |
        CLRWDT                  ; | |
        DECFSZ  DelayReg1,f     ; | |
        GOTO    BeepD1          ; | |
        MOVLW   Beep2           ; |
        MOVWF   BeepPort        ; |
        MOVLW   d’32’           ; | Delay 132Ti
        MOVWF   DelayReg1       ; | |
        NOP                     ; | |
        GOTO    BeepD2          ; | |
BeepD2:                         ; | |
        CLRWDT                  ; | |
        DECFSZ  DelayReg1,f     ; | |
        GOTO    BeepD2          ; | |
        DECFSZ  BeepCtrLo,f     ; |
        GOTO    BeepLoopJ1      ; |
        DECFSZ  BeepCtrHi,f     ; |
        GOTO    BeepLoopJ2      ; |
        NOP                     ; |
        MOVLW   Beep0           ; |
        MOVWF   BeepPort        ; |

        MOVLW   d’20’           ; Wait ~10ms (reset gap)
        MOVWF   Counter1        ; |
ResetGapL1:                     ; |
        MOVLW   d’124’          ; | Wait ~500us
        MOVWF   DelayReg1       ; | |
ResetGapL2:                     ; | |
        CLRWDT                  ; | |
        DECFSZ  DelayReg1,f     ; | |
        GOTO    ResetGapL2      ; | |
        DECFSZ  Counter1,f      ; |
        GOTO    ResetGapL1      ; |
        BSF     _FirstTime

        Coil_On                 ; Turn coil on
        MOVLW   d’6’            ; Wait ~6ms
        MOVWF   Counter1        ; |
ResetDelayL1:                   ; |
        MOVLW   d’250’          ; |
        MOVWF   DelayReg1       ; |
ResetDelayL2:                   ; |
        CLRWDT                  ; |
        DECFSZ  DelayReg1,f     ; |
        GOTO    ResetDelayL2    ; |
        DECFSZ  Counter1,f      ; |
        GOTO    ResetDelayL1    ; |
        BCF     _PAGE0
        GOTO    ResetDelayDone

        end
```

Using the microID™ Programmer

1.0 INTRODUCTION

The following is a description of how to program Microchip's MCRF2XX family of RFID products. A contactless programmer (PG103001), user interface software (RFLAB™), and a host computer are needed to program the MCRF2XX devices. The device can also be programmed in a standard terminal mode (i.e., c:\windows\terminal.exe) rather than the RFLAB. See Figure 5-1 for the programming sequence.

The microID programmer requires an external power supply (+9 VDC, >750 mA). The RFLAB software runs under Microsoft® (MS) Windows® 95 environment only. The programmer communicates with a host computer via an RS-232 serial interface at 9600 baud, 8 data bits, 1 stop bit, and no parity.

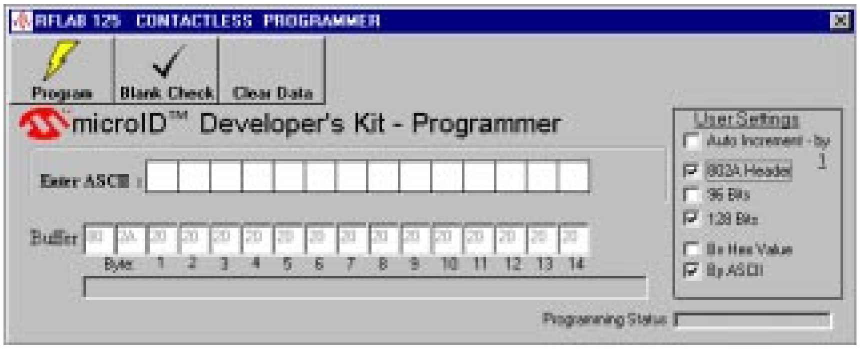
Since the MCRF2XX is a One-Time-Programmable (OTP) device, only a blank (unlocked) device can be programmed by the programmer. Therefore, the programmer first checks the status of the memory in the device before initiating programming. A blank device contains an array of all '1's.

The device can be programmed with 16 bytes (128 bits) or 12 bytes (96 bits) of data length. Once the MCRF2XX enters its programming mode, it sets a lock bit at the same time. If the programming is interrupted for any reason during the programming period, the programming will be stopped, and the device may be left partially programmed. The device will still be locked even though it has not been programmed completely. In this case, the programmer will return a fail code to the host computer.

Any device that has been programmed, either fully or partially, will remain in a locked status; therefore, it cannot to be reprogrammed. If programming has been successfully completed, the programmer will return a verification code to the host computer.

In order to program the MCRF2XX device, it is necessary to provide a proper programming signal level to the device. The device requires specific peak-to-peak voltages for programming. Since the voltage induced in the tag coil varies depending on the coil parameters, the output signal level of the programmer must be calibrated to provide a proper programming signal level at the tag coil. A detailed calibration procedure is described in Section 3.0.

FIGURE 1-1: RFLAB SOFTWARE RUNNING UNDER MS WINDOWS 95


RFLAB 125 CONTACTLESS PROGRAMMER
Program
Blank Check
Clear Data
microID™ Developer's Kit - Programmer

Enter ASCII :

Buffer: 80 2A 20 20 20 20 20 20 20 20 20 20 20 20 20 20

Byte: 1 2 3 4 5 6 7 8 9 10 11 12 13 14

User Settings

☐ Auto Increment - by 1

☑ 802A Header

☐ 96 Bits

☑ 128 Bits

☐ By Hex Value

☑ By ASCII

Programming Status:

RFLAB is a trademark of Microchip Technology Inc.

2.0 PROGRAMMING SIGNAL WAVEFORM

Figure 2-1 shows the waveform of the programming signal. Once the programmer sends a power-up and gap signal to the device, the device transmits back a verification bitstream in FSK. The verification signal represents the contents of the memory in the device. The blank device has all ‘1’s in its memory. A bit ‘1’ in FSK is represented by a low signal level for five cycles and a high signal level for an additional five cycles (Figure 2-1).

The device will respond with a nonmodulated (no data) signal if the device has not recognized the power-up signal. In this case, the power-up signal level should be calibrated to provide a proper signal level to the device. The calibration procedure is explained in Section 3.0.

After the device is verified as blank, the programmer sends a programming signal to the device. The programming data is represented by an amplitude modulation signal. Therefore, bit ‘1’ and ‘0’ are represented by a low-power (level) signal and a high power (level) signal, respectively, as shown in Figure 2-1. Each data bit is represented by 128 cycles of the carrier signal. An MCRF200 configured for 128 bits uses all bits in the transfer; an MCRF200 configured for 96 bits ignores bits 33 through 64, although they are present in the programming sequence. Therefore, for a 125 kHz carrier signal, it takes 1.024 ms for one data bit (128 cycles x 8 µs/cycles) and 131.072 ms for 128 data bits (128 cycles/bit x 8 µs/cycle x 128 bits).

A guard-band of Δt = 10 cycles (80 µs) should be kept at each end of a high-power (0) bit as shown in Figure 2-1. This is to prevent accidental programming or disturbing of adjacent bits in the array.

The memory array is locked at the start of the programming cycle. Therefore, when the device leaves the programming field, it locks the memory permanently, regardless of the programming status. The device should not be interrupted during the programming cycle.

The device transmits the programmed (data contents) circuits back to the programmer for verification. If the verification bitstream is correct, the programmer sends a verified signal (‘v’) to the host computer; otherwise, it sends an error message (‘n’, see Figure 5-1).

The programming signal level must be within a limit of the programming voltage window for successful programming. The calibration of the signal level is explained in Section 3.0.

2.1 Power-up, Gap, and Verification Signals

The programming signal starts with a power-up signal for 80 ~ 180 µS, followed by a gap signal (0 volt) for 50 ~ 100 µS. The purpose of these signals is to check whether the device is blank and establish a programming mode in the device. Once the device recognizes the power-up signal, it transmits back the contents of its memory. If the device transmits back with the blank bitstream (FSK with all ‘1’s), it is ready to be programmed. If the device is not blank, the programmer informs the host computer that it is nonprogrammable.

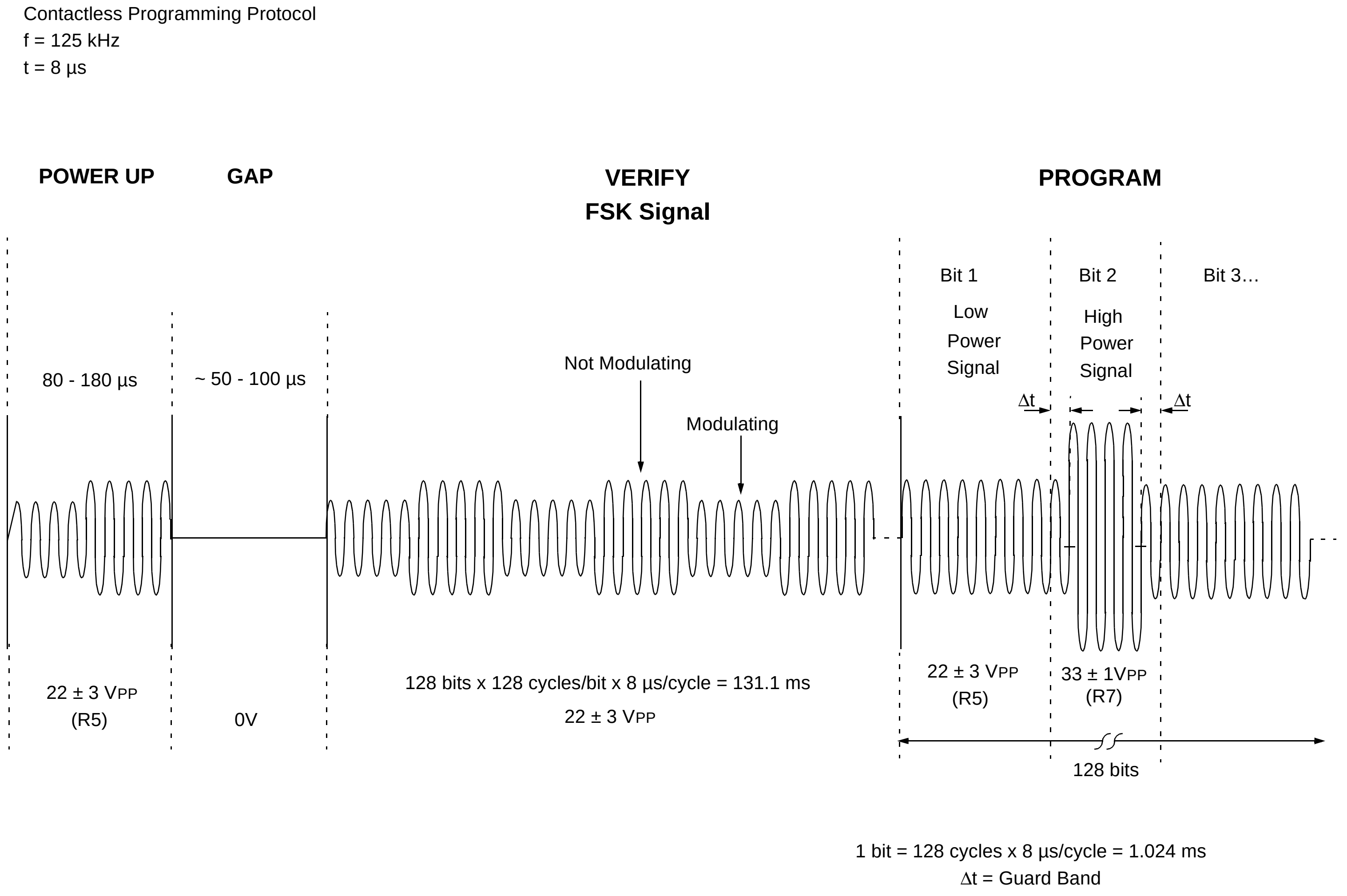
If the power-up signal level is out of the programming voltage range, the device will transmit back a non-modulated signal (no data). The nonmodulated signal has no variation in the amplitude (constant voltage signal). A variable resistor, R5 in the microID programmer, should be adjusted to provide a proper power-up signal level. A typical signal level is about 22 ± 3 VPP across the tag coil. This calibration procedure is described in Section 3.0.

2.2 Programming Sequence

Once the device has been verified blank for programming, the programmer sends a programming sequence to the device. The programming data entered in the RFLAB software is sent to the device via the programmer. The programming signal waveforms are shown in Figure 2-1. One bit of data is represented by 128 cycles of the carrier signal. It takes 131.072 ms to complete one programming cycle for the total of 128 data bits. An MCRF200 configured for 128 bits uses all bits in the transfer; an MCRF200 configured for 96 bits ignores bits 33 through 64, although they are present in the programming sequence. After the programming sequence, the device transmits back a verification bitstream. The programmer reports to the host computer the status of the programming.

The data is programmed only if the programming signal level is within the limit in the programming voltage requirement of the device. It takes several programming/verify cycles to completely program each bit of the MCRF200. The microID programmer uses ten (10) blind program/verify cycles before checking the final verify sequence for correct programming. Faster programmers can be designed by checking each program/verify cycle; after approximately 3 ~ 5 cycles, the device will verify correctly. Once a correct verify sequence is received, one additional program cycle should be run to ensure proper programming margin.

FIGURE 2-1: CONTACTLESS PROGRAMMING WAVEFORM



Default programming protocol = FSK, Fc/8/10, 128 bits
For 96-bit programming, bits 33 – 64 are ‘don’t care’, but all 128-bit cycles must be in the sequence.

Note: Low-power signal leaves bit = 1
High-power signal programs bit = 0

3.0 CALIBRATION OF PROGRAMMING VOLTAGE

If you are using your own tag coil (with resonant capacitor) with the MCRF200 or MCRF250, you may need to calibrate the programmer for your circuit. Follow this procedure, if you are unable to program your tag.

a) Open the programmer, and turn R5 and R6 full counter-clockwise. Remove the four screws at the back of the programmer.

b) Set up the programmer and calibration tag as shown in Figure 3-1.

Set Up:

- Connect the +9 VDC power supply to the programmer.
- Connect the RS-232 cable from the external serial port in the programmer box to a COM port in the host.
- Open up the RFLAB software on the host computer.
- Place the calibration tag in the center of the tag area on the programmer. A calibration tag is any tag using MCRF200 or MCRF250 silicon and your own coil and capacitor.

c) Run the programming software (RFLAB).

Power-up Signal Level:

d) Click the **Blank Check** button in the RFLAB software.

If the device is blank, a green bar appears in the window with a message indicating that it is blank. If the device is not blank or the power-up signal is out of range, a red bar appears in the window with an error message indicating that it is not blank. The variable resistor (R5) in the programmer should be adjusted to provide a proper “low-power” voltage level to the tag coil. A typical signal level is about 22 ± 3 VPP at the tag coil, but it can vary outside of this range.

R5: Turn clockwise in 1/16-inch increments

Repeat step (d) while adjusting R5. Once the device has been verified as a blank, turn it clockwise one more increment. Then move to the next step.

Programming Signal Level:

e) Click on the buttons in RFLAB for the appropriate data type and protocol for your tag.

f) Enter the programming data in the text box.

g) Click the **Program** button. This will send the programming data to the device. A typical signal level for programming is 33± 1 VPP at the tag coil, but can vary outside of this range.

h) After the device has been programmed, it transmits back the programmed data for verification.

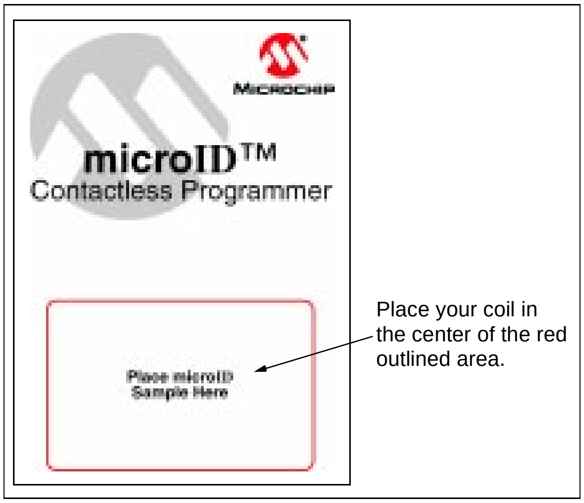
i) If the data has been programmed correctly, a green bar will appear for a few seconds with a message indicating *Programming successful*.

If the programming has been unsuccessful due to insufficient programming signal levels, a message indicating *Programming unsuccessful* will appear in the RFLAB. See Figure 1-1. In this case, R7 (“High Power”) must be adjusted to provide a proper programming signal level to the tag coil. Turn R7 clockwise in 1/16-inch increments, repeating steps (f) through (h) until programming is successful. Then turn R7 clockwise one more increment.

> **Note:** The MCRF200 or MCRF250 lock may be locked even if the programming cycle was unsuccessful; therefore, a new MCRF200 sample may be required for each pass through steps (f) through (h).

j) After programming is completed successfully, keep these R5 and R7 settings for future programming of your tags. Once this calibration has been done, remove the calibration tag from the programmer and reinstall the four screws.

FIGURE 3-1: MCRF2XX microID PROGRAMMER AND CALIBRATION TAG COIL ARRANGEMENT FOR PROGRAMMING SIGNAL LEVEL MEASUREMENT



Place your coil in the center of the red outlined area.

4.0 PROGRAMMING PROCEDURE

a) Set up the programmer and open up the RFLAB software on the host computer.

Set Up:

- Connect the +9 VDC power supply to the programmer.
- Connect from the external serial port in the programmer box to a COM port in the host computer using the RS-232 cable.

b) Place the RFID device at the center of the programmer.

c) Click **Blank Check** button if you want to check whether the device is blank. This button can also be used to verify that the device is assembled properly.

> **Note:** The device can’t be programmed unless it is blank.

d) Enter the programming data in the RFLAB and select appropriate data type.

e) If several devices are going to be programmed sequentially by any number, click the **Auto Increment** button and specify the increment number.

f) Click the **Program** button. This will send the data to the device.

g) If the data has been programmed correctly, there will be a green bar with a message indicating *Programming successful*.

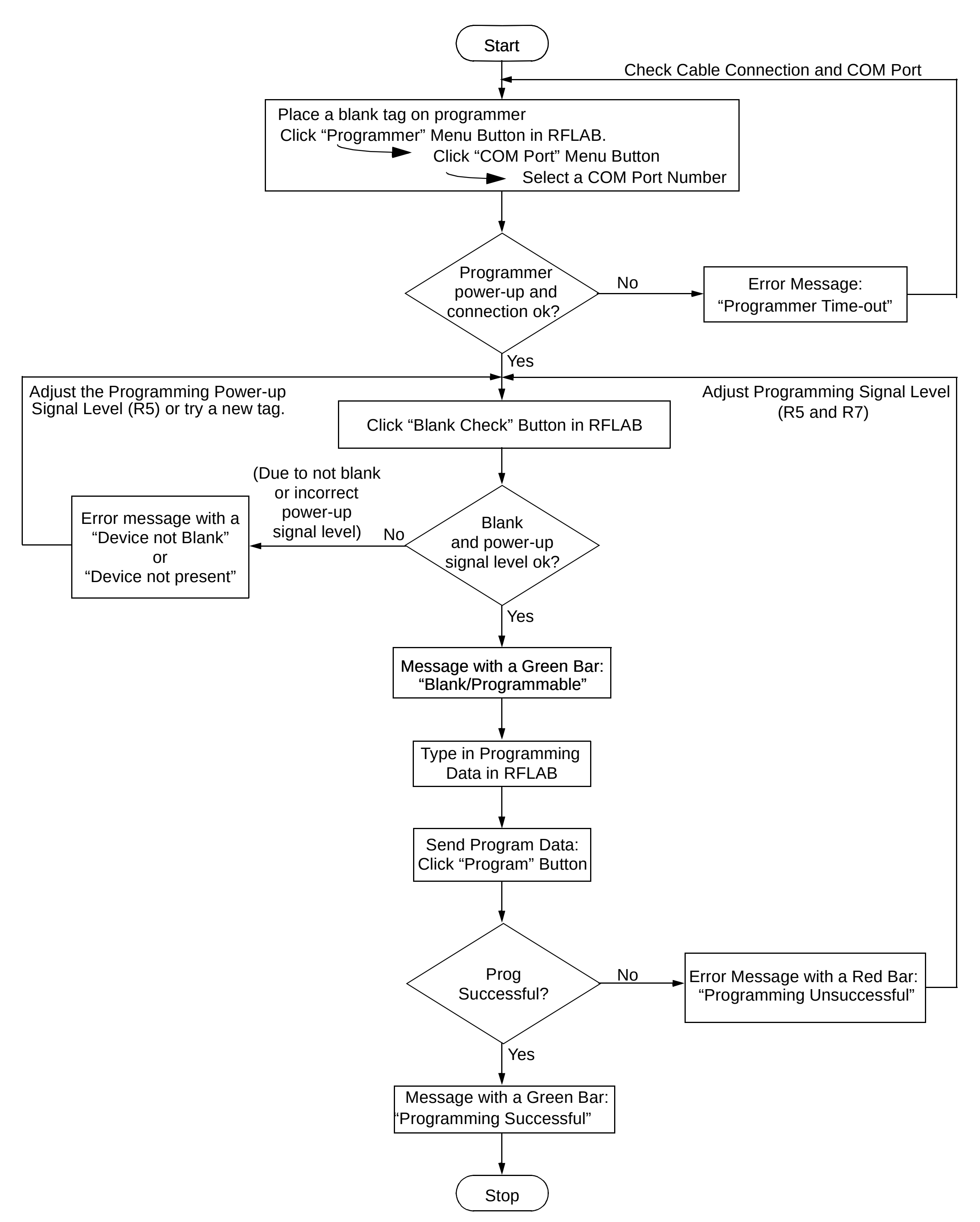
If the programming has been unsuccessful due to out-of-range in the programming signal level, a message and red bar will show up indicating *Programming unsuccessful.* In this case, the programming signal voltage may need to be calibrated for your tag. See the calibration procedure for the programming signal level in the previous section.

h) Repeat step (a) through (g) for other tags.

4.1 Error Conditions

If the host computer does not send programming data to the programmer for more than 3 seconds, the programmer will timeout and reset. If the programmer does not respond to the host computer, there will be an error message indicating *Programmer time out*. If invalid programming data is sent to the programmer during the loading of the program buffer, the programmer will return a message indicating *Invalid.*

FIGURE 4-1: PROGRAMMING FLOWCHART USING RFLAB


Start
Check Cable Connection and COM Port
Place a blank tag on programmer
Click “Programmer” Menu Button in RFLAB.
Click “COM Port” Menu Button
Select a COM Port Number
Programmer power-up and connection ok?
No
Error Message: “Programmer Time-out”
Yes
Adjust the Programming Power-up Signal Level (R5) or try a new tag.
Adjust Programming Signal Level (R5 and R7)
Click “Blank Check” Button in RFLAB
(Due to not blank or incorrect power-up signal level)
No
Error message with a “Device not Blank” or “Device not present”
Blank and power-up signal level ok?
Yes
Message with a Green Bar: “Blank/Programmable”
Type in Programming Data in RFLAB
Send Program Data: Click “Program” Button
Prog Successful?
No
Error Message with a Red Bar: “Programming Unsuccessful”
Yes
Message with a Green Bar: “Programming Successful”
Stop

5.0 PROGRAMMING IN A STANDARD TERMINAL MODE

In special cases, the device can also be programmed in a standard terminal mode by executing the `terminal.exe` program (c:\windows\terminal.exe) or by any customer production software. The programmer setup, signal waveforms, and calibration procedure are the same as programming with the RFLAB.

The following is a description of how to interface a host computer to Microchip’s contactless programmer without the use of RFLAB software. The programmer will check for a blank, unlocked MCRF2XX tag before initiating programming. Once programming has been completed, the programmer will return a pass or fail code. The programmer communicates at 9600 baud, 8 data bits, 1 stop bit, and no parity.

Figure 5-1 shows the programming flow and communication handshakes between host and programmer.

5.1 Programmer Wake-up

Sending an ASCII ‘W’ (57h) to the programmer on the RS-232 interface will tell the programmer to wake up and be prepared to receive commands. The programmer will reply with ASCII ‘R’ (52h) when it is ready.

5.2 Blank Check

Sending an ASCII ‘T’ (54h) will signal the programmer to read the part about being contactlessly programmed and check to see if it is blank (all 1’s) and unlocked. If the part is blank and unlocked, the programmer will reply with an ASCII ‘Y’ (59h) to signify programming should continue. If the part is not blank or not unlocked, the programmer will reply with an ASCII ‘N’ (4Eh) to indicate an error. It is always necessary to perform a blank check before programming MCRF2XX devices.

5.2.1 SENDING DATA TO THE PROGRAMMER

If the programmer responds with an ASCII ‘Y’, indicating that the part is blank, the PC can begin passing the 16 bytes of required data to the programmer data buffer. AnMCRF200 configured for 128 bits uses all 16 bytes of data in the transfer; when programming a 96-bit device, however, bits 33 through 64 are ‘don’t care’ and are ignored by the MCRF200. The data should be passed in ASCII equivalent hex bytes and the programmer will acknowledge the receipt of each byte by echoing back what it has received. For example, to program 05 hex data into the first byte, the PC would send ASCII ‘0’ (30h), the programmer would echo ‘0’ back. Next, the programmer would send ASCII ‘5’ (35h), and the programmer will echo back ‘5’. All of the data must be sent in UPPERCASE ASCII equivalent only. See Figure 5-1 for a typical programming sequence.

5.3 Program and Verify the Device

After 16 bytes of data have been received by the programmer, it is ready to begin programming the data buffer into the MCRF2XX. Sending an ASCII ‘V’ (56h) will tell the programmer to program the 16 bytes it has received and verify that the device has programmed properly. When the device programs properly, the programmer replies with ASCII ‘y’ (79h). If the programming was not successful, the programmer replies with ASCII ‘n’ (6Eh). A successful programming operation should take about 3 to 4 seconds per device.

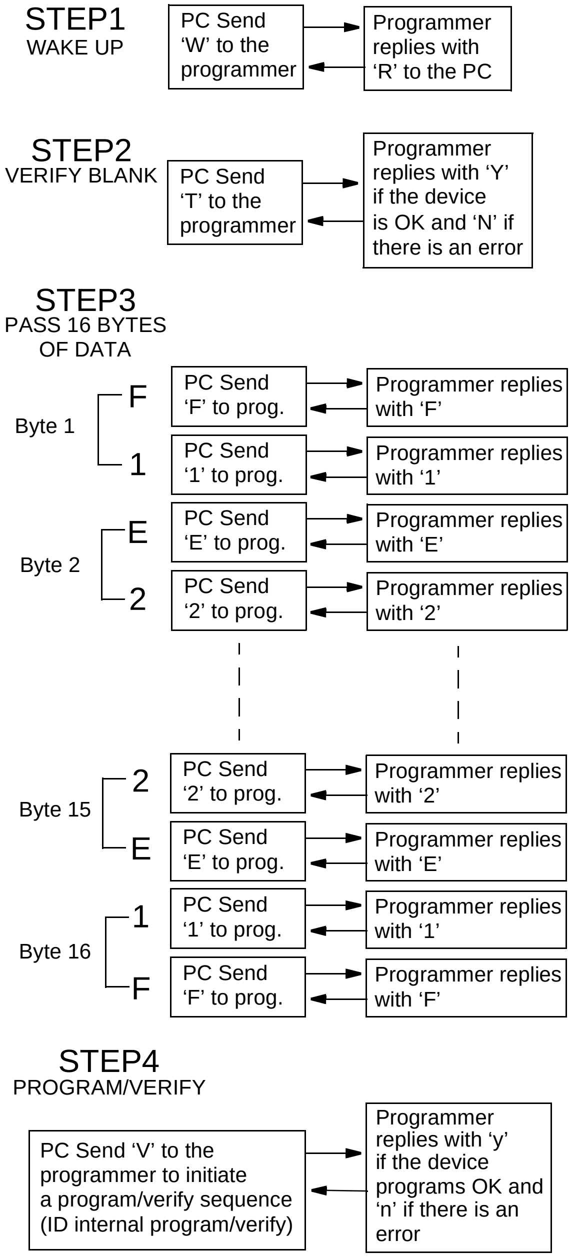
5.4 Error Conditions

If the PC does not send a byte to the programmer for more than 3 seconds, the programmer will timeout and reset. The entire programming sequence will need to be repeated, beginning with the programmer wake-up byte ASCII ‘W’.

If invalid bytes are sent to the programmer during the loading of the program buffer, the programmer will return an ASCII ‘I’ (49h). In this case, the entire programming sequence must be repeated, beginning with the programmer wake-up byte ASCII ‘W’.

FIGURE 5-1: TYPICAL SEQUENCE

The following is the programming sequence necessary to wake up the programmer, check if a MCRF2XX part is blank, unlocked and ready to be programmed, send F1E2D3C4B5A6978888796A5B4C3D2E1F ASCII data to the programmer, and instruct the programmer to program and verify the device.

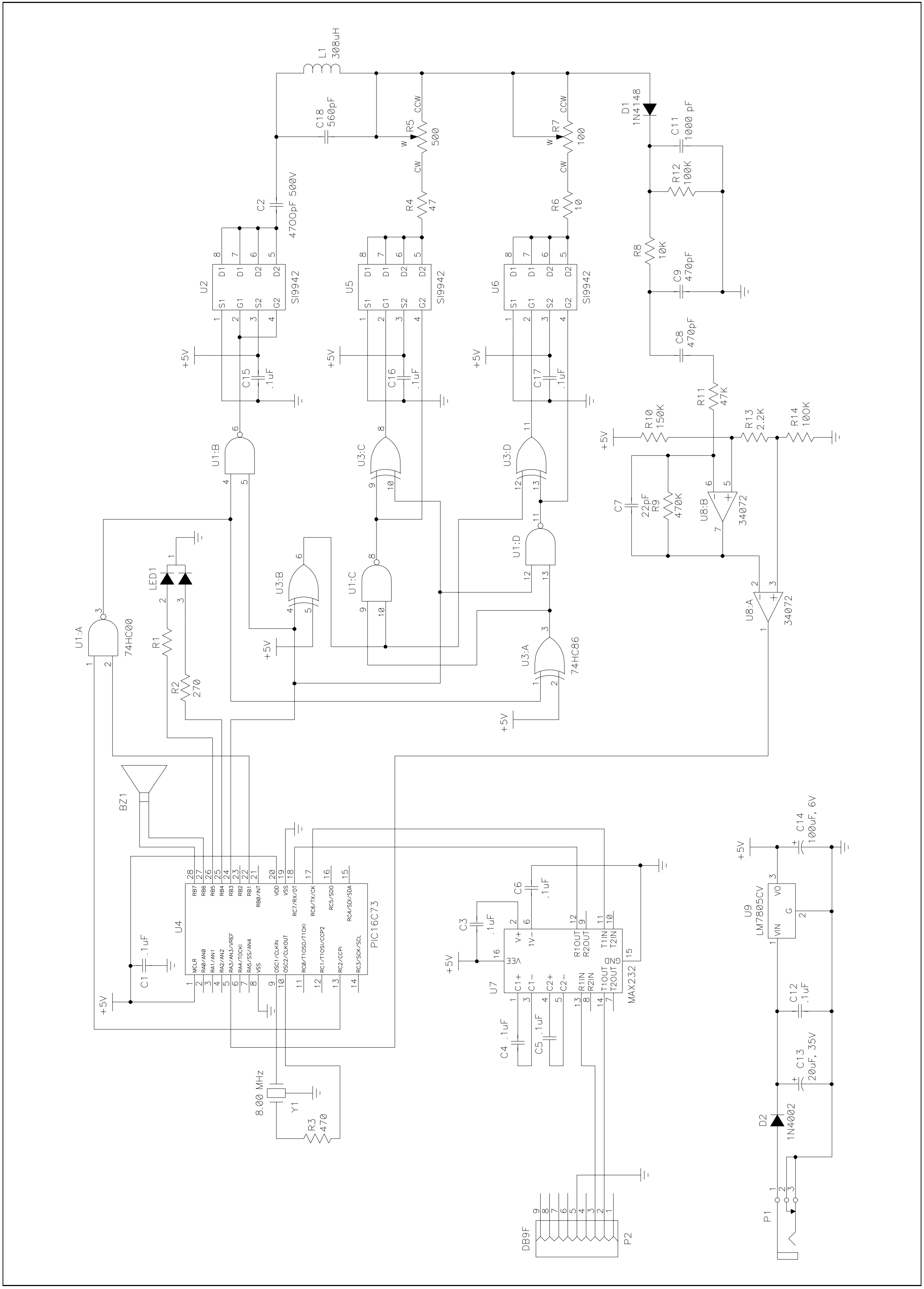

STEP1
WAKE UP
PC Send ‘W’ to the programmer
Programmer replies with ‘R’ to the PC
STEP2
VERIFY BLANK
PC Send ‘T’ to the programmer
Programmer replies with ‘Y’ if the device is OK and ‘N’ if there is an error
STEP3
PASS 16 BYTES OF DATA
Byte 1
F
PC Send ‘F’ to prog.
Programmer replies with ‘F’
1
PC Send ‘1’ to prog.
Programmer replies with ‘1’
Byte 2
E
PC Send ‘E’ to prog.
Programmer replies with ‘E’
2
PC Send ‘2’ to prog.
Programmer replies with ‘2’
Byte 15
2
PC Send ‘2’ to prog.
Programmer replies with ‘2’
E
PC Send ‘E’ to prog.
Programmer replies with ‘E’
Byte 16
1
PC Send ‘1’ to prog.
Programmer replies with ‘1’
F
PC Send ‘F’ to prog.
Programmer replies with ‘F’
STEP4
PROGRAM/VERIFY
PC Send ‘V’ to the programmer to initiate a program/verify sequence (ID internal program/verify)
Programmer replies with ‘y’ if the device programs OK and ‘n’ if there is an error

Note: See the signal waveforms and calibration procedure in Sections 2.0 and 3.0.

TABLE 5-1 ASCII CHARACTER SET

		Most Significant Characters							
	Hex	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Least Significant Characters	0	NUL	DLE	Space	0	@	P	‘	p
	1	SOH	DC1	!	1	A	Q	a	q
	2	STX	DC2	"	2	B	R	b	r
	3	ETX	DC3	#	3	C	S	c	s
	4	EOT	DC4	$	4	D	T	d	t
	5	ENQ	NAK	%	5	E	U	e	u
	6	ACK	SYN	&	6	F	V	f	v
	7	Bell	ETB	’	7	G	W	g	w
	8	BS	CAN	(	8	H	X	h	x
	9	HT	EM	)	9	I	Y	i	y
	A	LF	SUB	*	:	J	Z	j	z
	B	VT	ESC	+	;	K	[	k	{
	C	FF	FS	,	<	L	\	l	|
	D	CR	GS	-	=	M	]	m	}
	E	SO	RS	.	>	N	^	n	~
	F	SI	US	/	?	O	_	o	DEL

6.0 microID™ PROGRAMMER SCHEMATIC



7.0 microID™ PROGRAMMER BILL OF MATERIALS

Item #	Qty	Part #	Reference Designator	Part Description	Manufacturer	Vendor	Vendor Part #
1	1	ICA-328-S-ST	U4	SOCKET, 28 PIN,.300, MACHINED COLLET	SAMTEC		
2	1	-SPARE-	SP1, LED1, R1, R2	-SPARE- LOCATION DO NOT INSTALL			
3	1	PCC220CNCT-ND	C7	CAP SMT, 22 pF NPO 0805	PANASONIC		
4	2	0805N471J101NT	C8, C9	CAP SMT, 470 pF 5% 100V 0805	MALLORY		
5	1	CD15FC561JO3	C18	CAP, 560 pF, MICA, DIPPED, 300V, AX (0.234LS)	CORNELL DUBILIER	MOUSER	5982-15-300V560
6	1	ECU-V1H102JCX	C11	CAP SMT, 1000 pF 50V NPO CER, 0805	PANASONIC		
7	1	CD19FD472JO3	C2	CAP, 4700 pF, MICA, DIPPED, 500V, AX (0.344LS)	CORNELL DUBILIER	MOUSER	5982-19-500V4700
8	9	250R18Z104MV4E-6	C1, C3-C6, C12, C15-C17	CAP SMT, 0.1 μF 20% 50V 0805	JOHANSON	NEWARK	50F3674
9	1	ECS-H1ED106R	C13	CAP SMT, 10 μF, TANT ELEC, 25V, 7343	PANASONIC	DIGIKEY	PCT5106CT-ND
10	1	ECE-V0JA101SP	C14	CAP SMT, 100 μF, TANT ELEC, 6.3V, (VS-D)	PANASONIC	DIGIKEY	PCE3058CT-ND
11	1	LL4148	D1	DIODE SMT, 5uA, 100V, 500 mW, FAST SWITCHING, DL-35	DIODES INC	DIGIKEY	LL4148DITR-ND
12	1	DL4002	D2	DIODE SMT, RECTIFIER, 1N4002, 1A, 100V, DL-41	DIODES INC.	DIGIKEY	DL4002DITR-ND
13	1	3345P-1-101	R7	RES, POT, 100 OHM 1/2 RD WW ST SL	BOURNS	DIGIKEY	3345P-101-ND
14	1	3345P-1-501	R5	RES, POT, 500 OHM 1/2 RD WW ST SL	BOURNS	DIGIKEY	3345P-501-ND
15	1	ERJ-6GEYJ100	R6	RES SMT, 10 OHM 1/10W 5% TYPE 0805	PANASONIC		P10ACT-ND
16	1	ERJ-6GEYJ470V	R4	RES SMT, 47 OHM 1/10W 5% TYPE 0805	PANASONIC	DIGIKEY	P470ATR-ND
17	1	ERJ-6GEYJ471V	R3	RES SMT, 470 OHM 1/10W 5% TYPE 0805	PANASONIC		P470ATR-ND
18	1	ERJ-6GEYJ222V	R13	RES SMT, 2.2K OHM 1/10W 5% TYPE 0805	PANASONIC		P2.2KATR-ND
19	1	ERJ-6GEYJ103V	R8	RES SMT, 10K 1/8W 5% TYPE 0805	PANASONIC	DIGIKEY	P10KATR-ND

Item #	Qty	Part #	Reference Designator	Part Description	Manufacturer	Vendor	Vendor Part #
20	1	ERJ-6GEYJ473V	R11	RES SMT, 47K OHM 1/10W 5% TYPE 0805	PANASONIC	DIGIKEY	P473ATR-ND
21	2	ERJ-6GEYJ104V	R12, R14	RES SMT, 100K OHM 1/10W 5% TYPE 0805	PANASONIC	DIGIKEY	P100KATR-ND
22	1	ERJ-6GEYJ154V	R10	RES SMT, 150K OHM 1/8W 5% 0805	PANASONIC	DIGIKEY	P150KATR-ND
23	1	ERJ-6GEYJ474V	R9	RES SMT, 470K OHM 1/8W 5% 0805	PANASONIC	DIGIKEY	P470KATR-ND
24	1	MM74HC00M	U1	IC, SMT, 74HC00 QUAD 2 IN NAND (SO-14)	FAIRCHILD SEMICONDUCTOR	DIGIKEY	MM74HC00M-ND
25	3	NDS9942	U2, U5, U6	IC, SMT, 9942 MOSFET N-CH & P-CH 20V (SO-8)	FAIRCHILD SEMICONDUCTOR	DIGIKEY	NDS9942TR-ND
26	1	MM74HC86MX	U3	IC, SMT, 74HC86, QUAD XOR GATE (SO-14)	FAIRCHILD SEMICONDUCTOR	DIGIKEY	
27	1	PIC16C73A /P	U4	IC, PIC16C73A /P, PLASTIC DIP, 28P, 0.300	MICROCHIP		
28	1	MAX232ACSE	U7	IC, MAX232ACSE DUAL RS-232 TRANSMITTER/RCVR, (SO-16)	MAXIM	DIGIKEY	MAX232ACSE-ND
29	1	MC34072D	U8	IC, DUAL OP AMP, (SO-8)	MOTOROLA		
30	1	L7805CV	U9	IC, REG, +5V, 1.5A, 10%, TO-220	SGS THOMSON	MOUSER	511-L7805CV
31	1	EFO-EC8004A4	Y1	OSC, 8.00 MHz CER RESONATOR W/CAP 3 PIN	PANASONIC	DIGIKEY	PX800-ND
32	1	MCT0003-000	L1	INDUCTOR, 162 µH	CORNEL DUBILIER		
33	1	DE9S-FRS	P2	CONN, D-SUB 9P RECPT RT ANGLE	SPC TECHNOLOGY		
34	1	DJ005B	P1	JACK, POWER, 2.5mm DC PC MOUNT	LZR ELECTRONICS		

8.0 PROGRAMMER SOURCE CODE FOR PIC16C73

```
; #=#=#=#=#=#=#=#=#=#=# PROJECT Microchip Programmer Reader #=#=#=#=#=#=#=#=#=#=#
; #=#=#=#=#=#=#=#=#=#=#          16C73A module               #=#=#=#=#=#=#=#=#=#=#
; rfgopr5.asm
; PIC16C73A running at 8.5MHz, Ti = 0.47us
; Tcy = 16 Ti

; ///////////////////////////////////////////////////////////////////////////
; Revision history
; ///////////////////////////////////////////////////////////////////////////
;
; Ver    Date        Comment
;
; 1.00   10/24/97    Shannon/Hugh first pass
; 1.04   13 Feb 98   ADDED TIMEOUT TO TESTMOD
;
        LISTP=PIC16C73A
    INCLUDE “P16C73A.INC”
    __config b’11111111110010’
    ; Code Protect off, Brown-out detect on, Power-up timer on, WDT off,
    ;   HS oscillator

 constant StartPORTA    = b’000000’
 constant StartTRISA    = b’010111’

 #define _LED1          PORTB,4
 #define _LED2          PORTB,5
 #define _BUZZ1         PORTB,6
 constant StartPORTB    = b’00010010’
 constant StartTRISB    = b’00000100’
 constant StartOPTION   = b’10001000’
        ; Pullups disabled, TMR0 internal, WDT*1

COUNT1EQU0x20  ; COUNT REGISTER
DATA0EQU0x21
DATA1EQU0x22
DATA2EQU0x23
DATA3EQU0x24
DATA4EQU0x25
DATA5EQU0x26
DATA6EQU0x27
DATA7EQU0x28
DATA8EQU0x29
DATA9EQU0x2A
DATAAEQU0x2B
DATABEQU0x2C
DATACEQU0x2D
DATADEQU0x2E
DATAEEQU0x2F
DATAFEQU0x30
BIT EQU 0x31
OVERPROEQU0x32
DELAY1EQU0x33
DELAY2EQU0x34
DelayReg?H   = h’35’
DelayReg?L   = h’36’
CycleCtr?H   = h’37’
CycleCtr?L   = h’38’
TimerHi      = h’39’
TimerMid     = h’3A’
TimerLo      = h’3B’
BitCtr       = h’3C’
BO3          = h’3D’
RxByte       = h’3E’
TxByte       = h’3F’
ByteCtr      = h’40’
```

```
NoiseTimeout = h’41’
SampTimeout  = h’42’
CycleCtr2?L  = h’43’
CycleCtr2?H  = h’44’
 #define _RAW_DATA      PORTA,4
 #define _RS232OUT      PORTC,6
 #define _CARRY         STATUS,0
 #define _TMR2ON        T2CON,2
 #define _RS232IN       PORTC,7
 #define _ZERO          STATUS,2
 #define _COIL_PWR_0    PORTB,3       ; cycle at 30ms period (1=low power)
 #define _COIL_EN       PORTB,1

SKIP macro
        BTFSC   PORTA,7
 endm

; ***** Reset Vector

        org h’000’
        CLRF    STATUS
        CLRF    PCLATH
        CLRF    INTCON
        GOTO    RESET_A

; ***** Interrupt Vector - no interrupts yet

        org h’004’
        CLRF    STATUS
        CLRF    PCLATH
        GOTO    RESET_A

RS232StopBit                    ;[0] Delay >=208 cycles with _RS232OUT high
    BSF     _RS232OUT       ; |
    MOVLW   d’208’-d’12’+d’40’; |
DelayW12                        ;[0] Delay 12+W cycles
    MOVWF   DelayReg?L      ; |
Delay1                          ;[0] Delay 11+Delay cycles
    MOVLW   d’4’            ; |
Delay1L                         ; |
    SUBWF   DelayReg?L,f    ; |
    BTFSC   _CARRY          ; |
    GOTO    Delay1L         ; |
    COMF    DelayReg?L,W    ; |
    ADDWF   PCL,f           ; |
Delay07                         ;[0] Delay 7 cycles
    NOP                     ; |
Delay06                         ;[0] Delay 6 cycles
    NOP                     ; |
Delay05                         ;[0] Delay 5 cycles
    NOP                     ; |
Delay04                         ;[0] Delay 4 cycles
    RETLW   h’00’           ; |

RESET_A
        CLRWDT                  ; Initialise registers, clear watchdog timer
        CLRF    STATUS          ; | Access register page 0
        CLRF    FSR             ; | FSR=#0
        MOVLW   StartPORTA      ; | Initialise PORT registers
        MOVWF   PORTA           ; | |
        MOVLW   StartPORTB      ; | |
        MOVWF   PORTB           ; | |
        CLRF    INTCON          ; | Interrupts off
        MOVLW   b’110001’       ; | TMR1 prescale *8, on
        MOVWF   T1CON           ; | |
        MOVLW   b’0000000’      ; | TMR2 postscale *1, off, prescale *1
        MOVWF   T2CON           ; | |
        MOVLW   d’8’            ; | Duty on period = 8 Ti @@@
        MOVWF   CCPR1L          ; | |
        MOVLW   b’001100’       ; | CCP1 to PWM, 0,0 extra duty time @@@
        MOVWF   CCP1CON         ; | |
        MOVLW   b’00000000’     ; | A/D convertor OFF
        MOVWF   ADCON0          ; | |
        BSF     STATUS,RP0      ;^| Initialise TRIS registers
        MOVLW   StartTRISA      ;^| |
        MOVWF   TRISA           ;^| |
        MOVLW   StartTRISB      ;^| |
        MOVWF   TRISB           ;^| |
        MOVLW   0x82
        MOVWF   TRISC
        MOVLW   StartOPTION     ;^| Initialise OPTION register
        MOVWF   OPTION_REG      ;^| |
        MOVLW   d’15’           ;^| PR2=7 (period of TMR2=16) @@@
        MOVWF   PR2             ;^| |
        MOVLW   h’03’           ;^| (It says so on page 2-584)
        MOVWF   PCON            ;^| |
        MOVLW   b’110’          ;^| No analog inputs
        MOVWF   ADCON1          ;^| |
        BCF     STATUS,RP0      ; | |


        ; !!!!! set TRIS registers, and other hardware registers.
        BCF     T2CON,2; turn coil off
        CLRF    TMR2
        BCF     PORTB,3

        CALL    RS232On

BigLoop1
    CALL    RS232WaitForever
CheckRxByte
    MOVF    RxByte,W
    XORLW   ‘W’
    BTFSC   _ZERO
    GOTO    INTERRUPT
    CALL    RS232On
    MOVLW   ‘Q’
    CALL    RS232TxW
    GOTO    BigLoop1

INTERRUPT
    CALL    Delay07         ; LED1 on, LED2 on (orange/yellow)
    BSF     _LED1           ; |
    CALL    Delay07         ; |
    BSF     _LED2           ; |
    CALL    Delay07         ; |
INT_WAKEUP
        MOVLW   ‘R’
        MOVWF   RxByte
    CALL    RS232On         ; delay
    MOVF    RxByte,W        ; Transmit RxByte
    CALL    RS232TxW        ; |
    CALL    RS232Rx         ; Read byte from RS-232
    BTFSC   _CARRY          ; | (if timeout, goto INT_END)
    GOTO    INT_END         ; |
    MOVF    RxByte,W        ; if (RxByte<>#’T’)
    XORLW   ‘T’             ; |
    BTFSS   _ZERO           ; |
    GOTO    CheckRxByte     ; { goto CheckRxByte }


        MOVLW   d’10’
        MOVWF   CycleCtr?H
        CLRF    CycleCtr?L

Top1    BCFPORTB,3; SET FOR LOW VOLTAGE
    CALLDELAY  ; CALL A SMALL DELAY

GAP1; THIS IS THE ROUTINE THAT SETS THE GAP

        BCF     PORTB,3
        CALL    DELAY

    BSF T2CON,2 ; TURN ON THE COIL

    MOVLW0x32  ; MOVE 32 HEX TO W, NUMBER CYCLES BEFORE A GAP
    MOVWFCOUNT1; MOVW W INTO COUNT1
LOOP11DECFSZCOUNT1,1; DECREMENT COUNT 1 UNTIL IT IS ZERO
    GOTOLOOP11

    BCF T2CON,2 ; TURN OFF THE COIL

    MOVLW0x40  ; MOVE 10 HEX TO W, DURATION OF GAP
    MOVWFCOUNT1; MOVW W INTO COUNT1
LOOP21DECFSZCOUNT1,1; DECREMENT COUNT 1 UNTIL IT IS ZERO
    GOTOLOOP21

    BSF T2CON,2 ; TURN THE COIL BACK ON

        CALL    TWC             ; CALL A DELAY FOR AMP TO SETTLE
        CALL    TWC
        CALL    TWC
        CALL    TWC
        CALL    TWC


WaitFall1                       ; Wait for falling edge
WaitFall1A                      ; | Wait for high
        MOVLW   d’200’          ; | | Set timeout
        MOVWF   DelayReg?H      ; | | |
        CLRF    DelayReg?L      ; | | |
WaitFall1AL                     ; | | { {
        DECFSZ  DelayReg?L,f    ; | |     if (timeout)
        SKIP                    ; | |     |
        DECFSZ  DelayReg?H,f    ; | |     |
        SKIP                    ; | |     |
        GOTO    INT_ErrorN      ; | |     { goto INT_ErrorN }
        BTFSS   _RAW_DATA       ; | |   } until (_RAW_DATA==#1)
        GOTO    WaitFall1AL     ; | |   |
        NOP                     ; | |   |
        DECFSZ  DelayReg?L,f    ; | |   if (timeout)
        SKIP                    ; | |   |
        DECFSZ  DelayReg?H,f    ; | |   |
        SKIP                    ; | |   |
        GOTO    INT_ErrorN      ; | |   { goto INT_ErrorN }
        BTFSS   _RAW_DATA       ; | | } until (_RAW_DATA==#1)
        GOTO    WaitFall1AL     ; | |
WaitFall1B                      ; | Wait for low
        MOVLW   d’200’          ; | | Set timeout
        MOVWF   DelayReg?H      ; | | |
        CLRF    DelayReg?L      ; | | |
WaitFall1BL                     ; | | { {
        DECFSZ  DelayReg?L,f    ; | |     if (timeout)
        SKIP                    ; | |     |
        DECFSZ  DelayReg?H,f    ; | |     |
        SKIP                    ; | |     |
        GOTO    INT_ErrorN      ; | |     { goto INT_ErrorN }
        BTFSC   _RAW_DATA       ; | |   } until (_RAW_DATA==#0)
        GOTO    WaitFall1BL     ; | |   |
        NOP                     ; | |   |
        DECFSZ  DelayReg?L,f    ; | |   if (timeout)
        SKIP                    ; | |   |
        DECFSZ  DelayReg?H,f    ; | |   |
        SKIP                    ; | |   |
        GOTO    INT_ErrorN      ; | |   { goto INT_ErrorN }
        BTFSC   _RAW_DATA       ; | | } until (_RAW_DATA==#0)
        GOTO    WaitFall1BL     ; | | |

        CLRF    DelayReg?L      ; Clear timer
WaitFall2                       ; Time falling edge
WaitFall2A                      ; | Wait for high
WaitFall2AL                     ; | | { {
        NOP                     ;
        NOP                     ;
        INCF    DelayReg?L,f    ; | |     Increment timer
        BTFSC   DelayReg?L,7    ; | |     if timeout,
        GOTO    INT_ErrorN      ; | |     { goto INT_ErrorN }
        BTFSS   _RAW_DATA       ; | |   } until (_RAW_DATA==#1)
        GOTO    WaitFall2AL     ; | |   |
        NOP                     ; | |   |
        NOP                     ;
        NOP                     ;
        INCF    DelayReg?L,f    ; | |   Increment timer
        BTFSC   DelayReg?L,7    ; | |   if timeout,
        GOTO    INT_ErrorN      ; | |   { goto INT_ErrorN }
        BTFSS   _RAW_DATA       ; | | } until (_RAW_DATA==#1)
        GOTO    WaitFall2AL     ; | |
        NOP                     ;
WaitFall2B                      ; | Wait for low
WaitFall2BL                     ; | | { {
        NOP                     ;
        NOP                     ;
        INCF    DelayReg?L,f    ; | |     Increment timer
        BTFSC   DelayReg?L,7    ; | |     if timeout,
        GOTO    INT_ErrorN      ; | |     { goto INT_ErrorN }
        BTFSC   _RAW_DATA       ; | |   } until (_RAW_DATA==#0)
        GOTO    WaitFall2BL     ; | |   |
        NOP                     ; | |   |
        NOP                     ;
        NOP                     ;
        INCF    DelayReg?L,f    ; | |   Increment timer
        BTFSC   DelayReg?L,7    ; | |   if timeout,
        GOTO    INT_ErrorN      ; | |   { goto INT_ErrorN }
        BTFSC   _RAW_DATA       ; | | } until (_RAW_DATA==#0)
        GOTO    WaitFall2BL     ; | | |
        ; DelayReg?L*8Ti = period of signal
        ; period of _RAW_DATA on FSK = Tcy*10 = Ti*160
        ; DelayReg?L = 20 if FSK present
                                ; if period does not match FSK, goto INT_ErrorN
        MOVF    DelayReg?L,W    ; | if (DelayReg?L<14)
        ADDLW   low(0-d’14’)    ; | |
        BTFSS   _CARRY          ; | |
        GOTO    INT_ErrorN      ; | { goto INT_ErrorN }
        ADDLW   low(d’14’-d’22’); | if (DelayReg?L>=22)
        BTFSC   _CARRY          ; | |
        GOTO    INT_ErrorN      ; | { goto INT_ErrorN }

        MOVLW   d’7’            ; CycleCtr > 13*128=1664
        MOVWF   CycleCtr?H      ; |
        MOVLW   d’164’          ; |
        MOVWF   CycleCtr?L      ; |
TestGotLo
        DECFSZ  CycleCtr?L,f
        SKIP
        DECFSZ  CycleCtr?H,f
        SKIP
        GOTO    INT_ErrorN
        MOVLW   0x20
        MOVWF   COUNT1
        BTFSS   _RAW_DATA
        GOTO    TestGotHi
TestGotLoLoop
        BTFSS   _RAW_DATA
        GOTO    TestGotHi
        DECFSZ  COUNT1,1
        GOTO    TestGotLoLoop
        GOTO    MChip_Prog

TestGotHi
        MOVLW   0x20
        MOVWF   COUNT1
        BTFSC   _RAW_DATA
        GOTO    TestGotLo
TestGotHiLoop
        BTFSC   _RAW_DATA
        GOTO    TestGotLo
        DECFSZ  COUNT1,1
        GOTO    TestGotHiLoop
;END TEST FOR NO MODULATION

MChip_Prog
        BCF     _TMR2ON
        CALL    TWC

    CLRF    DATA0
    CLRF    DATA1
    CLRF    DATA2
    CLRF    DATA3
    CLRF    DATA4
    CLRF    DATA5
    CLRF    DATA6
    CLRF    DATA7
    CLRF    DATA8
    CLRF    DATA9
    CLRF    DATAA
    CLRF    DATAB
    CLRF    DATAC
    CLRF    DATAD
    CLRF    DATAE
    CLRF    DATAF

    MOVLW   ‘Y’             ; RxByte=’Y’
    MOVWF   RxByte          ; |
    MOVLW   DATAF           ; FSR=#DATAF
    MOVWF   FSR             ; |
    MOVLW   h’20’           ; ByteCtr=#h’20’
    MOVWF   ByteCtr         ; |
RS_ByteLoop                     ; {
    CALL    RS232On         ;   delay
    MOVF    RxByte,W        ;   Transmit RxByte on RS-232
    CALL    RS232TxW        ;   |
    CALL    RS232Rx         ;   Read RS-232 byte into RxByte
    BTFSC   _CARRY          ;   |(if timeout, goto INT_END)
    GOTO    INT_END         ;   |
    MOVF    RxByte,W        ;   BO3=RxByte
    MOVWF   BO3             ;   |
    MOVLW   h’30’           ;   if (BO3<#h’30’)
    SUBWF   BO3,W           ;   |
    BTFSS   _CARRY          ;   |
    GOTO    CheckRxByte     ;   { goto CheckRxByte }
    MOVWF   BO3             ;   BO3=BO3-#h’30’
    MOVLW   h’3A’-h’30’     ;   if (BO3>=#h’3A’-#h’30’)
    SUBWF   BO3,W           ;   |
    BTFSS   _CARRY          ;   |
    GOTO    RSDataJ1        ;   {
    MOVWF   BO3             ;     BO3=BO3-#h’3A’+#h’30’
    MOVLW   h’41’-h’3A’     ;     if (BO3<#h’41’-#h’3A’)
    SUBWF   BO3,W           ;     |
    BTFSS   _CARRY          ;     |
    GOTO    CheckRxByte     ;     { goto CheckRxByte }
    MOVWF   BO3             ;     BO3=BO3-#h’41’+#h’3A’
    MOVLW   h’47’-h’41’     ;     if (BO3>=#h’47’-#h’41’)
    SUBWF   BO3,W           ;     |
    BTFSC   _CARRY          ;     |
    GOTO    CheckRxByte     ;     { goto CheckRxByte }
    MOVLW   h’0A’           ;     BO3=BO3+#h’0A’
    ADDWF   BO3,f           ;     |
RSDataJ1                        ;   }
    SWAPF   BO3,W           ;   W = { BO3 swapped  if ByteCtr,0==#0
    BTFSC   ByteCtr,0       ;   |   { BO3          if ByteCtr,0==#1
    MOVF    BO3,W           ;   |
    IORWF   INDF,f          ;   INDF=INDF OR W
    BTFSC   ByteCtr,0       ;   if (ByteCtr,0==#1)
    DECF    FSR,f           ;   { FSR=FSR-#1 }
    DECFSZ  ByteCtr,f       ;   DEC ByteCtr
    GOTO    RS_ByteLoop     ; } until (ByteCtr==#0)

    CALL    RS232On         ; delay
    MOVF    RxByte,W        ; Transmit RxByte on RS-232
    CALL    RS232TxW        ; |
    CALL    RS232Rx         ; Read RS-232 byte into RxByte
    BTFSC   _CARRY          ; |( if timeout, goto INT_END)
    GOTO    INT_END         ; |
    MOVF    RxByte,W        ; if (RxByte!=#’V’)
    XORLW   ‘V’             ; |
    BTFSS   _ZERO           ; |
    GOTO    CheckRxByte     ; { goto CheckRxByte }



; ********

Top BCF PORTB,3 ; SET FOR LOW VOLTAGE
    CALLDELAY  ; CALL A SMALL DELAY

GAP ; THIS IS THE ROUTINE THAT SETS THE GAP

        BCF     PORTB,3
        CALL    DELAY

    BSF T2CON,2 ; TURN ON THE COIL

    MOVLW0x32  ; MOVE 32 HEX TO W, NUMBER CYCLES BEFORE A GAP
    MOVWFCOUNT1; MOVW W INTO COUNT1
LOOP1DECFSZCOUNT1,1; DECREMENT COUNT 1 UNTIL IT IS ZERO
    GOTOLOOP1

    BCF T2CON,2 ; TURN OFF THE COIL

    MOVLW0x40  ; MOVE 10 HEX TO W, DURATION OF GAP
    MOVWFCOUNT1; MOVW W INTO COUNT1
LOOP2DECFSZCOUNT1,1; DECREMENT COUNT 1 UNTIL IT IS ZERO
    GOTOLOOP2

    BSF T2CON,2 ; TURN THE COIL BACK ON
```

```
        MOVLWd’8’; MOVE 5 INTO THE W REGISTER
    MOVWFOVERPRO; THIS IS THE NUMBER OF OVERPROGRAMMING

        CALL    TWC             ; CALL A DELAY FOR AMP TO SETTLE
        CALL    TWC
        CALL    TWC
        CALL    TWC
        CALL    TWC

MODING  CALL    TESTMOD
        CALL    PROGRAM

        MOVLW   0x60
        MOVWF   COUNT1
BIGDLY  CALL    TWC             ; CALL A DELAY TO ALLOW THE AMP TO SETTLE
        DECFSZ  COUNT1,f
        GOTO    BIGDLY

        DECFSZ  OVERPRO,1       ; DECREMENT THE OVERPROGRAMMING NUMBER
    GOTOMODING ; GOTO LOOK FOR THE MODULATION TO STOP
        GOTOVERIFY

;****************************************

VERIFY
        CALL    TESTMOD         ; Wait for modulation to stop
                                ;% 167Ti of constant _RAW_DATA

StartWatch                      ; Wait >~Ttag (for mod to start again)

        MOVLW   h’00’           ; Delay >~262144Ti
        MOVWF   DelayReg?H      ; |
VerifyD1a                       ; |
        MOVLW   h’FF’           ; | delay 1021Ti
        MOVWF   DelayReg?L      ; | |
VerifyD1b                       ; | |
        CLRWDT                  ; | |
        DECFSZ  DelayReg?L,f    ; | |
        GOTO    VerifyD1b       ; | |
        DECFSZ  DelayReg?H,f    ; |
        GOTO    VerifyD1a       ; |
StopWatch                       ; |

        CLRF    BitCtr          ; BitCtr=#128
        BSF     BitCtr,7        ; |

VerifyL1                        ; {
;% reftime-1345
        CLRF    CycleCtr?L
;% reftime-1344
;% reftime-3-10*6-183*7
        MOVLW   d’10’           ;   set NoiseTimeout
        MOVWF   NoiseTimeout    ;   |
;% reftime-1-10*6-183*7
;% reftime-1-NTO*6-183*7
        MOVLW   d’183’          ;   set SampTimeout to 80Tcy
        MOVWF   SampTimeout     ;   |
;% reftime+1-NTO*6-183*7
;% reftime+1-NTO*6-STO*7

        BTFSC   _RAW_DATA
        GOTO    VerS1
        NOP

VerS0
;% reftime+4-NTO*6-STO*7
```

```
        DECFSZ  NoiseTimeout,f
        SKIP
        GOTO    VerFail
        BTFSC   _RAW_DATA
        GOTO    VerS1
VerGot0
;% reftime+3-NTO*6-STO*7
VerGot0a
;% reftime+3-NTO*6-STO*7
        CLRWDT
        DECFSZ  SampTimeout,f
        SKIP
        GOTO    SampleDone
        BTFSS   _RAW_DATA
        GOTO    VerGot0
        NOP
VerGot0b
;% reftime+3-NTO*6-STO*7
        CLRWDT
        DECFSZ  SampTimeout,f
        SKIP
        GOTO    SampleDone
        BTFSS   _RAW_DATA
        GOTO    VerGot0
        NOP
VerGotRise
;% reftime+3-NTO*6-STO*7
        CLRWDT
        DECFSZ  SampTimeout,f
        SKIP
        GOTO    SampleDone
        INCF    CycleCtr?L,f
        GOTO    VerGot1

VerS1
;% reftime+4-NTO*6-STO*7
        DECFSZ  NoiseTimeout,f
        SKIP
        GOTO    VerFail
        BTFSS   _RAW_DATA
        GOTO    VerS0
VerGot1
;% reftime+3-NTO*6-STO*7
VerGot1a
;% reftime+3-NTO*6-STO*7
        CLRWDT
        DECFSZ  SampTimeout,f
        SKIP
        GOTO    SampleDone
        BTFSC   _RAW_DATA
        GOTO    VerGot1
        NOP
VerGot1b
;% reftime+3-NTO*6-STO*7
        CLRWDT
        DECFSZ  SampTimeout,f
        SKIP
        GOTO    SampleDone
        BTFSC   _RAW_DATA
        GOTO    VerGot1
        NOP
VerGotFall
;% reftime+3-NTO*6-STO*7
        CLRWDT
        DECFSZ  SampTimeout,f
        SKIP
        GOTO    SampleDone
        INCF    CycleCtr?L,f
        GOTO    VerGot0

SampleDone
;% reftime+1-NTO*6-STO*7
;& STO=0
;% reftime+1-NTO*6
NoiseMargin
;% reftime+1-NTO*6
        NOP
        NOP
        NOP
        DECFSZ  NoiseTimeout,f
        GOTO    NoiseMargin
;% reftime+0-NTO*6
;% NTO=0
;% reftime+0

        BTFSC   DATAF,7
        GOTO    Verify1
        NOP

Verify0
;% 3 from ref time
; if ‘0’ bit, _DATA_IN cycles 10 times in 80 Tcy
; CycleCtr?L should be 20
        MOVF    CycleCtr?L,W
        ADDLW   low(0-d’18’)
        BTFSS   _CARRY
        GOTO    INT_Failure
        ADDLW   low(d’18’-d’22’)
        BTFSS   _CARRY
        GOTO    Bit_Verified
        GOTO    INT_Failure

Verify1
;% 3 from ref time
; if ‘1’ bit, _DATA_IN cycles 8 times in 80Tcy
; CycleCtr?L should be 16
        MOVF    CycleCtr?L,W
        ADDLW   low(0-d’14’)
        BTFSS   _CARRY
        GOTO    INT_Failure
        ADDLW   low(d’14’-d’18’)
        BTFSS   _CARRY
        GOTO    Bit_Verified
        GOTO    INT_Failure

Bit_Verified
;% 11 from ref time
        BCF     _CARRY
        BTFSC   DATAF,7
        BSF     _CARRY
        RLF     DATA0,f
        RLF     DATA1,f
        RLF     DATA2,f
        RLF     DATA3,f
        RLF     DATA4,f
        RLF     DATA5,f
        RLF     DATA6,f
        RLF     DATA7,f
        RLF     DATA8,f
        RLF     DATA9,f
        RLF     DATAA,f
        RLF     DATAB,f
        RLF     DATAC,f
        RLF     DATAD,f
        RLF     DATAE,f
        RLF     DATAF,f

;% 30 from ref time

        MOVLW   d’167’          ; Delay 670Ti
        MOVWF   DelayReg?L      ; |
        NOP                     ; |
VerDelay                        ; |
        CLRWDT                  ; |
        DECFSZ  DelayReg?L,f    ; |
        GOTO    VerDelay        ; |

;% 700 from ref time
;% (ref times 128*16Ti apart = 2048Ti apart)
;% -1348 from ref time
        DECFSZ  BitCtr,f        ;   DEC BitCtr
        GOTO    VerifyL1        ; } until (BitCtr==#0)

;***************************************

INT_Success
        CALL    RS232On
    MOVLW   ‘y’
    CALL    RS232TxW
    GOTO    BigLoop1

VerFail
INT_Failure

        CALL    RS232On
        MOVLW   ‘n’
        CALL    RS232TxW
        GOTO    BigLoop1

INT_END ; RS-232 TIMEOUT
    NOP
    GOTO    BigLoop1

INT_ErrorN
    CALL    RS232On
    MOVLW   ‘N’
    CALL    RS232TxW
    GOTO    BigLoop1

DELAYMOVLW0x05
    MOVWFDELAY1
HOLD4DECFSZDELAY1,1
    GOTOHOLD4
    RETLW0

; TWC lasts
TWC MOVLW0xB0  ; WRITE CYCLE TIMER SUBROUTINE
    MOVWFDELAY1
HOLD1MOVLW0x02
    MOVWFDELAY2
HOLD2DECFSZDELAY2,1
    GOTOHOLD2
    DECFSZDELAY1,1
    GOTOHOLD1
        RETLW0

BUFFERMOVLW0x58
    MOVWFDELAY1
```

```
HOLD3DECFSZDELAY1,1
    GOTOHOLD3
        NOP
        NOP
        RETLW0

TESTMOD; THIS ROUTINE TESTS THE RAW DATA LINE TO SEE IF THE
    ; PART IS MODULATING OR NOT

; This routine returns when _RAW_DATA stays constant for some time
; some time = 7Ti+32*5Ti = 167Ti = 10.4375Tcy

        MOVLW   d’7’            ; CycleCtr2 > 13*128=1664
        MOVWF   CycleCtr2?H     ; |
        MOVLW   d’164’          ; |
        MOVWF   CycleCtr2?L     ; |

TestModLo
        DECFSZ  CycleCtr2?L,f
        SKIP
        DECFSZ  CycleCtr2?H,f
        SKIP
        GOTO    INT_Failure
        MOVLW   0x20
        MOVWF   COUNT1
        BTFSS   _RAW_DATA
        GOTO    TestModHi
TestModLoLoop
        BTFSS   _RAW_DATA
        GOTO    TestModHi
        DECFSZ  COUNT1,1
        GOTO    TestModLoLoop
        RETLW   0
TestModHi
        MOVLW   0x20
        MOVWF   COUNT1
        BTFSC   _RAW_DATA
        GOTO    TestModLo
TestModHiLoop
        BTFSC   _RAW_DATA
        GOTO    TestModLo
        DECFSZ  COUNT1,1
        GOTO    TestModHiLoop
;END TEST FOR NO MODULATION
        RETLW   0

PROGRAM BCFPORTB,3; CLEAR THE HIGH VOLTAGE

        MOVLW0x07; MOVW 7 HEX INTO W
    MOVWFBIT   ; MOVE THIS INTO THE BIT COUNTER
WRITEFBTFSSDATAF,7 ; TEST MOST BYTE
    BSF PORTB,3 ; SET THE HIGH VOLTAGE
    CALLTWC    ; CALL THE WRITE CYCLE TIMER
    BCF     STATUS,C        ; CLEAR THE CARRY BIT
        BTFSC   DATAF,7         ; TEST THE MSB
        BSF     STATUS,C        ; SET THE CARRY BIT
        RLF     DATAF,1         ; ROTATE DATAF
        BCFPORTB,3; CLEAR THE HIGH VOLTAGE
        CALL BUFFER; CALL THE BUFFER TIMER
    DECF    BIT,1; DECREMENT BIT, SKIP IF ZERO
    BTFSS   BIT,7           ; SKIP IF SET
        GOTOWRITEF; GOTO WRITEF IF BIT IS NOT EQUAL TO ZERO
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP

        MOVLW0x07; MOVW 7 HEX INTO W
    MOVWFBIT   ; MOVE THIS INTO THE BIT COUNTER
WRITEEBTFSSDATAE,7 ; TEST MOST BYTE
    BSF PORTB,3 ; SET THE HIGH VOLTAGE
    CALLTWC    ; CALL THE WRITE CYCLE TIMER
    BCF     STATUS,C        ; CLEAR THE CARRY BIT
        BTFSC   DATAE,7         ; TEST THE MSB
        BSF     STATUS,C        ; SET THE CARRY BIT
        RLF     DATAE,1         ; ROTATE DATAF
        BCFPORTB,3; CLEAR THE HIGH VOLTAGE
        CALL BUFFER; CALL THE BUFFER TIMER
    DECF    BIT,1; DECREMENT BIT, SKIP IF ZERO
    BTFSS   BIT,7           ; SKIP IF SET
        GOTOWRITEE; GOTO WRITEE IF BIT IS NOT EQUAL TO ZERO
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP

    MOVLW0x07  ; MOVW 7 HEX INTO W
    MOVWFBIT   ; MOVE THIS INTO THE BIT COUNTER
WRITEDBTFSSDATAD,7 ; TEST MOST BYTE
    BSF PORTB,3 ; SET THE HIGH VOLTAGE
    CALLTWC    ; CALL THE WRITE CYCLE TIMER
    BCF     STATUS,C        ; CLEAR THE CARRY BIT
        BTFSC   DATAD,7         ; TEST THE MSB
        BSF     STATUS,C        ; SET THE CARRY BIT
        RLF     DATAD,1         ; ROTATE DATAF
        BCFPORTB,3; CLEAR THE HIGH VOLTAGE
        CALL BUFFER; CALL THE BUFFER TIMER
    DECF    BIT,1; DECREMENT BIT, SKIP IF ZERO
    BTFSS   BIT,7           ; SKIP IF SET
        GOTOWRITED; GOTO WRITEF IF BIT IS NOT EQUAL TO ZERO
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP

    MOVLW0x07  ; MOVW 7 HEX INTO W
    MOVWFBIT   ; MOVE THIS INTO THE BIT COUNTER
WRITECBTFSSDATAC,7 ; TEST MOST BYTE
    BSF PORTB,3 ; SET THE HIGH VOLTAGE
    CALLTWC    ; CALL THE WRITE CYCLE TIMER
    BCF     STATUS,C        ; CLEAR THE CARRY BIT
        BTFSC   DATAC,7         ; TEST THE MSB
        BSF     STATUS,C        ; SET THE CARRY BIT
        RLF     DATAC,1         ; ROTATE DATAF
        BCFPORTB,3; CLEAR THE HIGH VOLTAGE
        CALL BUFFER; CALL THE BUFFER TIMER
    DECF    BIT,1; DECREMENT BIT, SKIP IF ZERO
    BTFSS   BIT,7           ; SKIP IF SET
        GOTOWRITEC; GOTO WRITEC IF BIT IS NOT EQUAL TO ZERO
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP

    MOVLW0x07  ; MOVW 7 HEX INTO W
    MOVWFBIT   ; MOVE THIS INTO THE BIT COUNTER
WRITEBBTFSSDATAB,7 ; TEST MOST BYTE
    BSF PORTB,3 ; SET THE HIGH VOLTAGE
    CALLTWC    ; CALL THE WRITE CYCLE TIMER
    BCF     STATUS,C        ; CLEAR THE CARRY BIT
        BTFSC   DATAB,7         ; TEST THE MSB
        BSF     STATUS,C        ; SET THE CARRY BIT
        RLF     DATAB,1         ; ROTATE DATAF
        BCFPORTB,3; CLEAR THE HIGH VOLTAGE
        CALL BUFFER; CALL THE BUFFER TIMER
    DECF    BIT,1; DECREMENT BIT, SKIP IF ZERO
    BTFSS   BIT,7           ; SKIP IF SET
        GOTOWRITEB; GOTO WRITEB IF BIT IS NOT EQUAL TO ZERO
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP

    MOVLW0x07  ; MOVW 7 HEX INTO W
    MOVWFBIT   ; MOVE THIS INTO THE BIT COUNTER
WRITEABTFSSDATAA,7 ; TEST MOST BYTE
    BSF PORTB,3 ; SET THE HIGH VOLTAGE
    CALLTWC    ; CALL THE WRITE CYCLE TIMER
    BCF     STATUS,C        ; CLEAR THE CARRY BIT
        BTFSC   DATAA,7         ; TEST THE MSB
        BSF     STATUS,C        ; SET THE CARRY BIT
        RLF     DATAA,1         ; ROTATE DATAF
        BCFPORTB,3; CLEAR THE HIGH VOLTAGE
        CALL BUFFER; CALL THE BUFFER TIMER
    DECF    BIT,1; DECREMENT BIT, SKIP IF ZERO
    BTFSS   BIT,7           ; SKIP IF SET
        GOTOWRITEA; GOTO WRITEA IF BIT IS NOT EQUAL TO ZERO
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP

    MOVLW0x07  ; MOVW 7 HEX INTO W
    MOVWFBIT   ; MOVE THIS INTO THE BIT COUNTER
WRITE9BTFSSDATA9,7 ; TEST MOST BYTE
    BSF PORTB,3 ; SET THE HIGH VOLTAGE
    CALLTWC    ; CALL THE WRITE CYCLE TIMER
    BCF     STATUS,C        ; CLEAR THE CARRY BIT
        BTFSC   DATA9,7         ; TEST THE MSB
        BSF     STATUS,C        ; SET THE CARRY BIT
        RLF     DATA9,1         ; ROTATE DATAF
        BCFPORTB,3; CLEAR THE HIGH VOLTAGE
        CALL BUFFER; CALL THE BUFFER TIMER
    DECF    BIT,1; DECREMENT BIT, SKIP IF ZERO
    BTFSS   BIT,7           ; SKIP IF SET
        GOTOWRITE9; GOTO WRITE9 IF BIT IS NOT EQUAL TO ZERO
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP

    MOVLW0x07  ; MOVW 7 HEX INTO W
    MOVWFBIT   ; MOVE THIS INTO THE BIT COUNTER
WRITE8BTFSSDATA8,7 ; TEST MOST BYTE
    BSF PORTB,3 ; SET THE HIGH VOLTAGE
    CALLTWC    ; CALL THE WRITE CYCLE TIMER
    BCF     STATUS,C        ; CLEAR THE CARRY BIT
        BTFSC   DATA8,7         ; TEST THE MSB
        BSF     STATUS,C        ; SET THE CARRY BIT
        RLF     DATA8,1         ; ROTATE DATAF
        BCFPORTB,3; CLEAR THE HIGH VOLTAGE
        CALL BUFFER; CALL THE BUFFER TIMER
    DECF    BIT,1; DECREMENT BIT, SKIP IF ZERO
    BTFSS   BIT,7           ; SKIP IF SET
        GOTOWRITE8; GOTO WRITE8 IF BIT IS NOT EQUAL TO ZERO
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP

        MOVLW0x07; MOVW 7 HEX INTO W
    MOVWFBIT   ; MOVE THIS INTO THE BIT COUNTER
WRITE7BTFSSDATA7,7 ; TEST MOST BYTE
    BSF PORTB,3 ; SET THE HIGH VOLTAGE
    CALLTWC    ; CALL THE WRITE CYCLE TIMER
    BCF     STATUS,C        ; CLEAR THE CARRY BIT
        BTFSC   DATA7,7         ; TEST THE MSB
        BSF     STATUS,C        ; SET THE CARRY BIT
        RLF     DATA7,1         ; ROTATE DATAF
        BCFPORTB,3; CLEAR THE HIGH VOLTAGE
        CALL BUFFER; CALL THE BUFFER TIMER
    DECF    BIT,1; DECREMENT BIT, SKIP IF ZERO
    BTFSS   BIT,7           ; SKIP IF SET
        GOTOWRITE7; GOTO WRITE7 IF BIT IS NOT EQUAL TO ZERO
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP

    MOVLW0x07  ; MOVW 7 HEX INTO W
    MOVWFBIT   ; MOVE THIS INTO THE BIT COUNTER
WRITE6BTFSSDATA6,7 ; TEST MOST BYTE
    BSF PORTB,3 ; SET THE HIGH VOLTAGE
    CALLTWC    ; CALL THE WRITE CYCLE TIMER
    BCF     STATUS,C        ; CLEAR THE CARRY BIT
        BTFSC   DATA6,7         ; TEST THE MSB
        BSF     STATUS,C        ; SET THE CARRY BIT
        RLF     DATA6,1         ; ROTATE DATAF
        BCFPORTB,3; CLEAR THE HIGH VOLTAGE
        CALL BUFFER; CALL THE BUFFER TIMER
    DECF    BIT,1; DECREMENT BIT, SKIP IF ZERO
    BTFSS   BIT,7           ; SKIP IF SET
        GOTOWRITE6; GOTO WRITE6 IF BIT IS NOT EQUAL TO ZERO
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP

    MOVLW0x07  ; MOVW 7 HEX INTO W
    MOVWFBIT   ; MOVE THIS INTO THE BIT COUNTER
WRITE5BTFSSDATA5,7 ; TEST MOST BYTE
    BSF PORTB,3 ; SET THE HIGH VOLTAGE
    CALLTWC    ; CALL THE WRITE CYCLE TIMER
    BCF     STATUS,C        ; CLEAR THE CARRY BIT
        BTFSC   DATA5,7         ; TEST THE MSB
        BSF     STATUS,C        ; SET THE CARRY BIT
        RLF     DATA5,1         ; ROTATE DATAF
        BCFPORTB,3; CLEAR THE HIGH VOLTAGE
        CALL BUFFER; CALL THE BUFFER TIMER
    DECF    BIT,1; DECREMENT BIT, SKIP IF ZERO
    BTFSS   BIT,7           ; SKIP IF SET
        GOTOWRITE5; GOTO WRITE5 IF BIT IS NOT EQUAL TO ZERO
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP

    MOVLW0x07  ; MOVW 7 HEX INTO W
    MOVWFBIT   ; MOVE THIS INTO THE BIT COUNTER
WRITE4BTFSSDATA4,7 ; TEST MOST BYTE
    BSF PORTB,3 ; SET THE HIGH VOLTAGE
    CALLTWC    ; CALL THE WRITE CYCLE TIMER
    BCF     STATUS,C        ; CLEAR THE CARRY BIT
        BTFSC   DATA4,7         ; TEST THE MSB
        BSF     STATUS,C        ; SET THE CARRY BIT
        RLF     DATA4,1         ; ROTATE DATAF
        BCFPORTB,3; CLEAR THE HIGH VOLTAGE
        CALL BUFFER; CALL THE BUFFER TIMER
    DECF    BIT,1; DECREMENT BIT, SKIP IF ZERO
    BTFSS   BIT,7           ; SKIP IF SET
        GOTOWRITE4; GOTO WRITE4 IF BIT IS NOT EQUAL TO ZERO
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP

    MOVLW0x07  ; MOVW 7 HEX INTO W
    MOVWFBIT   ; MOVE THIS INTO THE BIT COUNTER
WRITE3BTFSSDATA3,7 ; TEST MOST BYTE
    BSF PORTB,3 ; SET THE HIGH VOLTAGE
    CALLTWC    ; CALL THE WRITE CYCLE TIMER
    BCF     STATUS,C        ; CLEAR THE CARRY BIT
        BTFSC   DATA3,7         ; TEST THE MSB
        BSF     STATUS,C        ; SET THE CARRY BIT
        RLF     DATA3,1         ; ROTATE DATAF
        BCFPORTB,3; CLEAR THE HIGH VOLTAGE
        CALL BUFFER; CALL THE BUFFER TIMER
    DECF    BIT,1; DECREMENT BIT, SKIP IF ZERO
    BTFSS   BIT,7           ; SKIP IF SET
        GOTOWRITE3; GOTO WRITE3 IF BIT IS NOT EQUAL TO ZERO
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP

    MOVLW0x07  ; MOVW 7 HEX INTO W
    MOVWFBIT   ; MOVE THIS INTO THE BIT COUNTER
WRITE2BTFSSDATA2,7 ; TEST MOST BYTE
    BSF PORTB,3 ; SET THE HIGH VOLTAGE
    CALLTWC    ; CALL THE WRITE CYCLE TIMER
    BCF     STATUS,C        ; CLEAR THE CARRY BIT
        BTFSC   DATA2,7         ; TEST THE MSB
        BSF     STATUS,C        ; SET THE CARRY BIT
        RLF     DATA2,1         ; ROTATE DATAF
        BCFPORTB,3; CLEAR THE HIGH VOLTAGE
        CALL BUFFER; CALL THE BUFFER TIMER
    DECF    BIT,1; DECREMENT BIT, SKIP IF ZERO
    BTFSS   BIT,7           ; SKIP IF SET
        GOTOWRITE2; GOTO WRITE2 IF BIT IS NOT EQUAL TO ZERO
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP

    MOVLW0x07  ; MOVW 7 HEX INTO W
    MOVWFBIT   ; MOVE THIS INTO THE BIT COUNTER
WRITE1BTFSSDATA1,7        ; TEST MOST BYTE
    BSF PORTB,3 ; SET THE HIGH VOLTAGE
    CALLTWC    ; CALL THE WRITE CYCLE TIMER
    BCF     STATUS,C        ; CLEAR THE CARRY BIT
        BTFSC   DATA1,7         ; TEST THE MSB
        BSF     STATUS,C        ; SET THE CARRY BIT
        RLF     DATA1,1         ; ROTATE DATAF
        BCFPORTB,3; CLEAR THE HIGH VOLTAGE
        CALL BUFFER; CALL THE BUFFER TIMER
    DECF    BIT,1; DECREMENT BIT, SKIP IF ZERO
    BTFSS   BIT,7           ; SKIP IF SET
        GOTOWRITE1; GOTO WRITEF IF BIT IS NOT EQUAL TO ZERO
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP

    MOVLW0x07  ; MOVW 7 HEX INTO W
    MOVWFBIT   ; MOVE THIS INTO THE BIT COUNTER
WRITE0BTFSSDATA0,7 ; TEST MOST BYTE
    BSF PORTB,3 ; SET THE HIGH VOLTAGE
    CALLTWC    ; CALL THE WRITE CYCLE TIMER
    BCF     STATUS,C        ; CLEAR THE CARRY BIT
        BTFSC   DATA0,7         ; TEST THE MSB
        BSF     STATUS,C        ; SET THE CARRY BIT
        RLF     DATA0,1         ; ROTATE DATAF
        BCFPORTB,3; CLEAR THE HIGH VOLTAGE
        CALL BUFFER; CALL THE BUFFER TIMER
    DECF    BIT,1; DECREMENT BIT, SKIP IF ZERO
    BTFSS   BIT,7           ; SKIP IF SET
        GOTOWRITE0; GOTO WRITE0 IF BIT IS NOT EQUAL TO ZERO
        RETLW   0

Delay12
        NOP
```

```
Delay11
        GOTO    Delay09
Delay09
        GOTO    Delay07

RS232On                         ;[1] Initialise RS-232
    BCF     _TMR2ON         ; | Turn coil off
    CALL    RS232StopBit    ; | Transmit stop bits
    CALL    RS232StopBit    ; | |
    CALL    RS232StopBit    ; | |
    CALL    RS232StopBit    ; | |
    CALL    RS232StopBit    ; | |
    CALL    RS232StopBit    ; | |
    CALL    RS232StopBit    ; | |
    CALL    RS232StopBit    ; | |
    CALL    RS232StopBit    ; | |
    CALL    RS232StopBit    ; | |
    RETLW   h’00’           ; | return

RS232WaitForever                ;[1] ~9600 baud
BigWaitL1                       ; | {
    CLRWDT                  ;!|
    BTFSS   _RS232IN        ; |   if (_RS232IN==#0)
    GOTO    RS232RxL1Done   ; |   { goto RS232RxL1Done }
    NOP                     ;!|
    GOTO    BigWaitL1       ; | } until (0)

RS232Rx                         ;[1] ~9600 baud
    MOVLW   d’16’           ; | Set timeout of ~2.9s
    MOVWF   TimerHi         ; | |
    CLRF    TimerMid        ; | |
    CLRF    TimerLo         ; | |
RS232RxL1                       ; | {
    CLRWDT                  ;!|
    BTFSS   _RS232IN        ; |   if (_RS232IN==#0)
    GOTO    RS232RxL1Done   ; |   { goto RS232RxL1Done }
    DECFSZ  TimerLo,f       ; | }
    GOTO    RS232RxL1       ; | |
    DECFSZ  TimerMid,f      ; | |
    GOTO    RS232RxL1       ; | |
    DECFSZ  TimerHi,f       ; | |
    GOTO    RS232RxL1       ; | |
    BSF     _CARRY          ; | return with error
    RETLW   h’00’           ; | |
               ; |
RS232RxL1Done                   ; |% 3 to (+6, +8, +10) - say 10us
               ; |% 10-104=-94
    MOVLW   d’90’           ;!|
    CALL    DelayW12        ;!|% 9
    CLRF    BitCtr          ; | BitCtr=#8
    BSF     BitCtr,3        ; | |
RS232RxLoop                     ; | {% 11
    MOVLW   d’181’          ;!|
    CALL    DelayW12        ;!|  % 205
    CLRF    BO3             ; |   BO3,1=_RS232IN
    BTFSC   _RS232IN        ; |   |
    INCF    BO3,f           ; |   |% 208
    BTFSC   _RS232IN        ; |   |% 1
    INCF    BO3,f           ; |   |
    BTFSC   _RS232IN        ; |   |
    INCF    BO3,f           ; |   |% 4
    RRF     RxByte,f        ; |   RR RxByte
    BCF     RxByte,7        ; |   RxByte,7=BO3,1
    BTFSC   BO3,1           ; |   |
    BSF     RxByte,7        ; |   |% 8
    DECFSZ  BitCtr,f        ; |   DEC BitCtr
    GOTO    RS232RxLoop     ; | } until (BitCtr==#0)
    BCF     _CARRY          ; | return with no error
    RETLW   h’00’           ; | |

RS232TxW                        ;[1] Transmit W on RS232 at ~9600 baud
    MOVWF   TxByte          ; | TxByte=W
    CALL    RS232StopBit    ; | stop bit
    CLRF    BitCtr          ; | BitCtr=#8
    BSF     BitCtr,3        ; | |
    BCF     _RS232OUT       ; | Start bit
    MOVLW   d’191’          ; | |
    CALL    DelayW12        ; | |
RS232TxLoop                     ; | {% 205
    BTFSS   TxByte,0        ; |   _RS232OUT=TxByte,0
    BCF     _RS232OUT       ; |   |% 207
    BTFSC   TxByte,0        ; |   |% 208
    BSF     _RS232OUT       ; |   |% 1
    RRF     TxByte,f        ; |   RR TxByte
    MOVLW   d’187’          ;!|
    CALL    DelayW12        ;!|  % 202
    DECFSZ  BitCtr,f        ; |   DEC BitCtr
    GOTO    RS232TxLoop     ; | } until (BitCtr==#0)
    GOTO    RS232TxJ1       ; |
RS232TxJ1                       ; |
    NOP                     ; |% 207
    BSF     _RS232OUT       ; | Stop bit
    RETLW   h’00’           ; | return

        end
```

NOTES:

NOTES:

NOTES:

NOTES:

MICROCHIP

WORLDWIDE SALES AND SERVICE

AMERICAS

Corporate Office

Microchip Technology Inc.
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 602-786-7200 Fax: 602-786-7277
Technical Support: 602 786-7627
Web: http://www.microchip.com

Atlanta

Microchip Technology Inc.
500 Sugar Mill Road, Suite 200B
Atlanta, GA 30350
Tel: 770-640-0034 Fax: 770-640-0307

Boston

Microchip Technology Inc.
5 Mount Royal Avenue
Marlborough, MA 01752
Tel: 508-480-9990 Fax: 508-480-8575

Chicago

Microchip Technology Inc.
333 Pierce Road, Suite 180
Itasca, IL 60143
Tel: 630-285-0071 Fax: 630-285-0075

Dallas

Microchip Technology Inc.
14651 Dallas Parkway, Suite 816
Dallas, TX 75240-8809
Tel: 972-991-7177 Fax: 972-991-8588

Dayton

Microchip Technology Inc.
Two Prestige Place, Suite 150
Miamisburg, OH 45342
Tel: 937-291-1654 Fax: 937-291-9175

Detroit

Microchip Technology Inc.
42705 Grand River, Suite 201
Novi, MI 48375-1727
Tel: 248-374-1888 Fax: 248-374-2874

Los Angeles

Microchip Technology Inc.
18201 Von Karman, Suite 1090
Irvine, CA 92612
Tel: 714-263-1888 Fax: 714-263-1338

New York

Microchip Technology Inc.
150 Motor Parkway, Suite 202
Hauppauge, NY 11788
Tel: 516-273-5305 Fax: 516-273-5335

San Jose

Microchip Technology Inc.
2107 North First Street, Suite 590
San Jose, CA 95131
Tel: 408-436-7950 Fax: 408-436-7955

AMERICAS (continued)

Toronto

Microchip Technology Inc.
5925 Airport Road, Suite 200
Mississauga, Ontario L4V 1W1, Canada
Tel: 905-405-6279 Fax: 905-405-6253

ASIA/PACIFIC

Hong Kong

Microchip Asia Pacific
RM 3801B, Tower Two
Metroplaza
223 Hing Fong Road
Kwai Fong, N.T., Hong Kong
Tel: 852-2-401-1200 Fax: 852-2-401-3431

India

Microchip Technology Inc.
India Liaison Office
No. 6, Legacy, Convent Road
Bangalore 560 025, India
Tel: 91-80-229-0061 Fax: 91-80-229-0062

Japan

Microchip Technology Intl. Inc.
Benex S-1 6F
3-18-20, Shinyokohama
Kohoku-Ku, Yokohama-shi
Kanagawa 222-0033 Japan
Tel: 81-45-471- 6166 Fax: 81-45-471-6122

Korea

Microchip Technology Korea
168-1, Youngbo Bldg. 3 Floor
Samsung-Dong, Kangnam-Ku
Seoul, Korea
Tel: 82-2-554-7200 Fax: 82-2-558-5934

Shanghai

Microchip Technology
RM 406 Shanghai Golden Bridge Bldg.
2077 Yan'an Road West, Hong Qiao District
Shanghai, PRC 200335
Tel: 86-21-6275-5700 Fax: 86 21-6275-5060

ASIA/PACIFIC (continued)

Singapore

Microchip Technology Singapore Pte Ltd.
200 Middle Road
#07-02 Prime Centre
Singapore 188980
Tel: 65-334-8870 Fax: 65-334-8850

Taiwan, R.O.C

Microchip Technology Taiwan
10F-1C 207
Tung Hua North Road
Taipei, Taiwan, ROC
Tel: 886-2-2717-7175 Fax: 886-2-2545-0139

EUROPE

United Kingdom

Arizona Microchip Technology Ltd.
505 Eskdale Road
Winnersh Triangle
Wokingham
Berkshire, England RG41 5TU
Tel: 44-1189-21-5858 Fax: 44-1189-21-5835

France

Arizona Microchip Technology SARL
Zone Industrielle de la Bonde
2 Rue du Buisson aux Fraises
91300 Massy, France
Tel: 33-1-69-53-63-20 Fax: 33-1-69-30-90-79

Germany

Arizona Microchip Technology GmbH
Gustav-Heinemann-Ring 125
D-81739 München, Germany
Tel: 49-89-627-144 0 Fax: 49-89-627-144-44

Italy

Arizona Microchip Technology SRL
Centro Direzionale Colleoni
Palazzo Taurus 1 V. Le Colleoni 1
20041 Agrate Brianza
Milan, Italy
Tel: 39-39-6899939 Fax: 39-39-6899883

9/8/98

DNV Certification, Inc.
USA
ACCREDITED
ANSI-RAB
REGISTRAR
DNV
DNV MSC
The Netherlands
Accredited by the RvA
ISO 9001 REGISTERED FIRM

Microchip received ISO 9001 Quality System certification for its worldwide headquarters, design, and wafer fabrication facilities in January, 1997. Our field-programmable PICmicro™ 8-bit MCUs, Serial EEPROMs, related specialty memory products and development systems conform to the stringent quality standards of the International Standard Organization (ISO).

Information contained in this publication regarding device applications and the like is intended for suggestion only and may be superseded by updates. No representation or warranty is given and no liability is assumed by Microchip Technology Incorporated with respect to the accuracy or use of such information, or infringement of patents or other intellectual property rights arising from such use or otherwise. Use of Microchip's products as critical components in life support systems is not authorized except with express written approval by Microchip. No licenses are conveyed, implicitly or otherwise, under any intellectual property rights. The Microchip logo and name are registered trademarks of Microchip Technology Inc. in the U.S.A. and other countries. All rights reserved. All other trademarks mentioned herein are the property of their respective companies.

Microchip Technology Inc.
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 602.786.7200 Fax: 602.899.9210