FM Receiver Project Technical Report

EEE2037
Laboratory, Design and Professional Studies IV
2016/17 Semester 2

Teddy Ng
Peng Young Tan
Herman Yau

Abstract

Between the months of February and May 2017, the project team undertook the design and construction of an FM receiver, in accordance with the guidelines set by the Laboratory, Design and Professional Studies IV module. This task necessitated the application of skills relating to both hardware and software in order to produce a functioning prototype.

This report details both the design and construction of the receiver hardware, and the implementation of software to operate the receiver. Additionally included are notes on novelties introduced into the design and considerations for improvement of future iterations of a similar project.

Table of Contents

1. Project Team

The FM project team consists of the following members:

Member	Roles
Teddy Ng	Development of the hardware, including manufacturing of enclosure and assembly of prototype board. Project management.
Peng Youn Tan	Development of the hardware, including building of test board, and prototype error checking. Aided with software development.
Herman Yau	Development of the software for the PIC18LF6490 microprocessor and AR1010 FM Chip.

2. Project Overview and Aims

The aims of this project are set by University of Surrey's Department of Electrical and Electronic Engineering lab manual in the following terms:
"Your goal is to produce a prototype FM receiver, making us of a PIC, an LCD display, and an audio amplifier chip. For the basic function of the FM receiver, you need to use the components provided from the lab." [1]

2.1. Project Planning

A brief meeting was held during the initial hours of the first lab session to allocate time and distribution of work in accordance with team members' strengths and weaknesses. A Gantt Chart was produced to identify tasks to be performed in subsequent weeks and assign approximate start times and durations. The chart was updated regularly, to review ongoing progress of the project and re-evaluate workload planning and allocation, where necessary. A full-size version of the chart below is included in Appendix E.

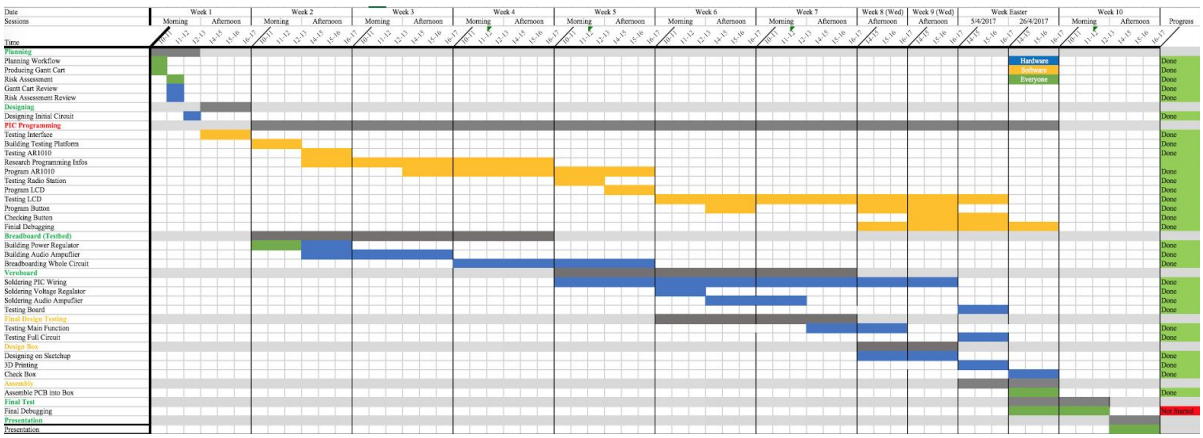


Table 2.1 Gantt Chart for the FM Receiver Project

3. Project Specification

Input:	3 x AA batteries, 4.5V / 1600-2400 mAh
Implementation:	Matrix Board / Stripboard Implementation
Output:	3.5 mm headphone jack output, capable of driving 35 ohms impedance headphones; and in-built stereo output

Frequency Range:	Tunable approximately over the UK broadcast band (87.5MHz to 108 MHz)
Software Functionalities:	The FM receiver should be able to switch channels (manually or by tuning), tune to preset frequencies and change volumes upon pressing the buttons. LCD should be able to display frequencies and volumes accordingly upon keypress.

4. Project Descriptions

4.1. Equipment

The following equipments are used for the FM Receiver Project:

- Electronic Prototype System (EPS) chassis rails
- Microsoft Visual C++
- MPLAB-X IDE
- PICkit™ 3 In-Circuit Debugger
- USB to I^2C converter

4.2. Hardware

4.2.1. Overall Structure

The hardware of the project is divided into 3 main modular blocks:

- the Power block, which regulates voltage supply to other blocks in the circuit;
- the Microcontroller block, which connects the microprocessor to the FM receiver chip used, and to other inputs and secondary outputs; and,
- the Amplifier block, which amplifies the audio signal from the FM receiver chip to an output level suitable to be played through speakers or headphones.

In the main, these blocks were adapted and modified from existing designs which can be found in the EE2037 FM Receiver Project materials and brief [1].

As further explained in the Novelties section, the blocks were designed to be modular. Hence, they could be tested individually on breadboard before being tested in combination; upon successful testing, they were then constructed with a similar methodology on the prototyping board.

4.2.2. Power Block

As the AR1010 FM receiver chip used later in the design cannot accept the approximate 4.5V supply given by 3 1.5V AA batteries, the power block is necessary to regulate its voltage supply to a suitable 3.3V. Meanwhile, the PIC18LF6490 microprocessor requires a voltage supply approaching 5V, hence the full voltage from the batteries must be fed to it. Summarily, the Power block's input is: one 4.5V DC supply from 3 AA batteries; and its outputs are: one 4.5V DC supply and one 3.3V DC supply, both referenced to a common ground, to be used as the supply rails for the rest of the circuit.

To achieve this, the LE33CZ voltage regulator was selected. Due to its low voltage dropout and corresponding power consumption, the LE33CZ is more suited for the design than other available alternatives with higher voltage dropout, such as circuits derived from the 78XX family of regulators. Additionally, a single SPST switch was included in series with the battery fixture to act as a power switch.

With these considerations, the final design for the Power block is as shown below:

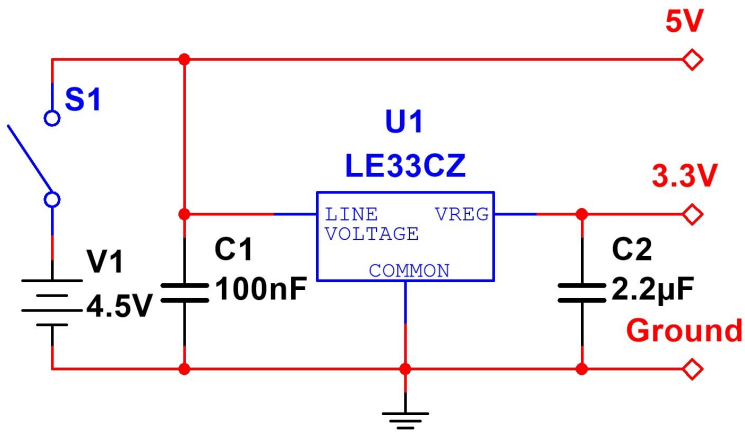


Figure 4.1 Circuit Diagram of Power Block

4.2.3. Microcontroller Block

As discussed in further detail in the Software section below, the PIC18LF6490 microprocessor is used to control all functionalities of the circuit. It must hence be electrically connected to user input (taken from a series of 8 buttons) in a known, standardised manner, such that commands may be properly received and transmitted; as well as to all relevant output modules (namely the AR1010 receiver chip and LCD screen).

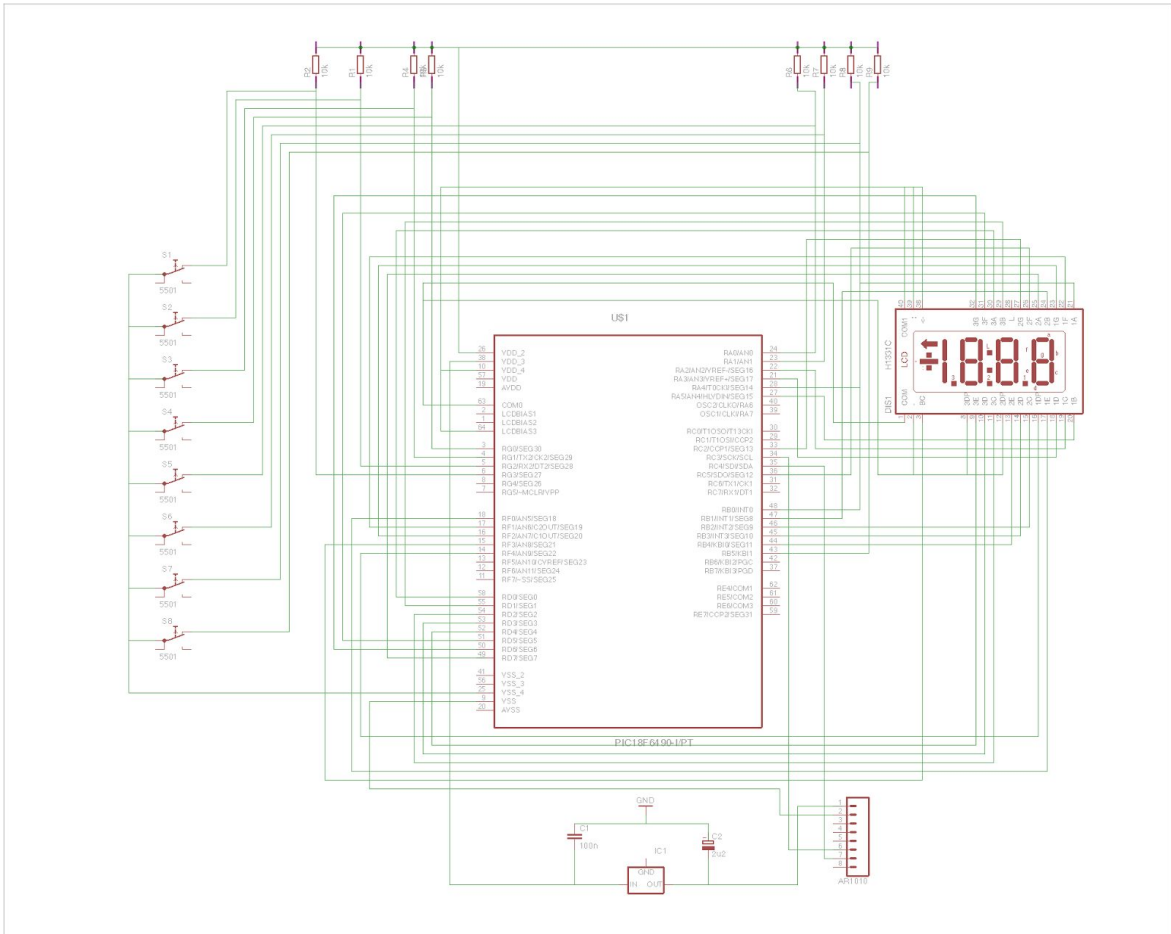


Figure 4.2 Circuit Diagram of Microcontroller Block

In addition to the wiring depicted above, the connections of the pushbuttons were implemented in such a way that voltages of the respective input pins on the microprocessor are high by default, connected to the positive supply rail across a 10kΩ resistor. Upon button press, the voltage will instead be connected to the ground rail and hence pulled low. This voltage change will be detected by the microprocessor, and the relevant data signal transmitted in accordance with the programming, as further discussed in the Software section.

The schematic of the implementation of an individual pushbutton is shown on the right. For ease of implementation, the required resistors were soldered together onto a separate piece of prototyping board and then connected later to the rest of the circuit.

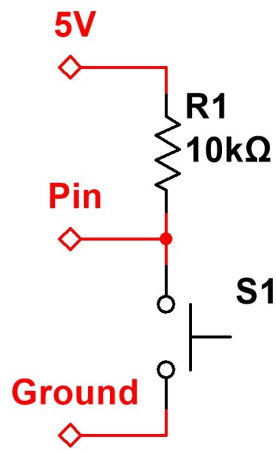


Figure 4.3. Circuit Diagram of Pushbutton

As the Microcontroller block is primarily concerned with the physical routing of wiring on the prototyping board, no schematic was designed for it. However, tables describing the connections used are included in Appendix B.

4.2.4. Amplifier Block

As the output signal from the FM receiver chip will not have sufficient power to drive audio hardware such as speakers or headphones, an additional block is required to amplify the signal to a suitable level.

The originally-suggested schematic did not limit the gain of the amplifier, which allows the greatest degree of amplification from the given operational amplifier (or op amp); in testing, output clipping was observed from input with amplitude as low as 10mV peak-to-peak. However, it is desirable that the audio signal should, as far as possible, maintain a high amplitude, up to and including the output channels of the FM receiver chip, such that the relative proportion of noise induced through electromagnetic interference (which is primarily amplitude-based) is as low as possible.

Prior to assembly, 100mV peak-to-peak was designated as the approximate output level of the FM receiver chip. Appropriate component values were then found experimentally to maximise the gain of the block without clipping by adding resistors at the op amp input to form the configuration of a non-inverting amplifier - these values were found to be 39Ω and 100Ω, the latter being the resistance of the feedback resistor. The FM receiver chip output was then regulated the desired value during software implementation.

As the audio output is required to be in stereo, two identical amplifier circuits are needed in order to keep the left and right audio channels separate. As such, two single op amps or one dual op amp should be used. From the available options, a TDA2822D dual low-voltage power amplifier was selected for its comparatively low minimum voltage supply and sufficient power amplification characteristics.

During implementation, the op amp itself was mounted on a separate piece of prototyping board and subsequently connected to the rest of the prototyping board. This allowed greater ease of construction, as well as the option to interchange multiple sets of op amps in the event that one may have been non-functional so as to facilitate testing.

In order to accommodate the differing needs of various users, the functionality to switch audio output between a set of built-in speakers and a TRS audio jack socket, capable of accepting a 3.5mm stereo TRS jack plug. The switching was accomplished through use of an SPDT switch to ground either of the speaker or plug outputs at a time.

As such, the final design of the Amplifier block is as shown below:

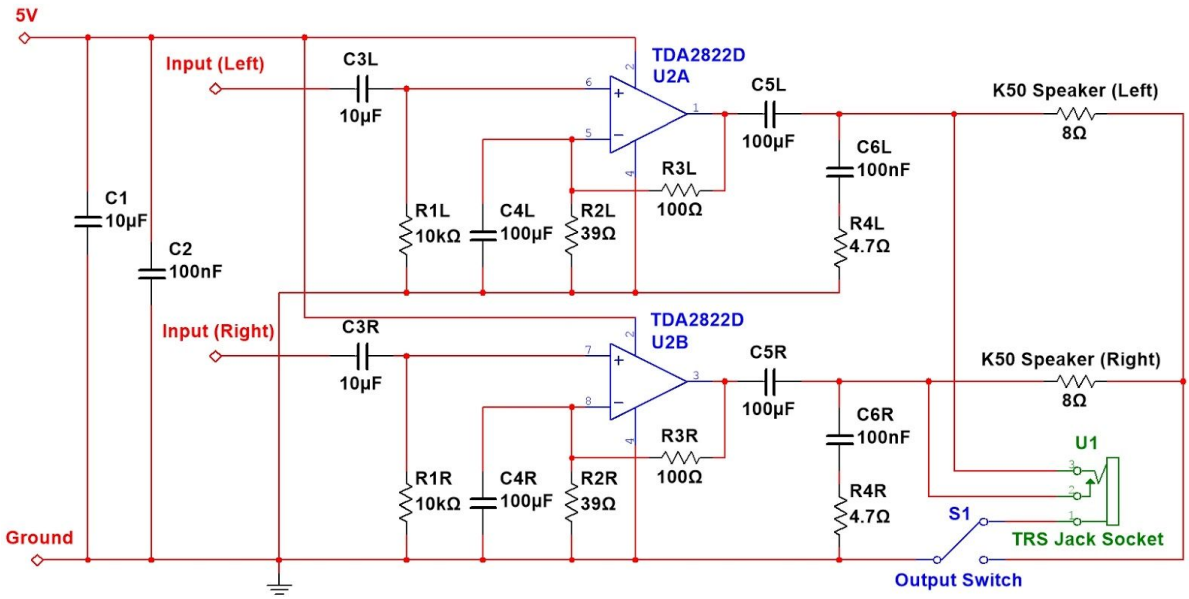


Figure 4.4 Circuit Diagram of Amplifier Block

4.2.5. Physical Implementation

While fabrication of a printed circuit board (or PCB) was initially considered, it was noted that use of a PCB would obviate some of the existing infrastructure in place:

Whereas the PCB could be constructed to conserve much of the wasted space on a prototyping board, the reduction would be limited by the size of the board provided on which the PIC18LF6490 microcontroller was mounted.

Additionally, components on a PCB would be spaced more closely together than on a prototyping board unless additional material was expended to deliberately fabricate the PCB to a large size.

Furthermore, multiple circuit boards would have to be fabricated if the modularity of the design is to be maintained, necessitating an overly large investment in time to wait for, check and (if necessary) refabricate the board for each module.

In view of these considerations, it was concluded that the design should be implemented on prototyping board instead of a PCB.

4.2.6. Design of Enclosure

The enclosure was designed on 'ProDesktop V8 Pro' - A CAD software, and export to 'Cure' - A print code generator, which translate the STL file into a GCODE file for the 3D printer - Ultimaker 2. The enclosure lid was also designed on 'ProDesktop V8 Pro', initially printed on the Ultimaker, but due to a machine breakdown issue, it was later laser cut using the Trotec Speedy300 laser-cutter with the speaker now mounted on the side. The lid was also printed

with less detail, thus allow the lid to be printed on a 2mm clear polycarbonate rather than a 6mm 3D printed PLA.

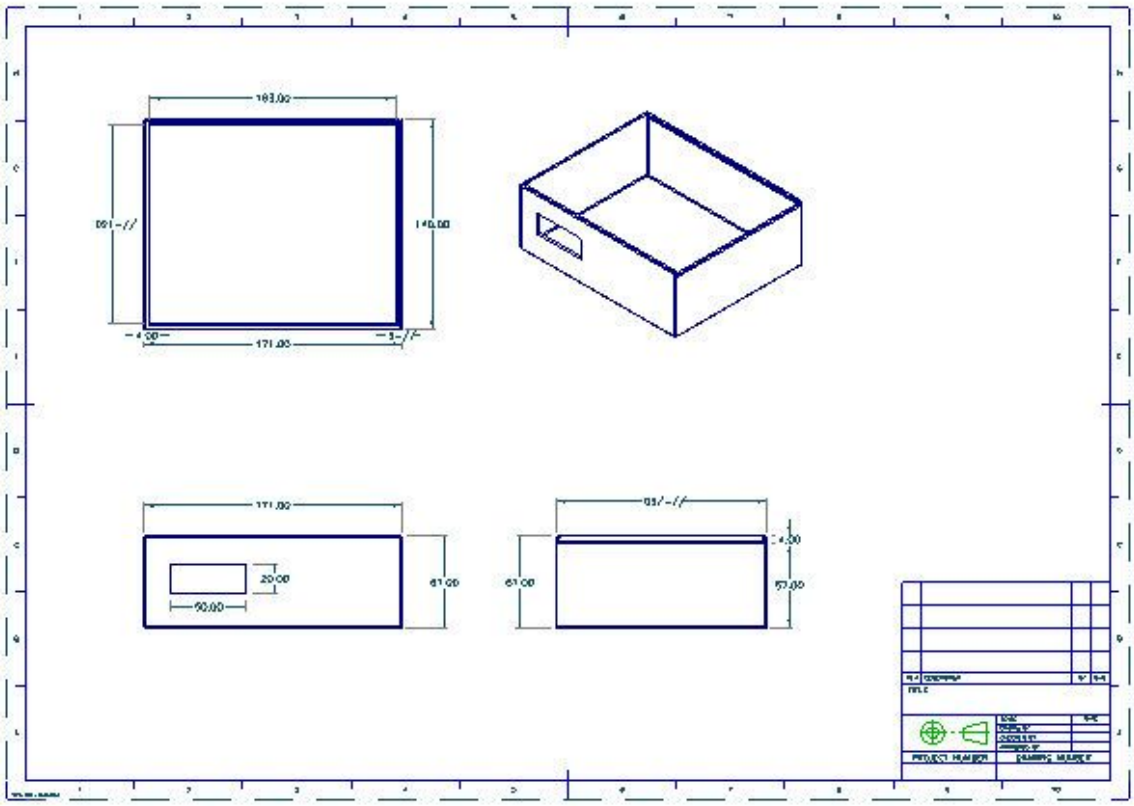


Figure 4.5 Top, Front and Side view of Enclosure

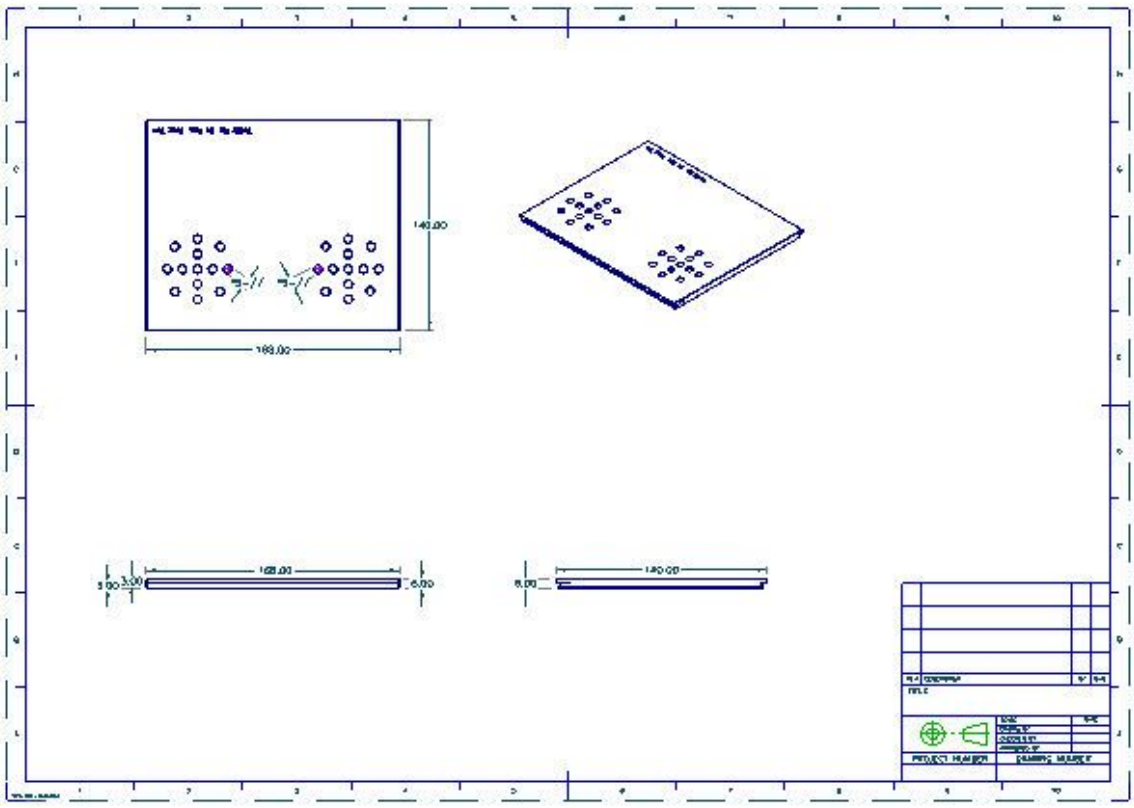


Figure 4.6 Top, Front and Side View of Enclosure Lid

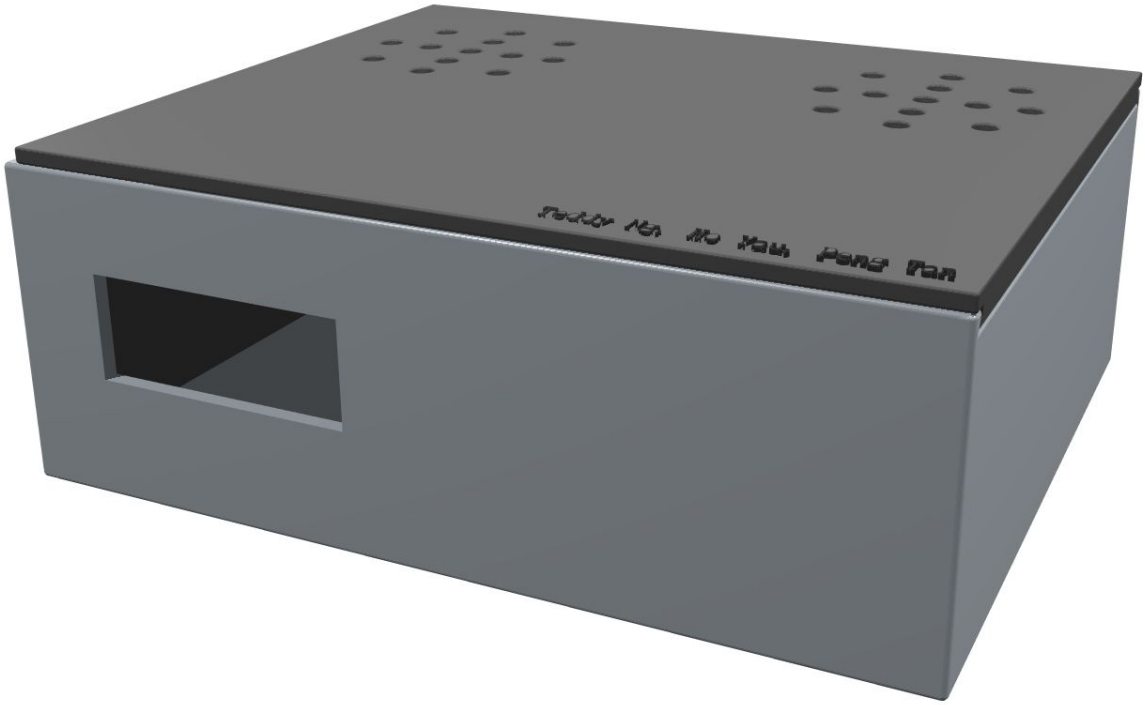


Figure 4.7 Rendered Model of the Completed Enclosure

4.3. Software

4.3.1. Initial Research

Prior to the start of the project, no members of the team had any pre-existing knowledge about microprocessor programming. Therefore, a variety of research was undertaken to study the underlying principles of Microchip® microprocessor operation, the I^2C (Inter-Integrated Circuit) Protocol and AR1010 FM radio chip used, and programming guides for both of them.

4.3.1.1. PIC18LF6490 Microcontroller

The PIC18LF6490 is a microcontroller developed by Microchip® Technology Inc. with a 8-bit CPU offering high computational power over a circuit board with low power at an economical price.
It is the main core where all functionalities are implemented and capable of controlling other modules on the board; in a sense, it is the "heart" of the FM receiver.

In order to program the PIC18LF6490 microcontroller, MPLAB PICkit™ 3 In-Circuit Debugger is used to connect it to a computer with MPLAB or MPLAB-X IDE installed.

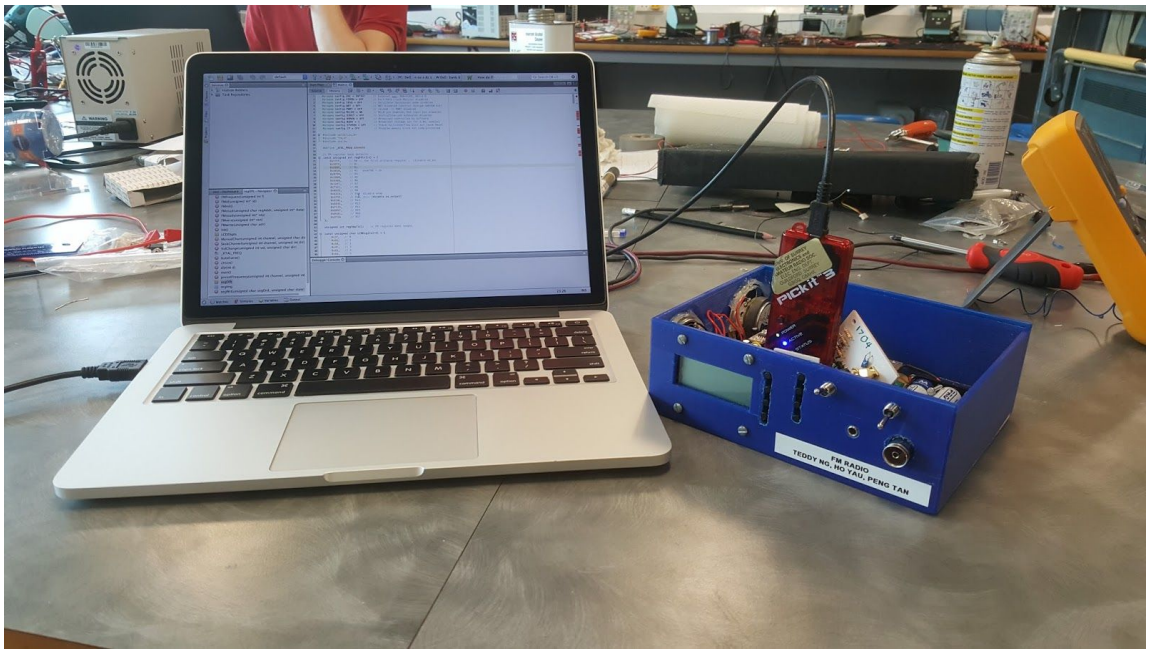


Figure 4.8 PICkit™ 3 In-Circuit Debugger Device, connected to laptop

4.3.1.2. I2C (Inter-Integrated Circuit) Protocol

For this project, this particular protocol was used for communication between the microprocessor (the master) and the AR1010 FM radio chip (the slave).

The I^2C Protocol is a message protocol used for communication between components which reside on the same circuit board with low transmission rate. The most unique feature of I^2C is that only 2 bus lines are required while still providing arbitration and collision detection.
The 2 bus lines are SDA (Data line) and SCL (Clock line). SCL is used to synchronise all data transfer over I^2C bus, while SDA is used for data transfer. When the protocol is used, the master/slave will initiate sequences of pulses which corresponds to '0's and '1's to write the desired registers.

4.3.1.3. AR1010 FM Chip Programming

The AR1010 is a microprocessor designed specifically for FM radio functions. In order to control the chip, a register map with description of bits operation was studied in detail.

ADDR	Alias	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
00H	R0									xo_en							EN
01H	R1			rds_en							rds_int_en	stc_int_en	deemp	mono	smute	hmute	
02H	R2							TUNE		CHAN<8:0>							
03H	R3	SEEKUP	SEEK	SPACE	BAND<1:0>			VOLUME<3:0>			SEEKTH<6:0>						
04H	R4																
05H	R4																
06H	R6																
07H	R7																
08H	R8																
09H	R9																
0AH	R10													seek_wrap			
0BH	R11	hilo_side													hiloctr_b1		hiloctrl_b2
0CH	R12																
0DH	R13											GPIO3<1:0>		GPIO2<1:0>		GPIO1<1:0>	
0EH	R14	VOLUME2<3:0>															
0FH	R15											rds_sta_en	rds_mecc<1:0>				rds_ctrl
10H	R16																
11H	R17																
12H	RSSI	RSSI<6:0>								IF_CNT<8:0>							
13H	STAT	READCHAN<8:0>									RDSR	STC	SF	ST			
14H	RBS	RBS1<1:0>		RBS2<1:0>		RBS3<1:0>		RBS4<1:0>									
15H	RDS1	RDS1<15:0>															
16H	RDS2	RDS2<15:0>															
17H	RDS3	RDS3<15:0>															
18H	RDS4	RDS4<15:0>															
19H	RDS5	rds_dsc<15:0>															
1AH	RDS6	rds_dfc<15:0>															
1BH	DEVID	VERSION<3:0>				MFID<11:0>											
1CH	CHIPID	CHIPNO<15:0>															

Figure 4.9 AR1010 FM Radio Chip Register Map

"pc2fm" Microsoft Visual C++ Project was provided by the University to facilitate study of the operation of AR1010 FM chip. It gives insight into how the chip can be controlled through I^2C and ultimately program the microprocessor to send instructions to it.

Frequency of FM signal cannot be directly fed into the AR1010 chip. A specific formula has to be used to calculate it, which is given by

$$Freq(MHz) = (690 + CHAN < 8:0 >)/10$$

The provided *FMfrequenc* function performs this calculation automatically, but the formula has to be manually applied after the channel seek operation since the returned value from *READCHAN<8:0>* by AR1010 is not the desired value. According to the Programming Guide for the AR1000/AR1010 Development Board, a set of pseudocode has to be followed precisely in order to control the FM chip for our desired functions. This is notably important for the implementation of a seek channel operation, such a function was listed in the specification. [2]

4.3.1.4. Seven-segment Display Programming

A 3.5 digit seven-segment display was used to display frequency and volume information from AR1010 FM chip via PIC microcontroller. The display uses a hexadecimal encoding of *".gfedcba"* by referring to the LCD connection layout and their corresponding PIC pin. The codings for the display can be referred in Appendix E.

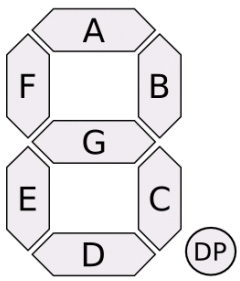


Figure 4.10. Individual Segments of Seven-segment Display

4.3.1.5. Pushbuttons Programming

As mentioned previously in the Hardware section, pushbuttons are connected to designated PIC inputs which are pulled to high via resistors and pulled down to ground whilst button is pushed. Due to mechanical imperfections, the signal level being read by the PIC microcontroller will jump between HIGH and LOW logic for a short period of time before stabilising which may lead to undesirable behaviours. This phenomenon is illustrated in the figure below:

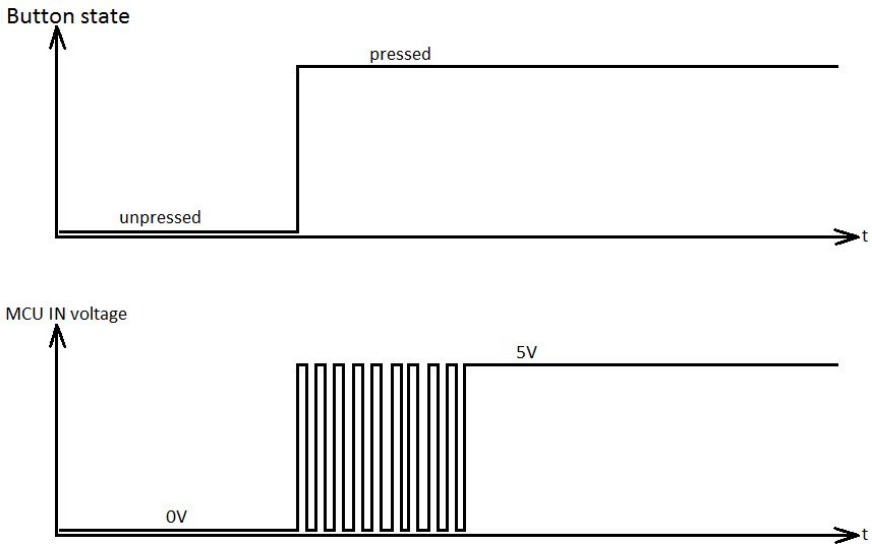


Figure 4.11 Switch Debouncing Problem Illustration

To tackle with the problem, a hardware solution where a smoothing capacitor or a software solution where a delay during the instability period can be used. It was decided that the implementation for pushbuttons detection would be utilising nested if-s. If a button is pushed, delay for a certain time to allow debouncing and check if the button is still pushed before sending the information to the PIC microcontroller for further actions.

4.3.2. Development Platform

4.3.2.1. MPLAB-X Integrated Development Environment

Programming for the PIC18LF6490 was done exclusively by the MPLAB-X IDE and C language with XC8 compiler. MPLAB-X IDE is a an integrated development environment developed by Microchip® Technology Inc. based on the open-source Oracle NetBeans IDE to provide a well-designed environment for microcontrollers and signal controllers programming.

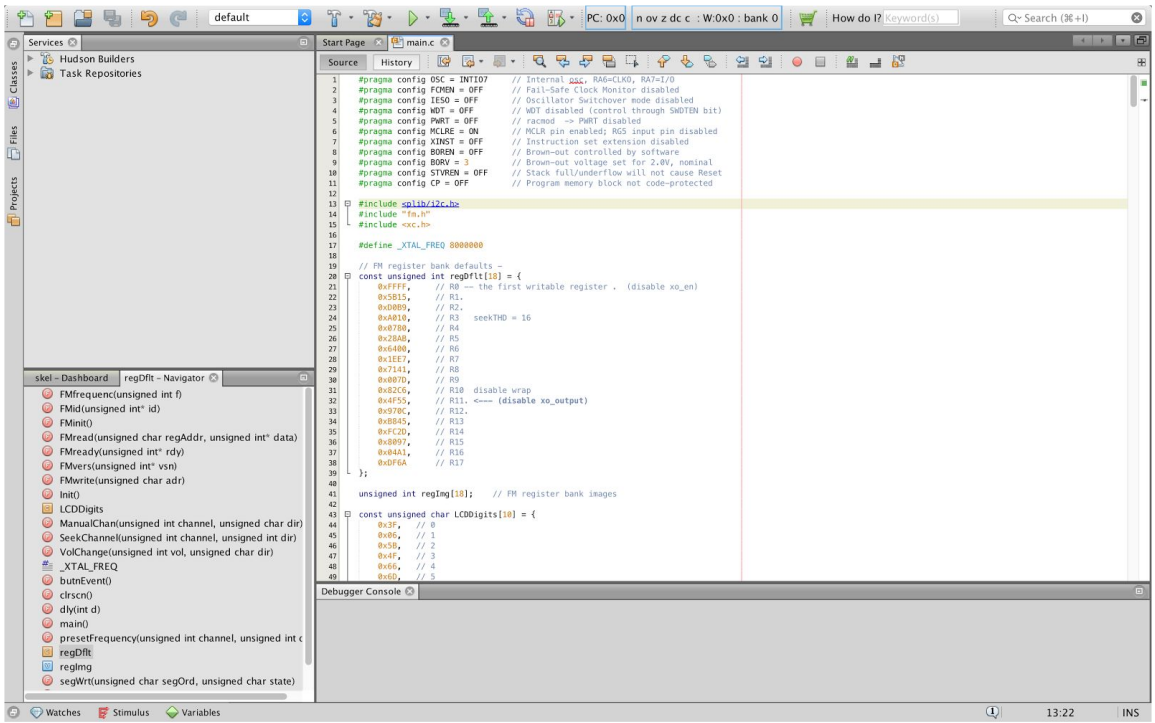


Figure 4.12 MPLAB-X IDE Interface

The software for the PIC18LF6490 was developed following the incremental build model, where the code was divided into different parts and built piece by piece until it is complete. The main benefit of this particular software development model is the flexibility provided, as all functionalities could be implemented with different modules with each responsible for a specific use. Also, it was easier to debug and apply adjustments for a smaller iteration. In return, a good planning and design prior to development was required, as well as a clear and complete definition of the whole system to ensure the final code adheres to the specifications.

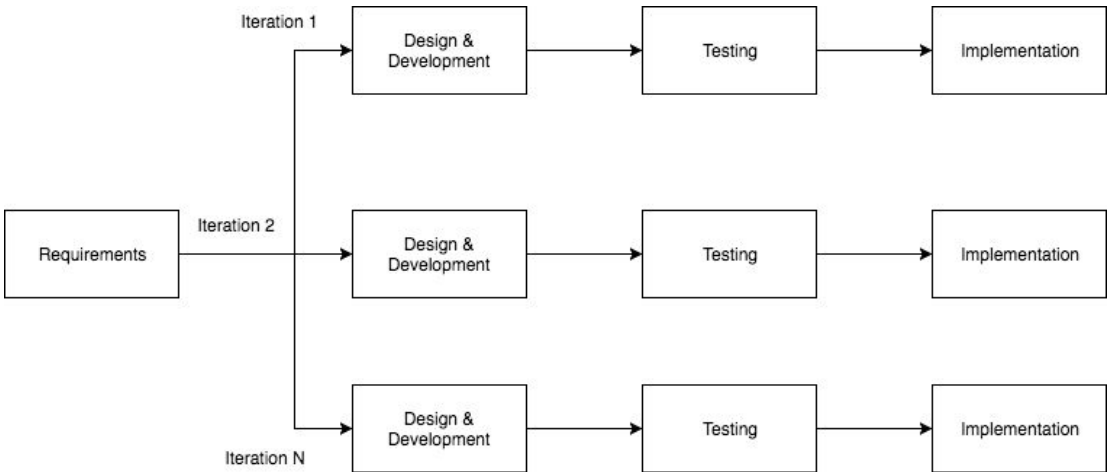


Figure 4.13 Incremental Software Development Model

4.3.2.2. Skel.X Skeleton Code

The original Skel.X skeleton code included a number of completed functions and global variables to aid development. These functions were reusable and code hierarchy could be created. Details for the provided global variables and functions in the original skeleton code can be found in Appendix D.

4.3.3. Final Product

To allow the PIC18LF6490 to properly communicate with the AR1010 FM chip (software-wise), the Programming Guide for AR1010 which was provided by the University and the PIC18LF6490 Data Sheet which is available from the public domain were carefully consulted. The relevant codes were then written into the skel.X MPLAB IDE Project, which is a skeleton code written in embedded C that was provided by the University. Below is a breakdown of the functions and global variables with explanations that are added to the skeleton code.

4.3.4. Global Variables

const unsigned char LCDDigits[10]

This is a constant character array which stores the hexadecimal encodings of the seven-segment display which is used on the LCD display.

const int VolData[2][19]

This is a constant 2D integer array which stores the integers for *VOLUME<3:0>* and *VOLUME2<3:0>* bits in register 3 and 14 respectively. The data can be found in Appendix.

const int presetFreq[4]

This is a constant 2D integer array which stores predefined frequencies.
The preset frequencies are set as follow: {880, 964, 977, 1046}. These frequencies are directly taken from pc2fm example code and are reused in the final product.

Frequencies	**Channel Description**
88.0 MHz	BBC R2
96.4 MHz	Eagle
97.7 MHz	BBC R1
104.6 MHz	BBC Surrey

4.3.5. Functions

unsigned char butnEvent()

The original template for this function is rewritten to the current version to simplify debouncing of buttons as well as incorporating button selection detection. This function reads the voltage level of the button ports, in this case the buttons are pulled to high when no buttons are pressed. When a button is pressed, the relevant port is pulled to low, in which case the button will initiate a 100ms delay for debouncing. Afterwards, it reads the same port again. If it is still pressed, a value is returned to main() to allow further actions to be taken.

unsigned char SeekChannel(unsigned int channel, unsigned int dir)

This function allows the AR1010 FM module to search for a channel with respect to seek direction and signal strength. It takes in two parameters from main function: *channel* and *dir*. *Channel* stores the current frequency where the FM module is located and *dir* indicates the direction of channel seek. Upon calling the function, it initiates steps to seek channel according to the pseudocode provided by AR1010 Programming Guide, as stated below[2]:

1. Set hmute bit
2. Clear TUNE bit
3. Set CHAN bits
4. Clear SEEK bit
5. Set SEEKUP/SPACE/BAND/SEEKTH bits
6. Enable SEEK bit
7. Wait STC flag (Seek/Tune Compete, in status register)
8. Clear hmute bit
9. Update Functions (optional) Remember to update CHAN with the seek result in READCHAN before next seek.

When channel seeking happens, an arrow will be displayed on the LCD to indicate it. Before returning new channel frequency from the FM module, it writes the LCD to display the new frequency.

unsigned int ManualChannel(unsigned int channel, unsigned char dir)

This function allows the AR1010 FM module to increment/decrement current frequency by 0.1 MHz for each call. As before, it takes in the same two parameters from main function: *channel* and *dir* with the same purpose.

void showFreq(unsigned int f)

This function allow the LCD to be written to display the current frequency where the FM chip is located at. It takes in a parameter f, which is the integer value for frequency.
The function first separates the integer into different digits, namely *ones, tens*, and *hundreds* by a simple division and modulus operation, followed by mapping to a predefined hexadecimal value dedicated to the LCD. If the frequency is higher than 1000 MHz, it also set '1' in the thousand digit to be displayed. Finally, it will go through a nested for loop where the digits are repeatedly shifted rightwards bitwise to allow LCD writing for individual segments.

void showVol(unsigned int vol)

This function allow the LCD to be written to display the current volume where the FM chip is set to. It takes in a parameter *vol*, which is the integer value for volume.
Like *showFreq* function, it first separates the integer into digits and mapping to predefined hexadecimal value dedicated to the LCD, followed by the same nested for loop to write individual digits.

unsigned int VolChange(unsigned int vol, unsigned char dir)

This function increments/decrements the volume of the AR1010 FM module. It takes in two parameters from the main function: *vol* and *dir*. *vol* contains the current volume of the FM chip and *dir* indicates the direction of volume change. Upon calling the function, the microprocessor clears the volume bits, followed by reading and shifting the new volumes to the those locations. Finally, it writes the registers and updates the LCD to display the new volume level. A total of 19 steps volume are implemented in this function in accordance with the AR1010 Programming Guide's suggestions.

5. Technical Challenges

5.1. Hardware

5.1.1. Complexity of Soldering

Due to the decision to implement the design on prototyping board, the vast majority of circuit connections had to be soldered manually. However, as the Microprocessor block required a large amount of connections in very close proximity to accommodate the 64 pins prevent on the PIC18LF6490, errors were easy to make and difficult to correct. There was no method to mitigate this apart from careful soldering and repeated continuity tests.

Due to wear from the repeated soldering and desoldering, some contact rings became detached from the prototyping board, requiring alternative contacts be found, and hence further complicating construction and generating unforeseen delays.

5.2. Software

5.2.1. Code Testing Inability

The software testing for the FM receiver was highly dependant on the hardware. Not only were the microcontroller and PICkit3™ only accessible at scheduled lab sessions, but the hardware was also not always readily available due to difficulties encountered during construction. As a result, progress on software development for the PIC18LF6490 microcontroller was constrained to the built-in MPLAB X IDE debugging console or, in some cases, unable to be tested at all. However, this testing process did on occasion serve to instead troubleshoot hardware problems.

5.2.2. LCD Display Method

The code for LCD output turned out to be to be a tricky implementation in the firmware due to a degree of awkwardness in the chosen mapping between the PIC's and the LCD's pins. This disallowed the use of LCDSE (LCD Segment Enable Register) for an easier and more readable implementation. This was overcome by an alternative method though with nested for loops and appropriate offsets for each digits accordingly.

6. Novelties

6.1. Modularity

Constructing the circuit in terms of multiple discrete blocks created a degree of modularity, allowing individual modules to be later substituted for alternatives which operate more efficiently or offer a greater variety of functionality.

Additionally, in constructing the prototype, the modules are mounted on separately-located pieces of prototyping board, hence reducing the degree of electromagnetic interference each module experiences from the others. It is particularly important to sufficiently isolate the clock and data channels between the microprocessor and FM receiver chip in the Microcontroller block from the high-power alternating currents in the Amplifier block, in order to reduce any consequent error signals.

6.2. Modified Enclosure

In order to improve user experience, several decisions were made regarding modifications to the enclosure during the construction process. To begin with, the decision was made early in the

construction stage to resize the enclosure to make it slightly more compact, portable and lightweight. Hence, not only would users find the receiver more convenient to transport from place to place, but also easier to designate a location for while in use.

This decision was not without trade-offs. A smaller enclosure also resulted in less surface area in which to install the various input and output modules. The decision was made to place the entirety of the user interface on a single side of the enclosure to improve accessibility. Consequently, the built-in speakers would have to be located either on the lid of the box or on another side.

However, owing to the simple sliding design of the lid, setting the speakers into the lid would have complicated the process of replacing the FM receiver's batteries, as each time, the speakers would have to be removed and replaced into a fixture designed to hold them in place. Hence, the decision was made to permanently set the speakers into an adjacent side of the enclosure to the user interface.

7. Improvements in the Future

7.1. Power-managed Modes Implementation

Due to time constraint, power-managed modes for the microprocessor was not written. This meant that the microprocessor would be in "Run" mode with both CPU and peripherals always on at all times, causing higher current to be drawn (14 μA typical). Current consumption could, however, be mitigated by implementation of an alternative, built-in "Idle" mode which powers off the CPU and drops the drawn current down to 5.8 μA typical instead.

7.2. Implementation of Tone Control

Whereas the AR1010 FM receiver chip can scale the entirety of its output signal to a given maximum voltage, it is incapable of performing attenuation over a limited range of frequencies. As the user is given the option of using their own audio hardware through the inclusion of the TRS audio jack socket, the output hardware may have any of a range of different characteristics. However, the circuit was tuned for the 8Ω built-in speakers; the bass, treble or even entirety of the audio may consequently be overly loud or soft, depending on the characteristics of the headphones used.

To mitigate this, the Amplifier block may be substituted with a modified version which includes a dedicated tone control circuit, such as the Baxandall network, which uses a network of capacitors and resistors to vary the cutoff frequencies for high- and low-pass filters. The design would be similar to some audio amplifiers made for EE2036 in 2016/17, hence this report will not go into especial detail regarding the implementation for brevity.

8. Operating Procedure

The FM receiver requires 3 AA alkaline battery to be powered. To access the batteries, simply slide the lid from the top of the box, replace the batteries as necessary and close the lid. Removal of batteries during prolonged periods without usage is recommended to prevent any possible leakage and damage to the circuitry.

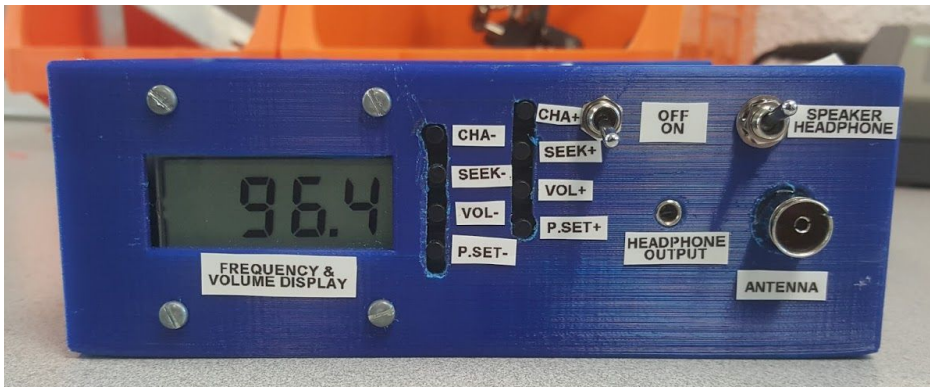


Figure 8.1 Front Paneling of Enclosure

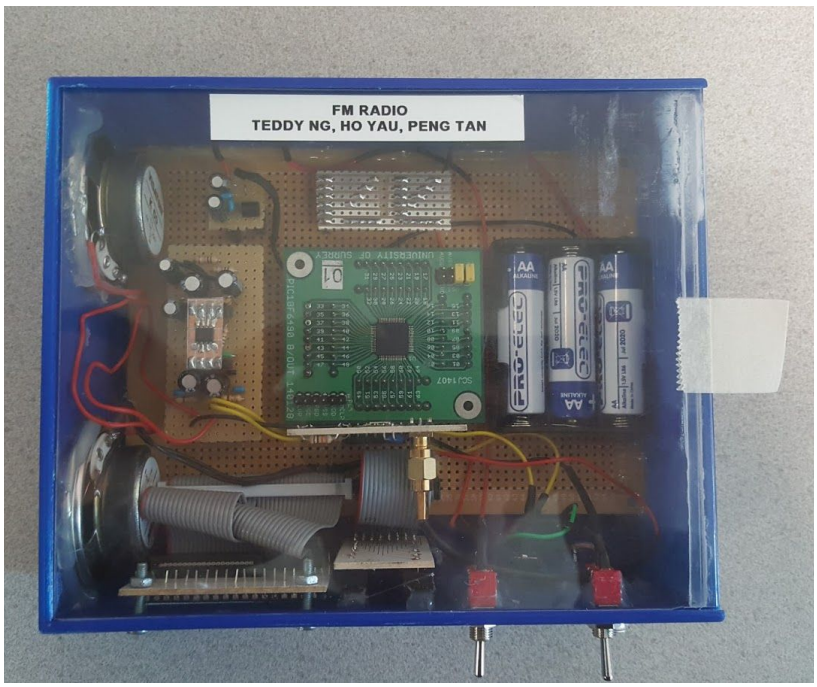


Figure 8.2 Top View of Enclosure

The FM receiver features speaker/headphone output, channel fine tuning, automatic channel seeking, 19-step volume control and 4 preset frequencies to choose from. All functionalities can be controlled via the on/off and output switches and the 8 pushbuttons set into the unit.

9. Conclusion

In conclusion, a product was designed and constructed that met all of the requirements set out in the specification, with a moderate amount of innovation. Additional consideration was taken to ensure the product is suited for the needs of the average user.

Embedded C and Microchip® libraries to control embedded systems were studied to improve competencies in C programming, and open source libraries used to code new components. The circuit was designed and developed from beginning to end on breadboard and prototyping board, with continual debugging of several issues with the prototype using continuity tests. Finally, an enclosure was designed and 3D-printed to house the finalised circuitry.

A Gantt chart was developed in the first week and was further improved and expanded it as the project progressed, ensuring an acceptable schedule was maintained. The project tasking was

divided into software and hardware teams, while allowing a degree of flexibility in task allocation; tasks were further compartmentalised into design of schematic, circuit construction and testing, software coding, and design of enclosure.

Some considerations were taken for improvements to future iterations of similar projects to further enhance user experience: implementation of improved power management via software, and inclusion of a tone control circuit via hardware.

10. References

[1] P. Aaen, “FM Receiver Project”, 2017
[2] Airoha Technology Corp, “AR1000 Programming Guide”

11. Appendix A - Supplementary External Documentation

PIC18F6390/6490/8390/8490 Datasheet
http://ww1.microchip.com/downloads/en/DeviceDoc/39629c.pdf

AR1010 Datasheet
http://www.rockbox.org/wiki/pub/Main/ArchosVision24c/AR1010_brief.pdf

TDA2882D Power Amplifier Datasheet
http://pdf.datasheetcatalog.com/datasheet/stmicroelectronics/1463.pdf

LE33CZ 3.3V Voltage Regulator Datasheet
http://www.st.com/content/ccc/resource/technical/document/datasheet/98/09/12/54/6e/d1/45/36/CD00000545.pdf/files/CD00000545.pdf/jcr:content/translations/en.CD00000545.pdf

Gantt Chart
https://drive.google.com/open?id=0B0w5J8cTtPUxSDAyb0RrM2IzQjg

12. Appendix B - Microcontroller Block Connection Tables

12.1. Table of Pushbutton Connections

PIC name	PIC pin	Pull-up pin	IDC pin	Button
RB0	48	2	10	Chan+
RB5	43	3	7	Chan-
RA0	24	4	12	PreSet+
RA1	23	5	5	PreSet-
RG0	3	6	14	Vol+
RG1	4	7	3	Vol-
RG2	5	8	16	SegTest
RG3	6	9	2	Error
Vss	9	-	1, 4, 6, 8, 9, 11, 13, 15	Switch return

12.2. Table of LCD Connections

PIC name	PIC pin	IDC pin	LCD pin	Varitronix name
RD0_SEG0	58	35	21	3A
RD1_SEG1	55	37	20	3B
RD2_SEG2	54	39	19	3C
RD3_SEG3	53	40	18	3D
RD4_SEG4	52	38	17	3E
RD5_SEG5	51	33	22	3F
RD6_SEG6	50	31	23	3G
RD7_SEG7	49	27	25	2A
RB1_SEG8	47	29	24	2B
RB2_SEG9	46	34	15	2C
RB3_SEG10	45	32	14	2D
RB4_SEG11	44	30	13	2E
RC5_SEG12	36	25	26	2F
RC2_SEG13	33	18	27	2G
RA4_SEG14	28	17	30	1A
RA5_SEG15	27	19	29	1B
RA2_SEG16	22	26	11	1C
RA3_SEG17	21	24	10	1D
RF0_SEG18	18	22	09	1E
RF1_SEG19	17	15	31	1F
RF2_SEG20	16	13	32	1G
RF3_SEG21	15	10	03	K
RF4_SEG22	14	36	16	DP3
RF4_SEG23	13	01	38	Z
COM0	63	28	12	DP2
COM0	63	21	08	DP1

COM0	63	20	28	COL
COM0	63	02	39	X
COM0	63	08	02	Y
COM0	63	06	01	COM
COM0	63	04	40	COM

13. Appendix C - Parts List

13.1. Miscellaneous

Part Description	**Quantity**
Prototyping Board	As necessary
Pin Headers and Sockets	As necessary
PIC18LF6490 Microprocessor	1
AR1010 FM Receiver Chip	1
Connector for External Antenna (Antenna not included)	1

13.2. Power Block

Part Description	**Quantity**
Battery Holder for 3 AA Batteries	1
1.5V AA Batteries	3
SPST Power Switch	1
Capacitor - 100nF	1
Capacitor - 2.2µF	1
STMicroelectronics LE33CZ Voltage Regulator	1

13.3. Microcontroller Block

Part Description	**Quantity**
Resistor - 10kΩ	8
Pushbutton	8
LCD Screen with Ribbon Cable Connector	1
Wiring Comb	As necessary

13.4. Amplifier Block

Part Description	Quantity
Capacitor - 100nF	3
Capacitor - 10µF	3
Capacitor - 100µF	4
Resistor - 4.7Ω	2
Resistor - 39Ω	2
Resistor - 100Ω	2
Resistor - 10kΩ	2
STMicroelectronics TDA2822D Dual Low-Voltage Power Amplifier	1
Visaton K50 8Ω Speaker	2
TRS 3.5mm Jack Socket	1
SPDT Audio Output Switch	1

14. Appendix D - Useful Materials for Software Development

14.1. Program Data for Volume Registers

VOLUME<3:0>	VOLUME2<3:0>
F	0
F	C
F	D
F	F
B	C
B	D
B	F
A	F
9	F
8	F

7	F
6	D
6	E
6	F
3	E
3	F
2	F
1	F
0	F

14.2. Hexadecimal Encodings for Seven-segment Display

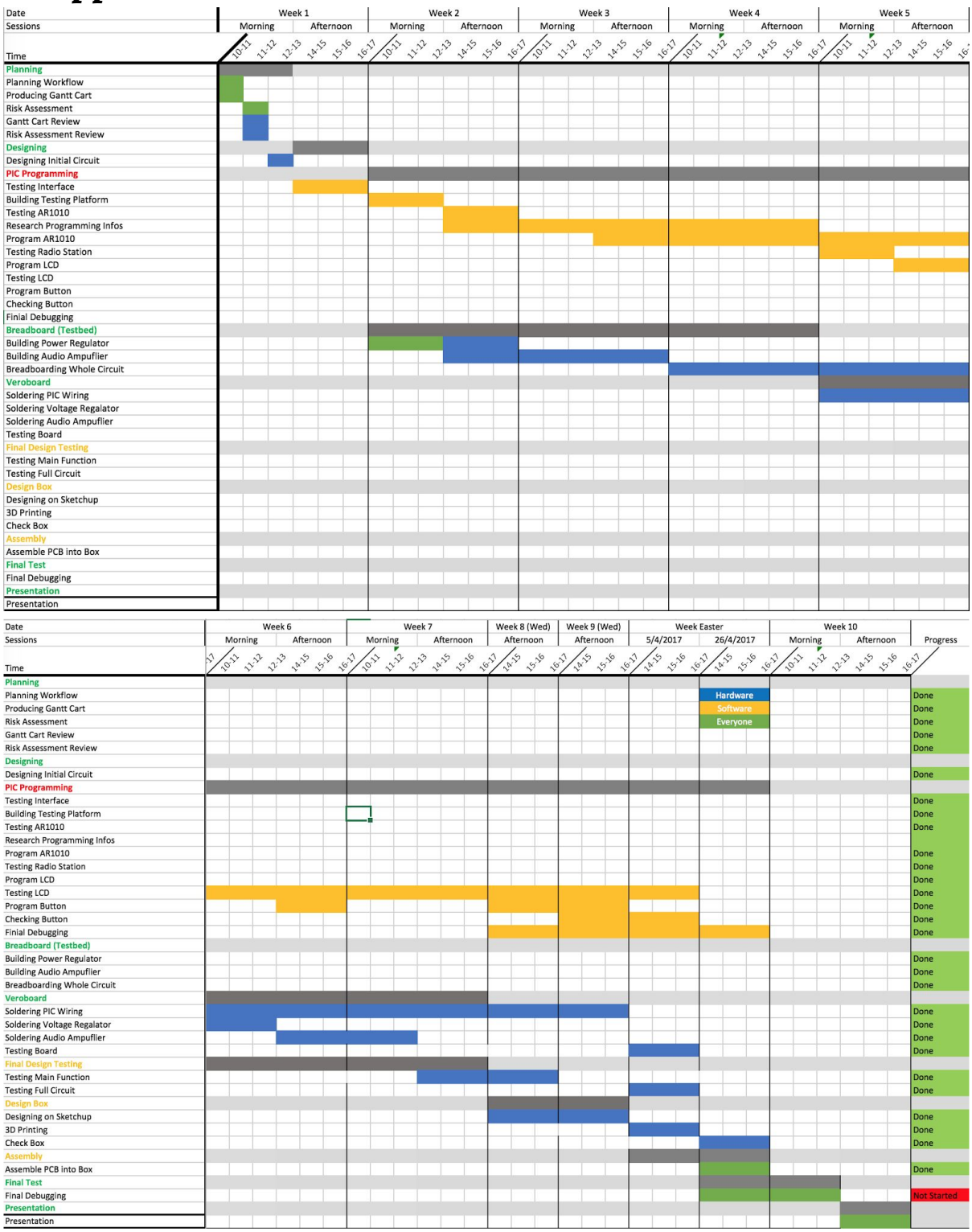
LCD Display	**Hexadecimal Encoding (.gfedcba)**
0	0x3F
1	0x06
2	0x5B
3	0x4F
4	0x66
5	0x6D
6	0x7D
7	0x07
8	0x7F
9	0x6F

14.3. Skel.X Skeleton Code Code Descriptions

Global Variables	**Description**
const unsigned int regDflt[18]	FM Register Bank Default
unsigned int regImg[18]	FM Register Bank Image, used by various functions. Main global variable array for read/write operation.

Functions	**Description**
void dly(int d)	Delay function to suspend program operation for d millisecond.
void clrscn()	Clear previously written LCD segments, i.e. turn off ALL LCD segments.
void Init()	Initialise the PIC microcontroller with designated bits.
void segWrt(unsigned char segOrd, unsigned char state)	Write a particular LCD segment specified by segOrd with a state specified by state, i.e. turn on/off LCD segment.
unsigned char FMwrite(unsigned char adr)	Write AR1010 register specified by adr.
unsigned char FMread(unsigned char regAddr, unsigned int *data)	Read AR1010 register specified by regAddr, and store the read bits into *data.
unsigned char FMready(unsigned int *rdy)	Check if AR1010 is ready for read/write.
unsigned char FMinit()	Initialise AR1010 with register bank default value.
unsigned char FMfrequenc(unsigned int f)	Tune AR1010 to new frequency specified by *f*.
unsigned char FMvers(unsigned int *vsn)	Fetch the AR1010 chip version.
unsigned char FMid(unsigned int *id)	Fetch the AR1010 chip ID.

15. Appendix E - Full-size Gantt Chart



16. Appendix F - PIC18LF6490 Software Codes

16.1. fm.h

```
#define XS          0          // Exit success
#define XF          1          // Exit fail
```

```
#define FMI2CADR                0x20            // Address (for writes) of FM module on I2C bus


#define DEVRD                   0x01            // Read not write an I2C device
#define FMCHIPVERSADR   0x1C            // Address of FM chip version
#define FMCHIPIDADR             0x1B            // Address of FM chip ID
#define FMCHIPSTSADR    0x13            // Address of FM chip status

#define FMASKMUTE               0x0002          // Register 1, bit 2
#define FMASKTUNE               0x0200          // Register 2, bit 9
#define FMASKSTATUS             0x0020          // Register 0x13, bit 5
#define FMASKSEEK               0x4000          // Register 3, bit 14
#define FMASKRDCHAN             0xFF80          // Register 2, channel number bits
#define FMASKVOL1       0x0780  // Register 3, Volume1
#define FMASKVOL2       0xF000  // Register 14, Volume2

#define BUTN1                   0b00000001      // Button number one
#define BUTN2                   0b00000010      // Button number two
#define BUTN3                   0b00000100
#define BUTN4                   0b00001000
#define BUTN5                   0b00010000
#define BUTN6                   0b00100000
#define BUTN7                   0b01000000
#define BUTN8                   0b10000000

#define LCDSEGDP3               22                      // LCD segment for decimal point
#define LCDSEGZ         23                      // LCD segment for Z

#define FMHIGHCHAN              (1080-690)      // Highest FM channel number
#define FMLOWCHAN               (875-690)
#define FALSE                   0
#define TRUE                    1

const int Vol_data[2][19] = {
  {15, 15, 15, 15, 11, 11, 11, 10, 9, 8, 7, 6, 6, 6, 3, 3, 2, 1, 0},
  {0, 12, 13, 15, 12, 13, 15, 15, 15, 15, 15, 13, 14, 15, 14, 15, 15, 15, 15}
};

const int presetFreq[4] = {881, 964, 977, 1046};

enum {                                                  // Global error numbers, enum type: enumerated
number
        GERNONE,                                        // No error 0
        GERWCOL,                                        // I2C write collision 1
        GERFINT,                                        // Could not initialize FM module 2
        GERFMID                                                 // Could not read chip ID (0x1010) 3
};

void Init();                                                            // Processor initialisation.
void dly(int d);
unsigned char FMread(unsigned char regAddr, unsigned int *data);
unsigned char FMwrite(unsigned char adr);                       // Write a new value to a register
unsigned char FMinit();                                                         // Initialise
the chip
unsigned char FMready(unsigned int *rdy);                       // Status is ready or busy
unsigned char FMid(unsigned int *id);                                   // Obtain ID number
void showFreq(unsigned int frequency);                  // Display the current f in MHz
unsigned char FMvers(unsigned int *vsn);                        // Obtain version number
unsigned int ManualChan(unsigned int channel, unsigned char dir);
unsigned int SeekChannel(unsigned int channel, unsigned int dir);
unsigned int Vol_change(unsigned int vol, unsigned char dir);
void LCDscn(unsigned char dir);
void showVol(unsigned int vol);
unsigned int presetFrequency(unsigned int channel, unsigned int dir);
//
```

```
// end receiveFM.h ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
//
//
```

16.2. main.c

```
#pragma config OSC = INTIO7     // Internal osc, RA6=CLKO, RA7=I/O
#pragma config FCMEN = OFF                  // Fail-Safe Clock Monitor disabled
#pragma config IESO = OFF                   // Oscillator Switchover mode disabled
#pragma config WDT = OFF       // WDT disabled (control through SWDTEN bit)
#pragma config PWRT = OFF      // racmod  -> PWRT disabled
#pragma config MCLRE = ON      // MCLR pin enabled; RG5 input pin disabled
#pragma config XINST = OFF     // Instruction set extension disabled
#pragma config BOREN = OFF     // Brown-out controlled by software
#pragma config BORV = 3        // Brown-out voltage set for 2.0V, nominal
#pragma config STVREN = OFF                 // Stack full/underflow will not cause Reset
#pragma config CP = OFF                     // Program memory block not code-protected

#include <plib/i2c.h>
#include "fm.h"
#include <xc.h>

#define _XTAL_FREQ 8000000

// FM register bank defaults -
const unsigned int regDflt[18] = {
            0xFFFF,    // R0 -- the first writable register .  (disable xo_en)
            0x5B15,    // R1.
            0xD0B9,    // R2.
            0xA010,    // R3   seekTHD = 16
            0x0780,    // R4
            0x28AB,    // R5
            0x6400,    // R6
            0x1EE7,    // R7
            0x7141,    // R8
            0x007D,    // R9
            0x82C6,    // R10  disable wrap
   0x4F55,    // R11. <--- (disable xo_output)
            0x970C,    // R12.
            0xB845,    // R13
            0xFC2D,    // R14
            0x8097,    // R15
            0x04A1,    // R16
            0xDF6A     // R17
};

unsigned int regImg[18]; // FM register bank images

const unsigned char LCDDigits[10] = {
   0x3F,   // 0
   0x06,   // 1
   0x5B,   // 2
   0x4F,   // 3
   0x66,   // 4
   0x6D,   // 5
   0x7D,   // 6
   0x07,   // 7
   0x7F,   // 8
```

```
    0x6F    // 9
};

/*
 * Obtain latest change in state for the pushbutton set.
 *
 * @return return a value to main() with respect to button push.
 */
unsigned char butnEvent() {
    if (PORTAbits.RA0 == 0){
        __delay_ms(100);
        if(PORTAbits.RA0 == 0)
            return 1;
    }
    if (PORTAbits.RA1 == 0){
        __delay_ms(100);
        if (PORTAbits.RA1 == 0)
            return 2;
    }

    if(PORTBbits.RB0 == 0){
        __delay_ms(100);
        if(PORTBbits.RB0 == 0)
            return 3;
    }

    if(PORTBbits.RB5 == 0){
        __delay_ms(100);
        if(PORTBbits.RB5 == 0)
            return 4;
    }

    if(PORTGbits.RG0 == 0){
        __delay_ms(100);
        if(PORTGbits.RG0 == 0)
            return 5;
    }

    if(PORTGbits.RG1 == 0){
        __delay_ms(100);
        if(PORTGbits.RG1 == 0)
            return 6;
    }

    if(PORTGbits.RG2 == 0){
        __delay_ms(100);
        if(PORTGbits.RG2 == 0)
            return 7;
    }

    if(PORTGbits.RG3 == 0){
        __delay_ms(100);
        if(PORTGbits.RG3 == 0)
            return 8;
    }
    return 0;                       // No changes
}
//
```

```
// end butnEvent ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
//


void dly(int d) {   //Delay function

        int i = 0;

        for ( ; d; --d)
                for (i = 100;  i;  --i) ;
}
//
// end dly ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
//

/*
 * Set all LCD segments to 0 (off, clear).
 *
 */
void clrscn() {

        int i = 0;
        unsigned char *CLEARptr;       // Pointer used to clear all LCDDATA


        for (       i = 0,
                        CLEARptr = (unsigned char *) &LCDDATA0;  // Point to first segment
                        i < 28;
                        i++)                    // Turn off all segments
                *CLEARptr++ = 0x00;
}
//
// end clrscn ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
//


void Init() {   //Initialize

        int i;

        OSCCON = 0b01110010;            // Select 8 MHz internal oscillator
        LCDSE0 = 0b11111111;            // Enable  LCD segments 07-00
        LCDSE1 = 0b11111111;            // Enable  LCD segments 15-08
        LCDSE2 = 0b11111111;            // Enable  LCD segments 23-16
        LCDSE3 = 0b00000000;            // Disable LCD segments 31-24
        LCDCON = 0b10001000;            // Enab LC controller. Static mode. INTRC clock
        LCDPS  = 0b00110110;            // 37 Hz frame frequency
        ADCON1 = 0b00111111;            // Make all ADC/IO pins digital
        TRISA = 0b00000011;         // RA0 and RA1 pbutton
        TRISB = 0b00100001;                                     // RB0 and RB5 pbutton
        TRISC = 0b00011000;                                     // RC3 and RC4 do the I2C bus
        TRISG = 0b11111111;                                     // RG0, RG1 & RG3 pbutton
        PORTA = 0;              // Set all Port A to LOW
        PORTB = 0;              // Set all Port B to LOW
        PORTC = 0;              // Set all Port C to LOW
   INTCONbits.TMR0IF = 0;       // Clear timer flag
   INTCONbits.GIE = 0;          // Set global interrupt bit
```

```
		//T0CON = 0b00000011;					// Prescale by 16, time required x16,
bit 5 set to 0 to enable PSA (prescale)
   T0CON = 0b00001000;		// No prescale, timer increments at 8Hz, timer overflow at 65536
   TMR0H = 0;			// Clear timer count
   TMR0L = 0;
   T0CONbits.TMR0ON = 1;		// Start timer
		OpenI2C( MASTER, SLEW_OFF);
		SSPADD = 0x3F;
}
//
// end Init ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
//



/*
 * Write an individual LCD segment.
 *
 * @param segOrd The segment ordinal.  Between 0 and 22.
 *
 * @param state Whether to turn the segment dark (true) or clear (false).
 */
void segWrt(unsigned char segOrd,  unsigned char state) {

		unsigned char bitSelect;
		unsigned char *LCReg;

		if (segOrd > 23) return;
		LCReg = (unsigned char *)&LCDDATA0 + (segOrd >> 3);
		bitSelect = 1 << (segOrd & 0x07);
		if (state) *LCReg  |=  bitSelect;			// Segment on
		else *LCReg &= ~bitSelect;				// Segment off
}
//
// end segWrt ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
//




/*
 * FMwrite() -  Write a two byte word to the FM module.  The new
 * register contents are obtained from the image bank.
 *
 * @param adr The address of the register in the FM module that needs
 * to be written.
 *
 *
 * @return XS on success or XF on error.
 *
 */
unsigned char FMwrite(unsigned char adr) {

		unsigned int  regstr;
		unsigned char firstByt;
		unsigned char secndByt;
		unsigned char rpy;
```

```
        firstByt = regImg[adr] >> 8;    //Shift away low byte data and store into firstByt
        secndByt = regImg[adr];

        StartI2C();                                     // Begin I2C communication
        IdleI2C();

        // Send slave address of the chip onto the bus
        if (WriteI2C(FMI2CADR)) return XF;
        IdleI2C();
        WriteI2C(adr);                          // Adress the internal register
        IdleI2C();
        WriteI2C(firstByt);                     // Ask for write to FM chip
        IdleI2C();
        WriteI2C(secndByt);
        IdleI2C();
        StopI2C();
        IdleI2C();
        return XS;
}
//
// end FMwrite ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
//
//



/*
 * FMread - Read a two byte register from the FM module.
 *
 * @param regAddr The address of the register in the module that needs
 *      to be read.
 *
 * @param data Where to store the reading.
 *
 * @return XS on success or XF on error.
 *
 */
unsigned char FMread(unsigned char regAddr, unsigned int *data) {

        unsigned char firstByt;
        unsigned char secndByt;

        StartI2C();                                     // Begin I2C communication
        IdleI2C();                                      // Allow the bus to settle

        // Send address of the chip onto the bus
        if (WriteI2C(FMI2CADR)) return XF;
        IdleI2C();
        WriteI2C(regAddr);                      // Address the internal register
        IdleI2C();
        RestartI2C();                           // Initiate a RESTART command
        IdleI2C();
        WriteI2C(FMI2CADR + DEVRD);   // Ask for read from FM chip
        IdleI2C();
        firstByt = ReadI2C();             // Returns the MSB byte
        IdleI2C();
        AckI2C();                                       // Send back Acknowledge
        IdleI2C();
```

```
	secndByt = ReadI2C();		// Returns the LSB of the temperature
	IdleI2C();
	NotAckI2C();
	IdleI2C();
	StopI2C();
	IdleI2C();
	*data = firstByt;
	*data <<= 8;
	*data = *data | secndByt;

	return XS;
}
//
// end FMread ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
//



/*
 * FMready - See if the FM module is ready.
 *
 * @param rdy Where to store the busy/ready status.  Will become
 * non-zero if the chip is ready, zero if busy.
 *
 *
 *
 * @return XS on success or XF on error.
 *
 */
unsigned char FMready(unsigned int *rdy) {

	unsigned int sts;

	if (FMread(FMCHIPSTSADR, &sts)  != XS) return XF;
	sts &= FMASKSTATUS;
	*rdy = sts ? TRUE : FALSE;
	return XS;
}
//
// end FMready ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
//


/*
 * FMinit() -  Initialise the FM module.
 *
 *
 * @return XS on success or XF on error.
 *
 */
unsigned char FMinit() {

	unsigned char ad;
	unsigned int dat;

	// Copy default FM register values to the image set -
	for(ad = 0; ad < 18; ad++) regImg[ad] = regDflt[ad];
```

```
	dat = regImg[0];	//dat = 0xFFFF
	regImg[0] &= ~1;	//regImg[0] = 0xFFFE
	if (FMwrite(0) != XS) return  XF;
	for(ad = 1; ad < 18; ad++) {
		if (FMwrite(ad) != XS) return XF;
	}

	regImg[0] = dat | 1;	//regImg[0] = 0xFFFF
	if (FMwrite(0) != XS) return XF;
	dly(20);
	while (FMready(&dat), !dat) dly(2);	//Ask if AR1010 is ready or not
	return XS;
}
//
// end FMinit ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
//
//




/*
 * FMfrequenc(f) -  Tune the FM module to new frequency.
 *
 *
 * @param f The new frequency as a multiple of 100 kHz.
 *
 * @return XS on success or XF on error.
 *
 */
unsigned char FMfrequenc(unsigned int f) {

	unsigned int dat;
	unsigned int cn;		// AR1010 channel number

	cn = f - 690;

	// NB AR1010 retunes on 0 to 1 transition of TUNE bit -
	regImg[2] &= ~FMASKTUNE;	//Set Tune bit to 0 in AR1010
	if (FMwrite(2) != XS) return XF;
	regImg[2] &= 0xfe00;		//Clear Channel bits
	regImg[2] |= (cn | FMASKTUNE);	//Write new channel and set tune bit to 1 into regimg
	if (FMwrite(2) != XS) return XF;	//write regimg into ar1010
	do {
		dly(2);
		if (FMready(&dat) != XS) return XF; //Ask if AR1010 is ready or not
	} while (!dat);

  return XS;
}
//
// end FMfrequenc ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
//
//


/*
 * FMvers - Obtain the FM chip version.
```

```
 *
 * @param vsn Where to store the version number.  Will become
 * 0x65B1 for vintage 2009 devices.
 *
 * @return XS on success or XF on error. *
 */
unsigned char FMvers(unsigned int *vsn) {
        if (FMread(FMCHIPVERSADR, vsn)  != XS) return XF;
        return XS;
}
//
// end FMvers ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
//


/*
 * FMid - Obtain the FM chip ID.
 * * @param id Where to store the ID number.  Will become
 * 0x1010 for AR1010 devices.
 *
 * @return XS on success or XF on error.
 */
unsigned char FMid(unsigned int *id) {

        if (FMread(FMCHIPIDADR, id)  != XS) return XF;
        return XS;
}
//
// end FMid ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
//




/*
 * SeekChannel() -  Find the next channel with suitably strong signal.
 *
 * @param up Set to non-zero for next channel up,
 *  zero for preset down.
 *
 * @return New found channel if it is valid, otherwise do not tune.
 *
 */
unsigned int SeekChannel(unsigned int channel, unsigned int dir) {

   segWrt(23, TRUE);
   unsigned int newchannel;
   dir <<= 15;
   unsigned int dat;

   regImg[1] |= FMASKMUTE;     //Set hmute bit
   if(FMwrite(1) != XS) return XF;

   regImg[2] &= ~FMASKTUNE;    //clear tune bit
   if(FMwrite(2) != XS) return XF;

   regImg[3] &= ~FMASKSEEK;    //clear seek bit
   if(FMwrite(3) != XS) return XF;
```

```
    regImg[3] |= dir;          //set SEEKUP bit
    regImg[3] |= 0x2000;
    regImg[3] |= 0x0008;
    if(FMwrite(3) != XS) return XF;

    regImg[3] |= FMASKSEEK;     //enable SEEK bit
    if(FMwrite(3) != XS) return XF;

    do {
       dly(2);
       if (FMready(&dat) != XS) return XF; //Ask if AR1010 is ready or not
    } while (!dat);

    segWrt(23, FALSE);
    regImg[1] &= ~FMASKMUTE;    //clear hmute bit
    if(FMwrite(1) != XS) return XF;

    if(FMread(FMCHIPSTSADR, &newchannel) != XS) return XF;
    regImg[3] &= ~(0x8000);
    if(FMwrite(3) != XS) return XF;
    newchannel = newchannel >> 7;
    newchannel += 690;

    if(newchannel <= 1080 && newchannel >= 875){
       FMfrequenc(newchannel);
       showFreq(newchannel);
       return newchannel;
    }
    else{
       return channel;
    }
}
//
// end nextChan ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
//
//

/*
 * ManualChan() -  Manually tuning channel, increasing/decreasing by 0.01 MHz each time.
 *
 * @param up Set to non-zero for increasing channel by 0.1 MHz,
 *  zero for 0.1 MHz down.
 *
 * @return XS on success or XF on error.
 *
 */
unsigned int ManualChan(unsigned int channel, unsigned char dir) {

    if(dir == TRUE){
       if(channel < 1080){
          channel++;
          FMfrequenc(channel);
          showFreq(channel);
       }
    }
    else if(dir == FALSE){
       if(channel > 875){
```

```
            channel--;
            FMfrequenc(channel);
            showFreq(channel);
        }
    }
    return channel;
}

//
// end ManualChan ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
//
//

/*
 * showFreq() - Display the frequency that the receiver chip is set to.
 */
void showFreq(unsigned int f) {
    int ones, tens, hundreds;
    int i, j;
    clrscn();
    segWrt(22, TRUE);
    ones = f % 10;
    ones = LCDDigits[ones];

    tens = (f/10) % 10;
    tens = LCDDigits[tens];

    hundreds = (f/100) % 10;
    hundreds = LCDDigits[hundreds];

    if(f >= 1000)
        segWrt(21, TRUE);

    for(i = 0; i <= 20; i++){
        for(j = 0; j <= 6; j++){
            if (((ones >> j) % 2) == 1)
                segWrt(j, TRUE);
            else
                segWrt(j, FALSE);
        }

        for(j = 0; j <= 6; j++){
            if (((tens >> j) % 2) == 1)
                segWrt(j + 7, TRUE);
            else
                segWrt(j + 7, FALSE);
        }

        for(j = 0; j <= 6; j++){
            if (((hundreds >> j) % 2) == 1)
                segWrt(j + 14, TRUE);
            else
                segWrt(j + 14, FALSE);
        }
    }

    return;
}
```

```
//
// end showFreq ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
//

/*
 * presetFrequency() - Set the FM chip to preset frequencies in rotation.
 * @return return the set channel back to main().
 */
unsigned int presetFrequency(unsigned int channel, unsigned int dir){
   int counter;

   switch(dir){
      case TRUE:
         for(counter = 0; counter <= 3; counter++){
            if(channel < presetFreq[counter] || channel == presetFreq[3]){
               FMfrequenc(presetFreq[counter]);
               showFreq(presetFreq[counter]);
               return presetFreq[counter];
            }
         }
         break;

      case FALSE:
         for(counter = 3; counter >= 0; counter--){
            if(channel > presetFreq[counter] || channel == presetFreq[0]){
               FMfrequenc(presetFreq[counter]);
               showFreq(presetFreq[counter]);
               return presetFreq[counter];
            }
         }
         break;
   }
   return channel;
}
//
// end presetFrequency ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
//

/*
 * showFreq() - Display the volume that the receiver chip is set to.
 */
void showVol(unsigned int vol) {
   int ones, tens;
   int i, j;
   clrscn();
   ones = vol % 10;
   ones = LCDDigits[ones];

   tens = (vol/10) % 10;
   tens = LCDDigits[tens];

   for(i = 0; i <= 13; i++){
      for(j = 0; j <= 6; j++){
         if (((ones >> j) % 2) == 1)
            segWrt(j, TRUE);
         else
            segWrt(j, FALSE);
      }
```

```
      for(j = 0; j <= 6; j++){
         if (((tens >> j) % 2) == 1)
            segWrt(j + 7, TRUE);
         else
            segWrt(j + 7, FALSE);
      }
   }

   return;
}
//
// end showVol ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
//

/*
 * VolChange() - Increase/decrease volume by one step depending on button input.
 */
unsigned int VolChange(unsigned int vol, unsigned char dir){ //should be correct
   switch (dir) {
      case TRUE:
         if(vol < 18){
            vol++;
            regImg[3] &= ~FMASKVOL1;
            regImg[14] &= ~FMASKVOL2;
            regImg[3] |= (Vol_data[0][vol] << 7);
            regImg[14] |= (Vol_data[1][vol] << 12);
            if(FMwrite(3) != XS) return XF;
            if(FMwrite(14) != XS) return XF;
            showVol(vol);
         }
         else
            vol = 18;
         break;

      case FALSE:
         if(vol > 0){
            vol--;
            regImg[3] &= ~FMASKVOL1;
            regImg[14] &= ~FMASKVOL2;
            regImg[3] |= (Vol_data[0][vol] << 7);
            regImg[14] |= (Vol_data[1][vol] << 12);
            if(FMwrite(3) != XS) return XF;
            if(FMwrite(14) != XS) return XF;
            showVol(vol);
         }
         else
            vol = 0;
         break;

      default: break;
   }
   return vol;
}
//
// end VolChange ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
//
```

```
void main(void) {

        unsigned char evt = 10;
        unsigned int ui;
   unsigned int vol = 18;
   unsigned int frequency = 964;
   int i, counter = 0;
        dly(20);
        Init();
        FMvers(&ui);                                                    // Check we
have comms with FM chip
        FMinit();
   vol = VolChange(vol-1, TRUE);
   showFreq(frequency);
   FMfrequenc(frequency);

   for(;;){
                evt = butnEvent();
                switch (evt) {
      case 1 : frequency = SeekChannel(frequency, TRUE); break;
      case 2 : frequency = SeekChannel(frequency, FALSE); break;
      case 3 : frequency = ManualChan(frequency, TRUE); break;
      case 4 : frequency = ManualChan(frequency, FALSE); break;
      case 5 : vol = VolChange(vol, TRUE); dly(1000); showFreq(frequency); break;
      case 6 : vol = VolChange(vol, FALSE); dly(1000); showFreq(frequency); break;
      case 7 : frequency = presetFrequency(frequency, TRUE); break;
                        case 8 : frequency = presetFrequency(frequency, FALSE); break;
      default: break;
    }
        }
}
//
// end main ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
//
```