UNIVERSITY COLLEGE LONDON
Department of Electronics and Electrical Engineering

RDS Encoder

Hamed Haddadi

Supervisor: Dr. Paul Brennan

Final Report

March 2003

Abstract

Radio Data Systems was development started 20 years ago in the European Broadcast Union, EBU. The developers aimed to ease the process of tuning in to a station, especially as the number of broadcasters was increasing and use of alternative frequencies to avoid interference would make it difficult to keep tuned to a certain station. The use of RDS would overcome this problem and will also enable the transmission of the Programme Service name (PS), traffic information and other useful features which will make the FM receivers more user-friendly.

University College London has got its own radio station, RARE FM, which requires an RDS encoder to enable them to transmit the name of the station. Commercial RDS encoders are relatively expensive and complex and they need a dedicated PC and network connection to enable the other features such as traffic information, which is not required by Rare FM. The purpose of this project is to build an isolated RDS encoder which will not need a PC and is simple enough to be used by a non-technical person in the station.

Some work has already been done last year to develop the encoder using analogue techniques. However any changes to the station name will require modification of components and switches and valid data has not been prepared for encoding and modulation alongside the FM carrier. The technique pursued in this project is using a micro-controller to control the data and bit-stream conversion and validations. This makes the system easier to operate and make it easier to change the settings of the transmitted data-stream if needed in future.

This report contains the background research and knowledge taken from the RDS standard. Objectives of the project and the approaches are also discussed within the appropriate section. The work done and the achievements of the project are also explained, as well as the time and budget planning and goals.

One of the major obstacles met in previous projects has been providing data and transmitting the correct information at the right time. In this project the data processing part has been investigated and planned first. This made sure that the new tasks are done before going through the development and improvement of previous attempts to make an RDS Encoder.

Acknowledgements:
The author would like to acknowledge the work done By Timothy Shaw and Richard Koch as their project in development of the RDS encoder and appreciate their valuable advice and ideas given prior to start of the project. Many thanks are also forwarded to Dr. Paul Brennan of the department for his help and supervision from the start of the research into the project and Gerald McBrearty and Andrew Moss of the EE laboratory for their valuable advice and support during the software development and hardware realisation of the project.

CONTENTS

List of Figure and Captions

1. An introduction to RDS

The use of more frequencies for the radio programmes in the VHF/FM range makes it difficult for an in-car radio to remain tuned to the desired programme as the stations have to constantly change frequencies in different regions to avoid interference. RDS employ an FM subcarrier to transmit steady stream traffic information and the station name. This is a real advantage over conventional radio systems as the sales of FM in-car radio systems were not growing at the desired rate. The RDS system allows the station to transmit its Programme Service Name (PS), an eight-character sequence identifying the station. This makes tuning to a station by frequency redundant. Another important addition is the PI code. This code allows receivers to automatically switch to the best available frequency for a particular station, especially useful on long car journeys where frequencies for the same station change to avoid interference patterns.

Following a long period of systems development in the 1970s and early 1980s, RDS is now implemented all over Western Europe, and in several other regions of the world. This was after the improvement of the in-car entertainment system developments. RDS had major advantages for the traveller. RDS can provide traffic information and filter out the unnecessary information when travelling in a specific route by recognizing the location codes. Nowadays RDS is implemented in most FM radios and virtually all in-car radio systems.

Even though the university radio station does not intend to transmit any traffic or news announcements, it still is an advantage if they can employ an RDS encoder to transmit the station name. Commercial RDS encoders are expensive and require a dedicated PC to operate them. The addition of a PC will make the system complicated for non-technical users and also the station's budget does not allow for such purchases. However it is possible to build an RDS encoder by using simple components and a microprocessor.

Some work has already been done to develop and RDS encoder. The knowledge gained from the previous students will be build upon of to make a functional unit within this project. The programmes for the microprocessor, provisionally PIC16F877, will be written and compiled using the MPLAB software and the circuit simulations will be done using the Proteus Virtual System Modelling software.

This report contains a brief review of the RDS standard, which includes the physical layer (hardware) and the data-link layer and message format (software) part of the encoder system. After the review of the theory, the software format, which has been already implemented, has been discussed. As there are various different comparisons made within the text between the different strategies and their implementation advantages and disadvantages, the implementation of the chosen strategy for each specific task is explained immediately after the technical discussion part. A brief description of the intended hardware layout, the project timescale and budget planning are also included in the discussions section.

2. Physical layer of RDS system

The RBDS standard, April 1998 was consulted to produce the following technical details of and RDS signal. RBDS standard is the American version of RDS and it is exactly similar in terms of operation and it was used as it is freely available on the web. The Radio Data System is intended for application to VHF/FM sound broadcasts in the range 87.5 to 108.0 MHz which may carry either stereophonic (pilot tone) or monophonic programs. The main objectives of RDS are to enable improved functionality for FM receivers and to make them more user friendly by using features such as Programme Identification (PI), Programme Service (PS) display and where applicable, automatic tuning for portable and in particular, car radios.

2.1 Subcarrier Frequency

During stereo broadcasts, the subcarrier will be locked to the third harmonic of the 19 kHz pilot-tone. The tolerance on the frequency of the 19 kHz pilot tone is ±2Hz, therefore the tolerance on the frequency of the subcarrier during stereo broadcast will be ± 6 Hz.
During monophonic broadcasts the frequency of the subcarrier will be 57 kHz ±6 Hz.

2.2 Subcarrier phase

During stereo broadcasts the subcarrier will be locked either in phase or in quadrature to the third harmonic of the 19 kHz pilot tone. The tolerance on this phase must be within ±10°, measured at the modulation input to the FM transmitter. Figure 1 shows the FM signal.

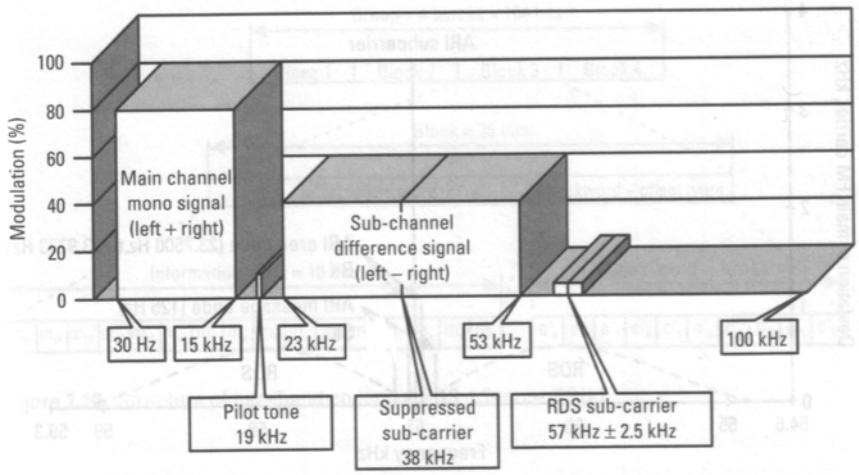


Figure 1: FM spectrum

Figures 2 and 3 represent the block diagrams of the transmitter and receiver.

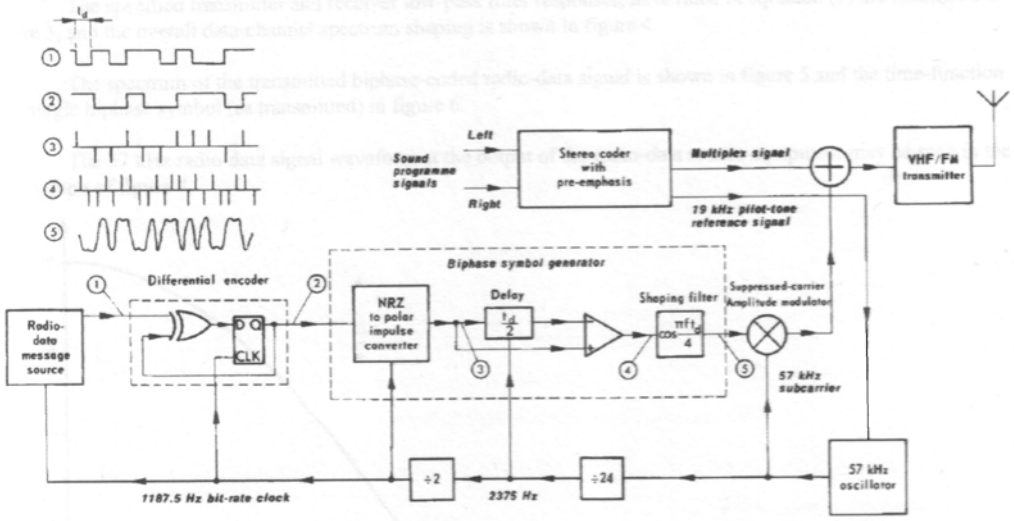


Figure 2: Block diagram of RDS encoder at the transmitter

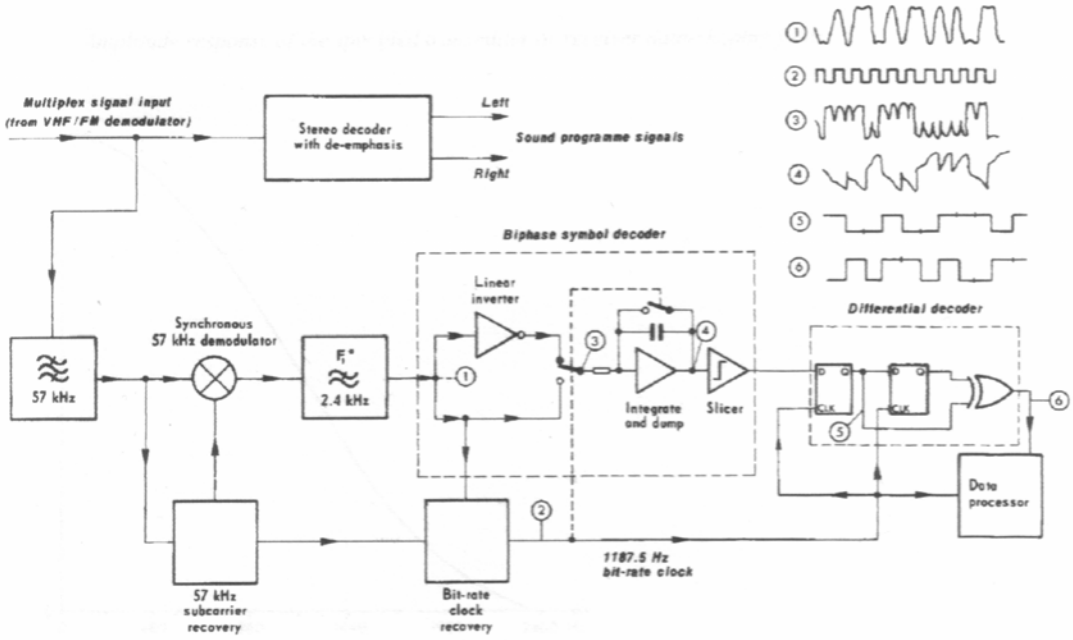


Figure 3: Block diagram of a typical RDS receiver

2.3 Subcarrier level

The deviation range of the FM carrier due to the un-modulated subcarrier is from ±1.0 kHz to ± 7.5 kHz. The recommended best compromise is ± 2.0 kHz. The decoder/demodulator should also operate properly when the deviation of the subcarrier is varied within these limits during periods not less than 10ms. The maximum permitted deviation due to the composite multiplex signal is ± 75 kHz.

2.4 Method of modulation

The subcarrier is amplitude-modulated by the shaped and bi-phase coded signal. The subcarrier is suppressed. This method of modulation may alternatively be thought of as a form of two-phase phase shift keying (PSK) with a phase deviation of ± 90°.

2.5 Clock Frequency and Data Rate

The basic clock frequency is obtained by dividing the transmitted subcarrier frequency by 48. The basic data rate of the system is therefore 1187.5 bit/s ± 0.125 bit/s.

2.6 Differential Coding

The source data at the transmitter are differentially encoded using table 1:

Previous output (at time t_{i-1})	New input (at t_i)	New Output (at time t_i)
0	0	0
0	1	1
1	0	1
1	1	0

Table 1: Differential encoding

Where t_i is some arbitrary time and t_{i-1} is the time one message-data clock period earlier, and where the message-data-clock rate is equal to 1187.5 Hz.

Thus when the input level is 0, the output remains unchanged from the previous output bit, and when an input 1 occurs, the new output bit is the complement of the previous output bit. In the receiver, the data may be decoded by the inverse process, as shown in table 2.

Previous input (at time t_i-1)	New input (at t_i)	New Output (at time t_i)
0	0	0
0	1	1
1	0	1
1	1	0

Table 2: Differential decoding

The data is thus correctly decoded whether or not the demodulated data signal in inverted.

2.7 Data Channel Spectrum Shaping

The power of the data signal at and close to the 57 kHz subcarrier is minimised by coding each source data bit as a bi-phase symbol.

This is done to avoid data-modulated cross talk in phase locked loop (PLL) stereo decoders, and to achieve compatibility with the ARI system. The principle of the process of generation was shown in Fig 2. In concept each source bit gives rise an odd impulse pair, e(t), such that logic level 1 at source gives:

$$e(t) = \delta(t) - \delta\,(t - t_d/2)$$

And logic 0 at source gives:

$$e(t) = \delta(t) + \delta\,(t - t_d/2)$$

These pairs are then shaped by a filter $H_T(f)$, to give the required band limited spectrum where:

$$H_T(f) = \begin{cases} \cos\dfrac{\pi f t_d}{4} & 0 \le f \le 2/t_d \\ 0 & f > 2/t_d \end{cases}$$

$$t_d = \frac{1}{1187.5}\ \text{s}$$

The data-spectrum shaping filtering has been split equally between the transmitter and the receiver (to give optimum performance in the presence of noise) so that, ideally, the data filtering at the receiver should be identical to that of the transmitter, i.e. as given above. The overall data-channel spectrum shaping $H_o(f)$ would then be 100% cosine roll-off.

The specified transmitter and receiver low-pass filter responses, as defined in previous equations, and the overall data-channel spectrum shaping is shown in figure 5.
The spectrum of the transmitted bi-phase coded radio-data signal is shown in figure 6 and the time-function of a single bi-phase symbol (as transmitted) in figure 7.

The 57 kHz radio-data signal waveform at the output of the radio-data source equipment may be seen in the photograph of figure 7.

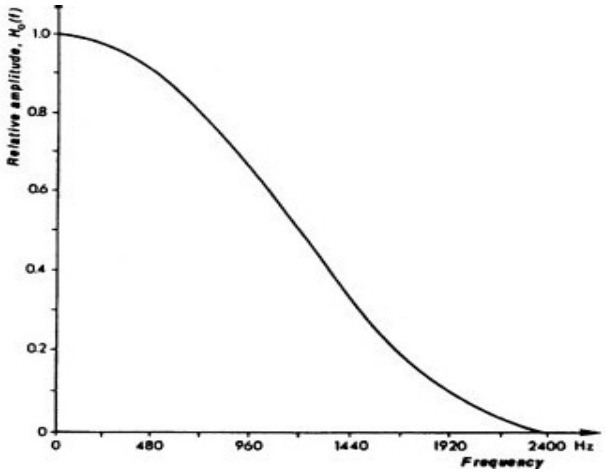


Figure 4: Amplitude response of the specified transmitter or receiver data-shaping filter

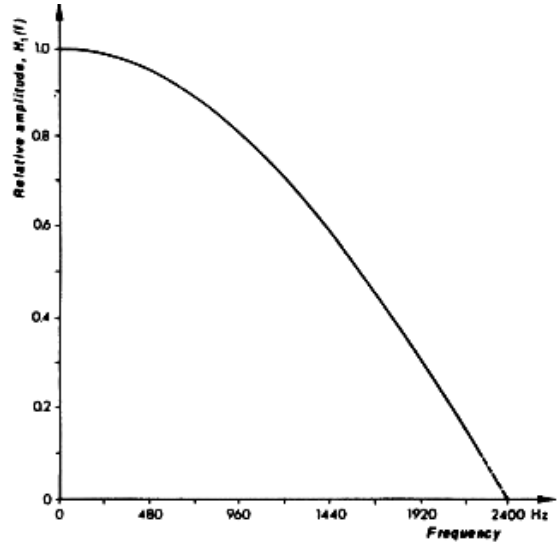


Figure 5: Amplitude response of the combined transmitter and receiver data-shaping filters

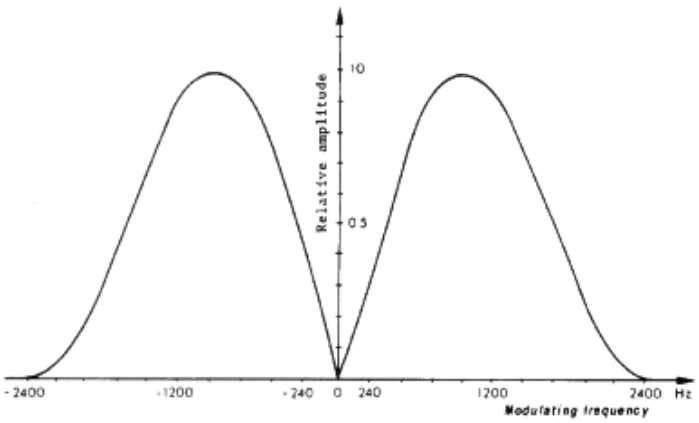


Figure 6: Spectrum of bi-phase coded radio-data signals

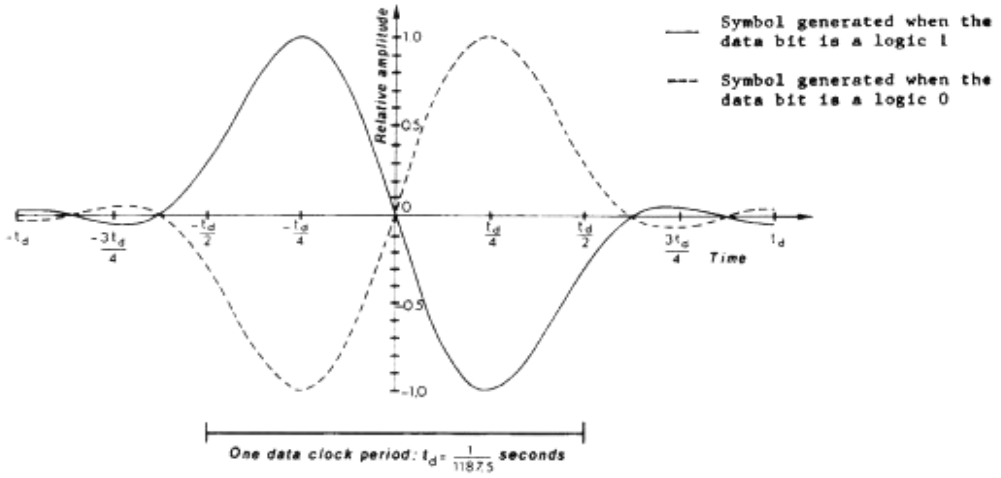


Figure 7: Time-function of a single bi-phase symbol

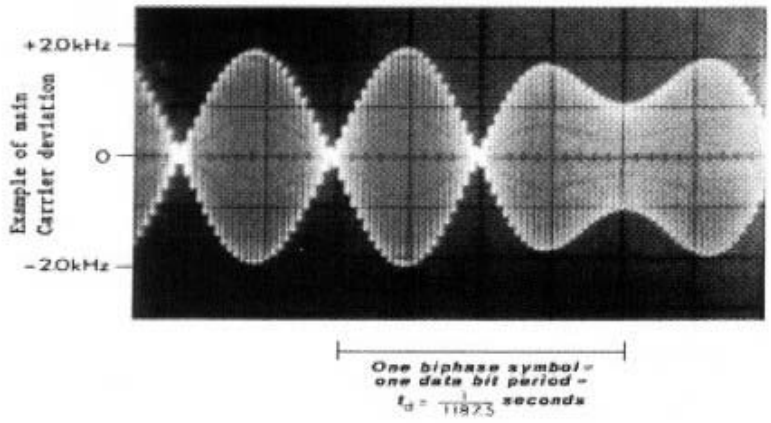


Figure 8: 57 kHz radio-data signals

3. Base-band coding of the RDS system

3.1 Base-band coding structure

Figure 9 shows the structure of the baseband coding. The largest element in the structure is called a "group" of 104 bits each. Each group comprises 4 blocks of 26 bits each. Each block comprises an information word and a checkword. Each information word comprises 16 bits. Each checkword comprises 10 bits.

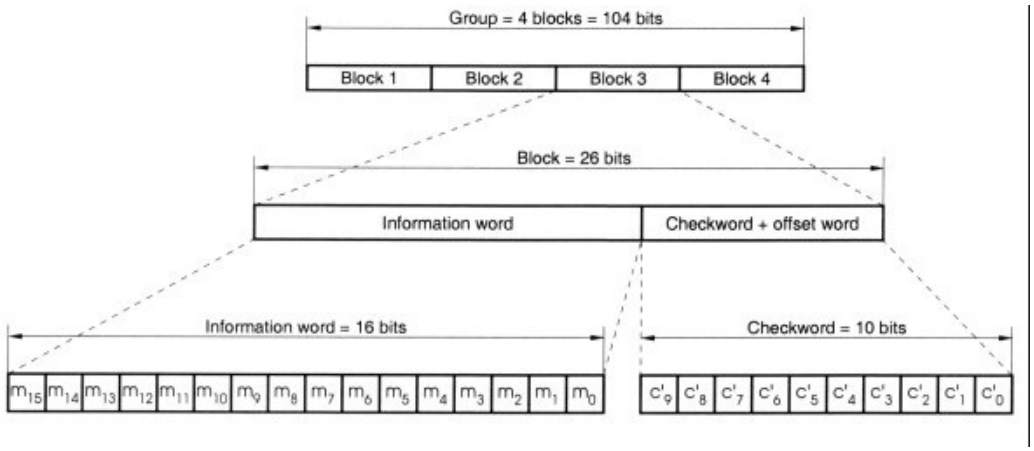


Figure 9: Structure of the baseband coding

3.2 Order of bit transmission

All information words and checkwords have their most significant bit (m.s.b) transmitted first (see figure 10). Thus the last bit transmitted in a binary number or address has weight 2^0.
The data transmission is fully synchronous and there are no gaps between the groups or blocks.

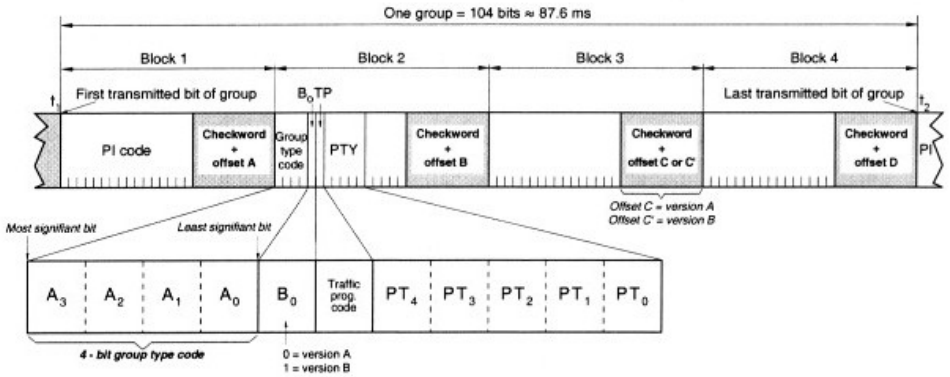


Figure 10: Message format and addressing

Information words and their use are explained in section 4, message format.

3.3 Error protection

Each transmitted 26-bit block contains a 10-bit checkword which is primarily intended to enable the receiver/decoder to detect and correct errors which occur in transmission. This checkword (i.e. $c'_9, c'_8 \ldots c'_0$ in figure 9) is the sum (modulo 2) of:

a) the reminder after multiplication by x^{10} and then division (modulo 2) by the generator polynomial g(x), of the 16-bit information word),
b) a 10-bit binary string d(x), called the 2offset word2,

Where the generator polynomial, g(x) is given by:

$$g(x) = x^{10} + x^8 + x^7 + x^5 + x^4 + x^3 + 1$$

The offset value, d(x), which is different for each block within a group is given in table 3.

Offset Word	Binary Value									
	D_9	D_8	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
A	0	0	1	1	1	1	1	1	0	0
B	0	1	1	0	0	1	1	0	0	0
C	0	1	0	1	1	0	1	0	0	0
C'	1	1	0	1	0	1	0	0	0	0
D	0	1	1	0	1	1	0	1	0	0

Table 3: The checkword offset values

The purpose of adding the offset word is to provide a group and block synchronisation system in the receiver/decoder. Because the addition of the offset is reversible in the decoder, the normal additive error correcting and detecting properties of the basic code are unaffected.
The error-protecting code has the following error-checking capabilities:

a) Detects all single and double errors in a block.
b) Detects any single error burst spanning 10 bits or less.
c) Detects about 99.8% of bursts spanning 11 bits and about 99.9% of all longer bursts.

The code is also an optimal burst error correcting code 5 and is capable of correcting any single burst of 5 bits or less.
The beginnings and ends of the data blocks ma be recognised in the receiver/decoder by using the fact that the error-checking decoder will detect block synchronisation slip as well as the additive errors. This system is made reliable by the addition of the offset words, which also serve to identify the blocks within the group.

3.4 Message format (session and presentation layer)

The main features of the message structure have been illustrated in figure 10. These may be seen to be:

1) The first block in every group always contains a Program Identification (PI) code. This information consists of a code enabling the receiver to distinguish between countries, areas in which the same programme is transmitted and the identification of the program itself.

2) The first four bits of the second block of every group are allocated to a four-bit code which specifies the application of the group. Groups will b referred to as types 0 to 15 according to the binary weighting of A_3, A_2, A_1, and A_0. For each type (0 to 15) two "versions" can be defined. The "version" is specified by the fifth bit (B_0) of block 2:
a) $B_0 = 0$: the PI code is inserted in block 1 only. This is called version A.
b) $B_0 = 1$: the PI is inserted in blocks 1 and 3 of all group type. This is version B.

3) The Programme Type code (PTY) and Traffic Program identification (TP) occupy fixed location in block 2 of every group. This is an identification number to be transmitted with each program item and which is intended to specify the current program type (e.g. news, sports…)

The above features are available in all of the 30 possible group types. The main objective of this project is to transmit the station name. Groups 0A and 0B are the most basic groups that without any need for other information, e.g. constant information feedback or traffic or text update, will transmit the station name. Block 3 of Group 0A consists of a list of alternative frequencies. This feature is for hand-over between different frequencies for stations which transmit over a wide geographical area and is not being used by the UCL radio station. Block 3 of group 0B simply repeats the PI code, with a different offset word, C'. Hence for the purpose of this project, group 0B is chosen for transmitting the station name. Figure 11 shows the format of the group type 0B.

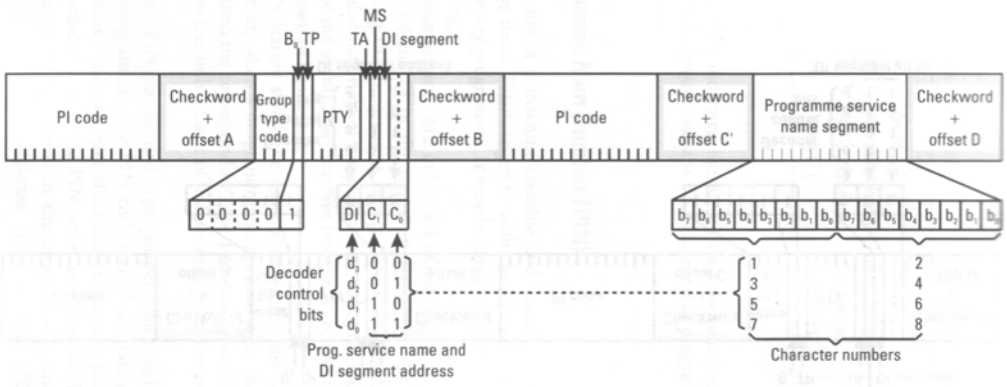


Figure 11: Basic tuning and switching information – Type 0B group

A total of four type 0B groups are required to transmit the entire Program Service (PS) name and therefore four type 0B groups will be required per second. The Program Service name comprises eight characters. It is the primary aid to listeners in program identification and selection. The PS name is to be used only to identify the station. This text may be changed as required by station, but shall not be scrolled or flashed or altered in a manner that would be distracting to the viewer (i.e. not more frequently than once per minute).

Notes on group 0B:
1. TA = Traffic Announcement code (1 bit)

2. M/S = Music-Speech switch code

3. DI = Decoder-Identification control code (4 bits). This code is transmitted as 1 bit in each 0B group. The Program Service name and DI segment address code (C_1 and C_0) serve to locate these bits in the codeword. Thus in a group with C_1C_0 = "00" the DI bit is d_3. These code bits are transmitted most significant bit (d_3) first. Table 4 demonstrates the DI bit-settings.

Settings	***Meaning***
Bit d_0 set to 0:	**Mono**
Bit d_0 set to 1:	**Stereo**
Bit d_1 set to 0:	**Not Artificial Head**
Bit d_1 set to 1:	**Artificial Head**
Bit d_2 set to 0:	**Not compressed**
Bit d_2 set to 1:	**Compressed**
Bit d_3 set to 0:	**Static PTY**
Bit d_3 set to 1:	**Dynamically switched PTY**

Table 4: DI code bits

4. Program Service name is transmitted as 8-bit character as defined in the 8-bit code-tables in annex E of RBDS standard [1]. Eight characters (including spaces) are allowed for each network and are transmitted as a 2-characteer segment in each group. These segments are located in the displayed name by the code bits C_1 and C_0 in block 2. The addresses of the characters increase from left to right in the display. The most significant bit (b_7) of each character is transmitted first.

4. Project objectives and strategies

As clearly suggested by the title, the main objective of this project is to build an RDS encoder to enable the UCL radio station, RARE FM, to transmit the station name on the FM spectrum. This task has been attempted in the Electronic & Electrical Engineering department. The main reason of failure of previous attempts has been the absence of valid data for transmission. The hardware design has been studied and areas for improvement have been identified. The focus of the first stage of the project has mainly been on development of the message format and session-presentation layer. The following objectives are set for the project:

- To build a complete (hardware & software) RDS Encoder to facilitate Program Service transmission for RARE FM.
- To provide valid data according to the European RDS standard for encoding.
- To design and implement a user-friendly interface to enable non-technical personnel to easily enter the desired PS name.
- To offer the facility to restore the settings hence avoid the need to enter the same data in case of power failure.
- To complete the unit within the allocated time and budget, 6 months and £100.

The choice of group 0B for the purpose of the encoding has been based on the fact that it will allow the transmission of PS with no need for any other information such as traffic or announcement. This means that the unit functions as a stand-alone without any need for further attention to provide data or network connection. Transmission of PS as the main objective of the project will enable easy tuning for the in-car radio units.

4.1 Providing frame data

There are various constants and variables within a group. It would be possible to enable the user to enter all the required data at the start-up part of the system. But this will only make the unit extremely hard to operate as all the values have to be set according to the RDS standard and any mistake will mean that the bit-stream is not verified and displayed at the receiver end. In order to avoid this, the constant parts of the group data are set within the program memory, according to RDS standard as explained below, and user is only required to enter the PS name.

1. **PI code**: Figure 12 shows the PI structure.

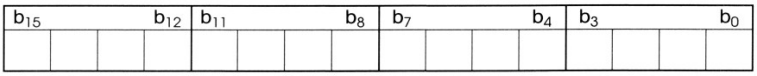


Figure 12: PI structure

These bits are all pre-set in according to RDS standard as followed:

Bits b_{15} to b_{12}: Country Code: These bits are set to 0xC or 0b1100 for UK.
Bits b_{11} to b_8: Area coverage: These bits are set to 0x0 or 0b0000 for local coverage.
Bits b_7 to b_0: Program reference number: These bits are set to 0x00 for not assigned.

2. **Group Type**: These bits are set to 0x0 for 0 group type.
3. **B_0**: This bit is set to 1 for group B type.
4. **TP**: This is set to 0 for no traffic program.
5. **PTY**: These bits are set to obooooo for not assigned.
6. **TA**: This is set to 0 for no traffic announcement.
7. **M/S**: This bit is set to 0 according to RDS standard by default for music.
8. **DI, C_1, C_0:** These bits are set in each group according to the group sequence in the stream as followed:

A) First group: 0b0 (d_3)00 for static PTY.
B) Second group: 0b0 (d_2)00 for not compressed.
C) Third group: 0b0 (d_2)00 for no artificial head.
D) Second group: 0b1 (d_2)00 for stereo.

9. Program Service name: Every group will transmit 2 characters, starting from the most significant character, e.g. [RA] in (RARE__FM) is located in first group.

4.2 Data input method for PS

Data can either be input via setting 64 switches for the 8 character ASCII code of the PS name or it can be input to the microcontroller via push button switches or a PS2 keyboard. The analogue method has been attempted before and is extremely difficult to operate. The more feasible design is to input the values to the microcontroller via a digital device:

A: PS2 keyboard. This method is extremely friendly but it introduces the complexity of writing a keyboard driver and also occupying much more space while adding the risk of breakages and damage in the busy studio environment.

B: Push button switches: Using three heavy duty push button switches, the user can decide whether to restore the previous settings or not, scrolling between YES and NO, and then pressing the SET switch. If he chooses not to restore the previous settings, he will scroll through the available set of ASCII characters, from SPACE through Z. By pressing the set button, the first character is set and stored in the EEPROM. The same routine is repeated for the other 7 characters and by pressing the SET button for the final character, the program will proceeds to CRC calculation. The disadvantages of this method are the cost of these switches and the fact that using this method, it will take a bit longer to input the data. But the cost can be compromised by the fact that the switches are not prone to damages in the studio environment. Figure 13 shows the design of this interface.

Figure 13: Data input via push-button switches

The following algorithm is implemented for polling for the switches for restoring data:

```
CHECK_SWITCH                ; Polls switches for decision to restore setting
GET_IT  btfsc   SWITCH_PORT,0
        call    D0_SET              ; restore
        btfsc   SWITCH_PORT,1
        call    D1_SET              : Don't restore
got     btfss   SWITCH_PORT,2
        goto    GET_IT
        movf    DECISION,W          ; Check if D2 is not pressed
        andlw   0xff                ; without making a  decision
        btfsc   STATUS,Z
        goto    GET_IT              ; if yes, loop until decision is made
        return
```

The following code demonstrates the acquisition of one PS character and storing it in the PIC EEPROM:

```
GET_PS8 btfsc   SWITCH_PORT,0
        call    CHAR_UP
        btfsc   SWITCH_PORT,1
        call    CHAR_DOWN
        btfss   SWITCH_PORT,2
```

```
        goto    GET_PS8
        return

CHAR_UP
        call    SWITCH_DELAY
        movf    POSITION,0      ; set display position
        call    SET_ADDR
        movf    CHARACTER,0     ; move character to W register
        call    check_max       ; check if reached the upper boundary
        movf    CHARACTER,0
        call    LCD_CHAR
        return

check_max                       ; it checks mor the upper character
        movwf   CHAR_TEMP       ; store the character locally
        movf    MAX_CHAR,0      ; now move MAX_CHAR to  W register
        subwf   CHAR_TEMP,0     ; subtract the two files
        btfsc   STATUS,Z        ; Is the result zero?? then we reached maximum
        goto    SET2MAX         ; keep the character as it is
        goto    INCREMENT       ; if not reached the max, increment the character

SET2MAX
        movf    MAX_CHAR,0      ; Keep MAX_CHAR as the character of chioce
        return

INCREMENT
        incf    CHARACTER,1     ; increment the character
        return

CHAR_DOWN
        call    SWITCH_DELAY
        movf    POSITION,0      ; set display position
        call    SET_ADDR
        movf    CHARACTER,0     ; move character to W register
        call    check_min       ; check if reached the lower boundary
        movf    CHARACTER,0
        call    LCD_CHAR
        return

check_min                       ; it checks mor the lower character
        movwf   CHAR_TEMP       ; store the character locally
        movf    MIN_CHAR,0      ; now move MIN_CHAR to  W register
        subwf   CHAR_TEMP,0     ; subtract the two files
        btfsc   STATUS,Z        ; Is the result zero?? then we reached minimum
        goto    SET2MIN         ; keep the character as it is
        goto    DECREMENT       ; if not reached the min, decrement the character

SET2MIN
        movf    MIN_CHAR,0      ; Keep MIN_CHAR as the character of chioce
        return


DECREMENT
        decf    CHARACTER,1     ; decrement the character
        return
```

EEPROM storage:

```
        movwf   PS_8            ; store in variable
        bsf     STATUS,RP1      ; Store in EEPROM
        bsf     STATUS,RP0
        btfsc   EECON1,WR       ; Make sure there is no other WRITE in progress
        call    DELAY_5MS
        bcf     STATUS,RP0
        movwf   EEDATA
        movlw   0x07
        movwf   EEADR
        bsf     STATUS,RP0
        bcf     EECON1,EEPGD
        bsf     EECON1,WREN
        movlw   0x55
        movwf   EECON2
        movlw   0xaa
        movwf   EECON2
        bsf     EECON1,WR
        bcf     EECON1,WREN
        clrf    STATUS          ; Select bank 0
```

4.3 Data display and user interface

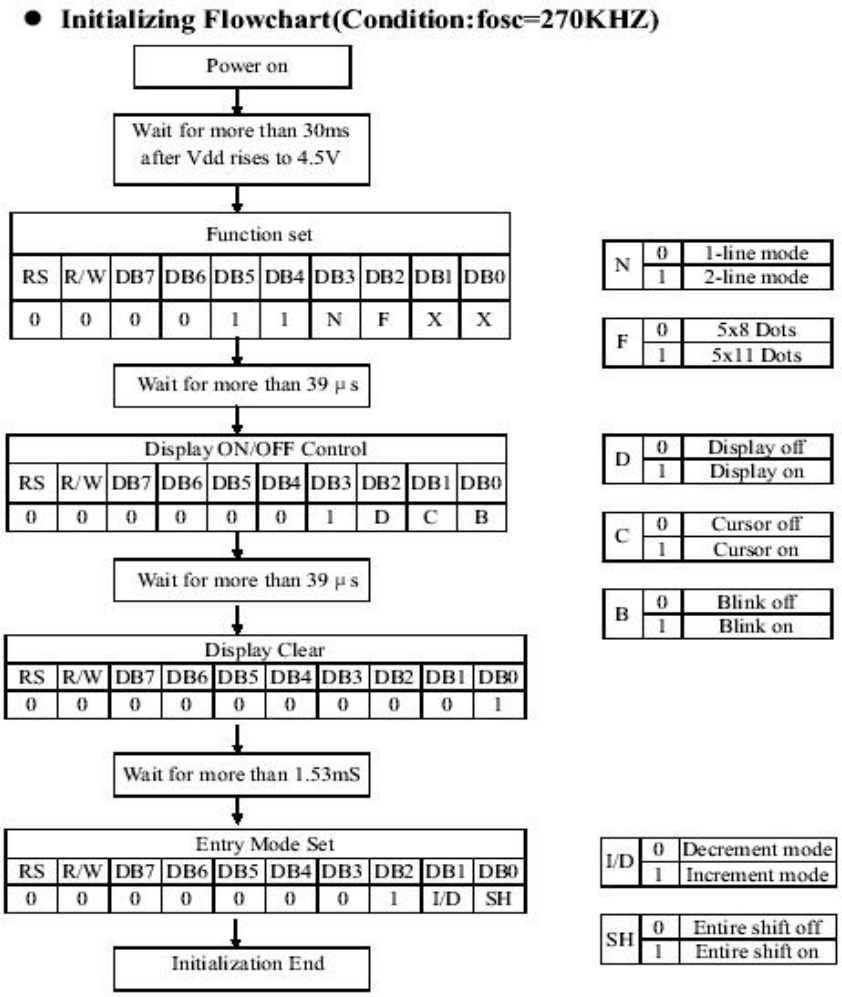
As there is a need for characters to be displayed for verification and during transmitting stage, there is need for a display. Two type of display are available:

A: Alphanumeric LED display

This display is cheap and easy to operate as each character is driven separately and there is no processing required, but in order to use this facility, there is a need for driving each character separately, this will mean either using many ICs, which will increase the complexity of the PCB and will introduce heat dissipation problems, or multiplexing the data line through one IC, which maybe very complicated for driving a large number of characters.

B: Using an intelligent LCD display

The new ranges of displays have eliminated the need for driving each character separately by using an internal chip and timing issues are dealt within the chip. But they are relatively expensive and a typical entry level can cost around £30. There is also need for writing an initialisation code for the display and one of the ports need to be constantly swapped between input and output to check the busy flag of the LCD to make sure there in no data loss. The initialisation code is as followed:



For the purpose of this project, a very cheap LCD module was obtained which would cost 60% less than the commercial ones available. But the initialisation was implemented in a different way than the standard data sheet and the code is shown below:

```
; INIT_LCD..... Module to initialise the display
;*********************************************************************
INIT_LCD
       call    DELAY_30MS

       movlw   0x38           ; 8-bit-interface, 2-lines... 0b00111000
       call    LCD_CMND
       call    DELAY_5MS
```

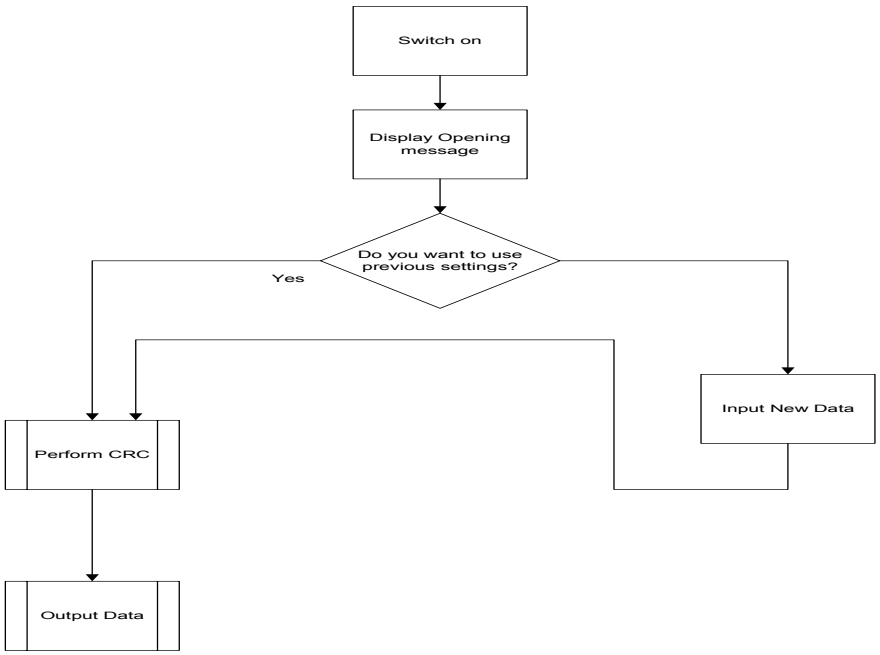
```
movlw   0x38            ; 8-bit-interface, 2-lines... 0b00111000
call    LCD_CMND
call    DELAY_125US

movlw   0x38            ; 8-bit-interface, 2-lines... 0b00111000
call    LCD_CMND
call    DELAY_125US

movlw   0x0F            ; display On, curser On, blink On 0b00001111
call    LCD_CMND
call    DELAY_5MS

movlw    0x01           ; display clear, 0b00000001
call    LCD_CMND
return
```

The delays have been implemented using a series of recursive loops. All the function calls and descriptions can be found in appendix A.

4.4 Data processing and input/output control

Providing the data at the right time for modulation with the FM signal has been the major obstacle of this project in previous attempts. Analogue and digital methods have both been tried and with great advantage, a digital source provides a much more versatile way of controlling and manipulating the data. Since the use of intelligent LCD and de-bounce switches will also emphasis the need for a microprocessor, this method was chosen for data control. The following is the control software data flow required for the encoder:

Top Level Software Flow Diagram



After studying the wide range of microcontrollers that are commercially available, Microchip's PIC16F877 was chosen for the data control. This is an 8 bit microprocessor with 8k program memory and 256 byte EEPROM data memory which both memory limits were enough for the application. The development environment, MPLAB IDE, is freely available on the web and from Microchip's website [2] and the programmer, PICstart PLUS, is available in the departmental laboratory. Figure 14 shows the pin-out diagram of PIC16F877.

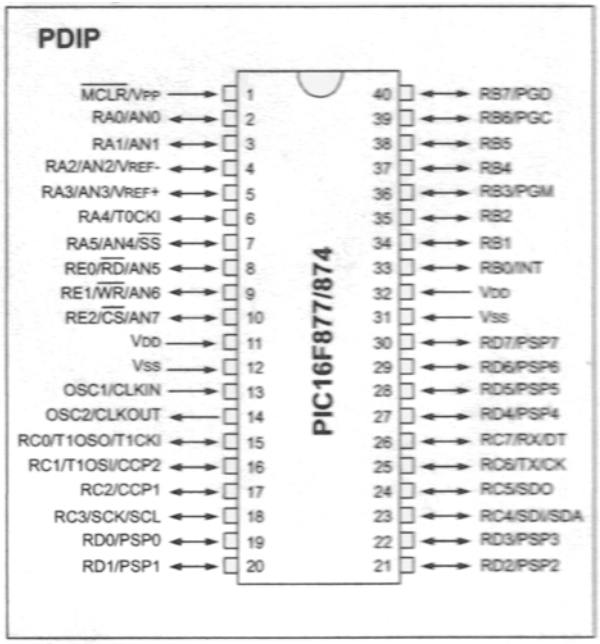


Figure 14: Pin-out diagram of PIC16F877 micro-processor

It operates with clock rates of up to 20MHz giving an instruction arte of up to 5 MIPS, so the clock frequency is equal to 0.2µs. A crystal oscillator was chosen to ensure that correct timings and clock rate are achieved. Figure 15 shows the connection of the crystal to the microprocessor.

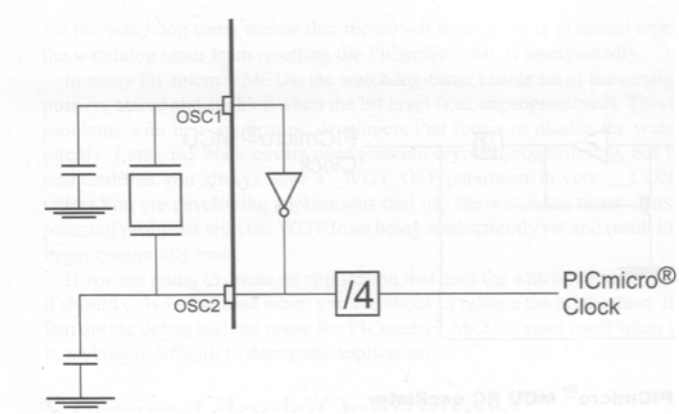


Figure 15: Clocking the PIC

The suggested values for the stabiliser capacitors were between 15-33 pF.

Another problem encountered was the fact that switches bounce back after being pressed down and released which would lead to many jumps in the character range with just one switch. This problem was solved by adding a short 0.2 second de-bounce delay after the switch port reading, which would clear the port from the previous switch voltage. The switches were also tied with 4k7 Ohm resistors to limit the current drawn from the power supply when the switches are being pressed often. Figure 16 demonstrates the data control and user-interface of the RDS encoder.

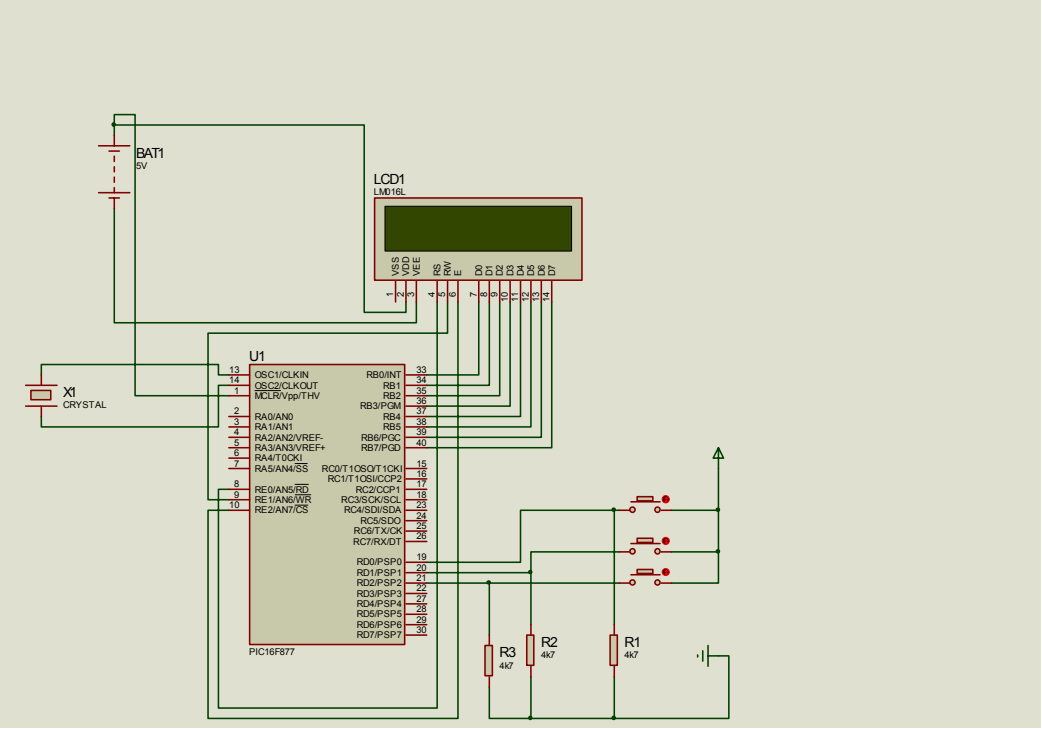


Figure 16: RDS encoder data input and user interface

4.5 CRC calculation

Every block in the group contains a 16 bit information word and a 10 bit checkword which is obtained by adding the result of the calculation of the Cyclic Redundancy Check to a specific offset, as explained in section 3.3. Calculation of the CRC can be done in the hardware as explained in the RDS standard. Figure 17 shows the hardware arrangement required to generate the 10 bit CRC value.

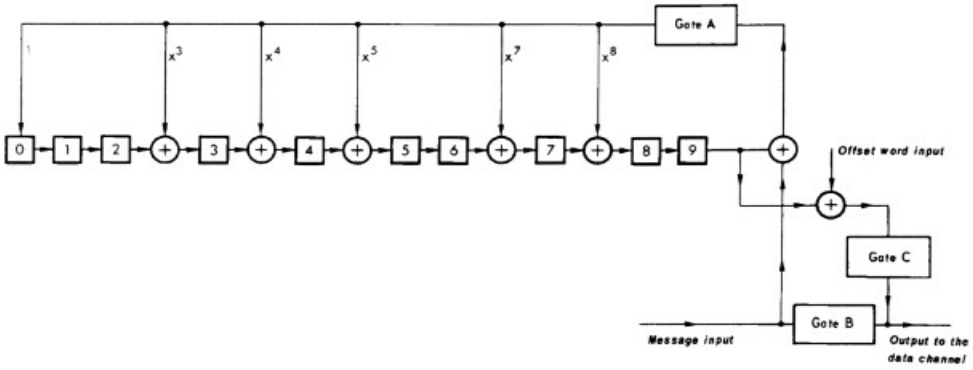


Figure 17: CRC generator circuit

This method can lead to a very high component count and greatly add to the complexity of the circuits. It has the advantage of being easily implemented as the method is pre-designed.

It is also possible to calculate the CRC value in the microprocessor. This will lead to a much less complex circuit and budget saving on component purchase. The only disadvantage of this method is the need to divide a 26 bit number by a 10 bit number, in an 8 bit microprocessor. This method was chosen to pursue whilst accepting the challenge of mathematical manipulations. Figure 18 shows the basic method of performing a modulo2 division.

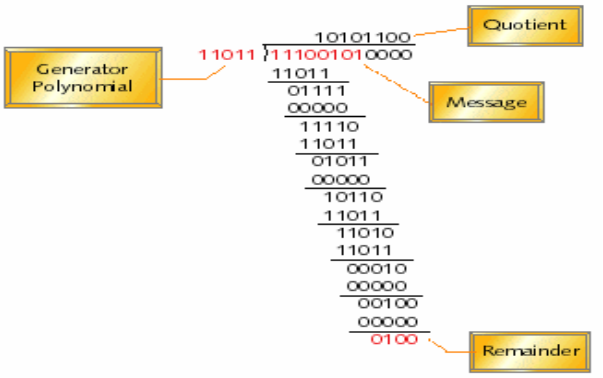


Figure 18: Modulo2 division

However, in a microprocessor, there are only 8-bit registers available, and there is no command for modulo2 division. However, using pseudo algorithm this process can be explained:

```
1. Load the register with zero bits.
2. Augment the message by appending W zero bits to the end of it.
While (more message bits)
Begin
3. Shift the register left by one bit, reading the next bit of the augmented
message into register bit position 0.
If (a 1 bit popped out of the register during step 3)
Register = Register XOR Poly.
End
```

The register now contains the remainder.

In practice the IF condition can be tested by testing the top bit of Register before performing the shift. Figure 19 demonstrates the implementation of the above technique. This method can be extended for larger numbers. In the case of the RDS block, 26 bits means that four 8-bit registers are used for the message data and the 10-bit polynomial will require two 8-bit registers. A buffer register and two working registers will be used, and data is left-rotated into the working registers, bit by bit, and whenever a 1 appears on the carry flag, both working registers or XORed with the generator polynomial registers. This process is repeated until all the 16 extra bits (in comparison with the 10-bit polynomial) are shifted out. The reminder in the working registers is the CRC value for the given 16-bit information word.

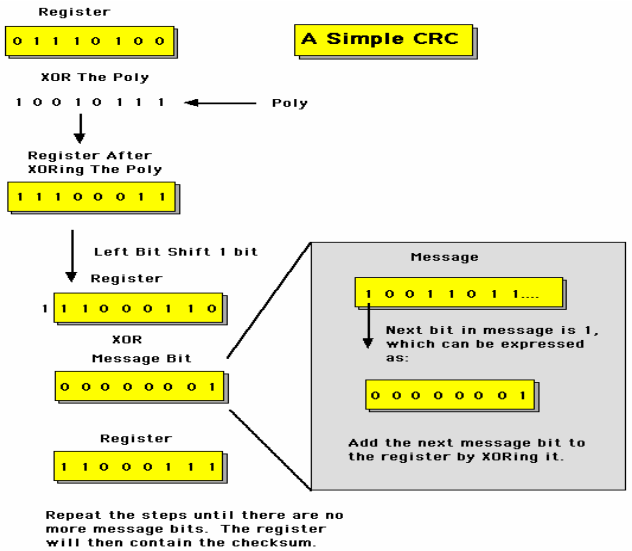


Figure 19: Modulo2 division using one register

The following is an extract from the assembly code generating the CRC:

```
CRC_INIT                        ; Initialise the registers
        clrf    CHAR_BUF        ; rotate left twice, through carry,
        bcf     STATUS,C        ; so that the 10^X factor can correctly
        rlf     CHAR_LOW,1      ; be implemented, i.e. by adding an 8bit
        rlf     CHAR_HIGH,1     ; all 0s register to the end of the data,
        rlf     CHAR_BUF,1      ; and shifting to left twice to have another
        bcf     STATUS,C        ; 2 obits.
        rlf     CHAR_LOW,1
        rlf     CHAR_HIGH,1
        rlf     CHAR_BUF,1

        movlw   0x10            ; 16 left shifts, for the CRC calculation
        movwf   ITERATIONS
        movf    CHAR_BUF,0      ; shifting all to the right place
        movwf   CRC_HIGH
        movf    CHAR_HIGH,0
        movwf   CRC_LOW
        movf    CHAR_LOW,0
        movwf   CRC_BUF1
        movlw   0x00            ; adding the last 8 zero's
        movwf   CRC_BUF2

        movlw   0x05            ; Storing the g(x) polynomial in registers
        movwf   GXPOLY_HIGH
        movlw   0xb9
        movwf   GXPOLY_LOW

        return

CRC_GEN                         ; CRC generator routine
        bcf     STATUS,C        ; clear the carry bit
        rlf     CRC_BUF2,1      ; left shift all the data by one bit,
        rlf     CRC_BUF1,1      ; using carry flag, so the carry 1 or 0
        rlf     CRC_LOW,1       ; will go to the LSB of next byte
        rlf     CRC_HIGH,1
        btfsc   CRC_HIGH,2      ; is the last bit a 1?
        call    DO_XOR          ; if yes, XOR the working registers with g(x)
        decfsz  ITERATIONS,1    ; Decrement iterations, more bits left?
        goto    CRC_GEN         ; if yes, do another bit
        return

DO_XOR                          ; Modulo 2 division with the g(x)
        movf    GXPOLY_LOW,0
        xorwf   CRC_LOW,1
        movf    GXPOLY_HIGH,0
        xorwf   CRC_HIGH,1
        return
```

4.6 Bit stream output

Data transmission tasks start after the CRC calculation steps. User is informed of the PS name currently being transmitted via the LCD display and the microprocessor constantly polls for the 1187.5 signal. As soon as this signal goes high, the m.s.b of first block of the first group is output to the output port. The block data is then rotated left and the carry will represent the next bit to be transmitted. This process is repeated 16 times for block information words and 8 times for each check-word value. For the top 2 bits of the check-word values, the data nibbles are swapped and then rotated left twice. In this way there is only need for two transmission cycles and many fewer instruction cycles. This is done in only 5 steps so it will have a frequency much higher than the 1187.5Hz signal. The next bit will be ready for transmission before the 1187.5 goes high again. After the first group, the second, third and fourth group are transmitted in turn and then the program loops back to the first group. The following code is a very small part of the transmission assembly code:

```
        movlw   0x02            ; Send the top two bits of PS checkword
        movwf   ITERATIONS
        movf    CHCKWRD_3_HIGH,0        ; Move the whole byte
        movwf   TRX_BUF
        swapf   TRX_BUF,1               ; Swap nibbles
        rlf     TRX_BUF,1               ; Rotate left twice, to bring the first bit
        rlf     TRX_BUF,1               ; of the two bits to the MSB location
        call    TRANSMIT_BUF_DATA

        movlw   0x08                    ; Send lower byte of PS checkword
```

```
        movwf   ITERATIONS
        movf    CHCKWRD_3_LOW,0
        movwf   TRX_BUF
        call    TRANSMIT_BUF_DATA

        return

TRANSMIT_BUF_DATA                       ; Transmit the data byte held in the buffer
        rlf     TRX_BUF,1               ; Rotate left once, in itself
        btfss   STATUS, C               ; Test the carry flag value
        call    TRANSMIT_0              ; is carry 0? transmit a 0
        btfsc   STATUS, C
        call    TRANSMIT_1              ; Is carry 1? transmit a 1
        decfsz  ITERATIONS,1            ; Decrement iteration until it reaches 0
        goto    TRANSMIT_BUF_DATA
        return

TRANSMIT_0
        btfss   RDS_PORT,SIGNAL         ; Is 1187.5 high?
        goto    TRANSMIT_0              ; No? then loop until it goes high
        bcf     RDS_PORT, RDS_DATA      ; Clear the port bit, 0 will be modulated
        return

TRANSMIT_1
        btfss   RDS_PORT,SIGNAL         ; Is 1187.5 high?
        goto    TRANSMIT_1              ; No? then loop until it goes high
        bsf     RDS_PORT, RDS_DATA      ; Set the port bit, 1 will be modulated
        return
```

Even though use of intelligent LCD and microprocessor has the disadvantage of taking a considerable amount of time to develop skills in a new programming language and its abilities, the time saved on designing complex circuits for the analogue techniques and the extremely user-friendly interface means that the encoder would be compensate for the software development time. The unit is much easier to operate and even upgrade in the future if any other facilities such as radio text are considered to be added to the system.

5. Software development tools

The software side of the project is done using various simulation tools for the hardware parts of the encoder block and assembly programming language is used for the microcontroller part of the radio message source. This section entails a brief description of the software tools used for software and hardware development.

5.1. MPLAB IDE

The assembly codes for the PIC microcontrollers used within the project are developed using MPLAB Integrated Development Environment. Figure 20 shows a screen dump of the MPLAB IDE software.

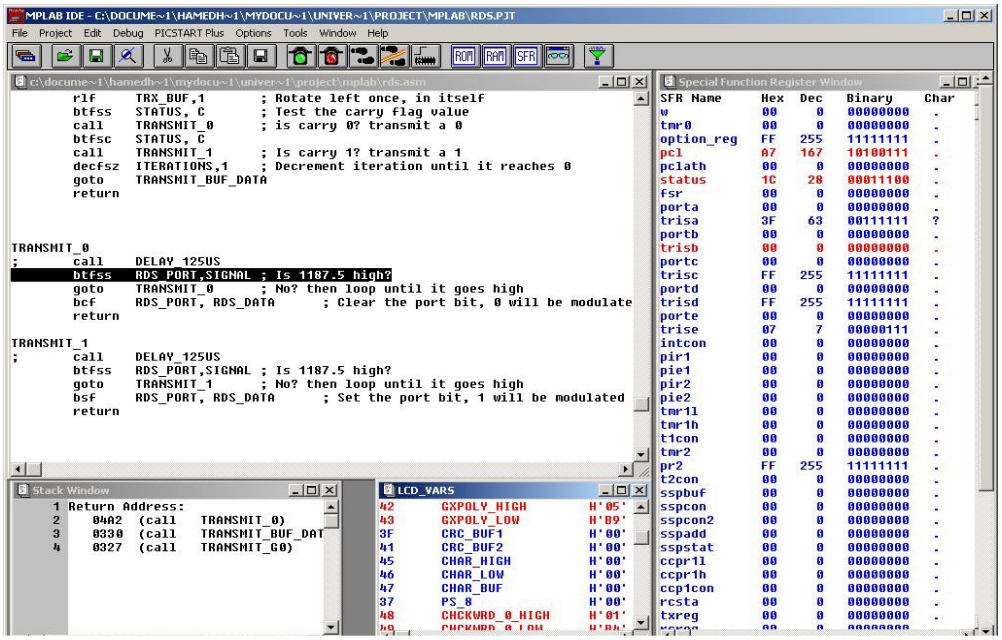


Figure 20: MPLAB IDE software

This software facilitates assembly code debugging, and it enables communication with the programmer PICstart PLUS, which loads the program .HEX file into the PIC. The software has been tested on the hardware circuit and it performs all the required action. The EEPROM data is also restored after power-up if requested by the user

The main objectives of the project in this part are achieved. An easy to operate user-interface is implemented and PS name is obtained from the user and the cyclic redundancy checks are performed. The user is also informed of the transmitted PS via the LCD interface and he will have the opportunity to restore the PS name in case the unit is turned off and then on for any reason.

5.2. Orcad PSpice

PSpice is the design software by Cadence Ltd allowing simulation and synthesis of circuit diagrams. All the circuit diagrams in this report have been designed using Orcad Design.

5.3. Proteus Virtual Circuit Modelling

Proteus VSM is a mixed signal simulation environment allowing an extensive range of microcontrollers and devices to be used within functional blocks. It also enables the use of oscilloscope and signal generators so simulation of various signals within an embedded system environment is possible using this software. Because of high license costs, an evaluation model was used with a low code limit and component count however it helped design of the PIC and LCD units. Figure 21 shows the interface of Protues VSM.

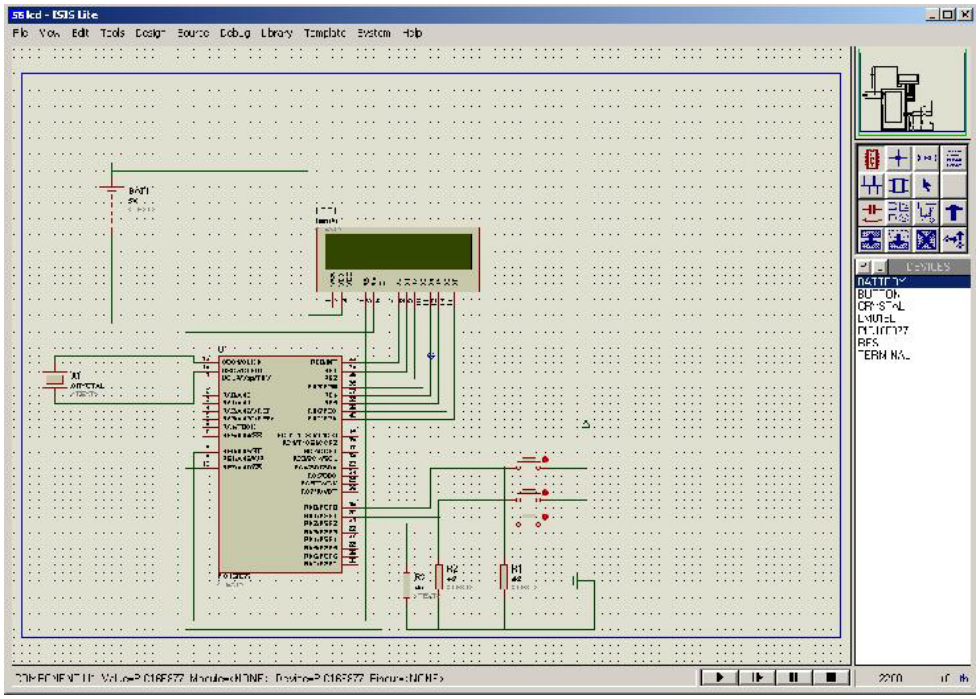


Figure 21: Proteus VSM software interface

5.4. Easy PC

Easy PC is the "Printed Circuit Board" design software provided by Number One Systems Ltd. It enables design of various PCBs and it has a simple user interface allowing control over drill holes and component spacing. This software was used to design the PCB for the radio message source block.

6. RDS encoder block circuits

Circuit blocks are all made according to RBDS standard [1] and using the block diagram of figure 2 on page 7. The circuit diagrams and where relevant the input and output are shown.

6.1. Radio Data Message source

The radio data message source as explained in section 4.4 is designed using PIC16F877 microcontroller and heavy duty push button switches. Pull-up resistors of 4k7 Ohms are used to limit the current on the PIC ports when switches are pushed down or when the 1187.5 signal goes high. An extra emergency battery holder, containing 3 AA batteries will be added to make sure that the module doesn't stop transmitting if disconnected for a short period of time. Figure 22 shows a picture of the data message source on breadboard and figure 23 shows the PCB design for this part of the encoder.

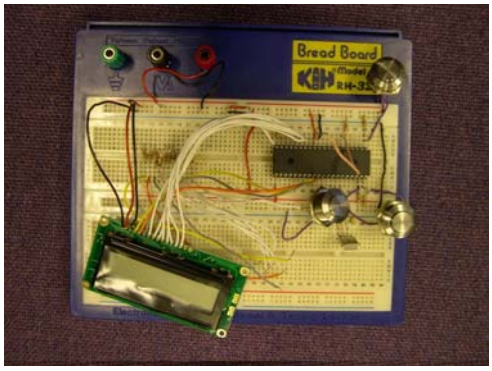


Figure 22: Data message source connected on breadboard

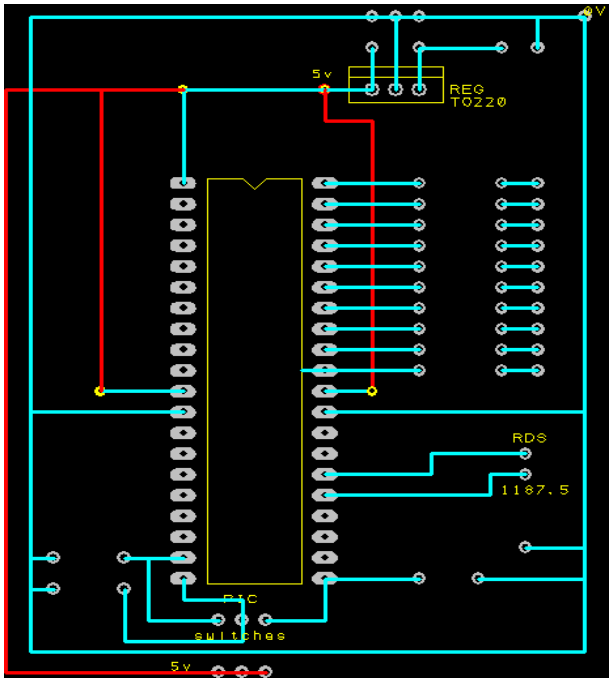


Figure 23: PCB design for the data message source block

6.2. Differential encoder

Data from the PIC port is passed to a differential encoder. This block is explained in the section 2.6, extracted from the RBDS [1] standard. The flip-flop used is a Philips 74HC74 D-type positive edge-triggered flip-flop. Exclusive-OR function was implemented using a Fairchild DM74LS86 2-input gate. Figure 24 shows the PSpice diagram of the differential encoder.

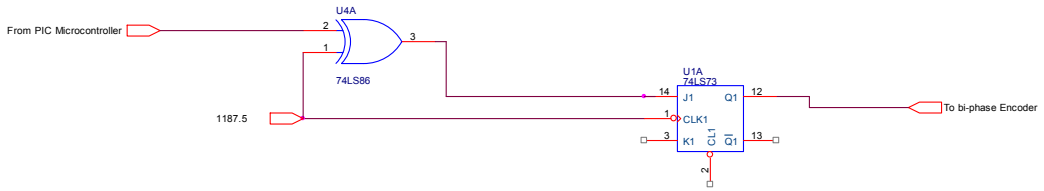


Figure 24: Differential encoder

6.3. Bi-phase symbol generator

The bi-phase symbol generator converts each bit to an odd pair of short pulses which are spaced one bit length. A "1" is converted to a +- pair, a "0" is converted to a -+ pair. The pulses are then passed through a 2nd Order Sallen-Key low-pass filter with a cosine-shaped transfer function. The combination of this filter and the inherent spectrum-shaping of the bi-phase scheme lead to a spectrum with a maximum near 1 kHz and zero amplitude at 0 and 2.4 kHz. Section 2.7 explains the theory behind bi-phase encoding. As delay and subtraction of short pulses is not possible in the flip-flops, another PIC microcontroller was used for generation and subtraction of data pulses, namely PIC16C622. The system is clocked with the 2375Hz clock generated from the divide-by-24 counter. The PIC starts of by polling for the 1187.5 clock. When this clock goes high, it also polls for the data line and output is decided based on table 5. In this case the previous input is zero as the input is only valid when the 1187.5 clock is high. The PIC generates the output pulse of short duration, about 51µs, by taking an output port high, then pulling it low after the delay. When a negative pulse is to be generated, one of the PIC ports goes high, but the output is connected to a fast inverting amplifier circuit, using MC33171 Op-amp with slew rate of 2V/µ. The next time that the 2375 signal goes high, the 1187.5 is low so the input data is considered to be zero. The output is then decided to be the opposite of the previous output, according to table 6.

Input N	**Previous Input (N-1)**	**N-(N-1) output**
+VE	0	**+VE**
-VE	0	**-VE**

Table 5: Output at 1187.5 data signal levels

Input N	**Previous Input (N-1)**	**N-(N-1) output**
0	+VE	**-VE**
0	-VE	**+VE**

Table 6: Output at 2375 zero pulse level

The following is an extract from the assembly code written for bi-phase symbol generation from the PIC.

```
poll11875
                btfss   PORTB,CLK11875
                goto    poll11875
                return
signal_pulse
                btfss   PORTB, RDS
                goto    PULSE_DOWN
                goto    PULSE_HIGH

poll2375
                btfss   PORTB, CLK2375
                goto    poll2375
                return

clear_pulse
                btfsc   MEMORY,0
                goto    PULSE_DOWN
                goto    PULSE_HIGH



PULSE_HIGH
                bsf             PORTB, PULSEHIGH
                call    DELAY_51US
                bcf             PORTB, PULSEHIGH
                bsf             MEMORY,0
                return

PULSE_DOWN
                bsf             PORTB, PULSEDOWN
                call    DELAY_51US
                bcf             PORTB, PULSEDOWN
                bcf             MEMORY,0
                return
```

Figure 25 shows the PSpice diagram of the PIC configuration for bi-phase symbol generation.

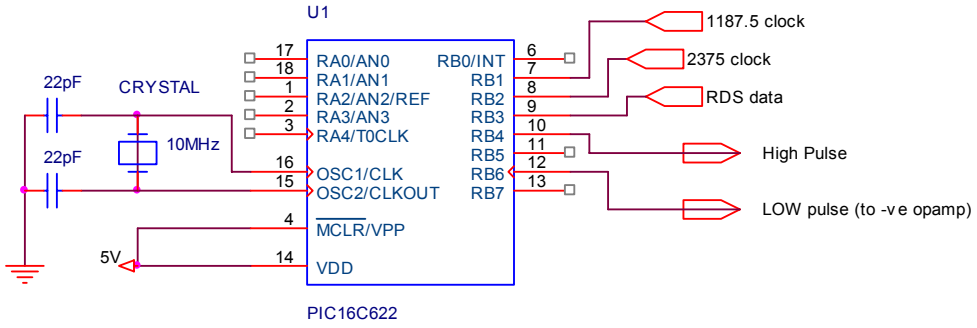


Figure 25: PIC16C622 for bi-phase symbol generation

Figure 26 shows the PSpice diagram of the inverting amplifier for negative pulse generation.

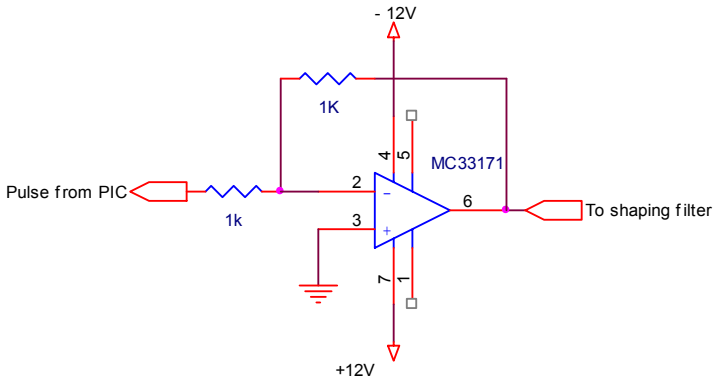


Figure 26: Inverting amplifier for negative pulses

6.4. Shaping filter

Data from the bi-phase symbol generator must go through a shaping filter as explained in section 2.7. This is a raised cosine filter with 100% roll-off and cut-off frequency of 1187.5Hz. A 2nd order Sallen-Key was devised for this purpose. The component values are calculated to give a cut-off at 1187.5Hz frequency, thus giving

$$C_1 = \frac{\sqrt{2}}{R\omega_0} \qquad \text{and} \quad C2 = \frac{1}{\sqrt{2}R\omega_0} \quad ,$$

where ω_0 is the cut-off frequency in radians. Using 10nf capacitors, R1 will be 18K9 Ohms and R2 will be 9k5 Ohms. Figure 27 shows a PSpice diagram of the shaping filter.

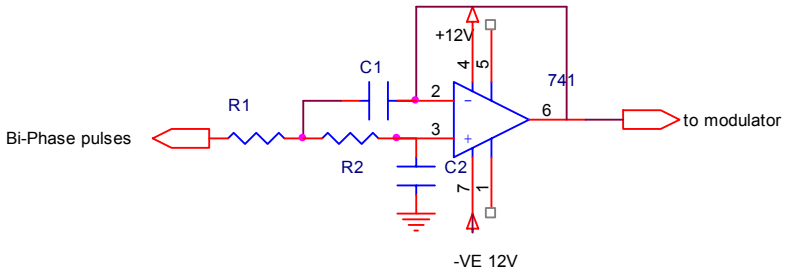


Figure 27: Shaping filter

6.5. 57 kHz signal generator

The RDS data has to be modulated with the FM signal and hence it needs to be clocked. The RDS carrier is at 57 kHz while the only signal available from the station stereo coder is the 19 kHz pilot tone. In order to generate a 57 kHz carrier in phase with the 19 kHz pilot tone, it is possible to use a filter to find the 3rd harmonic. This would need many active components and would require considerable phase compensation. As the 19 kHz pilot tone is a square wave, it can be fed into a Schmidt trigger circuit, then using a phase-locked-loop to generate a 57 kHz signal from the 3rd harmonic. A dedicated PLL chip, HC4046B by ST, is used and the voltage controlled oscillator's output is then fed into a divide-by-3 circuit.
Divide-by-3 part of the PLL is implemented using a pre-settable counter, HCC4018B, and 2 NAND-gates, on HCF4011B. Figure 28 shows the oscilloscope display of this division. It is not possible to get a 50% duty cycle from the circuit because of the configuration of the gates, but as PLL is edge-triggered, this causes no problems for the system.

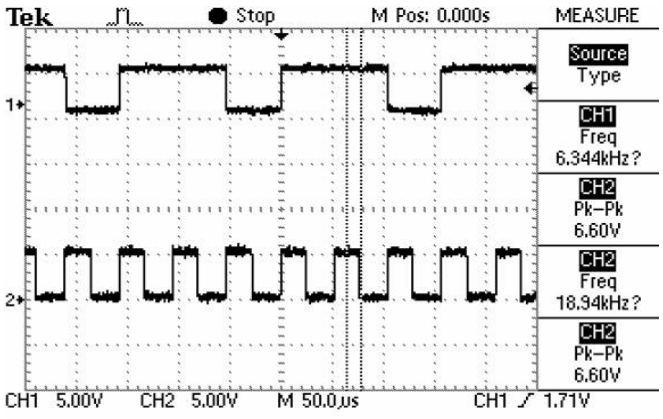


Figure 28: Divide-by-3 waveform

Using a few variable resistors at the VCO input and the R1 input on pin 11 it is possible to get a 57 kHz signal with less than 0.05% variation at times, which is due to variations in the 19 kHz signal input. Figure 29 shows a PSpice diagram of the complete PLL and divide-by-3 circuit.

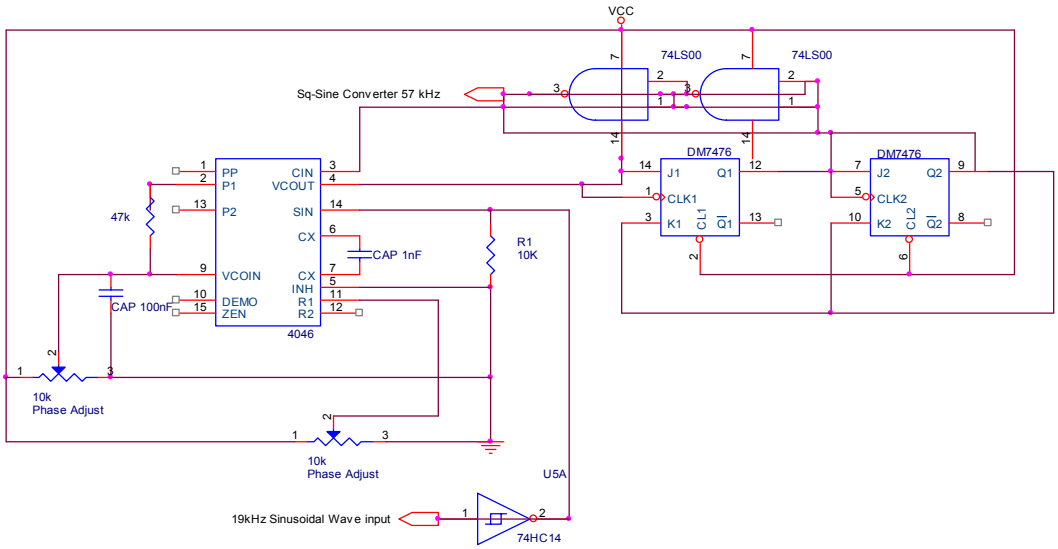


Figure 29: PLL and divide-by-3 circuit

Figure 30 shows the oscilloscope waveform of the 19 kHz input and the 57 kHz output of the PLL circuit.

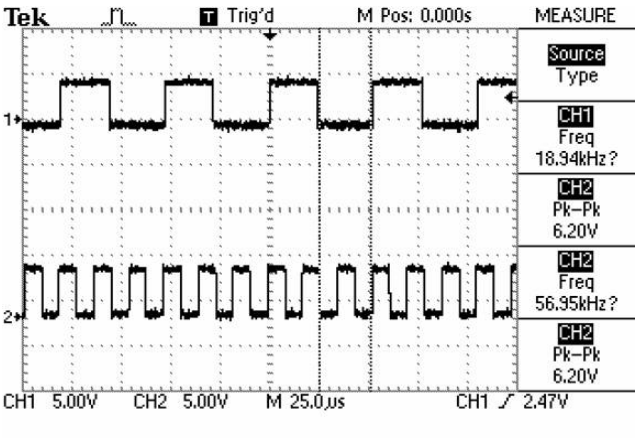


Figure 30: PLL waveform

6.6. Divide-by-24 counter

In order to generate the 2375 Hz signal for the bi-phase signal generator, the 57 kHz signal must be divided by 24. This is done in two stages using HCF4018B divide-by-N counters, dividing by 6 at the first stage, and dividing the output by 4 at the second stage. This signal is then used to clock the PIC for the bi-phase symbol generator. Divide-by-24 stage does not require any extra gates or components as the dividers allow straight division for even numbers. Figure 31 shows the PSpice diagram of the divide-by-24 circuit.

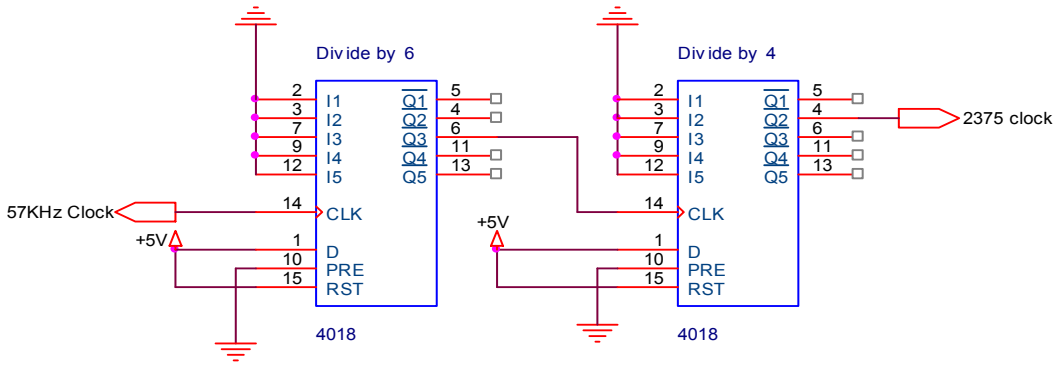


Figure 31: Divide-by-24 counter

Figure 32 shows the oscilloscope waveform of the divide-by-24 circuit. There is no phase shift introduces as result of this division.

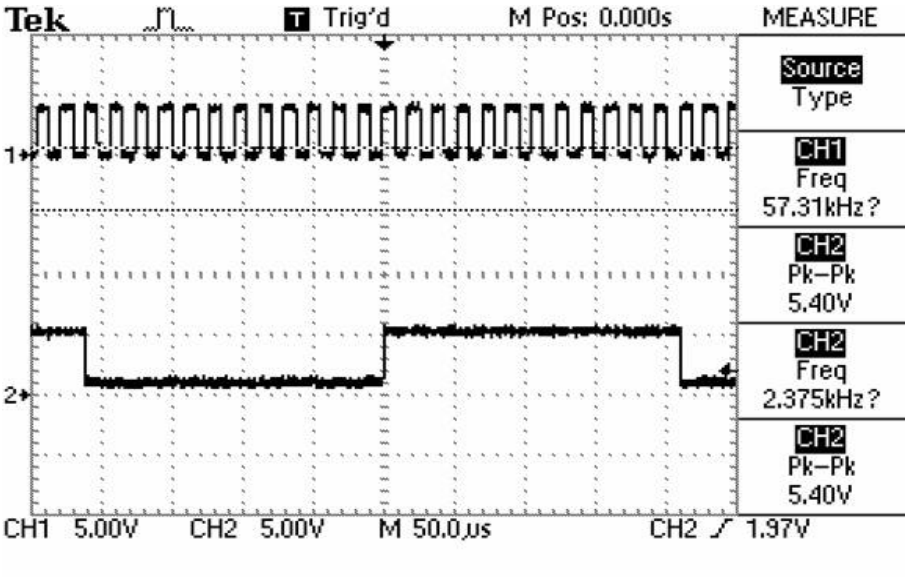


Figure 32: Divide-by-24 waveform

Figure 33 shows the circuit configuration for the PLL and divide-by-24 blocks.

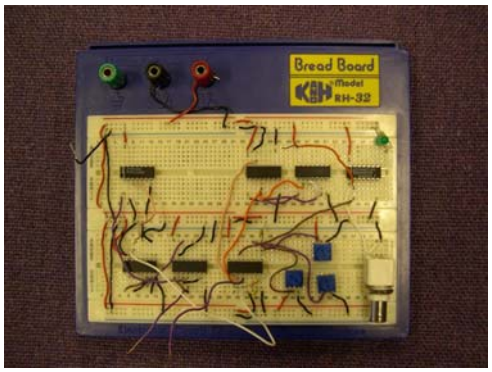


Figure 33: PLL and divide-by-24 circuits

6.7 Divide-by-2 counter

In order to generate the 1187.4 Hz clock for data output from the message source, the 2375 has to be divided by 2. This is done by using a simple DM74LS393N counter. Figure 34 shows the PSpice diagram of the circuit configuration.

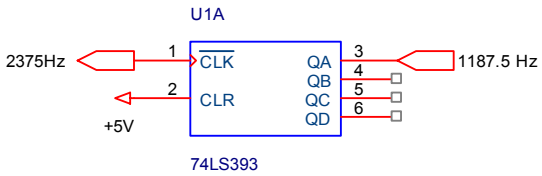


Figure 34: Divide-by-2 counter

Figure 35 displays the waveform output for the complete 57 kHz division to generate the 1187.5 Hz signal.

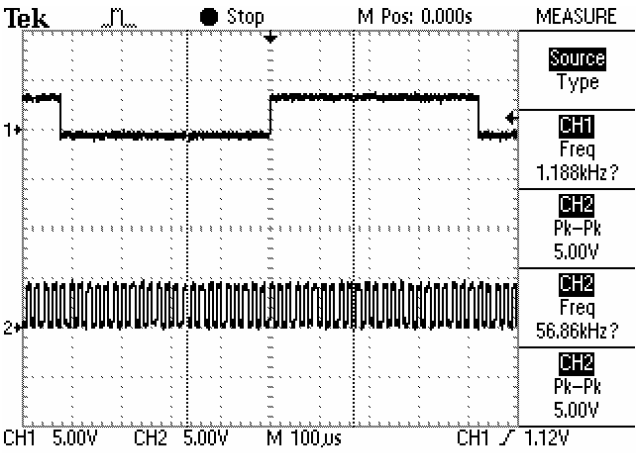


Figure 35: Divide-by-2 waveform

6.8. Modulation

The RDS bi-phase filtered symbols are modulated into the FM signal using double sideband suppressed carrier method. The carrier is generated using the 57 kHz clock generated by the PLL circuit. This is a square wave and it is converted to a sinusoidal wave for use by the modulator. This conversion is done by using an L-C tuned circuit with a centre frequency of 57 kHz. As the current drawn by the modulator is very little there is no need for transistor biasing. Figure 36 shows the PSpice diagram of the square-to-sinusoidal converter.

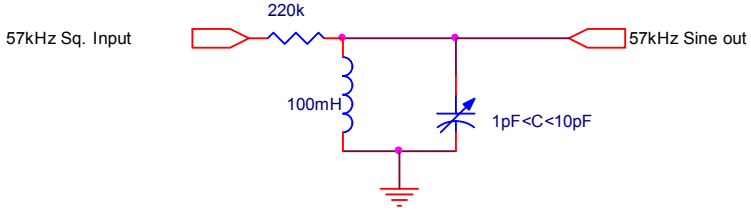


Figure 36: Square-to-sinusoidal wave converter

The output can be adjusted using the trimmer until the two waves or completely in-phase. Figure 37 shows the oscilloscope waveform output for the square-to sinusoidal waveform converter.

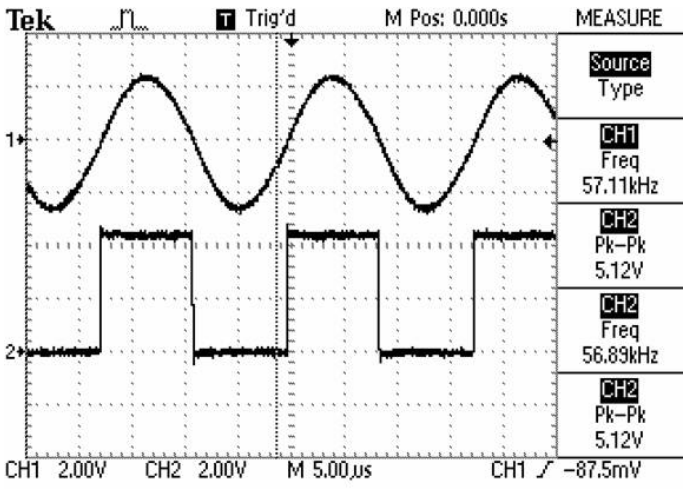


Figure 37: Square-to-sinusoidal waveform

Modulation of the data with the carrier is done using AD633 chip from Analog Devices. It is a low cost analogue multiplier which uses two external capacitors for double sideband suppressed carrier modulation. Figure 38 shows the circuit configuration of the modulator.

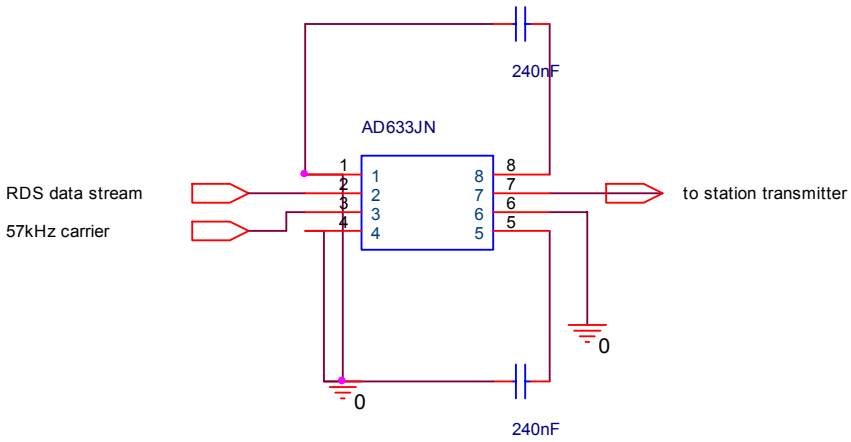


Figure 38: Modulator

Figure 39 displays the oscilloscope waveform of the modulator output, the input is a pure sine wave for demonstration purposes.

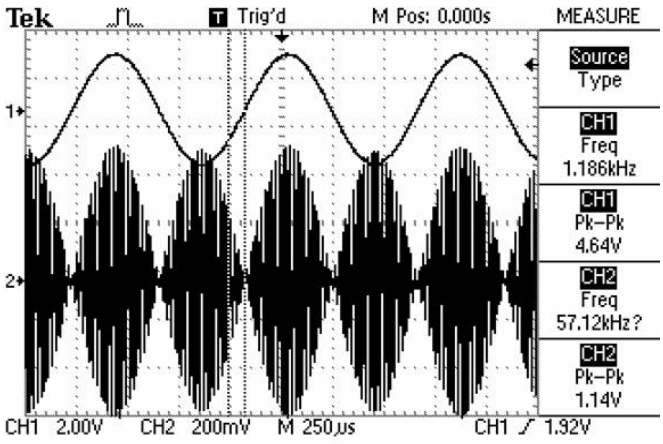


Figure 39: Modulator waveform

Figure 40 shows the complete modulator block and square-to sinusoidal converter circuit and the inverting regulator on breadboard.

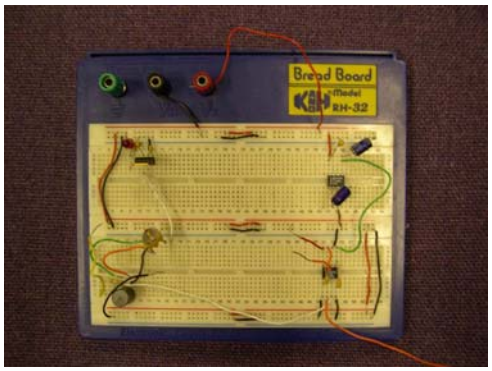


Figure 40: Modulator circuit

6.9 Power supply

Most of the components and circuits in the unit require 0V and +5V supplies. The modulator and inverting amplifier and filtering circuits operate in the range of -12V to +12V. Therefore a 12V, 400 mA adaptor is purchased to provide regulated DC power from mains. For the circuits requiring +5V supply, a positive voltage regulator is used to provide regulated DC voltages. Figure 41 shows the PSpice diagram of circuit configuration of the regulator.

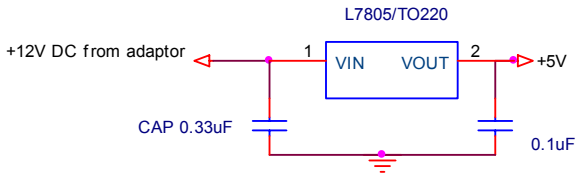


Figure 41: Positive voltage regulator

The -12V voltage is provided using LT1054 from Linear Technology, a switched capacitor voltage converter with regulator. It simply inverts the input voltage and uses two 100 µF capacitors. Figure 42 shows the PSpice diagram of the circuit configuration for the voltage inverter.

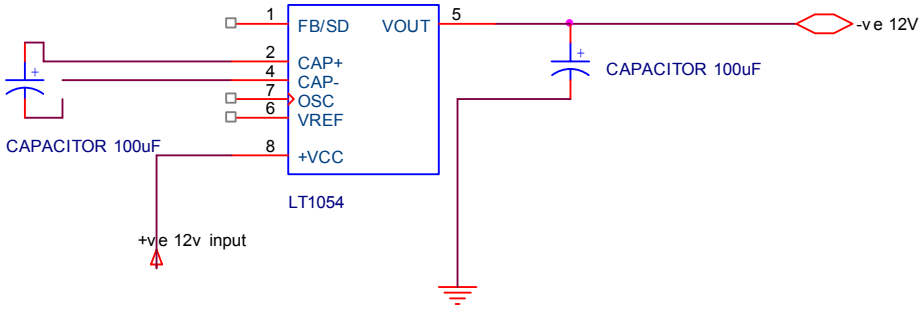


Figure 42: voltage inverter

7. Conclusions and future work

7.1 Objectives and achievements

The initial objectives of the project and achievements are discussed within this section.

Objective A) Provide RDS data stream for transmission.
This requirement has been fully met as the complete data message source unit has been designed and tested according to RBDS standard [1]. Valid data is output from the complete encoder circuit.

Objective B) Complete the real time control software.
This objective has been fully achieved as the control software completes the CRC error detections and then outputs the data at the right frequency when the 1187.5 signal goes high.

Objective C) Design and build a stand-alone RDS encoder circuit within budget
This objective is achieved based on the fact that all the functional modules and circuits are tested and synthesized. The PCB design for the data message source has been completed but the PCBs for the rest of the encoder circuits have not been completed yet. One of the obstacles in the design has been the lack of PCB auto-routing software that enables PCB design from schematic diagrams. Table 7 shows the bill for the complete list of components. It can be verified that the whole system has been built with a budget much less than 3rd year project budgets, £100. The only part that has not yet been purchased is a PCB rack for the system which will cost no more than £20.

Component	Quantity	Cost
Trimod LCD 16x2	1	£10.38
Push button switch	3	£14.04
PIC 16F877 MCU	1	£8.72
10 MHz crystal	2	£5.24
4018 Counter	3	£1.48
Power switch	1	£1.71
4046 PLL	1	£0.48
Emergency battery holder	1	£1.14
Inverting regulator	1	£4.06
Modulator	1	£0.95
PIC 16C622	1	£4.78
Regulator	2	£0.87
DC adaptor	1	£7.99
Misc. components	-	£5.00
Total	**£66.84**	

Table 7: Project component purchase price

7.2 Future work and improvements

The work to be done in the future includes designing PCBs for the rest of the module and mounting the components. Another task to be completed is extraction of the 19 kHz pilot-tone from the stereo-coder and feeding it into the system. The final task is to test the system on-air, when RARE FM launches officially in spring 2004.

8. Appendix

A. RDS.asm

```
;    Filename:     rds.asm                                                *
;    Date:          10th November 2002                                    *
;**********************************************************************
;
       list      p=16f877      ; list directive to define processor
       #include <p16f877.inc>  ; processor specific variable definitions

       __CONFIG _CP_OFF & _WDT_OFF & _BODEN_OFF & _PWRTE_ON & _XT_OSC & _WRT_ENABLE_ON
& _LVP_OFF & _DEBUG_OFF & _CPD_OFF


;***** VARIABLE DEFINITIONS

LCD_DATA       EQU     PORTB
LCD_CTRL       EQU     PORTE
SWITCH_PORT    EQU     PORTD   ; Switches
RDS_PORT       EQU     PORTC   ; The 1187.5Hz signal input & RDS output port


; PORT E, LCD control bits
RS             EQU     0       ; LCD Register-Select control line
RW             EQU     1       ; LCD Read/Write control line
E              EQU     2       ; LCD Enable control line

;PORT C, RDS data stream output
RDS_DATA       EQU     7       ; RDS data stream output, to be transmitted
SIGNAL         EQU     6       ; 1187.5 signal


LCD_BYTE       EQU     0x20    ; temporary register store for character byte to be sent
to lcd
CNT_DELAY1     EQU     0x21    ; temporary count register for 125 microsecond delay
routine
CNT_DELAY2     EQU     0x22    ; temporary count register 5 and 30 millisecond delay
routines
CNT_DELAY3     EQU     0x23    ; Temporary count for long delay
LCD_TEMP       EQU     0x24    ; Temporary register for LCD_BUSY function
CHAR_TEMP      EQU     0x25    ; Temporary storage for character comparisons

MS5            EQU     0x27    ; value to give a 5 millisecond delay in delay loop
MS30           EQU     0xe7    ; value to give a 30 millisecond delay in delay loop
US125          EQU     0x2a    ; value to give a 125 microsecond delay in delay loop
LCD_LINE1      EQU     0x00    ; Address of character 1, Line 1
LCD_LINE2      EQU     0x40    ; Address of character 1, line 2



PS_1           EQU     0x30    ; Registers to store the 8 character PS
PS_2           EQU     0x31
PS_3           EQU     0x32
PS_4           EQU     0x33
PS_5           EQU     0x34
PS_6           EQU     0x35
PS_7           EQU     0x36
PS_8           EQU     0x37

CHARACTER      EQU     0x38    ; Character to be displayed and changed on LCD
MAX_CHAR       EQU     0x39    ; The upper limit of the transmittable character range
MIN_CHAR       EQU     0x3a    ; The lower limit of the transmittable character range
DECISION       EQU     0x3b    ; The storage for user decision
POSITION       EQU     0x3c    ; Position of the character on the LCD


CRC_HIGH       EQU     0x3d    ; CRC calculation registers
CRC_LOW        EQU     0x3e
CRC_BUF1       EQU     0x3f
CRC_BUF2       EQU     0x41
GXPOLY_HIGH    EQU     0x42
GXPOLY_LOW     EQU     0x43
ITERATIONS     EQU     0x44
CHAR_HIGH      EQU     0x45
CHAR_LOW       EQU     0x46
CHAR_BUF       EQU     0x47

CHCKWRD_0_HIGH EQU     0x48    ; PS checkwords for 4 different groups
CHCKWRD_0_LOW  EQU     0x49
CHCKWRD_1_HIGH EQU     0x4a
CHCKWRD_1_LOW  EQU     0x4b
CHCKWRD_2_HIGH EQU     0x4c
CHCKWRD_2_LOW  EQU     0x4d
CHCKWRD_3_HIGH EQU     0x4e
```

```
CHCKWRD_3_LOW  EQU     0x4f

PI_LOW         EQU     0x50    ; PI words
PI_HIGH        EQU     0x51
PI_CHCK1_LOW   EQU     0x52    ; PI checkword with offset A for first block
PI_CHCK1_HIGH  EQU     0x53
PI_CHCK3_LOW   EQU     0x54    ; PI checkword with offset C' for third block
PI_CHCK3_HIGH  EQU     0x55

BLCK2_0HIGH    EQU     0x56    ; Block 2 data, first group
BLCK2_0LOW     EQU     0x57
BLCK2_0CHK_LOW EQU     0x58
BLCK2_0CHK_HIGH EQU    0x59

BLCK2_1HIGH    EQU     0x5a    ; Block 2 data, second group
BLCK2_1LOW     EQU     0x5b
BLCK2_1CHK_LOW EQU     0x5c
BLCK2_1CHK_HIGH EQU    0x5d

BLCK2_2HIGH    EQU     0x5e    ; Block 2 data, third group
BLCK2_2LOW     EQU     0x5f
BLCK2_2CHK_LOW EQU     0x60
BLCK2_2CHK_HIGH EQU    0x61

BLCK2_3HIGH    EQU     0x62    ; Block 2 data, fourth group
BLCK2_3LOW     EQU     0x63
BLCK2_3CHK_LOW EQU     0x64
BLCK2_3CHK_HIGH EQU    0x65

TRX_BUF        EQU     0x66    ; Data buffer, for bit-by-bit ouput


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

       org     0x00
RESET  goto    START


START


;      clrf    STATUS         ; Do initialization, Select bank 0
;      clrf    PORTE          ; ALL PORT output should output Low
;      clrf    PORTB          ; LCD data Port
;      clrf    PORTC          ; RDS signal IO port
;      clrf    PORTD          ; Switch port
;      bsf     STATUS, RP0    ; Select bank 1
;      clrf    TRISA          ; just to save power!
;      clrf    TRISE          ; PORT E output
;      clrf    TRISB          ; RB7-0 outputs
;      movlw   0x40           ; clear TRISC, apart from bit6 for 1187.5 input
;      movwf   TRISC
;      clrf    INTCON
;      bsf     OPTION_REG, NOT_RBPU
;                             ; disable pull-ups on port B
;      movlw   0xFF
;      movwf   ADCON1         ; Port E is digital
;      bcf     STATUS, RP0    ; Select bank 0
;
;      call    STORE_CONSTANT_DATA

;      call    INIT_LCD       ; Initialise the LCD

;      movlw   LCD_LINE1
;      call    SET_ADDR       ; LCD set on Line 1, character 1
;      call    GREETING       ; Greets the USER

;      call    LONG_DELAY

;      movlw   LCD_LINE2
       call    SET_ADDR       ; LCD set on Line 2, character 1
;      call    INTRODUCE

;      call    LONG_DELAY

;      call    LCD_CLEAR      ; clear display

;      movlw   LCD_LINE1
;      call    SET_ADDR       ; LCD set on line 1, character 1
;      call    DECIDE         ; Makes decision

;      call    LONG_DELAY

;      movlw   LCD_LINE2
;      call    SET_ADDR       ; LCD set on Line 2, character 1
;      call    QUESTION

;      call    CHECK_SWITCH   ; Check for answer
;      call    SWITCH_DELAY
```

```
;       call    LCD_CLEAR
;       movlw   LCD_LINE1
;       call    SET_ADDR        ; LCD cleared, line1
;       btfsc   DECISION,0      ; Skip to restore previous settings
;       call    GET_PS          ; Get and store the new PS
;       call    SWITCH_DELAY

;       call    RESTORE_PS      ; Restore PS from EEPROM


;       call    LCD_CLEAR
;       movlw   LCD_LINE1
;       call    SET_ADDR
;       call    CRC_BUSY        ; LCD displays that CRC is being calculated

;       call    LONG_DELAY

        call    CRC_CALC        ; calculate the CRC checkword values

        call    LCD_CLEAR
        movlw   LCD_LINE1
        call    SET_ADDR
        call    TRANSMIT_DISPLAY

        movlw   LCD_LINE2
        call    SET_ADDR
        call    DISPLAY_PS      ; Display the transmitted PS

        goto    TRANSMIT        ; Transmit the RDS groups


;*************************************************************************
; FUNCTION LIST
;*************************************************************************


STORE_CONSTANT_DATA             ; stores the values of constants for transmission

        movlw   0xc0            ; Move 11000000 to PI upper register
        movwf   PI_HIGH
        movlw   0x00            ; Move 00000000 to PI lower register
        movwf   PI_LOW
        movlw   0x00
        movwf   PI_CHCK1_HIGH   ; PI checkword with offset A for first block
        movlw   0xd5
        movwf   PI_CHCK1_LOW
        movlw   0x03
        movwf   PI_CHCK3_HIGH   ; PI checkword with offset C' for third block
        movlw   0x79
        movwf   PI_CHCK3_LOW


        movlw   0x08            ; Block2 data for the first group
        movwf   BLCK2_0HIGH
        movlw   0x08
        movwf   BLCK2_0LOW
        movlw   0x01
        movwf   BLCK2_0CHK_HIGH         ; Checkword with offset B
        movlw   0xc2
        movwf   BLCK2_0CHK_LOW


        movlw   0x08            ; Block2 data for the second group
        movwf   BLCK2_1HIGH
        movlw   0x09
        movwf   BLCK2_1LOW
        movlw   0x00            ; Checkword with offset B
        movwf   BLCK2_1CHK_HIGH
        movlw   0x6b
        movwf   BLCK2_1CHK_LOW


        movlw   0x08            ; Block2 data for the third group
        movwf   BLCK2_2HIGH
        movlw   0x0a
        movwf   BLCK2_2LOW
        movlw   0x02            ; Checkword with offset B
        movwf   BLCK2_2CHK_HIGH
        movlw   0xb0
        movwf   BLCK2_2CHK_LOW


        movlw   0x08            ; Block2 data for the fourth group
        movwf   BLCK2_3HIGH
        movlw   0x0f
        movwf   BLCK2_3LOW
        movlw   0x00            ; Checkword with offset B
        movwf   BLCK2_3CHK_HIGH
```

```
        movlw   0x58
        movwf   BLCK2_3CHK_LOW


        return

;*********************************************************************
; INIT_LCD..... Module to initialise the display
;*********************************************************************
INIT_LCD
        call    DELAY_30MS

        movlw   0x38            ; 8-bit-interface, 2-lines... 0b00111000
        call    LCD_CMND
        call    DELAY_5MS

        movlw   0x38            ; 8-bit-interface, 2-lines... 0b00111000
        call    LCD_CMND
        call    DELAY_125US

        movlw   0x38            ; 8-bit-interface, 2-lines... 0b00111000
        call    LCD_CMND
        call    DELAY_125US

        movlw   0x0F            ; display On, curser On, blink On 0b00001111
        call    LCD_CMND
        call    DELAY_5MS

        movlw    0x01           ; display clear, 0b00000001
        call    LCD_CMND

        return

GREETING
        movlw   'U'
        call    LCD_CHAR
        movlw   'C'
        call    LCD_CHAR
        movlw   'L'
        call    LCD_CHAR
        movlw   ' '
        call    LCD_CHAR
        movlw   ' '
        call    LCD_CHAR
        movlw   'R'
        call    LCD_CHAR
        movlw   'D'
        call    LCD_CHAR
        movlw   'S'
        call    LCD_CHAR
        movlw   ' '
        call    LCD_CHAR
        movlw   'E'
        call    LCD_CHAR
        movlw   'N'
        call    LCD_CHAR
        movlw   'C'
        call    LCD_CHAR
        movlw   'O'
        call    LCD_CHAR
        movlw   'D'
        call    LCD_CHAR
        movlw   'E'
        call    LCD_CHAR
        movlw   'R'
        call    LCD_CHAR
        return

INTRODUCE
        movlw   'B'
        call    LCD_CHAR
        movlw   'Y'
        call    LCD_CHAR
        movlw   ' '
        call    LCD_CHAR
        movlw   'H'
        call    LCD_CHAR
        movlw   'A'
        call    LCD_CHAR
        movlw   'M'
        call    LCD_CHAR
        movlw   'E'
        call    LCD_CHAR
        movlw   'D'
        call    LCD_CHAR
        movlw   ' '
        call    LCD_CHAR
        movlw   'H'
        call    LCD_CHAR
```

```
        movlw   'A'
        call    LCD_CHAR
        movlw   'D'
        call    LCD_CHAR
        movlw   'D'
        call    LCD_CHAR
        movlw   'A'
        call    LCD_CHAR
        movlw   'D'
        call    LCD_CHAR
        movlw   'I'
        call    LCD_CHAR
        return

DECIDE
        movlw   'R'
        call    LCD_CHAR
        movlw   'E'
        call    LCD_CHAR
        movlw   'S'
        call    LCD_CHAR
        movlw   'T'
        call    LCD_CHAR
        movlw   'O'
        call    LCD_CHAR
        movlw   'R'
        call    LCD_CHAR
        movlw   'E'
        call    LCD_CHAR
        movlw   ' '
        call    LCD_CHAR
        movlw   'S'
        call    LCD_CHAR
        movlw   'E'
        call    LCD_CHAR
        movlw   'T'
        call    LCD_CHAR
        movlw   'T'
        call    LCD_CHAR
        movlw   'I'
        call    LCD_CHAR
        movlw   'N'
        call    LCD_CHAR
        movlw   'G'
        call    LCD_CHAR
        movlw   'S'
        call    LCD_CHAR
        return

QUESTION
        movlw   'Y'
        call    LCD_CHAR
        movlw   'E'
        call    LCD_CHAR
        movlw   'S'
        call    LCD_CHAR
        movlw   '/'
        call    LCD_CHAR
        movlw   'N'
        call    LCD_CHAR
        movlw   'O'
        call    LCD_CHAR
        movlw   '?'
        call    LCD_CHAR
        return


CHECK_SWITCH                    ; Polls switches for decision to restore setting
GET_IT  btfsc    SWITCH_PORT,0
        call     D0_SET
        btfsc    SWITCH_PORT,1
        call     D1_SET
got     btfss   SWITCH_PORT,2
        goto    GET_IT
        movf    DECISION,W      ; Check if D2 is not pressed
        andlw   0xff            ; without making a  decision
        btfsc   STATUS,Z
        goto    GET_IT          ; if yes, loop until decision is made
        return


D0_SET
        call    SWITCH_DELAY
        movlw   0x48
        call    SET_ADDR        ; LCD set on Line 2, character 9
        movlw   'N'
        call    LCD_CHAR
        movlw   'O'
        call    LCD_CHAR
```

```
        movlw   ' '
        call    LCD_CHAR
        movlw   0x1
        movwf   DECISION
        return

D1_SET
        call    SWITCH_DELAY
        movlw   0x48
        call    SET_ADDR        ; LCD set on Line 2, character 9
        movlw   'Y'
        call    LCD_CHAR
        movlw   'E'
        call    LCD_CHAR
        movlw   'S'
        call    LCD_CHAR
        movlw   0x2
        movwf   DECISION
        return

PS_REQUEST
        movlw   'E'
        call    LCD_CHAR
        movlw   'N'
        call    LCD_CHAR
        movlw   'T'
        call    LCD_CHAR
        movlw   'E'
        call    LCD_CHAR
        movlw   'R'
        call    LCD_CHAR
        movlw   ' '
        call    LCD_CHAR
        movlw   'P'
        call    LCD_CHAR
        movlw   'S'
        call    LCD_CHAR
        movlw   ':'
        return

GET_PS
        call    PS_REQUEST      ; ask for PS to be entered
        movlw   ' '
        movwf   MIN_CHAR        ; set the lower boundary
        movlw   'Z'
        movwf   MAX_CHAR        ; set the upper boundary

        movlw   0x40            ; Set position, line 2 character 0
        movwf   POSITION
        call    SET_ADDR
        movlw   'A'
        movwf   CHARACTER
        call    LCD_CHAR
        call    GET_PS1
        movwf   PS_1            ; store in variable

        bsf     STATUS,RP1      ; Store in EEPROM
        bsf     STATUS,RP0
        btfsc   EECON1,WR       ; Make sure there is no other WRITE in progress
        call    DELAY_5MS
        bcf     STATUS,RP0      ; Write data and destination
        movwf   EEDATA
        movlw   0x00
        movwf   EEADR
        bsf     STATUS,RP0
        bcf     EECON1,EEPGD
        bsf     EECON1,WREN
        movlw   0x55
        movwf   EECON2
        movlw   0xaa
        movwf   EECON2
        bsf     EECON1,WR
        bcf     EECON1,WREN
        clrf    STATUS          ; Select bank 0

        call    SWITCH_DELAY

        movlw   0x41            ; Set position, line 2 character 1
        movwf   POSITION
        movf    CHARACTER,0
        call    LCD_CHAR
        call    GET_PS2
        movwf   PS_2            ; store in variable
        bsf     STATUS,RP1      ; Store in EEPROM
        bsf     STATUS,RP0
        btfsc   EECON1,WR       ; Make sure there is no other WRITE in progress
        call    DELAY_5MS
        bcf     STATUS,RP0
        movwf   EEDATA
```

```
movlw   0x01
movwf   EEADR
bsf     STATUS,RP0
bcf     EECON1,EEPGD
bsf     EECON1,WREN
movlw   0x55
movwf   EECON2
movlw   0xaa
movwf   EECON2
bsf     EECON1,WR
bcf     EECON1,WREN
clrf    STATUS          ; Select bank 0

call    SWITCH_DELAY

movlw   0x42            ; Set position, line 2 character 2
movwf   POSITION
movf    CHARACTER,0
call    LCD_CHAR
call    GET_PS3
movwf   PS_3            ; store in variable
bsf     STATUS,RP1      ; Store in EEPROM
bsf     STATUS,RP0
btfsc   EECON1,WR       ; Make sure there is no other WRITE in progress
call    DELAY_5MS
bcf     STATUS,RP0
movwf   EEDATA
movlw   0x02
movwf   EEADR
bsf     STATUS,RP0
bcf     EECON1,EEPGD
bsf     EECON1,WREN
movlw   0x55
movwf   EECON2
movlw   0xaa
movwf   EECON2
bsf     EECON1,WR
bcf     EECON1,WREN
clrf    STATUS          ; Select bank 0

call    SWITCH_DELAY

movlw   0x43            ; Set position, line 2 character 3
movwf   POSITION
movf    CHARACTER,0
call    LCD_CHAR
call    GET_PS4
movwf   PS_4            ; store in variable
bsf     STATUS,RP1      ; Store in EEPROM
bsf     STATUS,RP0
btfsc   EECON1,WR       ; Make sure there is no other WRITE in progress
call    DELAY_5MS
bcf     STATUS,RP0
movwf   EEDATA
movlw   0x03
movwf   EEADR
bsf     STATUS,RP0
bcf     EECON1,EEPGD
bsf     EECON1,WREN
movlw   0x55
movwf   EECON2
movlw   0xaa
movwf   EECON2
bsf     EECON1,WR
bcf     EECON1,WREN
clrf    STATUS          ; Select bank 0

call    SWITCH_DELAY

movlw   0x44            ; Set position, line 2 character 4
movwf   POSITION
movf    CHARACTER,0
call    LCD_CHAR
call    GET_PS5
movwf   PS_5            ; store in variable
bsf     STATUS,RP1      ; Store in EEPROM
bsf     STATUS,RP0
btfsc   EECON1,WR       ; Make sure there is no other WRITE in progress
call    DELAY_5MS
bcf     STATUS,RP0
movwf   EEDATA
movlw   0x04
movwf   EEADR
bsf     STATUS,RP0
bcf     EECON1,EEPGD
bsf     EECON1,WREN
movlw   0x55
movwf   EECON2
movlw   0xaa
```

```
movwf   EECON2
bsf     EECON1,WR
bcf     EECON1,WREN
clrf    STATUS          ; Select bank 0

call    SWITCH_DELAY

movlw   0x45            ; Set position, line 2 character 5
movwf   POSITION
movf    CHARACTER,0
call    LCD_CHAR
call    GET_PS6
movwf   PS_6            ; store in variable
bsf     STATUS,RP1      ; Store in EEPROM
bsf     STATUS,RP0
btfsc   EECON1,WR       ; Make sure there is no other WRITE in progress
call    DELAY_5MS
bcf     STATUS,RP0
movwf   EEDATA
movlw   0x05
movwf   EEADR
bsf     STATUS,RP0
bcf     EECON1,EEPGD
bsf     EECON1,WREN
movlw   0x55
movwf   EECON2
movlw   0xaa
movwf   EECON2
bsf     EECON1,WR
bcf     EECON1,WREN
clrf    STATUS          ; Select bank 0

call    SWITCH_DELAY

movlw   0x46            ; Set position, line 2 character 6
movwf   POSITION
movf    CHARACTER,0
call    LCD_CHAR
call    GET_PS7
movwf   PS_7            ; store in variable
bsf     STATUS,RP1      ; Store in EEPROM
bsf     STATUS,RP0
btfsc   EECON1,WR       ; Make sure there is no other WRITE in progress
call    DELAY_5MS
bcf     STATUS,RP0
movwf   EEDATA
movlw   0x06
movwf   EEADR
bsf     STATUS,RP0
bcf     EECON1,EEPGD
bsf     EECON1,WREN
movlw   0x55
movwf   EECON2
movlw   0xaa
movwf   EECON2
bsf     EECON1,WR
bcf     EECON1,WREN
clrf    STATUS          ; Select bank 0

call    SWITCH_DELAY

movlw   0x47            ; Set position, line 2 character 7
movwf   POSITION
movf    CHARACTER,0
call    LCD_CHAR
call    GET_PS8
movwf   PS_8            ; store in variable
bsf     STATUS,RP1      ; Store in EEPROM
bsf     STATUS,RP0
btfsc   EECON1,WR       ; Make sure there is no other WRITE in progress
call    DELAY_5MS
bcf     STATUS,RP0
movwf   EEDATA
movlw   0x07
movwf   EEADR
bsf     STATUS,RP0
bcf     EECON1,EEPGD
bsf     EECON1,WREN
movlw   0x55
movwf   EECON2
movlw   0xaa
movwf   EECON2
bsf     EECON1,WR
bcf     EECON1,WREN
clrf    STATUS          ; Select bank 0


return
```

```
GET_PS1 btfsc   SWITCH_PORT,0  ; Increment Character if Switch 0 pressed
        call    CHAR_UP
        btfsc   SWITCH_PORT,1  ; Decrement Character if Switch 1 pressed
        call    CHAR_DOWN
        btfss   SWITCH_PORT,2
        goto    GET_PS1
        return

GET_PS2 btfsc   SWITCH_PORT,0
        call    CHAR_UP
        btfsc   SWITCH_PORT,1
        call    CHAR_DOWN
        btfss   SWITCH_PORT,2
        goto    GET_PS2
        return

GET_PS3 btfsc   SWITCH_PORT,0
        call    CHAR_UP
        btfsc   SWITCH_PORT,1
        call    CHAR_DOWN
        btfss   SWITCH_PORT,2
        goto    GET_PS3
        return

GET_PS4 btfsc   SWITCH_PORT,0
        call    CHAR_UP
        btfsc   SWITCH_PORT,1
        call    CHAR_DOWN
        btfss   SWITCH_PORT,2
        goto    GET_PS4
        return

GET_PS5 btfsc   SWITCH_PORT,0
        call    CHAR_UP
        btfsc   SWITCH_PORT,1
        call    CHAR_DOWN
        btfss   SWITCH_PORT,2
        goto    GET_PS5
        return

GET_PS6 btfsc   SWITCH_PORT,0
        call    CHAR_UP
        btfsc   SWITCH_PORT,1
        call    CHAR_DOWN
        btfss   SWITCH_PORT,2
        goto    GET_PS6
        return

GET_PS7 btfsc   SWITCH_PORT,0
        call    CHAR_UP
        btfsc   SWITCH_PORT,1
        call    CHAR_DOWN
        btfss   SWITCH_PORT,2
        goto    GET_PS7
        return

GET_PS8 btfsc   SWITCH_PORT,0
        call    CHAR_UP
        btfsc   SWITCH_PORT,1
        call    CHAR_DOWN
        btfss   SWITCH_PORT,2
        goto    GET_PS8
        return

CHAR_UP
        call    SWITCH_DELAY
        movf    POSITION,0     ; set display position
        call    SET_ADDR
        movf    CHARACTER,0    ; move character to W register
        call    check_max      ; check if reached the upper boundary
        movf    CHARACTER,0
        call    LCD_CHAR
        return

check_max                      ; it checks mor the upper character
        movwf   CHAR_TEMP      ; store the character locally
        movf    MAX_CHAR,0     ; now move MAX_CHAR to  W register
        subwf   CHAR_TEMP,0    ; subtract the two files
        btfsc   STATUS,Z       ; Is the result zero?? then we reached maximum
        goto    SET2MAX        ; keep the character as it is
        goto    INCREMENT      ; if not reached the max, increment the character

SET2MAX
        movf    MAX_CHAR,0     ; Keep MAX_CHAR as the character of chioce
        return

INCREMENT
```

```
        incf    CHARACTER,1     ; increment the character
        return

CHAR_DOWN
        call    SWITCH_DELAY
        movf    POSITION,0      ; set display position
        call    SET_ADDR
        movf    CHARACTER,0     ; move character to W register
        call    check_min       ; check if reached the lower boundary
        movf    CHARACTER,0
        call    LCD_CHAR
        return

check_min                       ; it checks mor the lower character
        movwf   CHAR_TEMP       ; store the character locally
        movf    MIN_CHAR,0      ; now move MIN_CHAR to  W register
        subwf   CHAR_TEMP,0     ; subtract the two files
        btfsc   STATUS,Z        ; Is the result zero?? then we reached minimum
        goto    SET2MIN         ; keep the character as it is
        goto    DECREMENT       ; if not reached the min, decrement the character

SET2MIN
        movf    MIN_CHAR,0      ; Keep MIN_CHAR as the character of chioce
        return


DECREMENT
        decf    CHARACTER,1     ; decrement the character
        return


RESTORE_PS                      ; Restore the PS characters from EEPROM

        bsf     STATUS,RP1
        bcf     STATUS,RP0
        movlw   0x00
        movwf   EEADR
        bsf     STATUS,RP0
        bcf     EECON1, EEPGD
        bsf     EECON1, RD
        bcf     STATUS,RP0
        movf    EEDATA,0
        clrf    STATUS
        movwf   PS_1
        call    SWITCH_DELAY



        bsf     STATUS,RP1
        bcf     STATUS,RP0
        movlw   0x01
        movwf   EEADR
        bsf     STATUS,RP0
        bcf     EECON1, EEPGD
        bsf     EECON1, RD
        bcf     STATUS,RP0
        movf    EEDATA,0
        clrf    STATUS          ; Select bank 0
        movwf   PS_2
        call    SWITCH_DELAY

        bsf     STATUS,RP1
        bcf     STATUS,RP0
        movlw   0x02
        movwf   EEADR
        bsf     STATUS,RP0
        bcf     EECON1, EEPGD
        bsf     EECON1, RD
        bcf     STATUS,RP0
        movf    EEDATA,0
        clrf    STATUS          ; Select bank 0
        movwf   PS_3
        call    SWITCH_DELAY


        bsf     STATUS,RP1
        bcf     STATUS,RP0
        movlw   0x03
        movwf   EEADR
        bsf     STATUS,RP0
        bcf     EECON1, EEPGD
        bsf     EECON1, RD
        bcf     STATUS,RP0
        movf    EEDATA,0
        clrf    STATUS          ; Select bank 0
        movwf   PS_4
        call    SWITCH_DELAY
```

```
        bsf     STATUS,RP1
        bcf     STATUS,RP0
        movlw   0x04
        movwf   EEADR
        bsf     STATUS,RP0
        bcf     EECON1, EEPGD
        bsf     EECON1, RD
        bcf     STATUS,RP0
        movf    EEDATA,0
        clrf    STATUS          ; Select bank 0
        movwf   PS_5
        call    SWITCH_DELAY


        bsf     STATUS,RP1
        bcf     STATUS,RP0
        movlw   0x05
        movwf   EEADR
        bsf     STATUS,RP0
        bcf     EECON1, EEPGD
        bsf     EECON1, RD
        bcf     STATUS,RP0
        movf    EEDATA,0
        clrf    STATUS          ; Select bank 0
        movwf   PS_6
        call    SWITCH_DELAY


        bsf     STATUS,RP1
        bcf     STATUS,RP0
        movlw   0x06
        movwf   EEADR
        bsf     STATUS,RP0
        bcf     EECON1, EEPGD
        bsf     EECON1, RD
        bcf     STATUS,RP0
        movf    EEDATA,0
        clrf    STATUS          ; Select bank 0
        movwf   PS_7
        call    SWITCH_DELAY


        bsf     STATUS,RP1
        bcf     STATUS,RP0
        movlw   0x07
        movwf   EEADR
        bsf     STATUS,RP0
        bcf     EECON1, EEPGD
        bsf     EECON1, RD
        bcf     STATUS,RP0
        movf    EEDATA,0
        clrf    STATUS          ; Select bank 0
        movwf   PS_8
        call    SWITCH_DELAY


        return

CRC_BUSY
        movlw   'C'
        call    LCD_CHAR
        movlw   'R'
        call    LCD_CHAR
        movlw   'C'
        call    LCD_CHAR
        movlw   ' '
        call    LCD_CHAR
        movlw   'C'
        call    LCD_CHAR
        movlw   'A'
        call    LCD_CHAR
        movlw   'L'
        call    LCD_CHAR
        movlw   'C'
        call    LCD_CHAR
        movlw   'U'
        call    LCD_CHAR
        movlw   'L'
        call    LCD_CHAR
        movlw   'A'
        call    LCD_CHAR
        movlw   'T'
        call    LCD_CHAR
        movlw   'I'
        call    LCD_CHAR
        movlw   'O'
        call    LCD_CHAR
        movlw   'N'
```

```
        call    LCD_CHAR
        movlw   '.'
        call    LCD_CHAR
        return


CRC_CALC                        ; CheckWord calculation for the PS
        call    CRC_0           ; PS_1 and PS_2, for first transmission group
        call    CRC_1           ; PS_3 and PS_4, for second transmission group
        call    CRC_2           ; PS_5 and PS_6, for third transmission group
        call    CRC_3           ; PS_7 and PS_8, for fourth transmission group
        return

CRC_0                           ; Calculate the checkword for the first PS block
        movf    PS_1,0
        movwf   CHAR_HIGH       ; move PS_1 to the upper working register
        movf    PS_2,0
        movwf   CHAR_LOW        ; move PS_2 to the lower working register

        call    CRC_INIT        ; Initialise the registers
        call    CRC_GEN         ; Calculate CRC

        movf    CRC_HIGH,0      ; Load W with CRC upper part
        xorlw   0x01            ; Add upper part of offset D
        movwf   CHCKWRD_0_HIGH  ; Store in Checkword upper part
        movf    CRC_LOW,0       ; Load W with CRC lower part
        xorlw   0xb4            ; Add lower part of offset D
        movwf   CHCKWRD_0_LOW   ; Store in Checkword lower part
        return

CRC_1                           ; Calculate the checkword for the second PS block
        movf    PS_3,0
        movwf   CHAR_HIGH       ; move PS_3 to the upper working register
        movf    PS_4,0
        movwf   CHAR_LOW        ; move PS_4 to the lower working register

        call    CRC_INIT        ; Initialise the registers
        call    CRC_GEN         ; Calculate CRC

        movf    CRC_HIGH,0      ; Load W with CRC upper part
        xorlw   0x01            ; Add upper part of offset D
        movwf   CHCKWRD_1_HIGH  ; Store in Checkword upper part
        movf    CRC_LOW,0       ; Load W with CRC lower part
        xorlw   0xb4            ; Add lower part of offset D
        movwf   CHCKWRD_1_LOW   ; Store in Checkword lower part
        return

CRC_2                           ; Calculate the checkword for the third PS block
        movf    PS_5,0
        movwf   CHAR_HIGH       ; move PS_5 to the upper working register
        movf    PS_6,0
        movwf   CHAR_LOW        ; move PS_6 to the lower working register

        call    CRC_INIT        ; Initialise the registers
        call    CRC_GEN         ; Calculate CRC

        movf    CRC_HIGH,0      ; Load W with CRC upper part
        xorlw   0x01            ; Add upper part of offset D
        movwf   CHCKWRD_2_HIGH  ; Store in Checkword upper part
        movf    CRC_LOW,0       ; Load W with CRC lower part
        xorlw   0xb4            ; Add lower part of offset D
        movwf   CHCKWRD_2_LOW   ; Store in Checkword lower part
        return

CRC_3                           ; Calculate the checkword for the fourth PS block
        movf    PS_7,0
        movwf   CHAR_HIGH       ; move PS_7 to the upper working register
        movf    PS_8,0
        movwf   CHAR_LOW        ; move PS_8 to the lower working register

        call    CRC_INIT        ; Initialise the registers
        call    CRC_GEN         ; Calculate CRC

        movf    CRC_HIGH,0      ; Load W with CRC upper part
        xorlw   0x01            ; Add upper part of offset D
        movwf   CHCKWRD_3_HIGH  ; Store in Checkword upper part
        movf    CRC_LOW,0       ; Load W with CRC lower part
        xorlw   0xb4            ; Add lower part of offset D
        movwf   CHCKWRD_3_LOW   ; Store in Checkword lower part
        return


CRC_INIT                        ; Initialise the registers
        clrf    CHAR_BUF        ; rotate left twice, through carry,
        bcf     STATUS,C        ; so that the 10^X factor can correctly
        rlf     CHAR_LOW,1      ; be implemented, i.e. by adding an 8bit
        rlf     CHAR_HIGH,1     ; all 0s register to the end of the data,
        rlf     CHAR_BUF,1      ; and shifting to left twice to have another
        bcf     STATUS,C        ; 2 obits.
```

```
        rlf     CHAR_LOW,1
        rlf     CHAR_HIGH,1
        rlf     CHAR_BUF,1


        movlw   0x10            ; 16 left shifts, for the CRC calculation
        movwf   ITERATIONS
        movf    CHAR_BUF,0      ; shifting all to the right place
        movwf   CRC_HIGH
        movf    CHAR_HIGH,0
        movwf   CRC_LOW
        movf    CHAR_LOW,0
        movwf   CRC_BUF1
        movlw   0x00            ; adding the last 8 zero's
        movwf   CRC_BUF2

        movlw   0x05            ; Storing the g(x) polynomial in registers
        movwf   GXPOLY_HIGH
        movlw   0xb9
        movwf   GXPOLY_LOW

        return

CRC_GEN                         ; CRC generator routine
        bcf     STATUS,C        ; clear the carry bit
        rlf     CRC_BUF2,1      ; left shift all the data by one bit,
        rlf     CRC_BUF1,1      ; using carry flag, so the carry 1 or 0
        rlf     CRC_LOW,1       ; will go to the LSB of next byte
        rlf     CRC_HIGH,1

        btfsc   CRC_HIGH,2      ; is the last bit a 1?
        call    DO_XOR          ; if yes, XOR the working registers with g(x)
        decfsz  ITERATIONS,1    ; Decrement iterations, more bits left?
        goto    CRC_GEN         ; if yes, do another bit
        return

DO_XOR                          ; Modulo 2 division with the g(x)
        movf    GXPOLY_LOW,0
        xorwf   CRC_LOW,1
        movf    GXPOLY_HIGH,0
        xorwf   CRC_HIGH,1
        return

TRANSMIT_DISPLAY
        movlw   'T'
        call    LCD_CHAR
        movlw   'R'
        call    LCD_CHAR
        movlw   'A'
        call    LCD_CHAR
        movlw   'N'
        call    LCD_CHAR
        movlw   'S'
        call    LCD_CHAR
        movlw   'M'
        call    LCD_CHAR
        movlw   'I'
        call    LCD_CHAR
        movlw   'T'
        call    LCD_CHAR
        movlw   'T'
        call    LCD_CHAR
        movlw   'I'
        call    LCD_CHAR
        movlw   'N'
        call    LCD_CHAR
        movlw   'G'
        call    LCD_CHAR

        return


DISPLAY_PS                      ; Displays the currently transmitted PS
        movf    PS_1,0
        call    LCD_CHAR
        movf    PS_2,0
        call    LCD_CHAR
        movf    PS_3,0
        call    LCD_CHAR
        movf    PS_4,0
        call    LCD_CHAR
        movf    PS_5,0
        call    LCD_CHAR
        movf    PS_6,0
        call    LCD_CHAR
        movf    PS_7,0
        call    LCD_CHAR
        movf    PS_8,0
```

```
        call    LCD_CHAR

        return


TRANSMIT                    ; Transmit the RDS data-stream
        call    TRANSMIT_G0     ; Group 0
        call    TRANSMIT_G1     ; Group 1
        call    TRANSMIT_G2     ; Group 2
        call    TRANSMIT_G3     ; Group 3
        goto    TRANSMIT        ; Loop forever


TRANSMIT_G0                 ; Transmit the first group, with PS_1 & PS_2
        movlw   0x08            ; Send the top byte of PI
        movwf   ITERATIONS
        movf    PI_HIGH,0
        movwf   TRX_BUF
        call    TRANSMIT_BUF_DATA

        movlw   0x08            ; Send the lower bye of PI
        movwf   ITERATIONS
        movf    PI_LOW,0
        movwf   TRX_BUF
        call    TRANSMIT_BUF_DATA

        movlw   0x02            ; Send top two bits of PI checkword offset A
        movwf   ITERATIONS
        movf    PI_CHCK1_HIGH,0         ; Move the whole byte
        movwf   TRX_BUF
        swapf   TRX_BUF,1       ; Swap nibbles
        rlf     TRX_BUF,1       ; Rotate left twice, to bring the first bit
        rlf     TRX_BUF,1       ; of the two bits to the MSB location
        call    TRANSMIT_BUF_DATA

        movlw   0x08            ; Send lower byte of PI checkword offset A
        movwf   ITERATIONS
        movf    PI_CHCK1_LOW,0
        movwf   TRX_BUF
        call    TRANSMIT_BUF_DATA

        movlw   0x08            ; Send upper part of block 2
        movwf   ITERATIONS
        movf    BLCK2_0HIGH,0
        movwf   TRX_BUF
        call    TRANSMIT_BUF_DATA

        movlw   0x08            ; Send lower byte of block 2
        movwf   ITERATIONS
        movf    BLCK2_0LOW,0
        movwf   TRX_BUF
        call    TRANSMIT_BUF_DATA

        movlw   0x02            ; Send the top two bits of Block 2 checkword
        movwf   ITERATIONS
        movf    BLCK2_0CHK_HIGH,0       ; Move the whole byte
        movwf   TRX_BUF
        swapf   TRX_BUF,1       ; Swap nibbles
        rlf     TRX_BUF,1       ; Rotate left twice, to bring the first bit
        rlf     TRX_BUF,1       ; of the two bits to the MSB location
        call    TRANSMIT_BUF_DATA

        movlw   0x08            ; Send the lower byte of block2 checkword
        movwf   ITERATIONS
        movf    BLCK2_0CHK_LOW,0
        movwf   TRX_BUF
        call    TRANSMIT_BUF_DATA

        movlw   0x08            ; Send the top byte of PI
        movwf   ITERATIONS
        movf    PI_HIGH,0
        movwf   TRX_BUF
        call    TRANSMIT_BUF_DATA

        movlw   0x08            ; Send the lower bye of PI
        movwf   ITERATIONS
        movf    PI_LOW,0
        movwf   TRX_BUF
        call    TRANSMIT_BUF_DATA

        movlw   0x02            ; Send top two bits of PI checkword offset C'
        movwf   ITERATIONS
        movf    PI_CHCK3_HIGH,0         ; Move the whole byte
        movwf   TRX_BUF
        swapf   TRX_BUF,1       ; Swap nibbles
        rlf     TRX_BUF,1       ; Rotate left twice, to bring the first bit
        rlf     TRX_BUF,1       ; of the two bits to the MSB location
        call    TRANSMIT_BUF_DATA
```

```
        movlw   0x08            ; Send lower byte of PI checkword offset C'
        movwf   ITERATIONS
        movf    PI_CHCK3_LOW,0
        movwf   TRX_BUF
        call    TRANSMIT_BUF_DATA

        movlw   0x08            ; Send PS_1
        movwf   ITERATIONS
        movf    PS_1,0
        movwf   TRX_BUF
        call    TRANSMIT_BUF_DATA

        movlw   0x08            ; Send PS_2
        movwf   ITERATIONS
        movf    PS_2,0
        movwf   TRX_BUF
        call    TRANSMIT_BUF_DATA

        movlw   0x02            ; Send the top two bits of PS checkword
        movwf   ITERATIONS
        movf    CHCKWRD_0_HIGH,0       ; Move the whole byte
        movwf   TRX_BUF
        swapf   TRX_BUF,1       ; Swap nibbles
        rlf     TRX_BUF,1       ; Rotate left twice, to bring the first bit
        rlf     TRX_BUF,1       ; of the two bits to the MSB location
        call    TRANSMIT_BUF_DATA

        movlw   0x08            ; Send lower byte of PS checkword
        movwf   ITERATIONS
        movf    CHCKWRD_0_LOW,0
        movwf   TRX_BUF
        call    TRANSMIT_BUF_DATA

        return

TRANSMIT_G1                     ; Transmit the first group, with PS_3 & PS_4
        movlw   0x08            ; Send the top byte of PI
        movwf   ITERATIONS
        movf    PI_HIGH,0
        movwf   TRX_BUF
        call    TRANSMIT_BUF_DATA

        movlw   0x08            ; Send the lower bye of PI
        movwf   ITERATIONS
        movf    PI_LOW,0
        movwf   TRX_BUF
        call    TRANSMIT_BUF_DATA

        movlw   0x02            ; Send top two bits of PI checkword offset A
        movwf   ITERATIONS
        movf    PI_CHCK1_HIGH,0        ; Move the whole byte
        movwf   TRX_BUF
        swapf   TRX_BUF,1       ; Swap nibbles
        rlf     TRX_BUF,1       ; Rotate left twice, to bring the first bit
        rlf     TRX_BUF,1       ; of the two bits to the MSB location
        call    TRANSMIT_BUF_DATA

        movlw   0x08            ; Send lower byte of PI checkword offset A
        movwf   ITERATIONS
        movf    PI_CHCK1_LOW,0
        movwf   TRX_BUF
        call    TRANSMIT_BUF_DATA

        movlw   0x08            ; Send upper part of block 2
        movwf   ITERATIONS
        movf    BLCK2_1HIGH,0
        movwf   TRX_BUF
        call    TRANSMIT_BUF_DATA

        movlw   0x08            ; Send lower byte of block 2
        movwf   ITERATIONS
        movf    BLCK2_1LOW,0
        movwf   TRX_BUF
        call    TRANSMIT_BUF_DATA

        movlw   0x02            ; Send the top two bits of Block 2 checkword
        movwf   ITERATIONS
        movf    BLCK2_1CHK_HIGH,0      ; Move the whole byte
        movwf   TRX_BUF
        swapf   TRX_BUF,1       ; Swap nibbles
        rlf     TRX_BUF,1       ; Rotate left twice, to bring the first bit
        rlf     TRX_BUF,1       ; of the two bits to the MSB location
        call    TRANSMIT_BUF_DATA

        movlw   0x08            ; Send the lower byte of block2 checkword
        movwf   ITERATIONS
        movf    BLCK2_1CHK_LOW,0
        movwf   TRX_BUF
        call    TRANSMIT_BUF_DATA
```

```
        movlw   0x08            ; Send the top byte of PI
        movwf   ITERATIONS
        movf    PI_HIGH,0
        movwf   TRX_BUF
        call    TRANSMIT_BUF_DATA

        movlw   0x08            ; Send the lower bye of PI
        movwf   ITERATIONS
        movf    PI_LOW,0
        movwf   TRX_BUF
        call    TRANSMIT_BUF_DATA

        movlw   0x02            ; Send top two bits of PI checkword offset C'
        movwf   ITERATIONS
        movf    PI_CHCK3_HIGH,0         ; Move the whole byte
        movwf   TRX_BUF
        swapf   TRX_BUF,1       ; Swap nibbles
        rlf     TRX_BUF,1       ; Rotate left twice, to bring the first bit
        rlf     TRX_BUF,1       ; of the two bits to the MSB location
        call    TRANSMIT_BUF_DATA

        movlw   0x08            ; Send lower byte of PI checkword offset C'
        movwf   ITERATIONS
        movf    PI_CHCK3_LOW,0
        movwf   TRX_BUF
        call    TRANSMIT_BUF_DATA

        movlw   0x08            ; Send PS_1
        movwf   ITERATIONS
        movf    PS_3,0
        movwf   TRX_BUF
        call    TRANSMIT_BUF_DATA

        movlw   0x08            ; Send PS_2
        movwf   ITERATIONS
        movf    PS_4,0
        movwf   TRX_BUF
        call    TRANSMIT_BUF_DATA

        movlw   0x02            ; Send the top two bits of PS checkword
        movwf   ITERATIONS
        movf    CHCKWRD_1_HIGH,0        ; Move the whole byte
        movwf   TRX_BUF
        swapf   TRX_BUF,1       ; Swap nibbles
        rlf     TRX_BUF,1       ; Rotate left twice, to bring the first bit
        rlf     TRX_BUF,1       ; of the two bits to the MSB location
        call    TRANSMIT_BUF_DATA

        movlw   0x08            ; Send lower byte of PS checkword
        movwf   ITERATIONS
        movf    CHCKWRD_1_LOW,0
        movwf   TRX_BUF
        call    TRANSMIT_BUF_DATA

        return

TRANSMIT_G2                     ; Transmit the first group, with PS_1 & PS_2
        movlw   0x08            ; Send the top byte of PI
        movwf   ITERATIONS
        movf    PI_HIGH,0
        movwf   TRX_BUF
        call    TRANSMIT_BUF_DATA

        movlw   0x08            ; Send the lower bye of PI
        movwf   ITERATIONS
        movf    PI_LOW,0
        movwf   TRX_BUF
        call    TRANSMIT_BUF_DATA

        movlw   0x02            ; Send top two bits of PI checkword offset A
        movwf   ITERATIONS
        movf    PI_CHCK1_HIGH,0         ; Move the whole byte
        movwf   TRX_BUF
        swapf   TRX_BUF,1       ; Swap nibbles
        rlf     TRX_BUF,1       ; Rotate left twice, to bring the first bit
        rlf     TRX_BUF,1       ; of the two bits to the MSB location
        call    TRANSMIT_BUF_DATA

        movlw   0x08            ; Send lower byte of PI checkword offset A
        movwf   ITERATIONS
        movf    PI_CHCK1_LOW,0
        movwf   TRX_BUF
        call    TRANSMIT_BUF_DATA

        movlw   0x08            ; Send upper part of block 2
        movwf   ITERATIONS
        movf    BLCK2_2HIGH,0
        movwf   TRX_BUF
```

```
        call    TRANSMIT_BUF_DATA

        movlw   0x08            ; Send lower byte of block 2
        movwf   ITERATIONS
        movf    BLCK2_2LOW,0
        movwf   TRX_BUF
        call    TRANSMIT_BUF_DATA

        movlw   0x02            ; Send the top two bits of Block 2 checkword
        movwf   ITERATIONS
        movf    BLCK2_2CHK_HIGH,0      ; Move the whole byte
        movwf   TRX_BUF
        swapf   TRX_BUF,1       ; Swap nibbles
        rlf     TRX_BUF,1       ; Rotate left twice, to bring the first bit
        rlf     TRX_BUF,1       ; of the two bits to the MSB location
        call    TRANSMIT_BUF_DATA

        movlw   0x08            ; Send the lower byte of block2 checkword
        movwf   ITERATIONS
        movf    BLCK2_2CHK_LOW,0
        movwf   TRX_BUF
        call    TRANSMIT_BUF_DATA

        movlw   0x08            ; Send the top byte of PI
        movwf   ITERATIONS
        movf    PI_HIGH,0
        movwf   TRX_BUF
        call    TRANSMIT_BUF_DATA

        movlw   0x08            ; Send the lower bye of PI
        movwf   ITERATIONS
        movf    PI_LOW,0
        movwf   TRX_BUF
        call    TRANSMIT_BUF_DATA

        movlw   0x02            ; Send top two bits of PI checkword offset C'
        movwf   ITERATIONS
        movf    PI_CHCK3_HIGH,0        ; Move the whole byte
        movwf   TRX_BUF
        swapf   TRX_BUF,1       ; Swap nibbles
        rlf     TRX_BUF,1       ; Rotate left twice, to bring the first bit
        rlf     TRX_BUF,1       ; of the two bits to the MSB location
        call    TRANSMIT_BUF_DATA

        movlw   0x08            ; Send lower byte of PI checkword offset C'
        movwf   ITERATIONS
        movf    PI_CHCK3_LOW,0
        movwf   TRX_BUF
        call    TRANSMIT_BUF_DATA

        movlw   0x08            ; Send PS_1
        movwf   ITERATIONS
        movf    PS_5,0
        movwf   TRX_BUF
        call    TRANSMIT_BUF_DATA

        movlw   0x08            ; Send PS_2
        movwf   ITERATIONS
        movf    PS_6,0
        movwf   TRX_BUF
        call    TRANSMIT_BUF_DATA

        movlw   0x02            ; Send the top two bits of PS checkword
        movwf   ITERATIONS
        movf    CHCKWRD_2_HIGH,0       ; Move the whole byte
        movwf   TRX_BUF
        swapf   TRX_BUF,1       ; Swap nibbles
        rlf     TRX_BUF,1       ; Rotate left twice, to bring the first bit
        rlf     TRX_BUF,1       ; of the two bits to the MSB location
        call    TRANSMIT_BUF_DATA

        movlw   0x08            ; Send lower byte of PS checkword
        movwf   ITERATIONS
        movf    CHCKWRD_2_LOW,0
        movwf   TRX_BUF
        call    TRANSMIT_BUF_DATA

        return

TRANSMIT_G3                     ; Transmit the first group, with PS_1 & PS_2
        movlw   0x08            ; Send the top byte of PI
        movwf   ITERATIONS
        movf    PI_HIGH,0
        movwf   TRX_BUF
        call    TRANSMIT_BUF_DATA

        movlw   0x08            ; Send the lower bye of PI
        movwf   ITERATIONS
        movf    PI_LOW,0
```

```
movwf   TRX_BUF
call    TRANSMIT_BUF_DATA

movlw   0x02            ; Send top two bits of PI checkword offset A
movwf   ITERATIONS
movf    PI_CHCK1_HIGH,0       ; Move the whole byte
movwf   TRX_BUF
swapf   TRX_BUF,1       ; Swap nibbles
rlf     TRX_BUF,1       ; Rotate left twice, to bring the first bit
rlf     TRX_BUF,1       ; of the two bits to the MSB location
call    TRANSMIT_BUF_DATA

movlw   0x08            ; Send lower byte of PI checkword offset A
movwf   ITERATIONS
movf    PI_CHCK1_LOW,0
movwf   TRX_BUF
call    TRANSMIT_BUF_DATA

movlw   0x08            ; Send upper part of block 2
movwf   ITERATIONS
movf    BLCK2_3HIGH,0
movwf   TRX_BUF
call    TRANSMIT_BUF_DATA

movlw   0x08            ; Send lower byte of block 2
movwf   ITERATIONS
movf    BLCK2_3LOW,0
movwf   TRX_BUF
call    TRANSMIT_BUF_DATA

movlw   0x02            ; Send the top two bits of Block 2 checkword
movwf   ITERATIONS
movf    BLCK2_3CHK_HIGH,0      ; Move the whole byte
movwf   TRX_BUF
swapf   TRX_BUF,1       ; Swap nibbles
rlf     TRX_BUF,1       ; Rotate left twice, to bring the first bit
rlf     TRX_BUF,1       ; of the two bits to the MSB location
call    TRANSMIT_BUF_DATA

movlw   0x08            ; Send the lower byte of block2 checkword
movwf   ITERATIONS
movf    BLCK2_3CHK_LOW,0
movwf   TRX_BUF
call    TRANSMIT_BUF_DATA

movlw   0x08            ; Send the top byte of PI
movwf   ITERATIONS
movf    PI_HIGH,0
movwf   TRX_BUF
call    TRANSMIT_BUF_DATA

movlw   0x08            ; Send the lower bye of PI
movwf   ITERATIONS
movf    PI_LOW,0
movwf   TRX_BUF
call    TRANSMIT_BUF_DATA

movlw   0x02            ; Send top two bits of PI checkword offset C'
movwf   ITERATIONS
movf    PI_CHCK3_HIGH,0       ; Move the whole byte
movwf   TRX_BUF
swapf   TRX_BUF,1       ; Swap nibbles
rlf     TRX_BUF,1       ; Rotate left twice, to bring the first bit
rlf     TRX_BUF,1       ; of the two bits to the MSB location
call    TRANSMIT_BUF_DATA

movlw   0x08            ; Send lower byte of PI checkword offset C'
movwf   ITERATIONS
movf    PI_CHCK3_LOW,0
movwf   TRX_BUF
call    TRANSMIT_BUF_DATA

movlw   0x08            ; Send PS_1
movwf   ITERATIONS
movf    PS_7,0
movwf   TRX_BUF
call    TRANSMIT_BUF_DATA

movlw   0x08            ; Send PS_2
movwf   ITERATIONS
movf    PS_8,0
movwf   TRX_BUF
call    TRANSMIT_BUF_DATA

movlw   0x02            ; Send the top two bits of PS checkword
movwf   ITERATIONS
movf    CHCKWRD_3_HIGH,0       ; Move the whole byte
movwf   TRX_BUF
swapf   TRX_BUF,1       ; Swap nibbles
```

```
        rlf     TRX_BUF,1       ; Rotate left twice, to bring the first bit
        rlf     TRX_BUF,1       ; of the two bits to the MSB location
        call    TRANSMIT_BUF_DATA

        movlw   0x08            ; Send lower byte of PS checkword
        movwf   ITERATIONS
        movf    CHCKWRD_3_LOW,0
        movwf   TRX_BUF
        call    TRANSMIT_BUF_DATA

        return


TRANSMIT_BUF_DATA               ; Transmit the data byte held in the buffer
        rlf     TRX_BUF,1       ; Rotate left once, in itself
        btfss   STATUS, C       ; Test the carry flag value
        call    TRANSMIT_0      ; is carry 0? transmit a 0
        btfsc   STATUS, C
        call    TRANSMIT_1      ; Is carry 1? transmit a 1
        decfsz  ITERATIONS,1    ; Decrement iteration until it reaches 0
        goto    TRANSMIT_BUF_DATA
        return



TRANSMIT_0
;       call    DELAY_125US
        btfss   RDS_PORT,SIGNAL         ; Is 1187.5 high?
        goto    TRANSMIT_0      ; No? then loop until it goes high
        bcf     RDS_PORT, RDS_DATA      ; Clear the port bit, 0 will be modulated
        return

TRANSMIT_1
;       call    DELAY_125US
        btfss   RDS_PORT,SIGNAL         ; Is 1187.5 high?
        goto    TRANSMIT_1      ; No? then loop until it goes high
        bsf     RDS_PORT, RDS_DATA      ; Set the port bit, 1 will be modulated
        return



;*******************************************************************************
; LCD_CHAR
; Sends character to LCD
; Required character must be in W
;*******************************************************************************
LCD_CHAR
        movwf   LCD_TEMP        ; Character to be sent is in W
        call    LCDBUSY         ; Wait for LCD to be ready
        bcf     LCD_CTRL, RW    ; Set LCD in read mode
        bsf     LCD_CTRL, RS    ; Set LCD in data mode
        bsf     LCD_CTRL, E     ; LCD E-line High
        movf    LCD_TEMP, W
        movwf   LCD_DATA        ; Send data to LCD
        bcf     LCD_CTRL, E     ; LCD E-line Low
        return
;*******************************************************************************
; LCD_CMND
; Sends command to LCD
; Required command must be in W
;*******************************************************************************

LCD_CMND
        movwf   LCD_TEMP
        call    LCDBUSY         ; Wait for LCD to be ready
        bcf     LCD_CTRL,RS     ; Set LCD in read mode
        bcf     LCD_CTRL,RW     ; Set LCD in command mode
        bsf     LCD_CTRL,E      ; LCD E-line High
        movf    LCD_TEMP, W
        movwf   LCD_DATA        ; Send data to LCD
        bcf     LCD_CTRL,E      ; LCD E-line Low
        return

;*******************************************************************************
; LCDBUSY
; Returns when LCD busy-flag is inactive
; OK
;*******************************************************************************
LCDBUSY
        bsf     STATUS,RP0      ; Select Register page 1
        movlw   0xFF            ; Set PORTB for input
        movwf   TRISB
        bcf     STATUS, RP0     ; Select Register page 0
        bcf     LCD_CTRL, RS    ; Set LCD for command mode
        bsf     LCD_CTRL, RW    ; Setup to read busy flag
        bsf     LCD_CTRL, E     ; LCD E-line High
        movf    LCD_DATA, W     ; Read busy flag + DDram address
        bcf     LCD_CTRL, E     ; LCD E-line Low
```

```
        andlw   0x80            ; Check Busy flag, High = Busy
        btfss   STATUS, Z
        goto    LCDBUSY         ; Loop back if busy
        bcf     LCD_CTRL, RW
        bsf     STATUS, RP0     ; Select Register page 1
        movlw   0x00
        movwf   TRISB           ; Set PORTB for output
        bcf     INTCON, RBIF
        bcf     STATUS, RP0     ; Select Register page 0
        return
;*********************************************************************************
**********
;* SET_ADDR
        *
;* sets the start address in LCD DDRAM for writing characters to the LCD
        *
;* Load the lcd address you wish to write to into the w register before calling
routine        *
;*********************************************************************************
**********

SET_ADDR
        iorlw   0x80            ;combine address(a) in w to give 1aaa aaaa
        call    LCD_CMND        ;send byte in w to LCD data lines
        call    DELAY_125US     ;delay 125 microsecond
        return



;*************************************************************************
; LONG_DELAY
; Uses a very long delay to allow the user to see the data on the LCD
;*************************************************************************

LONG_DELAY
        movlw   0x4f            ; Decimal value to give a long delay
        movwf   CNT_DELAY3
rep_3   call    DELAY_30MS
        decfsz  CNT_DELAY3,1
        goto    rep_3
        return
;*************************************************************************
; SWITCH_DELAY
; Uses a  long delay to allow the switch to over-come debounce problems
;*************************************************************************

SWITCH_DELAY
        movlw   0x20            ; Decimal value to give a long delay
        movwf   CNT_DELAY3
rep_4   call    DELAY_30MS
        decfsz  CNT_DELAY3,1
        goto    rep_4
        return

;*********************************************************************************
**********
;* DELAY_125US
        *
;* uses repeated instruction cycles to create approximate 125 microsecond delay
        *
;* using a 4Mhz clock on the pic.(42x3 cycles of 1us)
                *
;*********************************************************************************
**********

DELAY_125US
        movlw   US125           ; decimal value 42 loaded into w register

        movwf   CNT_DELAY1      ; move 42 into cnt_delay1 register
repeat  decfsz  CNT_DELAY1,1    ; decrease count by 1 and check if zero (1 instruction
cycle)
        goto    repeat          ; decfsz will skip this if count was zero
        return

;*********************************************************************************
**********
;* DELAY_5ms
        *
;* uses repeated instruction cycles to create approximate 5ms delay (39x130x1 cycle of
1us)    *
;* using a 4Mhz clock on the pic. (130 because 125+5 cycles from this routine)
                *
;*********************************************************************************
**********

DELAY_5MS
        movlw   MS5             ;decimal value 39 loaded into w register
        movwf   CNT_DELAY2      ;move 39 into count2 register
```

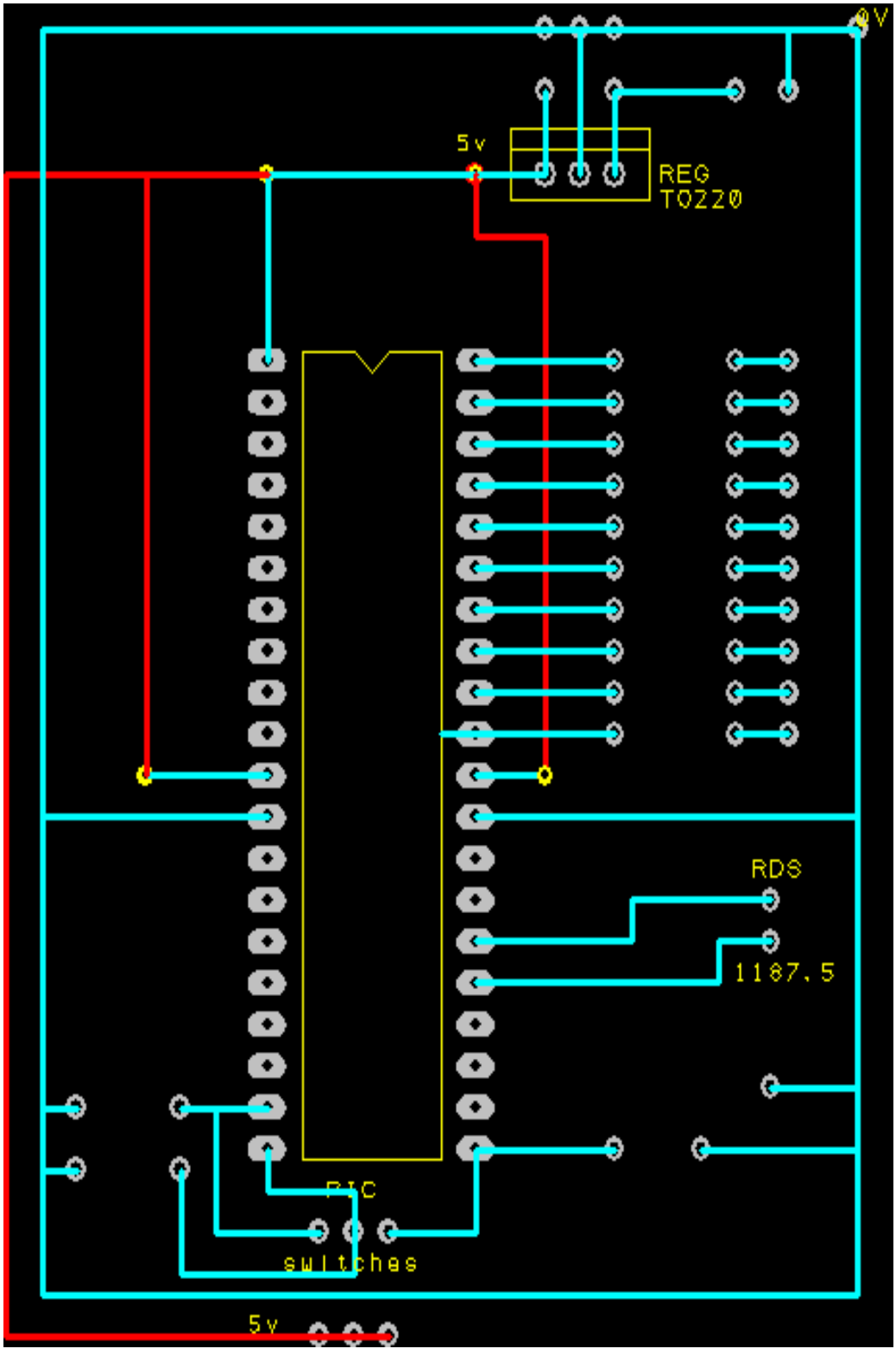
```
rep2    call    DELAY_125US     ;call routine for 125 microsecond delay(2 instruction
cycle)
        decfsz  CNT_DELAY2,F    ;decrease count2 by 1 and check if zero (1 instruction
cycle)
        goto    rep2            ;decfsz will skip this if count2 was zero(2 instruction
cycle)
        return

;*******************************************************************************
**********
;* END OF DELAY_5ms
        *
;*******************************************************************************
**********


;*******************************************************************************
**********
;* DELAY_30ms
        *
;* uses repeated instruction cycles to create approximate 30ms delay (231x130x1 cycle
of 1us) *
;* using a 4Mhz clock on the pic. (130 because 125+5 cycles from this routine)
              *
;*******************************************************************************
**********

DELAY_30MS
        movlw   MS30            ; decimal value 231 loaded into w register
        movwf   CNT_DELAY2      ; move 240 into cnt_delay2 register
rep_2   call    DELAY_125US     ; call routine for 125 microsecond delay
        decfsz  CNT_DELAY2,1    ; decrease count2 by 1 and check if zero (1 instruction
cycle)
        goto    rep_2           ; decfsz will skip this if count2 was zero
        return

;=============================================================================
; LCD_CLEAR
; Clears display and returns cursor to home position (upper-left corner).
;
;=============================================================================
LCD_CLEAR
        movlw   0x01            ; Move the value for lcd clear command
        call    LCD_CMND
        retlw   0x00

        END
```

B. Bi-Phase.asm

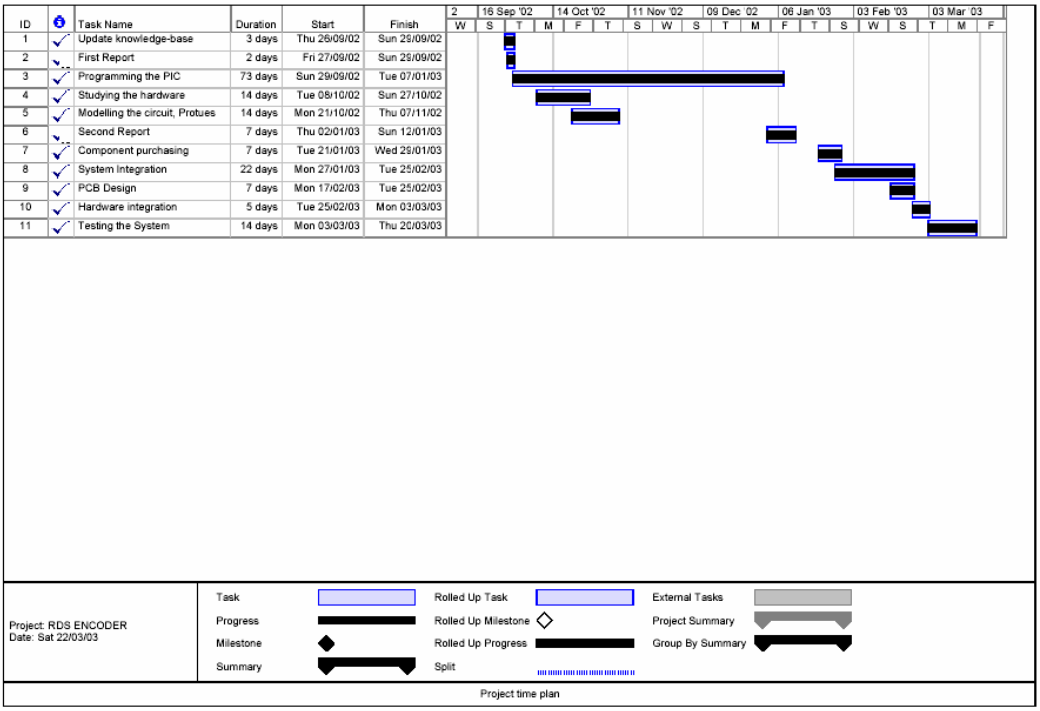
```
;*********************************************************************
;                                                                    *
;    Filename: biphase.asm                                           *
;    Date:     05/03/03                                              *
;                                                                    *
;    Author:  hhaddadi                                               *
;*********************************************************************
;                                                                    *
;    Notes:  To generate biphase symbols from port B                 *
;*********************************************************************

       list      p=16c622            ; list directive to define processor
       #include <p16c622.inc>        ; processor specific variable definitions

       __CONFIG   _CP_OFF & _WDT_OFF & _BODEN_ON & _PWRTE_ON & _XT_OSC


;***** VARIABLE DEFINITIONS*****************************************
CLK11875       EQU            1
CLK2375        EQU            2
RDS                   EQU            3


PULSEHIGH      EQU            4
PULSEDOWN      EQU            6

US51           EQU            0x21    ; value to give a 51 microsecond in delay loop
CNT_DELAY1     EQU            0x22    ; temporary count register for 51us delay routine
MEMORY         EQU            0x23

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

       org     0x00
RESET  goto    MAIN

MAIN

               clrf    STATUS                          ; Do initialization, Select bank 0
               bsf             STATUS,RP0                      ; Select Bank 1
               clrf    TRISA                           ; Port A as output to save power
               clrf    TRISB                           ; PORT B is output
               bsf             OPTION_REG,NOT_RBPU     ; Disable week Pull ups
               movlw   0x0E
               movwf   TRISB                           ; Declare pins 0&1 of  port B as
inputs
               clrf    STATUS
generate
               call    poll11875
               call    signal_pulse
               call    DELAY_51US              ; To by-pass the clock's high-signal
               call    DELAY_51US
               call    DELAY_51US
               call    DELAY_51US
               call    DELAY_51US
               call    DELAY_51US
               call    DELAY_51US
               call    DELAY_51US
               call    DELAY_51US
               call    poll2375
               call    clear_pulse
               call    DELAY_51US              ; To by-pass the clock's high-signa
               call    DELAY_51US
               call    DELAY_51US
               call    DELAY_51US
               call    DELAY_51US
               call    DELAY_51US
               call    DELAY_51US
               call    DELAY_51US

               goto    generate

poll11875
               btfss   PORTB,CLK11875
               goto    poll11875
               return
signal_pulse
               btfss   PORTB, RDS
               goto    PULSE_DOWN
               goto    PULSE_HIGH

poll2375
               btfss   PORTB, CLK2375
               goto    poll2375
               return

clear_pulse
               btfsc   MEMORY,0
```

```
                goto    PULSE_DOWN
                goto    PULSE_HIGH



PULSE_HIGH
                bsf             PORTB, PULSEHIGH
                call    DELAY_51US
                bcf             PORTB, PULSEHIGH
                bsf             MEMORY,0
                return

PULSE_DOWN
                bsf             PORTB, PULSEDOWN
                call    DELAY_51US
                bcf             PORTB, PULSEDOWN
                bcf             MEMORY,0
                return

;*************************************************************************
;* DELAY_51US                                   *
;* uses repeated instruction cycles to create approximate 51 microsecond delay*
;* using a 10Mhz clock on the pic.(42x3 cycles of 0.4us)
          *
;*************************************************************************

DELAY_51US
                movlw   US51            ; decimal value 42 loaded into w register

                movwf   CNT_DELAY1      ; move 42 into cnt_delay1 register
repeat  decfsz  CNT_DELAY1,1    ; decrease count by 1 and check if zero (1 instruction
cycle)
                goto    repeat          ; decfsz will skip this if count was zero
                return

                END                      ; directive 'end of program'
```

C. Data source PCB



D. Project time plan



9. References

1. United States RBDS Standard, April 9, 1998
2. www.microchip.com for MPLAB IDE use
3. Microchip PIC 16F87X Data Sheet
4. Microchip PICmicro Mid-Range MCU Family reference Manual
4. RDS: The Radio Data System: Dietmar Kopitz, Bev Marks
5. How to use intelligent LCDs: Julyan Ilet
6. Hitachi HD44780 LCD Controller/Driver manual
7. Microchip AN587 Interfacing PICmicro to an LCD module
8. TRIMOD LCD Specification
9. Microchip EEPROM Memory Programming specification
10. A Painless guide to CRC error detection algorithm: Ross Williams
11. Microchip AN730 CRC Generating and Checking
12. RDS Encoder Project: Tim Shaw (of EE)
13. RDS Encoder Project: Richard Koch (of EE)
14. HC-49 Quarts Crystal data sheet
15. AV Series Push Button Switches data sheet
16: Basic Communication Theory: John Pearson
17: CRC, Easier said than done: Michael Barr
18: www.farnell.co.uk
19: Electronic circuits III: Paul V Brennan, EEE
20: Digital Circuits: Chris Pitt, EEE
21: Microchip PIC16C622 datasheet

Author:
Hamed Haddadi, Department of Electronics & Electrical Engineering UCL, Torrington Place, London WC1E 7JE h.haddadi@ucl.ac.uk

12 Lionel House, 370 Portobello Rd, London W10 5RP hamedhaddadi@hotmail.com

10. Component datasheets