Application Note

RF Link Using the Z86E08

AN005801-Z8X0500

ZiLOG WORLDWIDE HEADQUARTERS • 910 E. HAMILTON AVENUE • CAMPBELL, CA 95008
TELEPHONE: 408.558.8500 • FAX: 408.558.8300 • www.ZiLOG.com

This publication is subject to replacement by a later edition. To determine whether a later edition exists, or to request copies of publications, contact:

ZiLOG Worldwide Headquarters
910 E. Hamilton Avenue
Campbell, CA 95008
Telephone: 408.558.8500
Fax: 408.558.8300
www.ZiLOG.com

Information Integrity

The information contained within this document has been verified according to the general principles of electrical and mechanical engineering. Any applicable source code illustrated in the document was either written by an authorized ZiLOG employee or licensed consultant. Permission to use these codes in any form, besides the intended application, must be approved through a license agreement between both parties. ZiLOG will not be responsible for any code(s) used beyond the intended application. Contact the local ZiLOG Sales Office to obtain necessary license agreements.

Document Disclaimer

Table of Contents

Acknowledgements

Project Lead Engineer

Mark Shaw

Application and Support Engineer

Mark Shaw

System and Code Development

Mark Shaw

RF Link Using the Z86E08

General Overview

Low power RF systems are an extremely popular means of transmitting wireless data. These units typically transmit less than 1 mW of power and operate over distances of 3 to 60 meters. The advantage of these systems is that when certified to meet local communications regulations, they do not require a license for operation. More than 60 million short-range wireless products are manufactured each year with sales expected to top 100 million systems by the year 2000.

This application note provides the schematics and software required to build an RF Link using a ZiLOG Inc Z86E08 microcontroller on the Protocol Board of the RF Monolithics (RFM), Virtual Wire, Radio Design Kit. This design kit allows the implementation of low-power wireless ASCII communications between two PCs with RS-232C serial ports. The kit contains two communications nodes with each node consisting of a protocol board and a data radio board. This kit is an excellent tool for evaluating the feasibility of a low-speed wireless data application as well as facilitating the development of the actual design.

The following list identifies some applications for short-range wireless data systems:

- Wireless bar-code and credit-card readers
- Wireless and bar-code label printers and credit-card receipt printers
- Smart ID tags for inventory tracking and identification
- Wireless automatic utility meter reading systems
- Communications links for hand-held terminals, HPCs and PDAs
- Wireless keyboards, joysticks, mice and game controls
- Portable and field data logging
- Location tracking (follow-me phone extensions, etc.)
- Sports telemetry
- Surveying system data links
- Engine diagnostic links
- Polled wireless security alarm sensors

- Authentication and access control tags

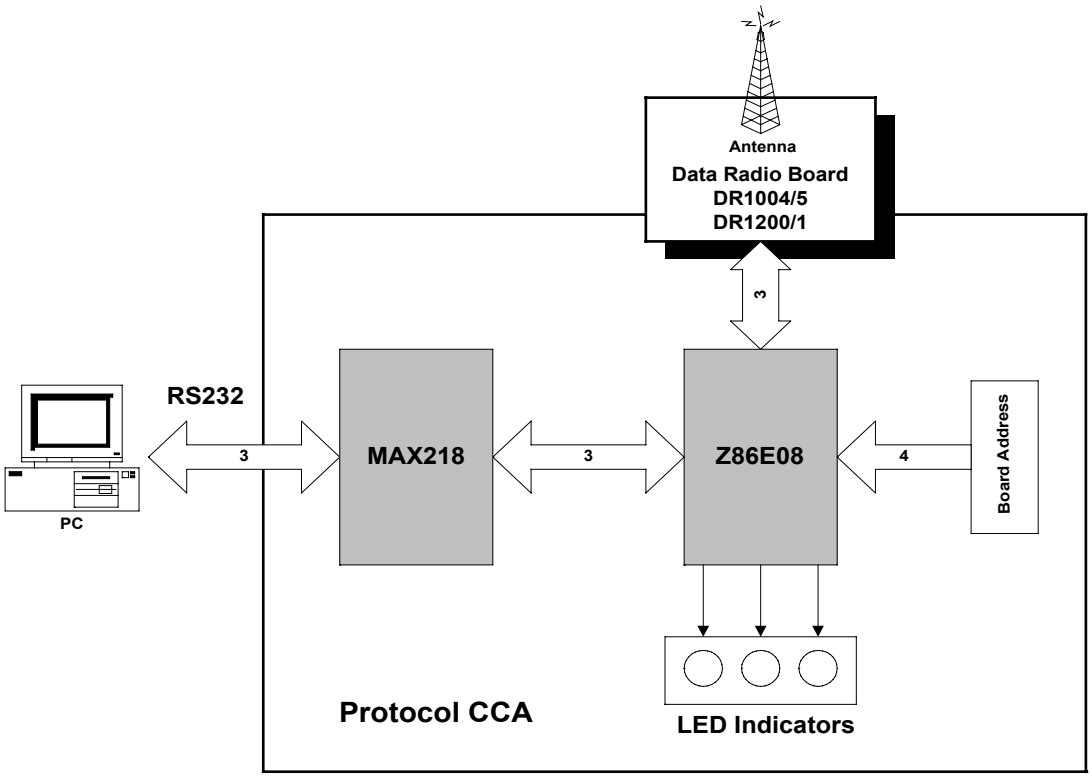
Further information concerning the hardware and software described by this application note can be found at the RFM web page at www.rfm.com.

Discussion

Hardware Description

Figure 1 contains the design kit block diagram. The kit contains two circuit card assemblies: The Data Radio CCA and the Protocol CCA.

Figure 1. Hardware Block Diagram



The Data Radio can be either the RFM, DR1004/5 or DR 1200/1 Data Radio CCA. This board contains the RF transceiver or receiver/transmitter pair, discrete circuitry, and the antenna necessary to generate and receive ~900-MHz RF communication. The transceiver includes provisions for on-off keyed (OOK) or amplitude-

shift keyed (ASK) modulation. This application uses OOK. The Data Radio operates from 3V power supplied by the Protocol CCA.

The Protocol CCA contains all of the circuitry necessary to interface to the Data Radio CCA and a PC, including RS-232 communication to the PC and packet protocol serial communication to the Data Radio CCA. The CCA also contains jumpers for setting a 4-bit board address and 3 LEDs for monitoring communications status. Power is supplied by three 1.5-V batteries.

Z8 Microcontroller

The microcontroller selected for this application is the Z86E08. It demonstrates a minimal cost and configuration for the application while maintaining the expected performance of the RF Link. The Z8 controls the four primary functions of the Protocol CCA. These functions are listed below and described in the Software Overview section.

- RS-232 communications to the PC
- RF serial communication to the Data Radio Board
- Interpretation of board address
- Control of LED indicators

The Z86E08 is an 18-pin device with 14 available I/O pins. The I/O pins are arranged into two 3-bit ports, Port 0 and Port 3, and one 8-bit port, Port 2. Port 0 (P02–P00) is a dedicated output port. Port 2 (P27–P20) is a bidirectional port with each pin independently configurable as input or output. Port 3 (P33–P31) is a dedicated input port.

Note: P32 and P31 are external interrupt sources and are therefore used by the serial communications inputs (RF and PC). Table 1 provides a pin-out and pin description.

Table 1. Z86E08 Pin Configuration

Pin #	Name	Symbol	Function	Direction
Pin 1	D3	P24	D3 (PC RCV) Diode Enable. Active Low	Output
Pin 2	D4	P25	D4 (RF RCV) Diode Enable. Active Low	Output
Pin 3	D5	P26	D4 (RXI) Diode Enable. Active Low	Output
Pin 4	PTT_	P27	Radio Transmit Enable. Active Low	Output
Pin 5	POWER	V_{CC}	Power	Input
Pin 6	XTAL2	XTAL2	Crystal Oscillator Clock	Input
Pin 7	XTAL1	XTAL1	Crystal Oscillator Clock	Input

Table 1. Z86E08 Pin Configuration (Continued)

Pin #	Name	Symbol	Function	Direction
Pin 8	232_RX	P31	RS232 Receive Data	Input
Pin 9	RRX	P32	Radio Receive Data	Input
Pin 10	N/A	P33	Unused	Input
Pin 11	RTX	P00	Radio Transmit Data	Output
Pin 12	232_TX	P01	RS232 Transmit Data	Output
Pin 13	SHDN_	P02	RS232 Transmit Disable. Active Low	Output
Pin 14	GND	GND	Ground	Input
Pin 15	ID0	P20	Board Address/ID Bit 0	Input
Pin 16	ID1	P21	Board Address/ID Bit 1	Input
Pin 17	ID2	P22	Board Address/ID Bit 2	Input
Pin 18	ID3	P23	Board Address/ID Bit 3	Input

Firmware Overview

The primary functions of the Protocol CCA reside in the firmware within the Z86E08. The majority of the processing consists of two major interrupt service routines: receiving PC RS-232 data (PC_RCV) and receiving RF serial data (RF_RCV). The processor waits in the main processing loop until one of these two interrupts is received. Figure 2 contains a high-level flowchart for these routines.

RS-232 Communication

The software features an RS-232 communication port controlled entirely by software. This port operates at 9600 baud, half-duplex, with no flow control. A data byte consists of 10 bits: 1 start bit (active Low), 8 data bits, and 1 stop bit (active High). Between transmissions, the port resets High. Receipt of a data byte is initiated by a falling edge of the 232_RX (P31) pin. After a byte is received, it is stored in RAM. When an entire message is received, the data is processed and appropriate actions taken. Figure 3 contains a flow diagram for receiving data on the RS-232 communications port.

RS-232 Packet Protocol

A PC message packet contains the data to be sent over the RF link. As shown below, the basic elements of a packet are the TO/FROM address, the packet number, the message size, and the message itself. All elements are 8-bit data segments. The first byte of a packet contains the TO/FROM address. The High nibble contains the TO node address of the Protocol CCA and the Low nibble contains the FROM node address. The second byte contains the packet number. The packet number is a message sequence number incremented each time an RF

message is successfully transmitted. A valid packet number is restricted to values 1 through 7. The number 8 is reserved for telemetry packets. The third byte contains the size. The size indicates the number of data bytes contained in the message. A message can contain up to 32 bytes. However, the first byte and most recent byte contain the start byte (0x02) and the stop byte (0x03) thereby limiting the actual data stream to 30 bytes.

PC Message Packet:

```
| TO/FROM | Packet#, 01hto08h | Size | Message (up to 32 bytes) |
```

When the host software is ready to transmit a message packet, it tests the availability of the communications port by sending the TO/FROM byte. If the TO/FROM byte is returned to the protocol CCA within 50ms, the host software controls the RS-232 communications interrupt and sends the remaining data bytes. This test applies to all message packets including the special messages discussed below. Figure 5 illustrates the timing and flow for a PC message packet transmission to the Protocol CCA.

Special messages are data packets intended only for the Protocol CCA. This message is a request for information or an instruction to perform a specific task. The packet format is the same as a PC message packet A special message is indicated by a Packet number of 00h. A list of special messages follows:

`Reset` instructs the receiving unit to perform a software reset:

```
| FROM/FROM | 00h | 01h | 30h |
```

`Send Node Address` requests the node address of the receiving unit:

```
| 00h | 00h | 01h | 31h |
```

`Set Node Address` sets the node address of the receiving unit to ADDR:

```
| 00h | 00h | 02h | 34h | ADDR |
```

`Run Self Test` instructs the receiving unit to run an internal SRAM memory test.

```
| FROM/FROM | 00h | 01h | 33h |
```

All communication from the PC to the Protocol CCA requires a response from the Protocol CCA to the PC in complete the transaction. Figure 6 contains the timing and flow for all of the special message transmissions between the PC and the Protocol CCA. A list of valid responses and a description of each follow.

The message from `Host too Long` indicates that the unit received more than the allowable 32 characters. Firmware limits data to a length of 32 bytes:

```
| FROM/FROM | 00h | 01h | 30h |
```

`Failed Self Test` indicates that the microcontroller failed the internal SRAM memory test:

```
| FROM/FROM | 00h | 01h | 33h |
```

`Passed Self Test` indicates that the microcontroller passed the internal SRAM memory test:

```
| FROM/FROM | 00h | 01h | 34h |
```

`Local Node Address` sends the local node address of the Protocol CCA. The address is contained in the FROM/FROM byte:

```
| FROM/FROM | 00h | 01h | 35h |
```

Link status messages are packets transmitted to the PC to indicate the status of the RF link. The purpose of these messages is to inform the PC whether an RF transmission was successful.

`ACK` indicates that the message is acknowledged, and that the RF packet has been transmitted and received successfully.

```
| TO/FROM | Packet # | En |     (n=1 to 8), number of retries
```

`NAK` indicates that the RF packet was transmitted and the receiving unit did not acknowledge receipt of the packet:

```
| TO/FROM | Packet # | DDh |
```

RF Communications Port

The software features an RF serial data port for communication with the Data Radio CCA. This port is controlled by software and operates at 9600 baud half-duplex. A data segment consists of 14 bits: 1 start bit (active Low), 12 DC-balanced data bits, and 1 stop bit (active High). To meet the polarity requirements of the data radio, all RF serial data is inverted in software before the data is placed on the output port. Receipt of data is initiated by a falling edge of the RRX (P32) pin. After a byte is received, it is stored in RAM. When an entire message is received, the data is processed. Figure 3 contains a flow diagram for receiving data on the RF Communication Port.

RF Packet Protocol

As shown below, the basic elements of a data packet are the TO/FROM address, the packet number, the message size, and the message. All the elements are 8-bit data segments. The first byte of a 232 message contains the TO/FROM address. The High nibble contains the TO node address of the Protocol CCA and the Low nibble contains the FROM node address. The second byte contains the packet number. The packet number is a message sequence number that is incremented

each time an RF message is successfully transmitted. A valid packet number is restricted to values 1 through 7. The number 8 is reserved for telemetry packets. The third byte contains the size. The size indicates the number of data bytes contained in the message. A message contains up to 32 bytes. Upon successful receipt of a message and verification of the TO/FROM address, the software processes the received data.

Data Message:

```
| 0x55h | TO/FROM | Packet # | Size | 0x02 | Message (up to 30
characters) | 0x03 | FCSHI | FCSLO|
```

Upon successful receipt and verification of an RF Packet, the software sends a receipt acknowledgment (ACK) to the originator. The acknowledgment consists of a start byte (0x55h), the TO/FROM address, the packet number, En (where n is the retry number), and the FCS.

Data Acknowledge:

```
| 0x55h | TO/FROM | Packet # | En | FCSHI | FCSLO |
```

After transmitting the data message, the message originator waits approximately 10ms for the data acknowledgment packet. If a timeout occurs, the originator delays an additional 25ms before re-sending the data packet. The data is resent up to seven times before the originator sends a `data not acknowledged` (NAK) message to the PC. Figure 7 illustrates the timing and flow for an RF message retry. Figure 8 illustrates the timing and flow for an RF communications timeout.

An additional transmit option is the Broadcast Mode. A broadcast message is intended for all the nodes rather than a single address. A TO/FROM address of 00h indicates to the receiving node that the message is a broadcast. The receiving node, therefore, does not respond with a data acknowledgment. Because `ACK` is not used, the originator sends the broadcast packet eight times to enhance the probability of reception. Figure 9 illustrates the message broadcast timing.

DC-Balancing the Receiver

To increase transmission efficiency, all the data is DC-balanced by the software before the data is transmitted. The software encodes the data to condition the transmitted signal for efficient AC coupling at the receiver. The encoding scheme used for this application is an 8- to 12-bit symbol conversion. Each 8-bit byte of data is encoded as 12-bits with six `1` bits and six `0` bits to balance the data. Also, it is important to limit the number of `1` and `0` pulses that occur consecutively. Consecutive identical pulses are limited to no more than three. To perform this conversion in software, a lookup table is used on each nibble of data to convert the nibble from 4 bits to the encoded 6 bits. Table 2 lists the conversion values for the encoding.

Table 2. DC Balance Encoding

8-bit Data	12-bit Encoding
0000	010101 (15h)
0001	110001 (31h)
0010	110010 (32h)
0011	010011 (13h)
0100	110100 (34h)
0101	100101 (25h)
0110	100110 (26h)
0111	000111 (07h)
1000	111000 (38h)
1001	101001 (29h)
1010	101010 (2Ah)
1011	001011 (0Bh)
1100	101100 (2Ch)
1101	001101 (0Dh)
1110	001110 (0Eh)
1111	011100 (1Ch)

To further increase the receiver efficiency, a 24-bit preamble is sent to the receiver prior to all RF message transmissions. This preamble is a string of 24 alternating 1 and 0 pulses. The purpose of the preamble is to train the receiver to optimum efficiency prior to sending actual data. The firmware detects a preamble and ignores the data stream.

RF Noise Immunity

Message validation is performed on all received data. This process determines as fast and reliably as possible whether an interrupt on the RRX input was caused by valid data or RF noise. Several tests are required to determine the validity of a message.

When an interrupt is received, the firmware determines whether the incoming data is a new message or the next byte of a message already in progress. If the data is a new message, the firmware verifies the first three bits of the preamble. If any of the pulses fail, the firmware immediately exits to the main loop to await the next interrupt. If the pulses are verified, then the firmware runs a delay loop to the end of the preamble before waiting for the next data bit.

When a 12-bit encoded data segment is received, the value is converted to 8-bit format. If either of the nibbles fails to match a valid value from the encoding table, the message is deemed invalid and the firmware immediately exits to the main loop to await the next interrupt.

When an entire message is received, the Frame Check Sequence (FCS) is calculated and verified. The FCS is calculated based on all of the bits in the message, excluding the start byte. The FCS used by this application is defined by ISO 3309. When the FCS is verified, the data is sent as a data message to the PC. If the FCS fails verification, the firmware immediately exits to the main loop to await the next interrupt.

LED Indicators

The Protocol CCA contains three LED indicators. The RXI LED indicates that the firmware is processing an interrupt of the RRX line. The RF RCV indicates that a valid RF Packet was received. The PC RCV indicates that a valid message from the PC was received.

Node Addressing

Each Protocol CCA contains a set of four jumpers to set the node address. This binary address is interpreted by the firmware to determine whether or not a message is intended for that node. Therefore, using four jumpers, 16 different addresses theoretically can be set. Because of the way the address is processed, only 15 node addresses are possible. The node address of the board is equal to the jumper value plus 1. Therefore, with no jumpers set, the node address is 1. The exception occurs when jumpers are placed on all pins, which is also interpreted as node address 1. Note that node address 0 is not a valid address. Node 0 is reserved for packets broadcast to all other nodes. Table 3 contains the interpretation table

Table 3. Node Address Table

Jumper Value (binary)	Node Address
0000	1
0001	2
0010	3
0011	4
0100	5
0101	6
0110	7
0111	8

Table 3. Node Address Table

Jumper Value (binary)	Node Address
1000	9
1001	10
1010	11
1011	12
1100	13
1101	14
1110	15
1111	1

Summary

This application provides a reliable and inexpensive start for developing a low-power wireless data communications application. The application code readily fits into the 2-KB program space of the Z86E08, utilizing approximately 1.7KB. If additional I/O or code space is required, the code transfers to larger microcontrollers in the Z8 family.

Technical Support

Assembling the Application Code

Any Z8 assembler can be used to assemble the application code, but ZiLOG Development Studio (ZDS) is recommended. This integrated suite of software tools allows for program file handling, editing, real-time emulation, and debugging when used with the appropriate emulator. Future versions of ZDS incorporate a C-Compiler, simulator and trace buffer. See ZiLOG's web page for news and free downloads of ZDS.

Place the `.ASM` file into it's own sub-directory. Invoke ZDS and select a new project from the file menu. Under `Target Selection`, select `Family`. Under `Master Select`, select `Z8`. Under `Project Target`, select `Z86E08`. Select the appropriate emulator type to be used. Browse to fill in the project name by clicking on the … key. Select the sub-directory containing the `.ASM` file, name the project (the extension is added), click `Save`, and the first ZDS screen re-appears with the project name, path and file extension filled in. If everything is acceptable, click `OK`.

Click on the Project tab and select Add to Project. Then select Files. Double click on the RFLink.asm file. This file is displayed in the project window. Next, click on the Build tab and select Build. The Output window displays the assembly results. The standard assembler and linker settings produce listing and hex files along with the ZDS files in the same sub-directory. Save the project and its files by clicking on the File tab to select these options. The ZDS Project File is included and when the ZDS is installed, the above steps can be omitted for program assembly.

Programming a One Time Programmable (OTP) is accomplished by selecting the OTP option with the hex code installed. Do not install the OTP until access to it is required, either for blank checking, verification, or programming. Insert a blank Z86E08 into the OTP socket and click on the program OTP selection. Differences exist between earlier GUIs and the ZDS, so take the time to read and understand the operation of the software in use. A recommendation is to pad unused memory locations with FFh before programming. If the padding is not consistently done, differences in the checksum occurs.

Source Code

```
;-------------------------------------------------------------------------
;           RFZ8.ASM (Version 1.0) 12/17/99
;           Asynchronous 232 and RF Interface for Z86x08
;           Baud Rate is 9600 for both channels
;           Half-Duplex Operation Mode.
;
;      Signal       Pin        Z86E08      Pin     Signal
;      -----------  -----     ---------    -----   -----------
;      D3           P24   -- | 1    18 |-- P23     ID3
;      D4           P25   -- | 2    17 |-- P22     ID2
;      D5           P26   -- | 3    16 |-- P21     ID1
;      PTT_         P27   -- | 4    15 |-- P20     ID0
;      Power        VCC   -- | 5    14 |-- GND     Ground
;      Crystal Out  XTAL2 -- | 6    13 |-- P02     SHDN_
;      Crystal In   XTAL1 -- | 7    12 |-- P01     232_TX
;      232_RX       P31   -- | 8    11 |-- P00     RTX
;      RRX          P32   -- | 9    10 |--
;                            ---------
;
;     SCLK = 6 MHz (FOSC = 12 MHz in OSC/2 Standard Mode)
;
;     Assembler: ZDS V3.0
;     Best if viewed in Courier New 9 pt with 7 space tabs
;-------------------------------------------------------------------------

      GLOBALS ON
      DEFINE      REGDATA, SPACE=RFILE

;*************************************************************************
;*               Global Definitions
;*************************************************************************
      SEGMENT     REGDATA
```

```
            DS    4           ; Ports
P0COPY      DS    1           ; %04
P2COPY      DS    1           ; %05
P3COPY      DS    1           ; %06
RCV_STATUS  DS    1           ; %07
FRAME_ERR   DS    1           ; %08
MY_ADDR     DS    1           ; %09
FROMFROM    DS    1           ; %0A
SERDATA     DS    1           ; %0B
RFDATA      DS    1           ; %0C
RETRY       DS    1           ; %0D
DATA_PTR    DS    1           ; %0E
SPCL_DATA   DS    1           ; %0F
            DS    14          ; %10-1D
COPYSIZE    DS    1           ; %1E
BYTECNT     DS    1           ; %1F
STARTBYTE   DS    1           ; %20
TOFROM      DS    1           ; %21
PACKET      DS    1           ; %22
SIZE        DS    1           ; %23
DATABUF     DS    1           ; %24
            DS    33          ; %25
FCSHI       DS    1           ; %46
FCSLO       DS    1           ; %47
ESTARTBYTE  DS    1           ; %48
ETOFROM     DS    1           ; %49
EPACKET     DS    1           ; %4A
EFCSHI      DS    1           ; %4B
EFCSLO      DS    1           ; %4C
            DS    19          ; %4D
TABLE       DS    16          ; %60-6F

;*****************************************************************************
;     Register Definitions
;*****************************************************************************
;
;  RCV_STATUS:
;  BIT0 - PC_RCV        Receiving Serial Data from PC
;  BIT1 - RF_DATA_WAIT  Waiting for RF Data (Preamble rcvd)
;  BIT2 - PC_TIMEOUT    Timeout of PC Channel occurred
;  BIT3 - RF_TIMEOUT    Timeout of RF Channel occurred
;  BIT4 - RF_ECHO_FAIL  Timeout of RF Channel awaiting echo
;  BIT5 - RF_ECHO_WAIT  Waiting for echo on RF channel
; BIT6 - EMPTY
;  BIT7 - BROADCAST     TOFROM=00 (from PC) for RF Broadcast
;
;*************************************************************
; RAM bank 0 (RP=00H for working register usage. Physical    *
;             addresses 00H to 0FH).                         *
;*************************************************************
;

PORT_GRP    .EQU  00h   ; Register Work Group 0
_P0         .EQU  r0    ; Port 0, 4-bit address
_P2         .EQU  r2    ; Port 2, 4-bit address
_P3         .EQU  r3    ; Port 3, 4-bit address
_P2COPY     .EQU  r5    ; Port 2 shadow register
```

```
;*************************************************************
; RAM bank 6 (RP=60H for working register usage. Physical    *
;             addresses 60H to 6FH).                         *
; This bank holds the table for DC balancing the RF output   *
; This table is stored in RAM to increase the lookup speed   *
;*************************************************************
;

_TABLE      .EQU   60h;
NIBBLE0     .EQU   15H    ;  NIBBLE = 0    010101
NIBBLE1     .EQU   31H    ;  NIBBLE = 1    110001
NIBBLE2     .EQU   32H    ;  NIBBLE = 2    110010
NIBBLE3     .EQU   23H    ;  NIBBLE = 3    100011
NIBBLE4     .EQU   34H    ;  NIBBLE = 4    110100
NIBBLE5     .EQU   25H    ;  NIBBLE = 5 100101
NIBBLE6     .EQU   26H    ;  NIBBLE = 6    100110
NIBBLE7     .EQU   07H    ;  NIBBLE = 7    000111
NIBBLE8     .EQU   38H    ;  NIBBLE = 8    111000
NIBBLE9     .EQU   29H    ;  NIBBLE = 9    101001
NIBBLE10    .EQU   2AH    ;  NIBBLE = 10   101010
NIBBLE11    .EQU   0BH    ;  NIBBLE = 11   001011
NIBBLE12    .EQU   2CH    ;  NIBBLE = 12   101100
NIBBLE13    .EQU   0DH    ;  NIBBLE = 13   001101
NIBBLE14    .EQU   0EH    ;  NIBBLE = 14   001110
NIBBLE15    .EQU   1CH    ;  NIBBLE = 15   011100

;*************************************************************
; RAM bank F (RP=F0H for working register usage. Physical
;             addresses F0H to FFH).
;*************************************************************
;

CTRL_GRP    .EQU  0F0h  ; Register Work Group F
_TMR        .EQU   r1   ;Timer Mode Register
_T1         .EQU   r2   ;Timer T1
_PRE1       .EQU   r3   ;T1 Prescaler
_P2M        .EQU   r6   ;Port 2 Mode Register
_P3M        .EQU   r7   ;Port 3 Mode Register
_P01M       .EQU   r8   ;Port 0/1 Mode Register
_IPR        .EQU   r9   ;Interrupt Priority Register
_IMR        .EQU   r11  ;Interrupt Mask Register
_SPL        .EQU   r15  ;Stack Pointer Low

;*************************************************************
;*               Constants Definitions
;*************************************************************
;
SCRATCHPAD  .EQU  50h   ;
PC_PRE      .EQU  07h   ;
PC_BAUD     .EQU  156   ; 104us ~(9.6kbps)
RF_BAUD     .EQU  156   ; 104us ~(9.6kbps)
BIT0        .EQU   01h  ; Bit 0 mask
BIT1        .EQU   02h  ; Bit 1 mask
BIT2        .EQU   04h  ; Bit 2 mask
BIT3        .EQU   08h  ; Bit 3 mask
BIT4        .EQU   10h  ; Bit 4 mask
BIT5        .EQU   20h  ; Bit 5 mask
BIT6        .EQU   40h  ; Bit 6 mask
BIT7        .EQU   80h  ; Bit 7 mask
MASK0       .EQU  0Eh   ; Bit 0 Mask
MASK1       .EQU  0Dh   ; Bit 1 Mask
```

```
MASK_ID     .EQU  0Fh   ; ID Mask
ON_232      .EQU  04h   ; Turn RS232 Drive On
OFF_232     .EQU  0FBh  ; Turn RS232 Drive Off
PTT_ON      .EQU  7Fh   ; Turn PTT (RF Xmt) On
PTT_OFF     .EQU  80h   ; Turn PTT (RF Xmt) Off
D3_ON       .EQU  0E0h  ; Turn on D3
D3_OFF      .EQU  10h   ; Turn off D3
D4_ON       .EQU  0D0h  ; Turn on D4
D4_OFF      .EQU  20h   ; Turn off D4
D5_ON       .EQU  0B0h  ; Turn off D5
D5_OFF      .EQU  40h   ; Turn off D5
RAM_TOP     .EQU  7Fh   ; Top of RAM
RAM_BOT     .EQU  07h   ; Bottom of RAM
RF_IMR      .EQU   01h  ; IMR for IRQ0 (RF_RCV)
PC_IMR      .EQU  04h   ; IMR for IRQ2 (PC_RCV)
PCRF_IMR    .EQU  05h   ; IMR for IRQ2/IRQ0
TMR1_IMR    .EQU  20h   ; IMR for IRQ5 (Timer1)
TMR0_IMR    .EQU  10h   ; IMR for IRQ4 (Timer0)
TMR10_IMR   .EQU  30h   ; IMR for IRQ5/IRQ4
PCTMR0_IMR  .EQU  14h   ; IMR for IRQ2/IRQ4
RFTMR0_IMR  .EQU  11h   ; IMR for IRQ2/IRQ4
RFTMR1_IMR  .EQU  21h   ; IMR for IRQ2/IRQ5

;*****************************************************************************
;               Interrupt Vector Table
;*****************************************************************************
;
      SEGMENT code

      VECTOR      IRQ0= RF_RCV
      VECTOR      IRQ1= START
      VECTOR      IRQ2= PC_RCV
      VECTOR      IRQ3= START
      VECTOR      IRQ4= TIMER0
      VECTOR      IRQ5= TIMER1
      VECTOR      RESET=      START

;*****************************************************************************
;     Initialization Section
;     Functions:
;     1) Initialize I/O ports
;     2) Clear memory (RAM)
;     3) Initialize DC Balance Table into RAM
;     4) Retrieve and process Protocol CCA address
;     5) Initialize Stack
;     6) Initialize interrupts and set interrupt priority
;*****************************************************************************
;
START:
            srp   #0h               ;
            ld    P2COPY,#0F0h      ; LEDs off, PTT off
            ld    P0COPY,#02h       ; serial ports High, SHDN_ High
            ld    P2, P2COPY        ;
            ld    P0, P0COPY        ;
            ld    P3M,#01h          ; Port3 -> Digital Inputs
            ld    P2M,#00fh         ; Port2 -> P24,P25,P26 Outputs
            ld    P01M,#04h         ; Port1 -> P01-P03 Inputs
            srp   #CTRL_GRP         ;
```

```
            ld    r15,#RAM_BOT      ; Clear RAM from 07h to 7Fh
initram:
            clr   @r15              ; RAM_BOT = 07h
            inc   r15               ; RAM_TOP = 7Fh
            cp    r15,#RAM_TOP+1    ;
            jr    nz,initram        ;

            srp   #_TABLE           ; This section initializes the
            ld    r0,#NIBBLE0       ; BALANCE TABLE into RAM memory
            ld    r1,#NIBBLE1       ;
            ld    r2,#NIBBLE2       ;
            ld    r3,#NIBBLE3       ;
            ld    r4,#NIBBLE4       ;
            ld    r5,#NIBBLE5       ;
            ld    r6,#NIBBLE6       ;
            ld    r7,#NIBBLE7       ;
            ld    r8,#NIBBLE8       ;
            ld    r9,#NIBBLE9       ;
            ld    r10,#NIBBLE10     ;
            ld    r11,#NIBBLE11     ;
            ld    r12,#NIBBLE12     ;
            ld    r13,#NIBBLE13     ;
            ld    r14,#NIBBLE14     ;
            ld    r15,#NIBBLE15     ;

            or    P2COPY,P2         ; Retrieve and process board addr
            ld    MY_ADDR,P2COPY    ;
            com   MY_ADDR           ;
            inc   MY_ADDR           ;
            and   MY_ADDR,#MASK_ID  ;
            ld    FROMFROM,MY_ADDR  ;
            swap  FROMFROM          ;
            or    FROMFROM,MY_ADDR  ;

            ld    SPL,#80h          ; Initialize stack
            ld    IPR,#%20          ; interrupt priority: 3>5>2>0>4>1
            ld    IMR,#PCRF_IMR     ; enable IRQ0 and IRQ2
            clr   IRQ               ; clear interrupts

;*****************************************************************************
;     Main Program
;*****************************************************************************

MAIN:
            or    P2COPY,#70h       ; Disable all diodes
            ld    P2,P2COPY         ;
            ei                      ; enable interrupts
            nop                     ; clear pipeline
            halt                    ; wait for next interrupt
            jr    MAIN              ;

;*****************************************************************************
;     Timeout Interrupt Service
;     INPUT: RCV_STATUS
;     DESCRIPTION:  Uses RCV_STATUS register to determine cause of
;     a data communications timeout of the PC or RF ports and resets
;     the communications accordingly.
```

```
;*********************************************************************
TIMER0:
            clr   tmr                ;
            tm    RCV_STATUS,#BIT7   ; test for Broadcast Echo
            jr    nz,BROADCAST             ;
            cp    RCV_STATUS,#BIT0   ; test for PC Data fail
            jr    z,PC_DATA_FAIL     ;
            tm    RCV_STATUS,#BIT5   ; test for Echo fail
            jr    nz,RF_ECHO_FAIL    ;
            tm    RCV_STATUS,#BIT1   ; test for RF Data Fail
            jr    nz,RF_DATA_FAIL    ;
            call  DELAY5MS           ; delay 5ms before returning
            ld    IMR,#PCRF_IMR      ; enable IRQ0 and IRQ2
            clr   RCV_STATUS         ;
            clr   IRQ                ; clear interrupts
            iret                     ;
RF_DATA_FAIL:
            call  DELAY5MS           ; 5 ms delay
            jp    RF_EXIT            ; exit RF_RCV module
RF_ECHO_FAIL:
            or    RCV_STATUS,#BIT4   ; RF_Echo Timeout
            clr   IRQ                ; clear interrupts
            jp    RF_RETRY           ; retry
PC_DATA_FAIL:
            jp    PC_EXIT            ; exit PC_RCV module
BROADCAST:
            call  PC_ECHO            ;
            jp    PC_WAIT10          ; wait 10ms for next byte
TIMEOUT10:
            ld    PRE0,#0F3h         ; 10ms Timeout
            ld    T0,#0FAh           ;
            ld    TMR,#03h           ;
            ret                      ;
TIMEOUT1:
            ld    PRE0,#3Ch          ; 1ms Timeout
            ld    T0,#14h            ;
            ld    TMR,#03h           ;
            ret                      ;

;*********************************************************************
;     PC_RCV         (Interrupt Service)
;     INPUT: Serial Data on P31
;     OUTPUT: SERDATA
;     DESCRIPTION: Enter on falling edge of P31 (IRQ2)
;     After a half bittime P31 is sampled again to validate
;     the Start bit. Then IRQ5 is enabled and T1 is setup
;     for bittime delay in continous mode.
;*********************************************************************

bitcnt      .EQU   r0          ; # of bits/word
rbuf        .EQU   r1          ; buffer
hbcnt       .EQU   r2          ; halfbit count
loopcnt     .EQU   r12         ;
rtxcnt      .EQU   r13         ; RF Retry count

PC_RCV:
            srp   #SCRATCHPAD        ;
            push  IMR                ; save present interrupt mask
```

```
            ld    PRE1,#07h          ; Modulo-N, PSC=1, TCLK=SCLK/4=1us
            ld    T1,#PC_BAUD        ; Baudrate = 9600
            and   P2COPY,#D3_ON      ; turn on PCRCV diode
            ld    P2,P2COPY          ;
            ld    hbcnt,#04          ; delay to half bit point
HALF_BIT:
            djnz  hbcnt,HALF_BIT     ; total delay ~ 25uS
            tm    P3,#02h            ; take Sample on P31: RX=0?
            jp    nz,PC_FAIL         ; if zero, Start bit is valid
START_OK:
            or    TMR,#0ch           ; load and enable T1
            ld    IMR,#TMR1_IMR      ; enable IRQ5
            clr   IRQ                ; clear interrupts
            ei                       ; enable interrupts
            ld    bitcnt,#08h        ; number of data bits/word
RECEIVE_LP:
            nop                      ; clear pipeline
            halt                     ; wait for next bit
            tm    P3,#BIT1           ; RX=0?
            jr    z,ZERO_IN          ; if zero, then reset carry bit
            scf                      ; if one, then set carry bit
            jr    COM_IN             ; "1" into buffer
ZERO_IN:
            rcf                      ; reset carry; "0" into buffer
COM_IN:
            rrc   rbuf               ; carry into MSB, LSB into Carry
            djnz  bitcnt,RECEIVE_LP  ; max loop delay = 108 SCLK (27us)
GET_STOP:
            nop                      ; clear pipeline
            halt                     ; wait for Stop Bit
            clr   TMR                ;
            pop   IMR                ;
            tm    P3,#BIT1           ; test Stop Bit

            jp    z,PC_FAIL          ; use BYTECNT to determine
            cp    BYTECNT,#0h        ; which word was received
            jr    z,PCTOFROM         ;
            cp    BYTECNT,#1h        ;
            jr    z,PCPACKET         ;
            cp    BYTECNT,#2h        ;
            jr    z,PCSIZE           ;
            jr    PCDATA             ;

PCTOFROM:
            ld    DATA_PTR,#24h      ; if TO/FROM indicates 'special msg'
            ld    TOFROM,rbuf        ; no echo is returned and code
            cp    rbuf,#0h           ; waits for next byte.
            jr    z,SPCLMSG          ; else go to LOCAL
            cp    rbuf,FROMFROM      ;
            jr    z,SPCLMSG          ;
            inc   BYTECNT            ;
            call  PC_ECHO            ; send echo to PC
            jp    PC_WAIT            ; wait for next byte
SPCLMSG:
            inc   BYTECNT            ; Special Message
            ld    RCV_STATUS,#BIT7   ; RCV_STATUS = Broadcast
            jp    PC_WAIT1           ; wait 1ms for next byte
PCPACKET:
```

```
            ld    PACKET,rbuf       ; Save PCPACKET
            inc   BYTECNT           ;
            jp    PC_WAIT10         ; wait 10ms for next byte
PCSIZE:
            ld    DATA_PTR,#24h     ; Save PCSIZE
            ld    SIZE,rbuf         ;
            ld    COPYSIZE,rbuf     ;
            inc   BYTECNT           ;
            jp    PC_WAIT10         ; wait 10ms for next byte
PCDATA:
            ld    @DATA_PTR,rbuf    ; store message DATA
            inc   DATA_PTR          ; increment RAM address
            dec   COPYSIZE          ;
            jr    z,PCPROC          ;
            jp    PC_WAIT10         ; wait 10ms for next byte
PCPROC:
            clr   TMR               ; Process Message Data
            cp    DATA_PTR,#44h     ; check if size > 32 bytes
            jr    gt,MSG_LONG       ; message is too long
            cp    PACKET,#0h        ; check if PACKET = 0
            jr    z,XMT_SPECIAL     ; message is 'special'
            call  FCS               ; calculate FCS
            clr   BYTECNT           ;
            clr   TMR               ;
            call  DELAY5MS          ;
            ld    rtxcnt,#8h        ; ld transmit counter = 8
            cp    TOFROM,#0h        ; Check TOFROM = 0 for 'broadcast'
            jr    z,RF_XMT_BCST     ; message
RF_XMT_CALL:
            call  RF_XMT            ; transmit RF Data
            call  DELAY3MS          ; delay 6ms before 10ms timeout
            call  DELAY3MS          ;
RFECHO_WAIT:
            call  TIMEOUT10         ; 10ms timeout for RF_ECHO
            ld    IMR,#RFTMR0_IMR   ; Enable IRQ0 and IRQ2
            clr   IRQ               ;
            iret                    ;
RF_XMT_BCST:
            call  RF_XMT            ; RF Broadcast 8X
            call  DELAY100          ; 100ms delay between broadcasts
            djnz  rtxcnt,RF_XMT_BCST ;
            call  PC_NAK            ; 'NAK' sent to PC after xmt
            jp    PC_EXIT           ; is completed.
TIMER1:
            clr   IRQ               ; clear interrupts
            iret                    ;
XMT_SPECIAL:                        ; Transmit response for 'Special'
            or    P0COPY,#ON_232    ; Enable 232 Drive
            ld    P0,P0COPY         ;
            ld    SERDATA,TOFROM    ; Echo
            call  PC_XMT            ; send echo to PC
            cp    DATABUF,#30h      ; Software Reset
            jr    z,RESET           ;
            cp    DATABUF,#31h      ; Unit Address
            jr    z,SEND_ADDR       ; send unit node address
            cp    DATABUF,#32h      ; Battery Status
            jr    z,SEND_BATT       ; send battery status
            cp    DATABUF,#33h      ; Self Test
```

```
            jr    z,SELF_TEST       ; self Self Test pass/fail
            cp    DATABUF,#34h      ; Set Address
            jr    z,SET_ADDR        ; s/w setting of node address
MSG_LONG:
            ld    SPCL_DATA,#30h    ; Message >32 Bytes
            jr    XMT_SPCL          ;
RESET:
            jp    START             ; Software Reset
SEND_ADDR:
            ld    SPCL_DATA,#35h    ; Send Address
            jr    XMT_SPCL          ;
SELF_TEST:
            call  RAMTEST           ; Self Test
            jp    START             ;
SEND_BATT:
            ld    SPCL_DATA,#32h    ; Assume battery ok
            jr    XMT_SPCL          ;
SET_ADDR:
            dec   DATA_PTR          ; S/W setting of Address
            ld    MY_ADDR,@DATA_PTR ;
            ld    FROMFROM,MY_ADDR  ;
            swap  FROMFROM          ;
            or    FROMFROM,MY_ADDR  ;
            ld    SPCL_DATA,#35h    ;
            jr    PC_EXIT           ;
XMT_SPCL:
            ld    SERDATA,FROMFROM  ; send FROMFROM
            call  PC_XMT            ;
XMT_SPCL1:
            ld    SERDATA,#0h       ; send '00h' to PC
            call  PC_XMT            ;
            cp    DATABUF,#34       ;
            jr    nz,XMT_SPCL2      ;
            ld    SERDATA,#0h       ; send '00h' to PC
            call  PC_XMT            ;
XMT_SPCL2:
            ld    SERDATA,#01h      ; send '01h' to PC
            call  PC_XMT            ;
            ld    SERDATA,SPCL_DATA ; send status byte to PC
            call  PC_XMT            ;
            jr    PC_EXIT           ; exit PC_RCV module
PC_WAIT:
            ld    RCV_STATUS,#BIT0  ; RCV_STATUS = PC_RCV
            ld    IMR,#PCTMR0_IMR   ; Enable IRQ2 and IRQ4
            clr   IRQ               ; clear interrupts
            iret                    ; return to MAIN loop
PC_WAIT10:
            ld    RCV_STATUS,#BIT0  ; RCV_STATUS = PC_RCV
            call  TIMEOUT10         ; 10ms timeout
            ld    IMR,#PCTMR0_IMR   ; Enable IRQ2 and IRQ4
            clr   IRQ               ; clear interrupts
            iret                    ; return to MAIN loop
PC_WAIT1:
            call  TIMEOUT1          ; 1ms timeout
            ld    IMR,#PCTMR0_IMR   ; enable IRQ2 and IRQ4
            clr   IRQ               ; clear interrupts
            iret                    ; return to MAIN loop
PC_FAIL;
```

```
            clr   TMR               ; exit PC_RCV
            pop   IMR               ; reload initial IMR and TMR
PC_EXIT:
            and   P0COPY,#OFF_232   ; Turn off 232 drive
            ld    P0,P0COPY         ;
            clr   RCV_STATUS        ;
            clr   BYTECNT           ;
            ld    IMR,#PCRF_IMR     ; enable IRQ0 and IRQ2
            clr   IRQ               ; clear interrupts
            iret                    ; return to MAIN loop
PC_ECHO:
            ld    SERDATA,TOFROM    ; Echo TOFROM back to PC
            or    P0COPY,#ON_232    ; Enable 232 drive
            ld    P0,P0COPY         ;
            call  PC_XMT            ; Send Echo
            and   P0COPY,#OFF_232   ; Disable 232 drive
            ld    P0,P0COPY         ;
            ret                     ;

;*************************************************************************
;     Transmit RS232 Data     (CALL)
;     Input:  SERDATA
;     Output: Serial Data on P01
;     Format: | start | data | stop |
;           start = active Low
;           data = 8 bits
;           stop = active High
;*************************************************************************
xbitcnt     .EQU   r3           ;
data1       .EQU   r4           ;
data2       .EQU   r5           ;
dummy       .EQU   r6           ;

PC_XMT:
            push  IMR               ;
            push  RP                ;
            srp   #SCRATCHPAD       ;
TX_LP:
            or    P0COPY,#BIT1      ; insure output is High
            ld    P0,P0COPY         ;
            ld    data1,SERDATA     ; load SERDATA into temp register
            ld    data2,#1          ; shift in stop bit

            rcf                     ; reset Carry
            rlc   data1             ; start bit into LSB
            rlc   data2             ; now all bits are in

            ld    xbitcnt,#10       ; number of bits (start,data,stop)
            ld    PRE1,#PC_PRE      ;
            ld    T1,#PC_BAUD       ; PC Baudrate = 9600 ~ 104us
            ld    TMR,#0ch          ; load and enable timer
            ld    IMR,#TMR1_IMR     ; enable IRQ5 bittime delay on T1
            clr   IRQ               ; clear interrupts
            ei                      ; enable interrupts
            nop                     ; clear pipeline
            halt                    ; wait 104us to send next bit
SEND_LP:
            rrc     data2           ; LSB into Carry
```

```
            rrc     data1           ; Carry into MSB, LSB into Carry
            jr      nc,ZERO_OUT     ; data bit zero?
            or      P0COPY,#BIT1    ; TX=1, '1' on output port
            jr      COM_OUT         ;
ZERO_OUT:
            and     P0COPY,#MASK1   ; TX=0, '0' on output port
            ld      dummy,#0        ; balance loop, 10 SCLK
COM_OUT:
            ld      P0,P0COPY       ; Output at port, 50 SCLK delay (TX=1)
            nop
            halt                    ; wait 104us to send next bit
            djnz  xbitcnt,SEND_LP   ;
            clr   TMR               ;
            pop   RP                ;
            pop   IMR               ;
            ret                     ;

;*********************************************************************
;     RF_XMT:
;     Performs transmission of RF Data Stream
;     Input:  STARTBYTE,TOFROM,PACKET,SIZE,DATABUF,FCSHI,FCSLO
;           (RAM Address 20h...)
;     Output: Serial Data on P01
;     Protocol:|0x55|TOFROM|PACKET|SIZE|0x02|DATA...|0x03|FCSHI|FCSLO|
;
;*********************************************************************
rf_ptr            .EQU  r0          ;
rfsize            .EQU  r1          ;
rfbits            .EQU  r2          ;
rfpass            .EQU  r3          ;
rfretry           .EQU  r4          ;
table_ls          .EQU  r6          ;
table_ms          .EQU  r7          ;
dcb_ls            .EQU  r8          ;
dcb_ms            .EQU  r9          ;
loop_cnt          .EQU  r10         ;
temp1             .EQU  r11         ;

RF_XMT:
            srp   #SCRATCHPAD       ;
            and   P2COPY,#PTT_ON    ; Clear PTT to enable RF Xmit
            ld    P2,P2COPY         ;
            call  RF_TRAIN          ; send preamble (DC Bal Training)
            ld    rf_ptr,#20h       ; pointer to start of data buffer
            ld    rfsize,SIZE       ; # of bytes to send
            add   rfsize,#4         ; Add Headers, FCS,
            ld    STARTBYTE,#55h    ; STARTBYTE = '55h'
XMT_LOOP:
            ld    SERDATA,@rf_ptr   ; XMT RF Data Loop
            call  RF_SEND           ; send byte
            inc   rf_ptr            ; increment data pointer
            djnz  rfsize,XMT_LOOP   ;
            ld    SERDATA,FCSLO     ; send FCSLO
            call  RF_SEND           ;
            ld    SERDATA,FCSHI     ; send FCSHI
            call  RF_SEND           ;
            or    P2COPY,#PTT_OFF   ; Disable RF Transmit
            ld    P2,P2COPY         ;
```

```
            ld    loop_cnt,#0FFh    ;
            or    RCV_STATUS,#BIT5  ; RCV_STATUS = RF_ECHO_WAIT
            call  DELAY3MS          ; delay 3ms
            ret                     ;
RF_SEND:
            ld    rfbits,#6         ; # of bits/pass
            ld    rfpass,#2         ; # of passes (2 6-bit xmits)
            push  IMR               ;
            ld    IMR,#TMR1_IMR     ; enable IRQ5 bittime delay on T1
            ld    PRE1,#07h         ; continous mode T1, TCLK=SLCK/4=1us
            ld    T1,#RF_BAUD       ; set T1 for baud rate 9600
            or    TMR,#0ch          ; load and enable timer
            clr   IRQ               ; clear interrupts
            ei                      ; enable interrupts
            nop                     ; clear pipeline
            halt                    ; wait 104us to send next bit
            or    P0COPY,#1         ; send start bit
            ld    P0,P0COPY         ;
DC_BAL:
            ld    table_ls,SERDATA  ; Retrieve DC Balanced Value
            ld    table_ms,table_ls ; for LS and MS Nibble of
            and   table_ls,#MASK_ID ; Data
            ld    dcb_ls,TABLE(table_ls); r8 = LS (6 bits)
            swap  table_ms          ;
            and   table_ms,#MASK_ID ;
            ld    dcb_ms,TABLE(table_ms); r9 = MS (6 bits)
RFTX_LOOP:
            rrc   dcb_ls            ; Inverted logic.
            jr    nc,ONE_OUT        ;
            and   P0COPY,#MASK0     ; data=1 sends a 0 on RTX.
            jr    RF_OUT            ;
ONE_OUT:
            or    P0COPY,#BIT0      ;
RF_OUT:
            clr   IRQ               ; Transmit loop (12 bits)
            nop                     ; clear pipeline
            halt                    ; wait 104us to send next bit
            ld    P0,P0COPY         ;
            djnz  rfbits,RFTX_LOOP  ; 6 bits/pass
            ld    dcb_ls,dcb_ms     ;
            ld    rfbits,#6         ; 2 passes
            djnz  rfpass,RFTX_LOOP  ;
            nop                     ; clear pipeline
            halt                    ; wait 104us to send next bit

            and   P0COPY,#0FEh      ; send stop bit
            ld    P0,P0COPY         ;
            pop   IMR               ;
            clr   IRQ               ; clear interrupts
            ret                     ;
RF_TRAIN:
            push  IMR               ; Output 24 bit alternating
            ld    rfpass,#12        ; 1,0,1,0 pulses to 'train'
            ld    IMR,#TMR1_IMR     ; the DC Balance of the RF
            ld    PRE1,#07h         ; Transmitter (preamble)
            ld    T1,#RF_BAUD       ; RF_BAUD = 9600
            or    TMR,#0ch          ; load and enable timer
            clr   IRQ               ; clear interrupts
```

```
            ei                      ; enable interrupts
TRAIN_LOOP:
            nop                     ; clear pipeline
            halt                    ; wait 104us to send next bit
            or    P0COPY,#1         ; send '1'
            ld    P0,P0COPY         ;
            clr   IRQ               ; clear interrupts
            ei                      ; enable interrupts
            nop                     ; clear pipeline
            halt                    ; wait 104us to send next bit
            and   P0COPY,#0FEh      ; send a '0'
            ld    P0,P0COPY         ;
            djnz  rfpass,TRAIN_LOOP ; loop 12 times
            pop   IMR               ;
            ret                     ;

;*****************************************************************************
;     RF_RCV:
;     Entry:  Enter module from IRQ0, falling edge of P32
;     INPUT:  Serial Data on P32
;     Definition: Module reacts to incoming data on P31 RF serial input.  Input
;     should be 1 start bit, 12 data bits, 1 stop bit.  Each bit is sampled
;     11X for verification. After 6 data bits have been received, the data
;     is run through the 6to4 module to convert from 12 bit data format to
;     8 bit data format.  After 12 bits are received, the data is combined
;     into a 8-bit word and stored in RAM (0x20 to 0x45)
;     Memory map for RAM is as follows:
;           Start Finish  Reg Description
;           0x20  0x20  r0    RF Start Byte (0x55)
;           0x21  0x21  r1    RF TO/FROM
;           0x22  0x22  r2    RFPACKET#
;           0x23  0x23  r3    RF Size
;           0x24  0x24  r4    Start byte (0x02)
;           0x24  0x43        RF Data
;           0x44  0x44  r4    Stop byte (0x03)
;           0x45  0x45  r5    RFFCSHI
;           0x46  0x46  r6    RFFCSLO
;       After each byte is received, the module will exit to the MAIN loop
;     to await the next interrupt.  If the wait loop times out (1ms), the
;     module will be exited and an error flag risen.  After the FCS data is
;     stored, the FCS module is called to calculate and verify the FCS
;     for the data stream.  If the FCS is verified, then the RFECHO module
;     is called and an ECHO/ACKNOWLEDGE is transmitted back on the RF link
;     to acknowledge receipt of the data.
;     BROADCAST:
;     In the special case of BROADCAST:  When a TOFROM=00h is detected, the
;     ECHO is skipped and the software enters a 100ms delay loop to wait out
;     the 8x broadcast of the message.
;     RF NOISE FILTERING:
;     1) 11X oversampling - Each bit is sampled 11X to determine the value
;     2) RF Data Validation - Each time the potential start of a message
;        is received (Training Pulses), the 1st, 2nd, and 3rd bits of the
;        training pulses are verified.  The purpose of this is to filter
;        RF noise and ensure that a valid message is being received
;        before delaying past the remaining pulses and entering the main
;        loop.
;*****************************************************************************
*
```

```
rx_ptr          .EQU  r0          ;
rfdata          .EQU  r1          ;
rffcshi         .EQU  r2          ;
rffcslo         .EQU  r3          ;
cmp_val         .EQU  r4          ;
bits            .EQU  r5          ;
nibcnt          .EQU  r6          ;
p32mask         .EQU  r7          ;
lonib           .EQU  r8          ;
hinib           .EQU  r9          ;
rf_size         .EQU  r10         ;
rftofrom        .EQU  r11         ;
pdata           .EQU  r12         ;
result          .EQU  r13         ;

RF_RCV:
            srp   #SCRATCHPAD     ;
            push  IMR             ;
            and   P2COPY,#D5_ON   ; Enable RXI Diode
            ld    P2,P2COPY       ;
RF_TRAIN_TEST:
            ld    PRE1,#07h       ; Verify first 3 pulses of
            ld    T1,#RF_BAUD     ; pulse train (preamble). Return
            ld    IMR,#TMR1_IMR   ; to main if any one is invalid
            or    TMR,#0ch        ; enable timer
            tm    RCV_STATUS,#BIT1 ; if RCV_STATUS = RF_DATA_WAIT
            jr    nz,RF_RCV_DATA  ; skip preamble check
RF_RCV_TRAIN:
            ld    nibcnt,#11      ;
NLOOP:
            nop                   ; delay to ~ middle of pulse
            djnz  nibcnt,NLOOP    ;
            ld    p32mask,#04h    ; P32 Test Mask
            call  RF_SAMPLE       ; test Sample#1=0
            cp    result,#16h     ; sample time ~ 40us (50-90us)
            jr    ugt,RF_INVALID  ; invalid bit if result > 16h
            clr   IRQ             ; clear interrupts
            ei                    ; enable interrupts
            nop                   ; clear pipeline
            halt                  ; wait 104us for next bit
            call  RF_SAMPLE       ; test for valid Sample#2
            cp    result,#16h     ; sample time ~ 40us (50-90us)
            jr    ult,RF_INVALID  ; invalid bit if result < 16h
            clr   IRQ             ; clear interrupts
            ei                    ; enable interrupts
            nop                   ; clear pipeline
            halt                  ; wait 104us for next bit
            call  RF_SAMPLE       ; Test for valid Sample#3
            cp    result,#16h     ; sample time ~40us (50-90us)
            jr    ult,RF_VALID    ; invalid bit if result > 16h
RF_INVALID:
            pop   IMR             ; if preamble test fails,
            clr   IRQ             ; restore IMR and return to
            iret                  ; MAIN loop
RF_VALID:
            ld    PRE1,#57h       ; if preamble test passes
            ld    T1,#RF_BAUD     ; delay 21 x 104us (21 pulses)
```

```
            ld    IMR,#TMR1_IMR     ; to start of message
            ld    TMR,#0ch          ;
            clr   IRQ               ; clear interrupts
            ei                      ; enable interrupts
            nop                     ; clear pipeline
            halt                    ; wait 104us for next bit
            or    RCV_STATUS,#BIT1  ; Set Status = RF_DATA_WAIT
            pop   IMR               ; restore IMR
            clr   IRQ               ; clear interrupts
            iret                    ; return to MAIN loop
RF_RCV_DATA:
            pop   IMR               ;
            ld    PRE1,#07h         ; set timer0 = 104us
            ld    T1,#RF_BAUD       ;
            ld    IMR,#TMR1_IMR     ;
            ld    TMR,#0ch          ; load and enable T1
            ld    nibcnt,#2         ; 2 6-bit nibbles/word
            clr   IRQ               ; clear interrupts
            ei                      ; enable interrupts
SB_SAMPLE:
            call  RF_SAMPLE         ; 11x sampling of input
            cp    result,#16h       ; Compare to 22 (14h)
            jp    gt,RF_FAIL        ; Error if not zero
RFNIB_LP:
            ld    bits,#6           ; number of data bits
            clr   r12               ;
RFRCV_OLP:
            nop                     ; clear pipeline
            halt                    ; wait 104us for next bit
RFRCV_SAMPLE:
            call  RF_SAMPLE         ; 11x sampling of input
            cp    result,#16h       ; compare to 22 (16h)
            jr    gt,LOGIC1         ; if > 16, data = 1
LOGIC0:
            rcf                     ; set carry flag
            rrc   rfdata            ; rotate '1' into byte
            djnz  bits,RFRCV_OLP    ;
            jr    STORE             ;
LOGIC1:
            scf                     ; clear carry flag
            rrc   rfdata            ; rotate '0' into byte
            djnz  bits,RFRCV_OLP    ;
STORE:
            rr    rfdata            ; format data and convert
            rr    rfdata            ; dc balanced (12 bit) data
            and   rfdata,#3Fh       ;
CONV6TO4:
            cp    rfdata,#7h        ; verify 7 <= R13 <= 38
            jp    lt,RF_FAIL        ; DC Balanced data is invalid
            cp    rfdata,#38h       ; if not withing this range
            jp    gt,RF_FAIL        ;
SIX_OK:
            cp    rfdata,#15h       ; if 15, then return 0
            jr    ne,CHECK1         ; '15' corresponds to value '0'
            clr   rfdata            ; in the DC Balance Table
            jr    SIXOUT            ;
CHECK1:
            cp    rfdata,#1Ch       ; elseif 1C, then return 15
```

```
            jr    ne,CHECK2         ; '1C' corresponds to value '15'
            ld    rfdata,#0fh       ; in the DC Balance TAble
            jr    SIXOUT            ;
CHECK2:
            and   rfdata,#0fh       ; elseif return Low nibble
SIXOUT:
            djnz  nibcnt,LONIBBLE   ;
HINIBBLE:
            ld    hinib,rfdata      ; store results for hinibble
            jr    COMBINE           ;
LONIBBLE:
            ld    lonib,rfdata      ; store results for lonibble
            jr    RFNIB_LP          ;
COMBINE:
            swap  hinib             ; swamp nibbles to format data
            or    hinib,lonib       ;
            ld    RFDATA,hinib      ; RFDATA <= RESULT
RFSTOP:
            nop                     ; clear pipeline
            halt                    ; wait 104us
            clr   TMR               ;
            tm    RCV_STATUS,#BIT5  ; if RCV_STATUS = RF_ECHO_WAIT
            jr    z,RF_PROCESS      ; then received data is RF ECHO data

;*****************************************************************************
;     RFECHO_PROCESS:
;     Entry:  Enter module from RF_RCV
;     Definition: Module takes data from RF_RCV and determines where
;     in SRAM each byte should be stored.
;*****************************************************************************
RFECHO_PROCESS:
            cp    BYTECNT,#0        ; use BYTECNT to determine which
            jr    z,RFETOFROM       ; databyte was received
            cp    BYTECNT,#1        ;
            jr    z,RFEPACKET       ;
            cp    BYTECNT,#2        ;
            jr    z,RFESIZE         ;
            jr    RF_RETRY          ;
RFETOFROM:
            cp    RFDATA,TOFROM     ; verify TOFROM is valid
            jr    nz,RF_RETRY       ; if not, RETRY
            inc   BYTECNT           ;
            jp    RFECHO_WAIT       ; wait for next byte
RFEPACKET:
            cp    RFDATA,PACKET     ; verify PACKET #
            jr    nz,RF_RETRY       ; if not, RETRY
            inc   BYTECNT           ;
            jp    RFECHO_WAIT       ; wait for next byte
RFESIZE:
            cp    RFDATA,SIZE       ; verify SIZE is valid
            jr    nz,RF_RETRY       ; if not, RETRY
            inc   RETRY             ;
            call  PC_ACK            ; if valid, send ACK to PC
            jp    RF_XMT_EXIT       ;
RF_RETRY:
            call  DELAY100          ; 100ms delay between retries
            inc   RETRY             ;
            ld    RCV_STATUS,#BIT5  ; RCV_STATUS = RF_ECHO_WAIT
```

```
            cp    RETRY,#8h          ;
            jp    nz,RF_XMT_CALL     ; send RF Message
RF_XMT_FAIL:
            call  PC_NAK             ;
RF_XMT_EXIT:
            clr   RETRY              ; exit the module
            clr   BYTECNT            ; clear globals
            clr   RCV_STATUS         ;
            ld    IMR,#PCRF_IMR      ; enable IRQ0 and IRQ2
            clr   IRQ                ; clear interrupts
            iret                     ; return to MAIN loop
;*****************************************************************************
;     RF_PROCESS: (JUMP)
;     Entry:  Enter module from RF_RCV
;     Definition: Module takes data from RF_RCV and determines where
;     in SRAM each byte should be stored.
;*****************************************************************************
RF_PROCESS:
            cp    BYTECNT,#0h        ; use BYTECNT to determine which
            jr    z,RFSTARTBYTE      ; databyte was received
            cp    BYTECNT,#1h        ;
            jr    z,RFTOFROM         ;
            cp    BYTECNT,#2h        ;
            jr    z,RFPACKET         ;
            cp    BYTECNT,#3         ;
            jr    z,RFSIZE           ;
            jr    RFDATABUF          ;
RFSTARTBYTE:
            cp    RFDATA,#55h        ; verify the STARTBYTE = 55h
            jr    nz,RF_FAIL         ;
            ld    STARTBYTE,RFDATA   ;
            inc   BYTECNT            ;
            jr    RF_WAIT            ; wait for next byte
RFTOFROM:
            ld    rx_ptr,#24h        ; store TOFROM
            ld    TOFROM,RFDATA      ;
            inc   BYTECNT            ;
            jr    RF_WAIT            ; wait for next byte
RFPACKET:
            ld    PACKET,RFDATA      ; store PACKET #
            inc   BYTECNT            ;
            jr    RF_WAIT            ; wait for next byte
RFSIZE:
            ld    SIZE,RFDATA        ; store SIZE (# of data bytes)
            ld    rf_size,SIZE       ;
            add   rf_size,#2         ; add 2 to account for FCS bytes
            inc   BYTECNT            ; in determining message size
            jr    RF_WAIT            ; wait for next byte
RFDATABUF:
            ld    @rx_ptr,RFDATA     ; store message data bytes
            inc   rx_ptr             ; rf_size = # of bytes
            djnz  rf_size,RF_WAIT    ;

            dec   rx_ptr             ; Load FCS into global variables
            ld    EFCSHI,@rx_ptr     ; for compare with calculated
            dec   rx_ptr             ; values
            ld    EFCSLO,@rx_ptr     ;
            jr    RF_VERIFY          ; verification of RF Message
```

```
RF_WAIT:
            ld    RCV_STATUS,#BIT1  ; RCV_STATUS = RF_DATA_WAIT
            call  TIMEOUT1          ; (causes s/w to skip preamble test)
            ld    IMR,#RFTMR0_IMR   ; enable IRQ0 and IRQ4
            clr   IRQ               ; clear interrupts
            iret                    ; return to MAIN loop
RF_VERIFY:
            cp    TOFROM,#0h        ; if TOFROM = 0, message is a
            jr    z,PACK_VERIFY     ; 'BROADCAST' message
            ld    rftofrom,TOFROM   ; else verify TO Address
            swap  rftofrom          ;
            and   rftofrom,#0fh     ;
            cp    rftofrom,MY_ADDR  ; exit if address not equal to
            jr    nz,RF_EXIT        ; MY_ADDR
PACK_VERIFY:
            cp    PACKET,#8h        ; verify Valid Packet #
            jr    gt,RF_EXIT        ; if 1 <= PACKET <=8
            cp    PACKET,#1h        ; else PACKET # is invalid
            jr    lt,RF_EXIT        ; and module is exited
            call  FCS               ; verify FCS (checksum)
            cp    FCSHI,EFCSHI      ; compare calclated (FCSHI/LO)
            jr    nz,RF_EXIT        ; to the received (EFCSHI/LO)
            cp    FCSLO,EFCSLO      ; exit if FCS fails
            jr    nz,RF_EXIT        ;
            call  RF_ACK            ; Send 'ACK' to PC if FCS is okay
            call  PC_XMT_DATA       ; transmit Message to PC
            cp    TOFROM,#0h        ; Check if message is a 'BROADCAST'
            jr    z,DELAY1200       ; if 'BROADCAST', Delay 1.2ms
            jr    RF_EXIT           ; then exit module
RF_FAIL:
            tm    RCV_STATUS,#BIT5  ; if RCV_STATUS = RF_ECHO_WAIT
            jr    z,RF_EXIT         ; exit module
            clr   BYTECNT           ;
            jp    RF_RETRY          ; else RETRY
RF_EXIT:
            clr   BYTECNT           ; clear global variables
            clr   RCV_STATUS        ;
            ld    IMR,#PCRF_IMR     ; enable IRQ0 and IRQ2
            clr   IRQ               ; clear interrupts
            iret                    ; return to MAIN loop
DELAY1200:
            ld    r12,#240          ; 1200ms delay loop
D1200_LOOP:
            call  DELAY5MS          ; perform 240 5ms delay loops
            djnz  r12,D1200_LOOP    ;
            jr    RF_EXIT           ;

;*****************************************************************************
;     RF_SAMPLE:  (CALL)
;     Entry:  Enter module from RF_RCV
;     INPUT: Serial Data on P32
;     OUTPUT: 'result' (local variable)
;     Definition: Module sample P32 input 11 times.  P32 is masked and
;     a running sum is calculated in the 'result' register.  This 'result'
;     is used to determine whether the average sample was a '1' or '0' by
;     comparing the value to 16h (22).
;*****************************************************************************
;
RF_SAMPLE:
```

```
            clr   result            ; 11X Sampling of P32
            ld    result,P3         ; Sample #1
            and   result,p32mask    ;
            ld    pdata,P3          ; Sample #2
            and   pdata,p32mask     ;
            add   result,pdata      ;
            ld    pdata,P3          ; Sample #3
            and   pdata,p32mask     ;
            add   result,pdata      ;
            ld    pdata,P3          ; Sample #4
            and   pdata,p32mask     ;
            add   result,pdata      ;
            ld    pdata,P3          ; Sample #5
            and   pdata,p32mask     ;
            add   result,pdata      ;
            ld    pdata,P3          ; Sample #6
            and   pdata,p32mask     ;
            add   result,pdata      ;
            ld    pdata,P3          ; Sample #7
            and   pdata,p32mask     ;
            add   result,pdata      ;
            ld    pdata,P3          ; Sample #8
            and   pdata,p32mask     ;
            add   result,pdata      ;
            ld    pdata,P3          ; Sample #9
            and   pdata,p32mask     ;
            add   result,pdata      ;
            ld    pdata,P3          ; Sample #10
            and   pdata,p32mask     ;
            add   result,pdata      ;
            ld    pdata,P3          ; Sample #11
            and   pdata,p32mask     ;
            add   result,pdata      ; sum results
            ret                     ;

;*************************************************************
;     PC_XMT_DATA:      (CALL)
;     INPUT: RF DATA   SRAM ADDR 0x21 to End of data
;     OUTPUT: Serial Data on P01
;     Description: Transmits Rf data received by RRX to the host
;     software (PC).
;*************************************************************
;
pc_ptr            .EQU  r0          ;
pc_size           .EQU  r1          ;

PC_XMT_DATA:
            call  DELAY5MS          ;
            and   P2COPY,#D4_ON     ; turn on RFRCV Diode
            ld    P2,P2COPY         ;
            or    P0COPY,#ON_232    ; enable 232
            ld    P0,P0COPY         ;
            ld    pc_ptr,#24h       ; set pointer to DATA Buffer
            ld    pc_size,SIZE      ;
            ld    SERDATA,TOFROM    ; transmit protocol info
            call  PC_XMT            ; send TOFROM byte
            ld    SERDATA,PACKET    ;
            call  PC_XMT            ; send PACKET byte
            ld    SERDATA,SIZE      ;
```

```
            call  PC_XMT              ; send SIZE byte
PCXMT_LOOP:
            ld    SERDATA,@pc_ptr     ; send message data
            call  PC_XMT              ;
            inc   pc_ptr              ;
            djnz  pc_size,PCXMT_LOOP;

            and   P0COPY,#OFF_232     ; Disable 232
            ld    P0,P0COPY           ;
            or    P2COPY,#D4_OFF      ; Turn off RFRCV Diode
            ld    P2,P2COPY           ;
            ret                       ;
RF_ACK:                               ; sends RF_ACK to  RF msg originator
            call  DELAY5MS            ; Delay 10ms to allow xmtr
            call  DELAY5MS            ; power to stabilize
            and   P2COPY,#PTT_ON      ; Enable RF Transmit
            ld    P2,P2COPY           ;
            call  RF_TRAIN            ; send preamble (DC Balance Train)
            ld    SERDATA,TOFROM      ; Transmits Acknowledge to RF Link
            call  RF_SEND             ; send TOFROM byte
            ld    SERDATA,PACKET      ;
            call  RF_SEND             ; send PACKET byte
            ld    SERDATA,SIZE        ;
            call  RF_SEND             ; send SIZE byte
            or    P2COPY,#PTT_OFF     ; Disable RF Transmit
            ld    P2,P2COPY           ;
            ret                       ;
PC_NAK:
            call  DELAY5MS            ; sends NAK to PC (retry failed 8X)
            or    P0COPY,#ON_232      ; Enable 232
            ld    P0,P0COPY           ;
            ld    SERDATA,TOFROM      ; format = |TOFROM|PACKET|DDh|
            call  PC_XMT              ; send TOFROM byte
            ld    SERDATA,PACKET      ;
            call  PC_XMT              ; send PACKET byte
            ld    SERDATA,#0DDh       ;
            call  PC_XMT              ; send 'DDh' byte
            and   P0COPY,#OFF_232     ; Disable 232
            ld    P0,P0COPY           ;
            ret                       ;
PC_ACK:
            call  DELAY5MS            ; sends ACK to PC
            or    P0COPY,#ON_232      ; Enable 232
            ld    P0,P0COPY           ; format = |TOFROM|PACKET|En|
            ld    SERDATA,TOFROM      ; n = RETRY
            call  PC_XMT              ; send TOFROM byte
            ld    SERDATA,PACKET      ;
            call  PC_XMT              ; send PACKET byte
            ld    SERDATA,#0E0h       ;
            or    SERDATA,RETRY       ; send 'En' byte
            call  PC_XMT              ;
            and   P0COPY,#OFF_232     ; Disable 232
            ld    P0,P0COPY           ;
            ret
DELAY5MS:
            di                        ; 5ms Delay routine
            push  IMR                 ;
            ld    PRE1,#7Bh           ;
```

```
            ld    T1,#0FAh          ;
            ld    IMR,#TMR1_IMR     ; enable interrupts
            ld    TMR,#0Ch          ;
            clr   IRQ               ; clear interrupts
            ei                      ; enable interrupts
            nop                     ; clear pipeline
            halt                    ; wait 5ms
            pop   IMR               ; restore IMR
            ret                     ;
DELAY3MS:
            di                      ; 3ms delay routine
            push  IMR               ;
            ld    PRE1,#0F3h        ;
            ld    T1,#04Bh          ;
            ld    IMR,#TMR1_IMR     ; enable IRQ5
            ld    TMR,#0Ch          ;
            clr   IRQ               ; clear interrupts
            ei                      ; enable interrupts
            nop                     ; clear pipeline
            halt                    ; wait 3ms
            pop   IMR               ; restore IMR
            ret                     ;
RAMTEST:
            push  RP                ; RAM Self Test
            srp   #0h               ;
            ld    R13,TOFROM        ;
            ld    DATA_PTR,#10h     ;
RAMWR:
            ld    @DATA_PTR,#0AAh   ; ld 'AA' into memory from
            inc   DATA_PTR          ; address '10h' to '76'
            cp    DATA_PTR,#76h     ; avoids overwriting data and
            jr    nz,RAMWR          ; stack
            ld    DATA_PTR,#10h     ;
RAMRD:
            cp    @DATA_PTR,#0AAh   ; read memory and verify 'AA'
            jr    nz,RAMFAIL        ;
            inc   DATA_PTR          ;
            cp    DATA_PTR,#76h     ;
            jr    nz,RAMRD          ;
            jr    RAMPASS           ;
RAMFAIL:
            ld    r15,#33h          ; send '33h' if RAM fails
            jr    RAMCLR            ; clear RAM
RAMPASS:
            ld    r15,#34h          ; send '34h' if RAM passes
RAMCLR:
            ld    DATA_PTR,#10h     ; clear RAM from '10h' to '76h'
RAMCLR1:
            clr   @DATA_PTR         ;
            inc   DATA_PTR          ;
            cp    DATA_PTR,#76h     ;
            jr    nz,RAMCLR1        ;
RAM_XMT:
            ld    SERDATA,#0        ; transmit test status to PC
            call  PC_XMT            ; send '00' to PC
            ld    SERDATA,#0        ;
            call  PC_XMT            ; send '00' to PC
            ld    SERDATA,#1        ;
```

```
            call  PC_XMT            ; send '01' to PC
            ld    SERDATA,r15       ;
            call  PC_XMT            ; send '33h' to '34h' to PC
            pop   RP                ; restore RP
            ret                     ;
DELAY100:
            push  RP                ; 100ms delay routine
            srp   #SCRATCHPAD       ;
            ld    r12,#20           ;
D100_LOOP:
            call  DELAY5MS          ; perform 20 5ms delays
            djnz  r12,D100_LOOP     ;
            pop   RP                ;
            ret                     ;

;*********************************************************************
;     FCS:  Performs FCS Checksum calculation
;
;           Input:  DATABUF (all data contained in SRAM)
;           Output: FCSHI,FCSLO
;
;           Output placed at end of the data buffer
;           This section uses register group 50h as a scratchpad
;*********************************************************************

fcs_ptr     .EQU  r0    ;
fsize       .EQU  r2    ;
fbitcnt     .EQU  r3    ;
fdata1      .EQU  r4    ;
fdata2      .EQU  r5    ;
fcsls       .EQU  r6    ;
fcsms       .EQU  r7    ;
ftemp1      .EQU  r8    ;
ftemp2      .EQU  r9    ;

FCS:
            srp   #SCRATCHPAD       ; set register pointer to scratchpad
            ld    fsize,SIZE        ; calculate # bytes to process
            add   fsize,#03h        ; by adding (size +3)
            ld    fcs_ptr,#21h      ; FCS starts with TO/FROM byte

            ld    fcsls,#0ffh       ; preset FCS
            ld    fcsms,#0ffh       ;
FCS21:
            ld    fbitcnt,#08h      ; Process 8 bits/byte
            ld    fdata1,@fcs_ptr   ;
            inc   fcs_ptr           ; increment for next pass
FCS22:
            ld    ftemp1,fdata1     ;
            and   ftemp1,#1         ; get LSB
            rr    fdata1            ; rotate for next pass

            ld    fdata2,fcsls      ;
            and   fdata2,#1         ; LS byte of FCS

;*********************************************************************
;     16 bit shift of FCS
;*********************************************************************
```

```
            rcf                     ;
            rrc   fcsms             ; from buffer
            rrc   fcsls             ; from FCS
;*****************************************************************************
;     Compare Bits saved from buffer and from LS bit of LS byte of FCS
;*****************************************************************************
            ld    ftemp2,ftemp1     ; Use r8 as temp
            xor   ftemp2,fdata2     ;
            jp    z,FCS23           ; jump if r8=0
            xor   fcsms,#84h        ; xor with poly
            xor   fcsls,#08h        ;
FCS23:
            djnz  fbitcnt,FCS22     ; go for another bit
            djnz  fsize,FCS21       ; go for another byte
            ld    FCSHI,fcsls       ; store FCS data in data buffer
            inc   DATA_PTR          ;
            ld    FCSLO,fcsms       ;
            ret                     ; return for FCS in r7,r6
;*****************************************************************************
;     Add FCS Buffer to XMIT
;*****************************************************************************
ADD_FCS:
            ld    fsize,#8h         ;
RLOOP1:
            rrc   fcsms             ;
            rlc   ftemp1            ;
            djnz  fsize,RLOOP1      ;
            com   ftemp1            ;
            ld    @fcs_ptr,ftemp1   ;
            ld    FCSHI,ftemp1      ;

            inc   fcs_ptr           ;
            ld    fsize,#8h         ;
RLOOP2:
            rrc   fcsls             ;
            rlc   ftemp1            ;
            djnz  fsize,RLOOP2      ;
            com   ftemp1            ;
            ld    @fcs_ptr,ftemp1   ;
            ld    FCSLO,ftemp1      ;
            ret                     ;

            END
```

Flow Diagrams

Figure 2. Software Flow Block Diagram

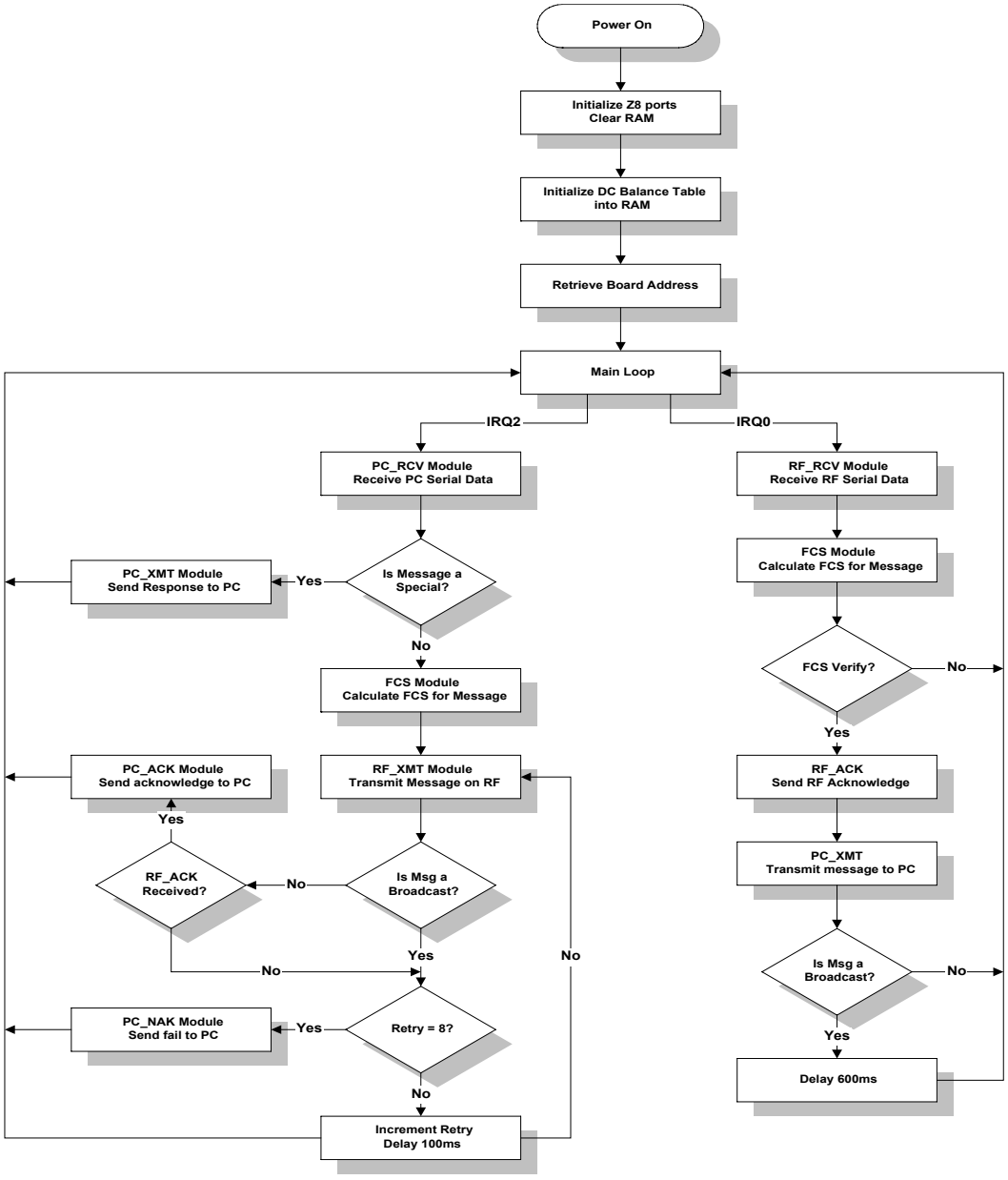
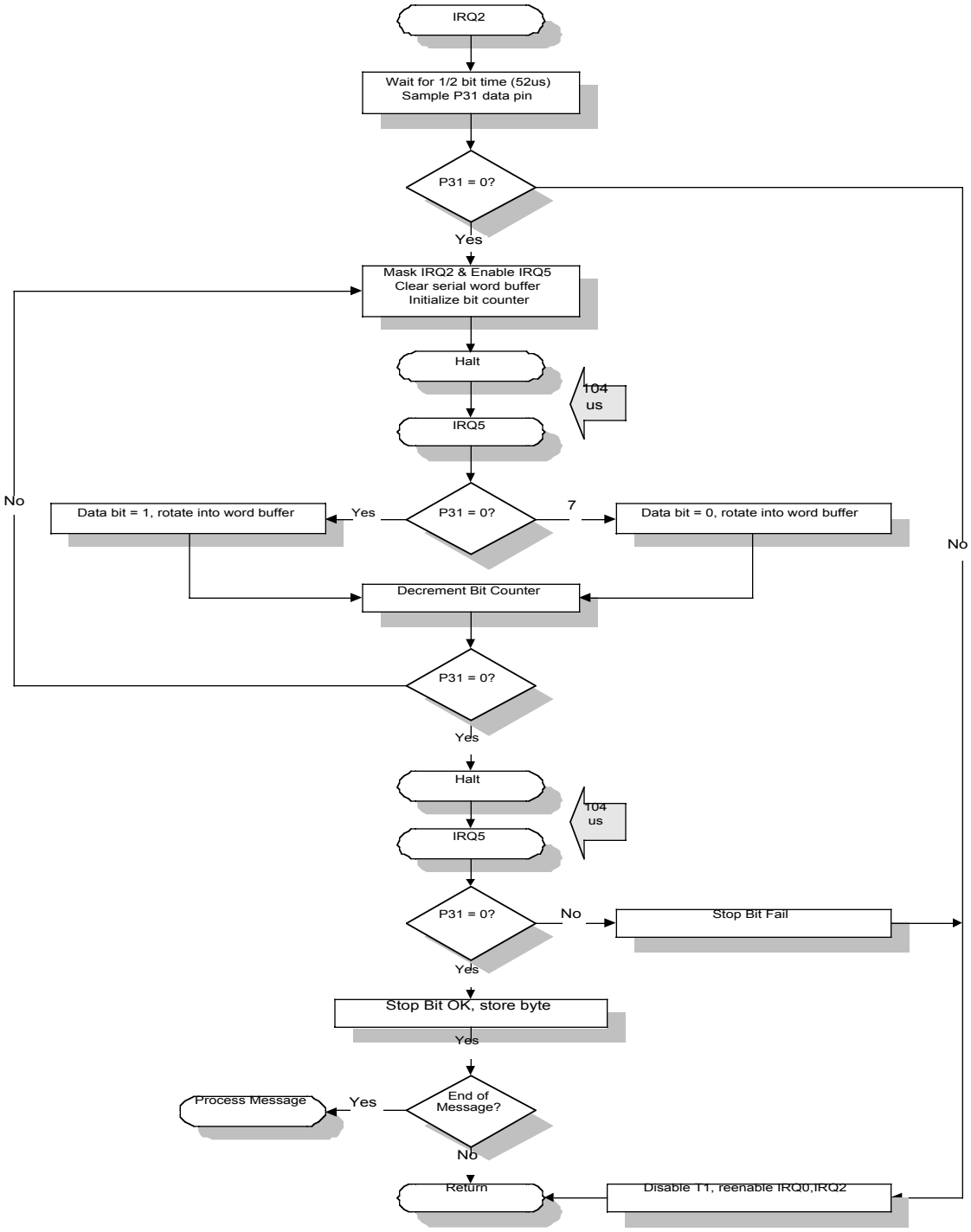


Figure 3. Receiving RF Serial Communication Flow Diagram



Timing Diagrams

Figure 4. PC Data Message Packet Transmission

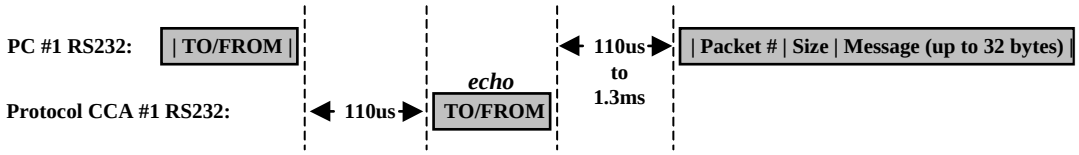


Figure 5. PC Special Message Packet Transmission

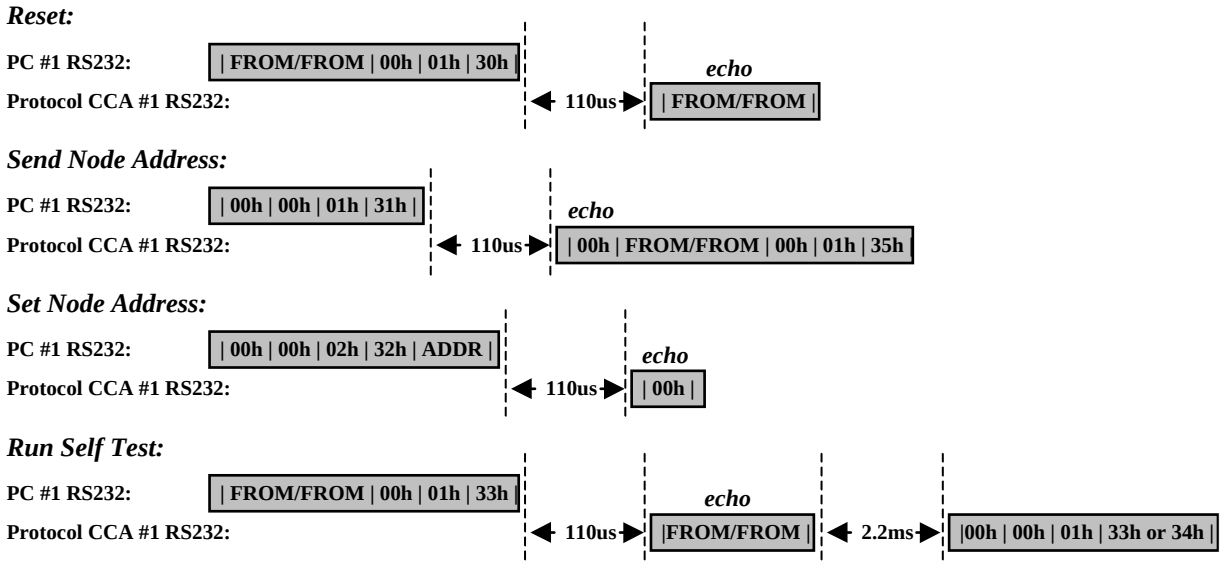


Figure 6. Typical Timing for Message Transmission

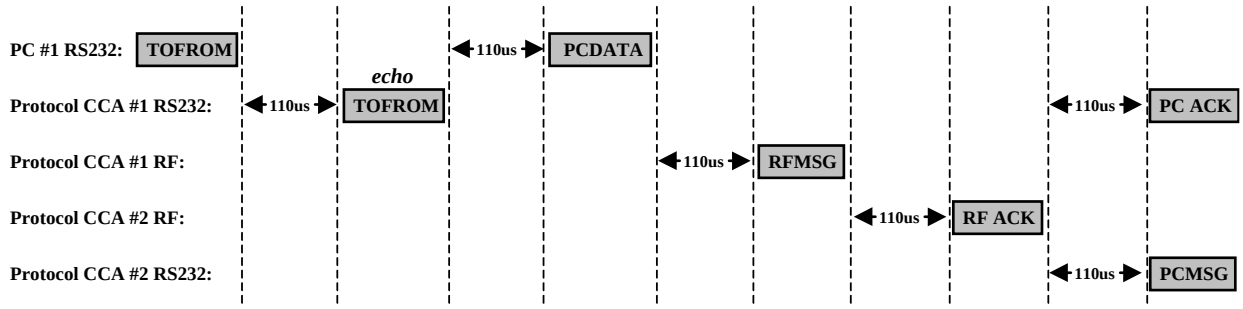


Figure 7. RF Retry Timing

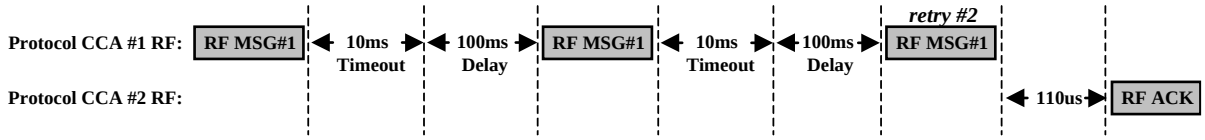


Figure 8. RF Communications Timeout

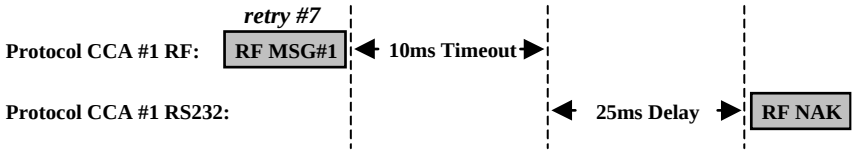


Figure 9. Broadcast Timing

Timing Diagram Glossary

The glossary below defines terms used in the timing diagrams above (Figures 4 through 9).

TOFROM. The first nibble is the node address of the intended receiver. The second nibble is the node address of the sender.

PCDATA. All remaining bytes of a PC data message packet, excluding the TOFROM address.

RF MSG. The RF message packet.

RF ACK. A message to the RF packet originator acknowledging receipt of a valid message packet.

PCMSG. A PC message packet.

PC ACK. A message to the PC acknowledging successful transmission and receipt of an RF message packet.

RF NAK. A message from Protocol CCA to the PC indicating that the RF communications timed out without receiving a valid RF message acknowledgment.

Test Procedure

Equipment Used

Testing the application requires the following items:

- 2 Protocol CCAs with programmed Z86E08
- 2 DR1200/1-DK or 2 DR1004/5-DK Data Radios
- Two Windows 95/98-based PCs, each with:
  - RFM® Terminal Program
  - One available 9-pin serial port
  - Z86CCP01ZEM (CCP Emulator)
  - 8V @0.8 Amp power supply (for emulator power)

General Test Setup and Execution

For simplicity, exercise the application by using two programmed Z86E08 devices in the Protocol CCAs rather than running the application from the emulator. If an emulator is used for one of the nodes, then two serial ports are required on that PC: one for the emulator and one for the applications RS-232 interface.To assemble the code and to program a device, follow the instructions as detailed in the Assembling the Application Code section.

The RFM® Terminal Program provides a medium for sending plain-text ASCII messages from one PC for display on another PC. The program can be obtained either by purchasing the Virtual Wire Design Kit or from the RFM web page at www.rfm.com. To install the program, create a directory on Version 2 of the Terminal Program (`vwt97v02.exe`). When the files are installed, edit the `vwt97.cfg` file. The `.cfg` file is a one-line ASCII file containing the communications port number, baud rate, and TO node address. Edit the line as follows:

```
*COM1:""9600""2*
```

Ensure that the Data Radio is properly installed onto the Protocol CCA. Set the node address of each Protocol CCA by removing or installing jumpers on the address pins. Note the address of each Protocol CCA. Attach each Protocol CCA to a PC serial port either directly or through a serial cable.

Test Results

Apply power to the Protocol CCAs by moving the power switch to the ON position. Start the terminal program Execute the Terminal Program Executable File (`.exe`) on both PCs to start the terminal program.by Upon execution, the program automatically polls the Protocol CCA for the node address. When the program obtains the node address, the program displays the terminal screen. The following list identifies the commands that can be executed from the terminal screen.

```
Esc: End Task
ALT-A: Read Node Number
ALT-B: Broadcast Mode
ALT-C: Clear Screen
ALT-D: Decrement TO Node Address
ALT-H: Help Screen
ALT-I: Increment TO Node Address
ALT-R: Protocol CCA Reset
ALT-S: Protocol CCA Self Test
ALT-X: Exit Program
CTL-N: Software Setting of Node Address
Function Key F1: Sends multi-packet test message (30 bytes)
```

The terminal screen contains three windows. The MESSAGES RECEIVED window displays message packets received from other nodes. The ENTER MESSAGE TO SEND window receives message packets to be sent to other nodes. The MESSAGES SENT window contains a history of message packets sent to other nodes. Note that this window displays the start byte and end byte characters of the message packet as `J` and © respectively. This window also displays the packet number and the status of sent packages. The status is indicated by several different messages. If the packet is transmitted successfully, the status indicates RX OK ON retry with retry being the number of packet transmission attempts before the messages were successfully acknowledged. If after eight tries, the packet remains unacknowledged, a LINK FAULT status is displayed. If the PC loses communication with the Protocol CCA, a TIME-OUT UNIT NOT RESPONDING status is displayed.

References

RFM® Virtual Wire® Development Kit Manual. RF Monolithics, Inc., 1999.

RFM® ASH Transceiver Designer’s Guide. RF Monolithics, Inc., 1999.

ZiLOG Z8® Microcontroller User’s Manual. UM97Z8X0104. ZiLOG Inc., 1997.

Appendix

Figure 10. RF Link Schematic