Voice Activated Remote Control System

Post Mortem

Project Team:	Alex Cheng Jeff Liu Gary Liaw Roger Lum Colin Ng Jason I-chih Wang
Contact Person:	Roger Lum fls-340@sfu.ca
Submitted to:	Dr. Andrew Rawicz Steve Whitmore School of Engineering Science Simon Fraser University

Issued Date:	December 19, 2002

Freedom • Life Systems

Executive Summary

The VoiceIR is a voice-activated remote control system. This product allows those physically disabled to enjoy the convenience of remote controls. The versatility and simplicity of this product is unique among current solutions.

The post-mortem report reviews the composition of the product and traces over the technical issues faced in developing each aspect of the product. The solutions to each technical issue are presented and its impact on the product development. It also provides a design of our finished product, with emphasis on the aspects that are different from the original design specifications.

As a reflection on the prototype, this report also presents the final cost of the product, estimated production cost, and an actual timeline. Team dynamics and project organization are discussed as well as a plan for future development provided.

Table of Contents

Freedom Life Systems

List of Figures

List of Tables

1 Introduction

The VoiceIR is a voice activated remote control system. This product allows disabled individuals to enjoy the convenience of remote controls through trainable voice commands. The user is able to control any device that uses infrared (IR) remotes simply by talking into a wireless microphone. Through the versatility of the design, the user is able to control devices located anywhere in the house.

1.1 Scope

This document describes the technical problems encountered in development of a workable prototype, and details the solutions that we used to surpass these issues.

The post-mortem report also provides some design depictions, especially those sections that have changed or have been added to the original design specifications. In the earlier written design specifications, we presented a theoretical design of our product, with our chosen components and our understanding to the best of our knowledge at that time. The post-mortem provides a detailed design of the actual finished prototype, including the modifications made to accommodate technical issues and improvements made from our increased technical knowledge of the components included in our project. In terms of cost, there is a cost analysis, describing the final costs to the product, and the estimated production cost.

As part of the reflection on the finished project, team dynamics and project organization are discussed. To improve the project, future work to be done is presented as well.

1.2 Acronyms

CCU	central control unit
EEPROM	electronically erasable programmable read only memory
FRS	family radio service
IR	infrared
LCD	liquid crystal display
LED	light emitting diode
RF	radio frequency

1.3 Intended Audience

This document is intended for project management to evaluate the progress of the product team after the 13 week prototype-development stage. It also provides a way to help measure the success of the project team, as well as make decisions regarding the future of the project.

2 System Overview

A system overview of the VoiceIR product is presented in Figure 2.1. This figure shows a user engaging the VoiceIR system to control various devices with voice commands. The purpose of this overview is to allow the reader to gain familiarity with the organization of functionality of the product.

The VoiceIR system consists of three modules. The user first needs to train the VoiceIR system at the central control unit (CCU). The CCU will store the user's voice and the corresponding remote control signals. After initial training, the user simply speaks commands into the wireless microphone in order to control target devices. The wireless microphone sends the voice commands to the central control unit for processing. The CCU interprets the voice signal and looks up the appropriate IR signal. A command signal is sent to the IR Module. The IR Module receives the signals and transmits the appropriate IR signal to target devices. To control numerous devices, the user may place multiple IR Modules in different locations in front of each target device.

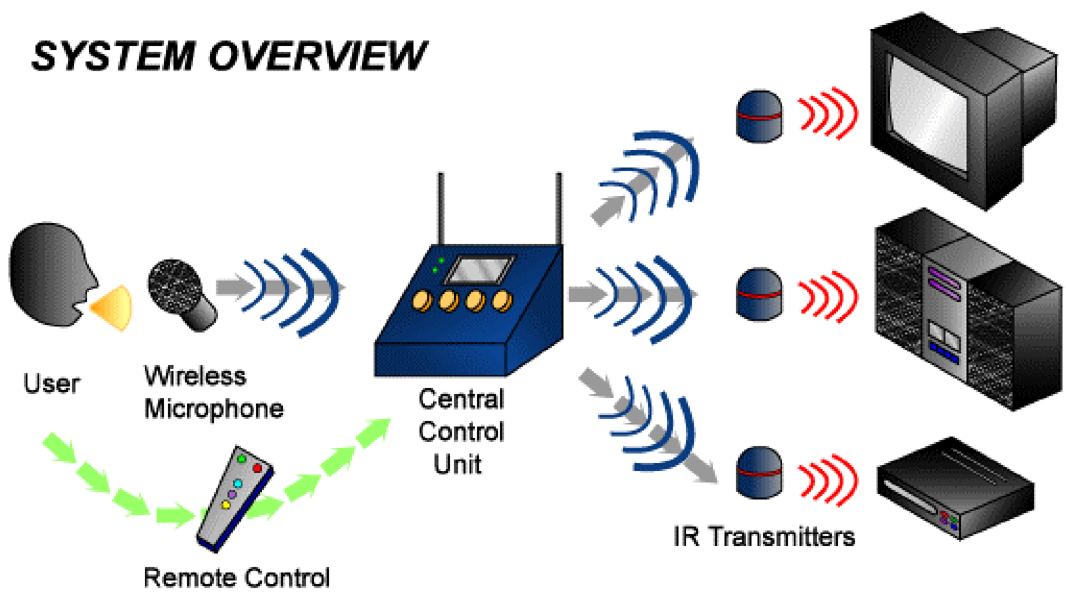

Figure 2.1: System Overview

3 Wireless Microphone

Originally, we had planned to use a wireless microphone FM transmitter that one of our group members had made in high school and use a pocket radio as a FM receiver. However, we had a lot of problems with this combination. We were unable to get stable tuning from our inexpensive pocket radio since the tuning knob on the radio was not very consistent. The inaccurate tuning meant that the voice signal out of the FM receiver had too much static for the voice recognition chip. As an attempted solution, we purchased a hobby FM receiver kit that we hoped would be more stable for tuning. This receiver kit used a variable inductor for tuning and we were unable to get a consistent frequency between the wireless microphone and the FM receiver.

A commercial wireless microphone transmitter and receiver set would cost too much and the voice quality it provides is not required for our product. We eventually settled on a pair of FRS radios, which can be tuned to a specific channel. The voice quality was sufficient for our voice recognition chip.

4 Central Control Unit

The central control unit is the main component of the product. The following sections discuss the technical issues encountered while developing this component, as well as some details about design changes.

4.1 Technical Issues

4.1.1 VoiceDirect Interface

The VoiceDirect does not provide as many status pins as we had originally anticipated to allow the HC11 to display complete status messages from the VoiceDirect to the LCD. For example, the VoiceDirect has only one error pin, which does not reveal information as to what type of error occurred. The VoiceDirect, however, does have a speaker through which it delivers messages to the user, though it cannot be readily interfaced with the HC11. The VoiceDirect also lacks pins to tell the user how many words are currently stored in memory.

We decided to make use of the speaker that came with the VoiceDirect to convey messages to the user, as this will save valuable memory space on the HC11 by eliminating the need to process messages and displaying it on the LCD.

The number of words recorded is tabulated by the HC11. This is important because the HC11 needs to associate a word with a signal sequence. The HC11 also needs to know when to inactivate the voice train sequence when 14 words have been trained.

As we were checking for possible error status for VoiceDirect, we noticed that the "training complete" audio message indicates an error. This error occurs in several situations. One situation is when the user keeps silent when VoiceDirect asks the user to say a word. When this error occurs, an error pin is toggled and the voice train process is aborted. Compared to all other error types, this "training complete" error does the voice train process only once and does not asks the user to repeat like all other error situations. This "training complete" error also occurs when all 15 words have been programmed onto VoiceDirect. In this case, the error does not indicate that word training failed, but it indicates that word train has succeeded and all 15 words have been used. From this example, we cannot distinguish whether a word has been successfully trained based on one error pin.

Based on our observation of the error situations, the only solution other than reading in the speaker feedback of VoiceDirect is to check the error pin for a set amount of time and decide that a word has been successfully obtained if the error pin does not get pulled high. We experimentally determined a length of time where if an error is to occur, it must occur within that time. We then used that measured time and incorporated it in the HC11 program. If an error is received within that time, then we ask the user to repeat training. If no error is detected, then we determine the training to be successful.

4.1.2 Infrared Signals

We were unable to fully characterize infrared signals until we could get our infrared transceiver working, which was accomplished after the design specifications had been written. The infrared signals turned out to be more complicated than we had anticipated. The figure below shows a typical infrared sequence.

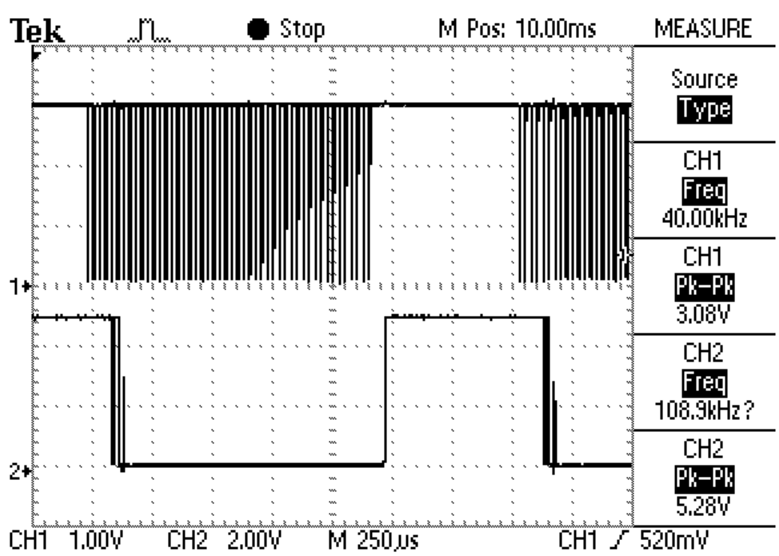

Figure 4.1: Infrared Signal (Unfiltered and Filtered)

The top signal in the figure above is an unfiltered IR signal. The signal can be interpreted as a digital signal, modulated onto a carrier. The frequency of the carrier is between 30 and 40 kHz. The precise carrier frequency is not critical to the operation transmitting an infrared signal. To save memory, we only need to store the outer envelope of the infrared signal. A filter was designed to extract the envelope. The design synopsis of this module describes the functionality of the filter. The lower signal in Figure 4.1 shows the filtered output.

The HC11 is able to toggle a pin at the required frequency of approximately 35 kHz. Our choice of RF transmitters/receivers can only transmit at 5000 bps, which is less than 35 kHz. Only the envelope of the IR signal is sent to the IR Modules, and the modulation is redone there. The details are presented in the design of the IR Module.

Another complication arose from the different timing of IR signals from different brand of remote controls. We originally thought there was a constant minimum signal length. We had to program logic into the HC11 to detect the different timing before an IR signal can be sampled.

4.1.3 RF Chips

We spent a lot of time trying to get our RF chips to work. The chips are designed to be surface mounted. We soldered leads on the chips and plugged it into a breadboard, but were unable to get it to transmit. It was very difficult to debug the RF chips with their limited amount of pins. We were concerned with stray capacitance and moved the circuit onto a perforated board. We also tried using ground-planes and better antennas to attempt to get the chip to work. In the end, we discovered that during the soldering process, we must have applied heat too long and burned out the chips.

4.2 Design Synopsis

This section describes the design of the Central Control Unit as evident in our prototype, with emphasis on the aspects that are different from our original design specification.

The central component in this module is the HC11 microcontroller. It controls the interfacing between all components within the CCU and the menu system that the user interacts with to train the device. The complete source code for the HC11 can be found in Appendix 1.

4.2.1 Voice Chip Interface

The VoiceDirect is connected to the HC11 via its four buttons and the error status pin, as well as its 8 output pins. The 8 output pins are passed through an encoder to reduce the number of pins that it uses on the HC11 ports. Although the HC11 chip has enough pins for direct connection, we observed that these 8 pins can easily be encoded to a 4-bit number. This saves valuable memory space from being used to get data from port pins and convert them to a 4-bit number, which will then be used to match the proper IR sequence.

To control the voice chip, HC11 needs to simulate button pushes. We noticed that the button pins connected to the voice chip are active low, therefore we needed to pull the pins to 0 V. However, by configuring HC11 pins to output pins, problem arose because the internal circuitry in the voice chip drives a pin to high. This creates the situation of connecting two outputs together.

The solution to properly simulate button pushes is to configure the HC11 pins to input by default. Whenever there is a need to simulate button pushes, we then reconfigure pins to output only for a short period of time (~200ms). After triggering the 0V status, we then reset the pins back to input only and let the voice chip pull up the voltages by itself.

4.2.2 Filter Design

The purpose of the filter is to extract the envelope of the IR signal. In order to filter out the carrier frequency (30~40 kHz) from the baseband (at a maximum of 800 Hz) we had to first low-pass filter the signal, then use a comparator to reconstruct the shape of the digital pulses.

The filter stage consists of a non-inverting amplifier and a low pass filter. The cutoff frequency of the filter is set at 1kHz. An amplifier is used to amplify the signal to offset the attenuation by the filter so that there is a bigger margin of error for setting the threshold voltage of the comparator. At the comparator side, the threshold voltage is set to achieve the highest duty cycle in order to most accurately reproduce the IR pulses. The comparator also acts as a DC shifter, shifting the 10.5 V output of the filter down to 5 V with 0 V corresponding to an IR pulse. The circuit diagram of the filter can be found in Appendix 2.

4.2.3 IR Signal Storage

We need to have the IR signal stored in memory maintained when the Central Control Unit is unplugged. We originally wanted to use a backup battery to keep the IR signals stored in RAM intact, but have since implemented a better solution.

During IR training, the sequence will initially be stored in RAM, and then copied over into the EEPROM. The IR sequence is permanently stored in EEPROM and would not erase when the unit is unplugged. The overhead associated with moving the signal from RAM into EEPROM is approximately 300 ms.

4.2.4 Power Supply

In order to supply the appropriate voltages to each device in the Central Control Unit, we used LM317 adjustable voltage regulators. In our design specifications, we had resistive voltage dividers, but this does not maintain steady voltages with fluctuations in the power supply. We used one voltage regulator for each device and powered all the voltage regulators with a 12 V AC adaptor.

Appendix 3 provides detail of the how the voltage regulator works, including the relationship for estimating output voltage.

5 IR Module

The IR Module is placed in front of the device the user wishes to control. It accepts incoming signals from the Central Control Unit and modifies it to be an output for IR transmitting.

The figure below gives an overview of the IR Module.

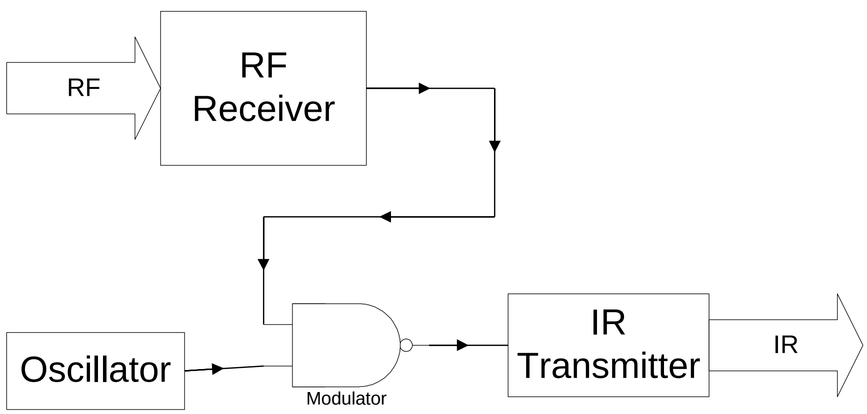

Figure 5.1: IR Module Overview

The next sections discuss the technical issues faced in developing this module and some design details.

5.1 Technical Issues

5.1.1 IR Complications

As stated in Section 4.1.2, the IR signal is more complicated than we had originally thought. The RF transmitter/receiver pair is unable to transmit the carrier of the IR signal, because of the limited transmission rate of the RF pair.

However, the carrier frequency is not critical, as long as it is between 30 and 40 kHz. We recreated the carrier at the IR Module and modulated it with the envelope of the IR signal to create the required signal to drive the IR transmitter. Originally, we planned to use a

PIC micro-controller to module the signal. After some development with the chip, we thought of a simple solution using an oscillator circuit and some logic gates.

5.2 Design Synopsis

5.2.1 Oscillator Circuit

A timer circuit is used to create oscillation at approximately 37 kHz. This circuit adds oscillation to the incoming RF signal to re-create the entire IR signal.

5.2.2 Entire Module

A detailed circuit diagram of the IR Module can be found in Appendix 4.

The oscillation from the oscillator circuit is modulated with the digital RF signal through a logic gate, and the corresponding output is used to drive the IR transmitter. Because the carrier frequency does not have to be precise, we are able to use the same carrier circuit for any IR signal.

6 Individual Contributions

The section below lists the contributions made by each person to the project.

6.1 Alex Cheng

Alex was first responsible in investigating the capabilities of several microcontrollers. Upon picking the Motorola 68HC11 series, he then took charge of the program design of the HC11 coding. He was also an important contributor in determining the functional and design aspects of the VoiceIR.

The functional modules of the assembly code include LCD control, button control, IR capture, VoiceDirect control, IR match, RF send, EEPROM erasure and IR storage. To accomplish the tasks within a limited EEPROM space, he also needed to find references on special techniques in capturing signals and determining pulse widths.

As other members complete their necessary parts for the final product, he then incrementally tested and interfaced each component and made sure they work as designed. When all components were interfaced, he did most of the software testing and debugging.

6.2 Gary Liaw

Gary was in charge of handling all financial aspects of the project team, which including keeping track of spending and purchasing parts. His initial task was to assemble and research the voice recognition module, the VoiceDirect 364. Gary's knowledge of filters and op-amp circuits were extremely helpful in designing the IR filter. As well, Gary helped with all the hardware interfacing, as he was very knowledgeable about interfacing issues.

Gary also had the role of the devil's advocate and offered suggestions and criticism of most group decisions. This is done through keeping an objective view as well as offering constructive criticism.

At the end of the project, Gary did the final editing of the presentation.

6.3 Jeff Liu

Jeff was initially responsible for the RF chips, soldering and testing the chips. He then worked on the LCD, including writing the HC11 code for controlling the LCD as well as interfacing the LCD with the HC11. He assisted Gary in implementing the IR filter and investigated voice-activation on the FRS.

Jeff also investigated and implemented the encoder circuit used in the interfacing between the voice chip's status lines and the HC11.

Near the end of the project, Jeff spent many hours on cleaning up all the circuits and arranging the Central Control Unit to its final form.

6.4 Roger Lum

Roger was the group leader and was in charged of creating timelines and assigning intermediate tasks. He also helped with all purchasing decisions and spent some time helping to characterize IR signals. He also worked on the PIC chip programming, which was later abandoned, and designed and made the IR Modules. This involved researching and creating the oscillator circuit.

6.5 Jason Wang

Initially, Jason was assigned to the task of developing a wireless microphone system. He researched possible solutions and tried initially to incorporate his high-school wireless microphone project. He was responsible for purchasing FM receivers and trying to use them as a receiver for the wireless microphone. In the end, the FRS solution was used instead of FM receivers because of the sound quality.

Jason also had extensive soldering experience and provided continuous soldering and assembly support for many parts and devices, including the IR transceiver and the RF chips. He also was instrumental in debugging and finally getting the RF circuits to work.

Jason also worked on constructing the voltage regulator circuits used to power different components in the CCU from a single AC adapter. He also performed the first examinations and assisted in characterizing IR remote signals.

Jason also worked on the PIC chip programming, which was later abandoned. Subsequently he participated in the PIC chip's alternative.

At the end of the project, Jason helped create the presentation slide shows and make the casing for the Central Control Unit.

6.6 Colin Ng

Colin's initial research focused on infrared transceivers. He built the supporting circuitry so the IR transceivers were able to receive and transmit IR signals. He was responsible for attempting to acquire sample components of all the parts.

Colin also worked on the voltage regulator circuits used to power different components in the CCU from a single AC adapter. He also assisted in a lot of the soldering.

At the end of the project, Colin was involved in developing and creating the presentations for the group presentation. In addition, Colin also aided in the design and creation of the prototype cases.

7 Group Dynamics and Structure

Our group functioned as a team of equal individuals, with one group leader to help make final decisions in case of disputes. The group leader determined the timeline and helped establish milestones with input from the entire group.

Besides a defined team leader, we also appointed a team recorder and a devil's advocate. The team recorder was instrumental in keeping track of all our decisions and providing minute meetings. The devil's advocate helped to rationalize our decisions and forced consideration of other alternatives. All decisions were made after a brief discussion and each group member had the opportunity to voice their opinion. However, for sake of not being bogged down by the decision-making process, the group leader has the final say even if there is not total group consent. Once a decision had been made, everyone accepts the decision.

With a larger group, there are always inherent difficulties with communication and keeping everybody satisfied. We kept regular minute meetings and continuously redefined the tasks that needed to be accomplished so everybody has clear idea of what needs to be done.

There were not any major group dynamic problems, except some minor issues near the end of the project when everyone was stressed and short-tempered. Overall, the group worked very well and the slight hierarchy in the group helped to keep the group working without any major stalls.

8 Project Organization

The project was split into multiple stages. As a group, we came up with a rough idea of how the product would work. We each researched a different technical aspect of the project and what kinds of components were required to accomplish each aspect.

Based on the individual research, we ordered the main components in our product. Each group member was in charge of a different component and did further research into how to interface the component and the major performance characteristics of the component.

All of the hardware development was done in parallel. Each person worked to get their individual component working with the appropriate supporting circuit and mounting it on perforated boards or soldering it to interface with a breadboard. We assigned one person to work on the HC11 software, which was continuously developed throughout the project timeline.

We started interfacing components as early as possible. Interfacing components created many issues and in the process, we discovered some issues which we had not foreseen, such as the complications involved with IR signals.

The figure below shows a time chart of the actual time spent on each component.

9 Budget

Table 9.1 below outlines our expected material cost for the prototype as well as the actual cost of the materials needed to build the prototype. The expected costs are taken from our Wighton Fund application, since the budget proposed in the proposal was submitted without the slightest idea of what was required.

Table 9.1: Prototype Cost

	Expected Cost	**Actual Cost**
Wireless Mic and Central Control Unit		
Microcontroller EVB	85	69
Voice Recognition Chip	90	77.5
RF Transmitter and antenna	30	16.36
IR Receiver	10	3
Wireless Microphone	30	60
LCD 16x4	40	36.52
Misc total	50	110.50
- breadboards		*35*
- encoder		*3.50*
- ac adapter		*12*
- speaker		*15*
- voltage regulators		*8*
- perf boards		*6*
- case (box + acrylic)		*30*
- other		*5*
Sub Total	**$335**	**$376.88**
IR Module		
RF receiver	35	21.80
IR transmitter	10	3
Misc	10	9.60
- perf boards		*4.50*
- case and battery holder		*3.60*
- NE555 timer		*0.50*
- other		*1*
Sub Total	**$55**	**$34.40**
Total	**$390**	**$411.28**

We spent more than the estimated cost to build the prototype mainly because we predicted our miscellaneous costs incorrectly. We overlooked, as well as underestimated the cost for, some miscellaneous items. For example, we failed to take into account the cost of the breadboards and AC adaptor, as well as the cost for the case. Other extra material costs include the unexpected cost of the speakers.

The cost of the VoiceIR prototype will be much more inexpensive in mass production. Table 9.2 outlines the cost of the prototype in production including some optimization. These optimizations include the use of PCB and using customized FRS module, without changing the functionality of the prototype. Cost of the items are in unit price when purchased in thousands.

Table 9.2: Estimated Prototype Production Cost

Item	Cost	
Wireless Mic and Central Control Unit		
RF tranmitter + antenna	11	
FRS module	30	
HC11	15	
VoiceDirect 364 Module	30	
IR Tranceiver	1	
LCD 16 x 4	23	
PCB	5	
Encoder	0.70	
AC adapter	8	
Speaker	2.50	
Voltage Regulators (LM317T)	2	
Misc (caps and resistors)	5	
Case (box + acrylic)	5	
Sub Total		**$138.20**
IR Module		
PCB	5	
NE555 Timer	0.30	
Case with Battery Holder	2	
RF Receiver	16	
IR Tranceiver	1	
Misc (caps and resistors)	0.50	
Sub Total		**$24.80**
Total (Wireless Mic, CCU & 1 IR module)		**$163**
Retail price		**$300**

We estimate that it will cost us $163 in material to manufacture each VoiceIR with one IR Module. Additional IR modules will cost us $25 each. The estimated retail price for the VoiceIR with one IR module is approximately $300 after taking into account of distribution and manufacturing costs. Additional IR Modules are estimated at $35 retail each.

Overall we spent a total of $648.52 in building the prototype. The extra spending outside of the building material can be attributed to our many trials with different wireless microphone solutions, burnt RF transmitter and receivers, and purchase of spare parts. We also spent quite a bit of money on perishable items such as batteries for our IR module and FRS, as well as spray paint for the case.

We received funding from various sources. We received $150 from the Engineering Student Society Endowment Fund (ESSEF) in exchange for our HC11 and Voice Direct EVBs at the end of the semester. We also received $310 from the Wighton fund. The sum of the two funds, combined with a total of $300 in individual contributions accounted for our entire budget.

10 Future Project Work

This section deals with future work and improvements that can be made to the VoiceIR, to bring it closer to a finished product.

10.1 IR Module Identification

In our design, each IR Module receives IR commands and just transmits it blindly. This would create problems if two devices being controlled use identical remotes.

Future work will have each IR Module identified with a unique tag, that way only the desired IR Module will transmit.

10.2 IR Signal Storage

In our existing product, IR signals are stored in the EEPROM of the HC11 chip. Currently, the HC11 only has 2 KB of EEPROM, which only allows storage of only 4 IR signals per voice command. This can easily be increased by adding external memory. We have available ports on the HC11 which can be used to interface this external memory.

10.3 Voice Recognition Chip

Currently, our voice recognition chip, the Sensory VoiceDirect 364 EVB, is very limited in its features. It only recognizes up to 15 words and there are not enough status pins, which forces us to use the speaker output. Ideally, we would prefer all status messages through the LCD. As well, individual words cannot be erased, only the entire memory. Because we are using an EVB, we are unable to use the voice chip in “slave” mode. There are more control and status options when the chip is in slave mode.

The voice recognition chip also has a low accuracy rate. It is very susceptible to background noise and usually requires multiple tries before it accepts a word.

By placing the chip in slave mode, we should have the ability to control it better as well as check up to 60 words.

Future project work on the voice recognition chip would be to add some noise filters to ensure the voice signal is clean and this should improve the accuracy of the voice chip. A new voice chip without an EVB should be used in “slave” mode, which will give better control of the chip. With the enhanced control, all status messages can be displayed on the LCD, instead of relying on the speaker output.

10.4 Wireless Microphone System

Currently, we are using a pair of FRS radios as a wireless microphone system. This was the most inexpensive system we could find that would still give us the required voice quality. Most commercial wireless microphone systems are too expensive, with the voice quality beyond what we require. The existing solution is not very elegant, since ideally the user will be wearing a headset connected to a small transmitter that can be clipped onto a belt. Have a headset will also keep the volume fairly consistent.

Future project work would involve researching the FRS technology and seeing whether it is feasible to purchase apply the technology in our own wireless microphone system, instead of buying a commercial pair of radios. This way, a more elegant and usable microphone system can be implemented.

10.5 HC11 Microcontroller

The HC11 currently is on an EVB. To reduce cost and space, the controller should be taken off the EVB.

10.6 PCB

The finished product should have all the circuits on printed circuit boards. This will further reduce the size of the product and ensure better connections between components.

Future project work would be to design a PCB layout and place all the components on a PCB.

10.7 IR Transceivers

Currently, our IR transceiver has a very limited range. It is only able to transmit a metre, which greatly restricts the distance the IR Module can be placed from the target device. An IR Module too close to the target device would probably restrict the placement of this module and this would be an inconvenience. We settled on our current IR transceiver because we were able to get free samples.

Future project work would be to investigate better IR transceivers and purchase better transceivers in the bulk quantity if the product is to be manufactured.

10.8 Usability

There are a lot of usability issues with the prototype. The buttons can probably be made larger in the finished product. As well, larger LCD, possibly colour, would make it easier to navigate through the menu. Usability is more of a concern regarding a final product than the prototype.

Currently, there is no feedback to the user that a voice command has been accepted. Instead, the user just keeps on repeating the voice command until it works. There needs to be some kind of notification, possibly through a speaker on the wireless microphone, to tell the user whether a voice command has been recognized or not.

There also needs to more of a focus in making the product more usable for those physically disabled. These could include people with speech impediments, which might interfere with voice recognition.

Appendix 1: HC11 Code

```
* Parallel I/O Port Registers *
PORTA EQU   $1000       ;PortA
PACTL EQU   $1026       ;Port A control reg
PORTC EQU   $1003       ;PortC
PORTB EQU   $1004       ;PortB
PORTCL      EQU   $1005       ;PortC control
DDRC  EQU   $1007       ;PortC direction
PORTD EQU   $1008       ;PortD
DDRD  EQU   $1009       ;PortD direction
PORTE EQU   $100A       ;PortE
SCCR2 EQU   $102D       ;SCI control register 2
PPROG EQU   $103B       ;EEPROM control reg
PBROT EQU   $1035       ;EEPROM block protection reg
TCNT  EQU   $0E
TIC1  EQU   $10
TIC2  EQU   $12
TIC3  EQU   $14
TOC1  EQU   $16
TOC2  EQU   $18
TOC3  EQU   $1A
TOC4  EQU   $1C
TOC5  EQU   $1E
TCTL1 EQU   $20
TCTL2 EQU   $21
TMSK1 EQU   $22
TFLG1 EQU   $23
TMSK2 EQU   $24
TFLG2 EQU   $25
TRAM  EQU   $0030

* Constants *
CMDREG      EQU   $AF         ;Set Enable = 0 and RS = 0 for writing
instruction
DATAREG     EQU   $10         ;Set RS = 1 and for writing data
ENABLE      EQU   $40         ;Set Enable = 1 for LCD write

LINE1 EQU   $80         ;Line 1 starting address
LINE2 EQU   $C0         ;Line 2 starting address
LINE3 EQU   $90         ;Line 3 starting address
LINE4 EQU   $D0         ;Line 4 starting address

IRBASE      EQU   $FE20
;IR1  EQU   $FE00       ;IR sequence address, 32 bytes each seq
;IR15 EQU   $FFC0

COUNT EQU   $29         ;counter
COUNT2      EQU   $2A         ;2nd counter
PTIME EQU   $2B         ;2B and 2C
HPTIME      EQU   $2D         ;2D and 2E
;notice that the following overlaps the RAM after TRAM.
```

```
;these RAM are used only during EEPROM programming/erasing, so it's
okay
RTIME EQU   $2F         ;Rise time (2bytes)
DTIME EQU   $31         ;down time (2bytes)
TOUT  EQU   $33         ;timeout counter
OCMODE      EQU   $34
LCDTEM1     EQU   $35         ;temporarily store the return address
after LCD display
LCDTEM2     EQU   $37
LHPTIME     EQU   $39         ;used for input capture
IRRPT EQU   $3A
IRCNT EQU   $3C

* The address for the IR storing, EEPROM programming protocol *
IRNUM EQU   $00         ;Number of IR signals
IRSEQ EQU   $01         ;Actual IR sequence, from $01 - $1F
VCNUM EQU   $20         ;Voice command number
EEPS  EQU   $21         ;EEPROM programing start burn-value
NCYCLE      EQU   $22         ;Number of cycles to run the
program/erase
INITADD     EQU   $23         ;Initial address for the prog/erase
procedure, from $23-$24
INCV  EQU   $25         ;Increment value for index register X
EEPE  EQU   $26         ;EEPROM programing end burn-value
SRCADD      EQU   $27         ;Source address - used only for
programing, from $27-$28



************************* MAIN *************************
********************************************************
* Main Program *
      ORG   $F800       ;EEPROM starting address
BEGIN:
      LDS   #$FF        ;Initialize stack pointer
      CLR   PBROT       ;Clear EEPROM block protect
      LDAB  #$E0        ;set up port C for 3 output, 5 input
      STAB  DDRC
      CLR   PORTC       ;for debug only

      LDX   #SCCR2            ;turn off SCI and use Port D
      BCLR  0,X,$08
      CLR   DDRD        ;all input for port D

      LDX   #PACTL            ;setup port A pin 7
      BSET  0,X,$80           ;setup pin 7 for output
      SEI               ;Disable interrupt

LCD:  JSR   LCD_INIT          ;Initialize LCD

M1:   LDX   #STR_READY  ;LCD display
      PSHX
      LDX   #STR_BLANK
      PSHX
```

```
      LDX   #STR_TRAIN
      PSHX
      LDX   #STR_CLRMEM
      PSHX
      LDAA  #$4
      JSR   DISPLAY            ;end of LCD display
      LDX   #$1000             ;loads X with regbase
      JSR   LDELAY
MAIN: LDAA  PORTE
      TSTA
      BNE   ISR
      BRA   MAIN        ;Wait for button push
*******************************************************
*******************************************************




*================ Interrupt Service Routine =================*
ISR:
;     LDAA  PORTE       ;load port E value to ACCA
      LSRA
      BCS   ABORT       ;Abort is pushed, nothing happens at this
stage.
      LSRA
      BCS   TRAIN       ;Train voice command is pushed
      LSRA
      BCS   CLEARALL    ;Clear all voice memory is pushed
      LSRA
      JMP   IR_MATCH

ABORT:
;     JMP   IR_MATCH    ;use if voice fails
      BRA   M1

TRAIN:
      JSR   VT          ;jump to Voice Train subroutine
      JSR   LDELAY
;     JSR   IRCAP       ;use if voice fails
      BRA   M1

CLEARALL:
      PSHX              ;LCD display
      LDX   #STR_CLRMEM
      PSHX
      LDX   #STR_NO
      PSHX
      LDX   #STR_BLANK
      PSHX
      LDX   #STR_YES
      PSHX
      JSR   DISPLAY
      PULX              ;end of LCD display
```

```
      JSR   CLRMEM            ;jump to clear memory subroutine
      BRA   M1
*=========== End of Interrupt Service Routine =================*




*=================== Voice Training =======================*
VT:
      LDAA  VCCOUNT
      LDAB  #$1
      STAB  COUNT2            ;ACCB is used as success/failure flag
      CMPA  #$E         ;check if VCCOUNT is already 14
      BLO   VT2         ;if lower, then do process. If not, quit
      RTS

VT2:  LDX   #$1000            ;regbase
      TST   VCINIT
      BNE   TREDO1            ;if initial word is there, then go to
train mode

      PSHX              ;LCD display
      LDX   #STR_GW
      PSHX
      LDX   #STR_ABORT
      PSHX
      LDAA  #$2
      JSR   DISPLAY
      PULX              ;end of LCD display

GREDO:      JSR   GMODE       ;jump to gate word train mode
      JSR   WFE         ;jump to Wait For Error
      BRSET $A,X,$1,TABORT    ;if ABORT button is held down, then quit
                        ;was TABORT
      JSR   LDELAY            ;THE MAGIC
      TST   COUNT2            ;check if B is set or cleared
      BNE   GREDO       ;if B is set, then redo gate word
      JSR   VRESET            ;if B is cleared, then no error. Jump to
reset
      JSR   SUCCESS           ;display Success!

      LDX   #VCINIT           ;erase VCINIT. Now VCINIT is $FF instead
of default 0
      STX   INITADD
      JSR   EEBYTE
      RTS

TABORT:     JSR   LDELAY
      BRA   VRESET            ;quits the Voice train procedure
```

```
TREDO1:
       PSHX                 ;LCD display
       LDX    #STR_TW
       PSHX
       LDX    #STR_ABORT
       PSHX
       LDAA   #$2
       JSR    DISPLAY
       PULX                 ;end of LCD display

TREDO:        JSR    TMODE        ;jump to train word mode
       JSR    WFE          ;wait for error
       BRSET $A,X,$1,TABORT    ;if ABORT is pressed, then quit
                            ;was TABORT when it last worked
       JSR    LDELAY
       TST    COUNT2             ;check if B flag is cleared
       BNE    TREDO        ;if not cleared = error occurred --> do TREDO
again

TDONE:        JSR    VRESET             ;reset the voice chip so that it
listens to words
       JSR    SUCCESS            ;display Success!
       JSR    IRCAP        ;this should be in the VT subroutine
       RTS

WFE:   LDAA   #$15         ;loads a count value
       STAA   COUNT
WFE2:  LDY    #$FFFF             ;loads Y
WFE3:  BRSET $3,X,$1,VRESET    ;check if error pin is toggled. If it is,
reset the voice chip
       DEY
       BNE    WFE3         ;delay until Y = 0
       DEC    COUNT
       BNE    WFE2         ;delay until count = 0
       CLR    COUNT2                   ;if no error, clear B pin
       RTS

VRESET:       INC    COUNT2             ;SHOULD NOT NEED THIS
       JSR    CDELAY
       BSET   $7,X,$10
       BCLR   $3,X,$10

VDELAY:       JSR    CDELAY             ;do a short delay
       BCLR   $7,X,$1F           ;set direction of PORTC to OOOIIIII
       JSR    LDELAY             ;do a long delay
       RTS

GMODE:        BSET   $7,X,$02
       BCLR   $3,X,$02
       BRA    VDELAY

TMODE:        JSR    CLRL         ;clear the listening mode light
       BCLR   $7,X,$08           ;pin 3 (recog) input mode
```

```
      BCLR  $3,X,$04          ;pin 2 --> 0
      BRA   VDELAY

CLRW: JSR   CLRL        ;clear listening light
      BCLR  $3,X,$0C          ;clear both pin 2 and 3
      JSR   LDELAY            ;need a long delay for clearing memory
      BRA   VDELAY

CLRL: ;clear listening mode's light
      BSET  $7,X,$0C
      BCLR  $3,X,$08          ;clear pin 2
      JSR   CDELAY
      BCLR  $7,X,$0C
      JSR   LDELAY
      BSET  $7,X,$0C
      RTS

CDELAY:     LDY   #$FFFF            ;~200ms delay
      JSR   SDELAY
      RTS
*=============== End of Voice Training ======================*




*================== IR Matching =====================*
IR_MATCH:

      ;PORT A should come with a value matched to the voice
IRMC: LDAB  #$20
      LDY   #IRBASE-$20

IRM1: ABY
      DECA              ;ACCA should contain voice # at this point
      BNE   IRM1

      LDAA  0,Y
      STAA  IRNUM
      TSTA
      BEQ   IRM2        ;send IR only if valid IR exists
      CMPA  #$5
      BLS   IRM3
IRM2: JMP   M1          ;not RTS because JSR was not called
      LDY   #$FE20
      LDAA  0,Y
      STAA  IRNUM
IRM3: INY
      STY   IRRPT
      LDAA  #$5
      STAA  IRCNT
*================ End of IR Matching =================*
```

```
*=================== IR Sending =====================*
IR_SEND:
      LDX   #$1000           ;loads reg base -- just in case. Probably
don't need
      LDY   IRRPT
      LDD   0,Y        ;
      STD   PTIME      ;this is the pulse width time
      INY
      INY              ;now it points to the actual IR sequence

      LDAA  #$2
      STAA  OCMODE           ;operate in mode '2'

      LDAA  #$8
      STAA  TMSK1,X          ;enable interrupt OC5
      STAA  TFLG1,X          ;clears the OC5 flags
      LDAA  #$01       ;set OC5 pin to toggle for debug. It can be set
to 0
      STAA  TCTL1,X          ;Set OC5 to toggle mode
      LDD   TCNT,X           ;loads main counter
      ADDD  PTIME
      STD   TOC5,X           ;stores a pulse width to OC5 for timeout
      INC   TOUT       ;sets TOUT
      CLI              ;enable interrupt

IR_SEN2:    LDAA  #$8        ;
      STAA  COUNT      ;send 8 times per byte
      LDAA  0,Y
      CMPA  #$FF       ;check if a byte is full of 1's
      BLO   S1         ;
      DEC   COUNT2           ;
      BEQ   DODLY      ;if COUNT2 is zero -- do a long delay since
there are 2 $FF's.
      BRA   S2         ;
S1:   LDAB  #$2        ;
      STAB  COUNT2           ;resets COUNT2 value

S2:   TST   TOUT
      BNE   S2         ;loops until OC5 is triggered
      INC   TOUT       ;resets TOUT
      LSLA             ;ACCA shift left
      BCS   DDONE1           ;if carry is 1, then go clear pin 7
      BSET  0,X,$80          ;if carry is 0, then set pin 7
      BRA   DDONE

DDONE1:     BCLR  0,X,$80
      BRA   DDONE      ;this is not redundent -- need this for
balanced timing

DDONE:      DEC   COUNT
      BNE   S2         ;if 8 counts have not expired, do again
      INY              ;if 1 byte has been sent, then increment Y and
load new byte
```

```
      BRA   IR_SEN2

DODLY:      SEI               ;if 2 $FFs have been detected, then
disable interrupt
      CLR   TCTL1,X           ;no more toggling for OC5
      CLR   TMSK1,X
      PSHY
      LDY   #$42F6            ;~60ms
      JSR   SDELAY
      PULY
      DEC   IRCNT
      BNE   IR_SEND
      JSR   LDELAY            ;do a long delay (~1.5 seconds)
      DEC   IRNUM       ;see if there's any more IR signals left
      BNE   IRRELAY           ;if yes, then go back to send
      JMP   M1          ;if not, jump back to start menu loop
                        ;not RTS because JSR was not called
IRRELAY:    JMP   IRM3
*================ End of IR Sending ==================*




*================ Clear Memory ====================*
;Clear memory should reset everything to 'pre-programmed' condition
CLRMEM:     PSHX              ;LCD display
      LDX   #STR_CLRMEM
      PSHX
      LDX   #STR_NO
      PSHX
      LDX   #STR_BLANK
      PSHX
      LDX   #STR_YES
      PSHX
      LDAA  #$4
      JSR   DISPLAY
      PULX
      JSR   LDELAY            ;need a long delay
CLRM: BRSET $A,X,$01,CSKIP
      BRSET $A,X,$04,CGO
      BRA   CLRM
CGO:  JSR   CLRW        ;clear Voice Chip memory
      LDX   #VCCOUNT    ;this will erase both VCCOUNT and VCINIT
      STX   INITADD
      JSR   EEBYTE
      LDX   #VCINIT
      STX   INITADD
      JSR   EEBYTE

      CLR   $0000
      CLR   $0001
      LDAA  #$2         ;this will program both to $0
```

```
      STAA  NCYCLE
      LDX   #VCCOUNT
      STX   INITADD
      LDX   #$0000
      STX   SRCADD
      JSR   EEPROG

      LDX   #IRBASE          ;this will clear all EEPROM allotted for
IR seq
      STX   INITADD
      LDAA  #$1C
      STAA  NCYCLE
      JSR   EEROW

      JSR   SUCCESS
CSKIP:      RTS
*============== End of Clear Memory =================*




*============= EEPROM Programming/Erasing Module =================*
;When jump to EEPROG
;     Need NCYCLE, INITADD, SRCADD
;When jump to EEBYTE
;     Need INITADD
;When jump to EEROW
;     Need NCYCLE, INITADD

EEPROG:
      LDAA  #$02
      STAA  EEPS
      LDAA  #$03
      STAA  EEPE
      LDAA  #$1        ;increment by 1 every time
      STAA  INCV
      BRA   EECOPY
EEBYTE:
      LDAA  #$16
      STAA  EEPS
      LDAA  #$17
      STAA  EEPE
      LDAA  #$1
      STAA  NCYCLE
      STAA  INCV
      BRA   EECOPY
EEROW:
      LDAA  #$0E
      STAA  EEPS
      LDAA  #$0F
      STAA  EEPE
      LDAA  #$10
      STAA  INCV
```

```
      ;go straight to EECOPY without branching
EECOPY:
      LDX   #TRAM
      LDY   #EESTART
ECOPY:      LDAA  0,Y
      STAA  0,X
      INY
      CPY   #EEDONE
      BEQ   ECEND
      INX
      BRA   ECOPY
ECEND:      JMP   TRAM

EESTART:
      LDX   INITADD           ;load target address to X
      LDY   SRCADD            ;if programing not called, this is unused
EES:  LDAB  EEPS        ;load burn-value
      STAB  PPROG       ;program to EEPROM program reg
      LDAA  #$FF        ;for erasing
      CMPB  #$5
      BHS   NX1         ;branch if it was for erasing
      LDAA  0,Y         ;if not erasing, need to load proper value
NX1:  STAA  0,X         ;store ACCA value to target address
      LDAB  INCV        ;load the proper incremental value
      ABX               ;update X
      INY               ;update Y - not needed if erasing
      LDAB  EEPE
      STAB  PPROG
      PSHY
      LDY   #$0B29            ;for ~10ms
DE1:  DEY
      BNE   DE1
      PULY
      DEC   NCYCLE            ;decrement the number of cycles left
      BNE   EES
      CLR   PPROG       ;clear program reg
      JMP   EEDONE
EEDONE:     RTS
*========== End of EEPROM Programming/Erasing Module ===============*




*================= IR Capturing ==========================*
*need calibration
IRCREDO:    PSHX              ;this runs only on REDOs
      LDX   #STR_IRE
      PSHX
      LDAA  #$1
      JSR   DISPLAY
      PULX
      JSR   LDELAY
```

```
IRCAP:      LDX   #$1000            ;this should, by default to be
#$1000
      CLR   IRNUM
      CLR   TCTL1,X           ;OC does not affect pin
      LDY   #$FFFF
      JSR   SDELAY
      INY               ;after SDELAY, Y = $0000. Increment = $0001 =
address of memory to store IR

IRCAP2:     LDAA  #$01
      STAA  OCMODE            ;set the mode to '1' -- check for 15ms
timeout
      STAA  TOUT        ;timeout counter
      CLR   PORTC       ;turn off all lights
      LDD   #$7000            ;
      STD   PTIME       ;defaults the PTIME to something high so it can
be replaced

      PSHX              ;LCD display
      LDX   #STR_TIR
      PSHX
      LDX   #STR_REDOIR
      PSHX
      LDX   #STR_TRAIN
      PSHX
      LDX   #STR_DONE
      PSHX
      LDAA  #$4
      JSR   DISPLAY
      PULX              ;end of LCD display

IRSTALL:    BRSET $A,X,$1,IRCREDO   ;if ABORT is pressed, jump to start
      BRSET $A,X,$2,IRC ;if 2nd button is pressed, new sequence
      BRSET $A,X,$4,IR_DONE   ;if 3rd button, jump to store
      BRA   IRSTALL

IRC:  PSHX              ;LCD display
      LDX   #STR_WAIT
      PSHX
      LDAA  #$1
      JSR   DISPLAY
      PULX              ;end of LCD display

IRCW: BRCLR 0,X,$4,IRT  ;if input is low, start timing
      BRA   IRCW
IRT:  LDD   TCNT,X            ;load main timer to D
      STD   DTIME       ;store it to DTIME
      BSET  TMSK1,X,$8  ;enable interrupt for OC5
      JSR   DOC         ;do the delay for OC5
      LDAA  #$8         ;turn on OC5
      STAA  TFLG1,X
      CLI               ;enable interrupt
IRCW2:      BRSET 0,X,$4,IRCW3
      BRA   IRCW2
```

```
IRCW3:       BRCLR 0,X,$4,IRT2
      TST   TOUT
      BEQ   IRCEND
      BRA   IRCW3       ;if IR finishes, it would stall here -- use OC5
to jump to IRCEND
IRT2: LDD   TCNT,X
      STD   RTIME
      SUBD  DTIME
      CPD   PTIME
      BHS   IRT3
      STD   PTIME
IRT3: LDD   RTIME
      STD   DTIME
      JSR   DOC
      BRA   IRCW2

IR_DONE:
      JMP   IR_STORE

IRCEND:      SEI
      LDD   PTIME ;PTIME contains 2 pulse widths
      LSRD        ;shifts ACCD to the right -- same as dividing by 2
      STD   PTIME ;now PTIME contains 1 pulse width
      STD   0,Y
      INY
      INY
      LSRD
      ;SUBTRACT HPTIME by a set amount for the new signal capture
method
      SUBD  #$64
      STD   HPTIME      ;now HPTIME contains half a pulse width with
50us less

      INC   TOUT  ;used in the IR signal capture module
      INC   OCMODE            ;OCMODE --> 2
      JSR   LDELAY            ;do a long delay
      CLR   PORTC       ;turn off all lights
      PSHX              ;LCD display
      LDX   #STR_RPT
      PSHX
      LDAA  #$1
      JSR   DISPLAY
      PULX              ;end of LCD display

;     JMP   IRWAIT            ;jump to IRWAIT
      ;should go smoothly to IRWAIT
*================= End of IR Capture ===================*



*================= IR signal capture ===================*
;must turn off the IC functions (if applicable) before going here
;should turn on OC3 and OC5 before going here
```

```
;make sure index X is $1000
IRWAIT:      BRCLR 0,X,$4,DOIR ;if input is 0V (IR detected), jump out
of loop
      BRA   IRWAIT            ;if nothing, then do IRWAIT again

DOIR: LDD   TCNT,X
      ADDD  HPTIME
      STD   TOC5,X            ;do a delay after half the pulse width
      LDAA  #$08
      STAA  TFLG1,X           ;clear the flags for both OC5
      CLI

SEQ:  LDAA  #$08        ;load 8 to counter
      STAA  COUNT       ;to counter

SEQ2: TST   TOUT        ;timeout counter -- controlled by OC3. Happens
every PTIME
      BNE   SEQ2
      ASL   0,Y         ;shift memory
      LDD   TOC5,X
      ADDD  #$14        ;adds 20 cycles (10us) to it
      STD   TOC5,X
      CLR   LHPTIME

LHP:  BRSET 0,X,$4,STOREL
      CLR   COUNT2
LHP2: JSR   IRDELAY
      LDAA  LHPTIME
MHP:  BRSET 0,X,$4,STOREM
      TSTA
      BEQ   J2          ;okay -- no problem
      BRA   CUTOC       ;must cut back on the timing
STOREL:     INC   LHPTIME
      INC   COUNT2
      INC   0,Y
      BRA   LHP2
STOREM:     TSTA
      BNE   J2          ;okay -- no problem
CUTOC:      LDD   TOC5,X            ;these 2 are different. Must delay
subtract time by 50us
      SUBD  #$64
      STD   TOC5,X

J2:   INC   TOUT        ;resets the TOUT back to 1
      DEC   COUNT       ;unaccounted for since the slight change of
design
      BNE   SEQ2
      INY               ;inc. Y for next RAM location
      CPY   #$001F            ;
      BHI   IR_REDO           ;if Y = $0020, then quit this process --
memory is full.
      LDAA  COUNT2            ;check if COUNT2 > 15
      CMPA  #$F
      BHI   INCIR       ;if higher, an IR signal is completed
```

```
      BRA   SEQ         ;do SEQ again if IR signal is not completed

INCIR:      SEI               ;disable interrupt
      INC   IRNUM       ;increment IRNUM after signal finishes
      LDAA  #$FF
      STAA  PORTC
      JSR   LDELAY            ;do a long delay
      CLR   PORTC
      JMP   IRCAP2            ;(3) wait for new IR signals

IR_REDO:    SEI               ;disable interrupt
      PSHX              ;LCD display
      LDX   #STR_IRLONG
      PSHX
      LDAA  #$1
      JSR   DISPLAY
      PULX
      JSR   LDELAY            ;end of LCD display
      JMP   IRCREDO           ;redo IR if it's too long
*================= End of IR signal capture ===================*
```

```
*================= Output Compare ===================*
;use OC5 for 1) timeout when determining IR pulse width and 2) to get
IR signal
;should turn off when not using it

;OC5 does timeouts
OC5S: LDAB  OCMODE            ;check what mode this should operate
      CMPB  #$1         ;compare with mode 1
      BHI   OC5S2       ;if higher, then jump
      BRA   OC5E        ;reset flag and return from interrupt

OC5S2:      LDD   TOC5,X            ;this was originally TCNT -- which
might be the cause of bug
      ADDD  PTIME
      STD   TOC5,X

OC5E: CLR   TOUT
      LDAB  #$8
      STAB  TFLG1,X
      RTI

DOC:  ;Delay Output Compare
      LDD   TCNT,X
      ADDD  #$7530            ;delay for about 15ms more
      STD   TOC5,X
      RTS
```

```
*============== End of Output Compare ================*




*==================== IR Storing =======================*
IR_STORE:
      LDAA   VCCOUNT
      INCA
      STAA   VCNUM
      LDAB   #$20
      LDX    #IRBASE-$20 ;figure out where to store to

IRS1: ABX
      DECA
      BNE    IRS1

      STX    INITADD          ;store IR sequence to proper address
      LDAA   #$20        ;run the program loop for 32 times
      STAA   NCYCLE
      LDX    #$0000           ;initial address to get the stuff
      STX    SRCADD
      JSR    EEPROG           ;go program

      LDX    #VCCOUNT    ;Erase VCCOUNT
      STX    INITADD
      JSR    EEBYTE

      LDX    #VCCOUNT    ;Replace VCCOUNT with new value
      STX    INITADD
      LDX    #VCNUM
      STX    SRCADD
      LDAA   #$1         ;program only 1 cycle
      STAA   NCYCLE
      JSR    EEPROG


      JSR    SUCCESS
      RTS
*================= End of IR Storing ====================*




*==================== Delays ======================*
LDELAY:      LDAA   #$5         ;~1.5 second delay
      STAA   COUNT
```

```
      LDAA  #$40
      STAA  PORTC
      PSHY
LD2:  LDY   #$FFFF
      BSR   SDELAY
      DEC   COUNT
      BNE   LD2
      PULY
      LDAA  #$20
      STAA  PORTC
      RTS

IRDELAY:    PSHY
      LDY   #$D         ;about 50us per IRDELAY
      BSR   SDELAY
      PULY
      RTS

SDELAY:     DEY               ;(4)
      BNE   SDELAY            ;(3)
      RTS               ;(5)
*================= End of Delays =====================*




*===================== LCD Component ====================*
* Write Message to LCD *
WRITE_MESS:
      PULX
      STX   LCDTEM2
      PULX
NEXT_CHAR:
      LDAB  0,X         ;Load the letter to be written into ACCB
      CMPB  #$00        ;Check whether it is a null letter
      BEQ   END_MESS    ;End the write message process when a null
letter is encountered
      JSR   WRITE_DATA  ;Write the letter to LCD
      INX
      BRA   NEXT_CHAR   ;Write next letter
END_MESS:
      LDX   LCDTEM2
      PSHX
      RTS

* Write a letter to LCD *
WRITE_DATA:
      STAB  PORTB       ;Write data to PortB
      ORAA  #DATAREG    ;Set RS = 1 for writing data to LCD
      STAA  PORTA
      JSR   DELAY       ;Delay for 1 ms
      EORA  #ENABLE           ;Enable LCD write
```

```
      STAA   PORTA
      JSR    DELAY        ;Delay for 1 ms
      EORA   #ENABLE            ;Disable LCD write
      STAA   PORTA
      JSR    DELAY        ;Delay for 1 ms
      RTS

* Write instruction to LCD*
WRITE_INST:
      STAB   PORTB        ;Write data to PortB
      ANDA   #CMDREG      ;Set RS = 0 for writing instruction to LCD
      STAA   PORTA        ;Write RS to PortA
      JSR    DELAY        ;Delay for 1 ms
      EORA   #ENABLE            ;Enable LCD write
      STAA   PORTA
      JSR    DELAY        ;Delay for 1 ms
      EORA   #ENABLE            ;Disable LCD write
      STAA   PORTA
      JSR    DELAY        ;Delay for 1 ms
      RTS

* LCD initialization *
LCD_INIT:
      PSHA                ;Push ACCA into stack
      PSHB                ;Push ACCB into stack
      LDAB   #$0F         ;Delay for 15 ms
      JSR    LONG_DELAY
      LDAB   #$30         ;Set 8-bit processing, 'work', and 5 X 7 dots
      JSR    WRITE_INST
      LDAB   #$4          ;Delay for 4 ms
      JSR    LONG_DELAY
      LDAB   #$38         ;Set 8-bit processing, 'ignore', and 5 X 7 dots
      JSR    WRITE_INST
      LDAB   #$08         ;Display off, cursor off, and blink off
      JSR    WRITE_INST
      LDAB   #$01         ;Clear display
      JSR    WRITE_INST
      LDAB   #$2          ;Delay for 2 ms
      JSR    LONG_DELAY
      LDAB   #$06         ;Cursor moves from left to right and with no
shift
      JSR    WRITE_INST
      LDAB   #$0C         ;Display on, cursor off, and blink off --Change
it later
      JSR    WRITE_INST
      PULB                ;Retrieve ACCB
      PULA                ;Retrieve ACCA
      RTS

* Delay for 1 ms *
DELAY:
      PSHY                ;Store index Y to stack
      LDY    #$0147             ;Set delay counter of 1ms into Y
DELAY_LOOP:
```

```
      DEY                 ;Decrement Y
      BNE   DELAY_LOOP    ;Loop until Y = 0
      PULY                ;Retrieve index Y
      RTS

* Delay for n ms where n is set by the programmer *
LONG_DELAY:
      STAB  COUNT         ;Store desired delay time to counter
LONG_LOOP:
      TST   COUNT         ;Check if counter = 0
      BEQ   FINISH              ;Quit loop
      DEC   COUNT         ;Decrement counter
      JSR   DELAY         ;Delay for 1 ms
      BRA   LONG_LOOP     ;Go back to the loop until counter = 0
FINISH:
      RTS
*============== End of LCD Component ====================*




*============== Write Premade lines ====================*
;X contains the address for 4 lines
;ACCA contains count (the number of lines. must be greater than 0)
DISPLAY:    PSHA
      LDAB  #$1
      JSR   WRITE_INST    ;clears the display
      PULA
      PULX
      STX   LCDTEM1
      DECA
      BEQ   DIS1
      DECA
      BEQ   DIS2
      DECA
      BEQ   DIS3          ;when A is >3, then do all

DIS4: LDAB  #LINE4
      JSR   WRITE_INST
      JSR   WRITE_MESS
DIS3: LDAB  #LINE3
      JSR   WRITE_INST
      JSR   WRITE_MESS
DIS2: LDAB  #LINE2
      JSR   WRITE_INST
      JSR   WRITE_MESS
DIS1: LDAB  #LINE1
      JSR   WRITE_INST
      JSR   WRITE_MESS
DIS0: LDX   LCDTEM1
      PSHX
      RTS
```

Freedom Life Systems

```
SUCCESS:
       PSHX
       LDX    #STR_SUC
       PSHX
       LDAA   #$1
       JSR    DISPLAY
       PULX
       JSR    LDELAY
       RTS
*============ End of Write Premade lines ================*



* Define The Strings *
STR_ABORT:
       DB     'Abort'
       DB     $00
STR_READY:
       DB     'Ready'
       DB     $00
STR_BLANK:
       DB     ' '
       DB     $00
STR_TRAIN:
       DB     'Train'
       DB     $00
STR_CLRMEM:
       DB     'Clear Memory'
       DB     $00
STR_GW:
       DB     'Gate Word'
       DB     $00
STR_TW:
       DB     'Train Word'
       DB     $00
STR_SUC:
       DB     'Success!'
       DB     $00
STR_RPT:
       DB     'Repeat'
       DB     $00
STR_TIR:
       DB     'Train IR'
       DB     $00
STR_IRLONG:
       DB     'IR too long'
       DB     $00
STR_WAIT:
       DB     'Waiting'
       DB     $00
STR_DONE:
       DB     'Done'
       DB     $00
```

Freedom Life Systems

```
STR_REDOIR:
      DB    'Redo IR'
      DB    $00
STR_IRE:
      DB    'IR Erased'
      DB    $00
STR_YES:
      DB    'Yes'
      DB    $00
STR_NO:
      DB    'No'
      DB    $00

* Variables *
VCCOUNT:    ;should be set to 0 for both VCCOUNT & VCINIT
            ;** OR, let "FF" equal to initial condition -- can save
some memory
      DB    $0          ;Number of voice commands stored

VCINIT:
      DB    $0          ;$0 for no gate word, $FF if yes


*========= IR sequence storage ===============*


*======= End of IR sequence storage ===========*

      ORG   $FFE0
      FDB   OC5S

* IRQ Vector *
;     ORG   $FFF2       ;IRQ vector
;     FDB   ISR         ;Point to interrupt service routine



* Reset Vector To Point To The Start Of Program *
      ORG   $FFFE
      FDB   BEGIN
```

Appendix 2: IR Filter Circuit

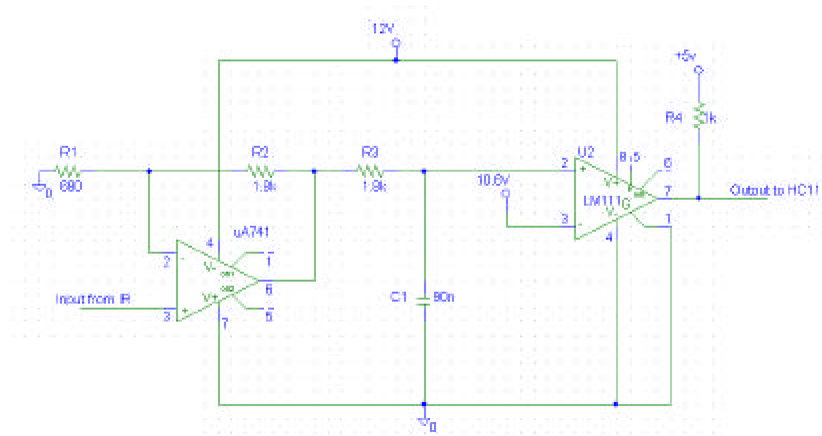

Figure A2.1: IR Filter Circuit

Appendix 3: Voltage Regulators

Figure A3.1 shows the supporting circuitry used with a voltage regulator.

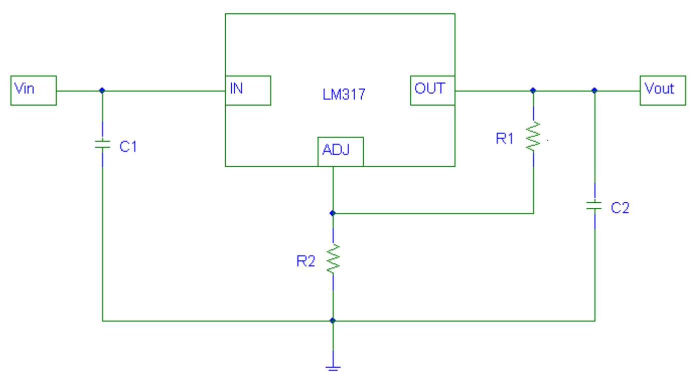

Figure A3.1: Voltage Regulator Circuit

The capacitors C1 = 0.1 µF and C2 = 1.0 µF are used to filter out instabilities in the voltage levels. The output voltage of the regulator is determined by the external resistors R1 and R2. The output voltage is related to R1 and R2 by

$$R2 = \left(\frac{Vout}{1.25} - 1 \right) 270$$

We chose R1 = 270 ? and varied R2 to produce the desired voltage. Table A3.1 shows the voltages produced for each device and the corresponding R2 value.

Freedom Life Systems

Table A3.1: Device Voltages

Device	Voltage (V)	R2 (?)
IR receiver	3	378
HC11 controller	5	810
Speaker	9	1674
LCD	5	810
Voice rec. chip	5	810
FRS	4.5	702
RF transmitter	3	478
Comparator	12	2322

Appendix 4: IR Module Circuit

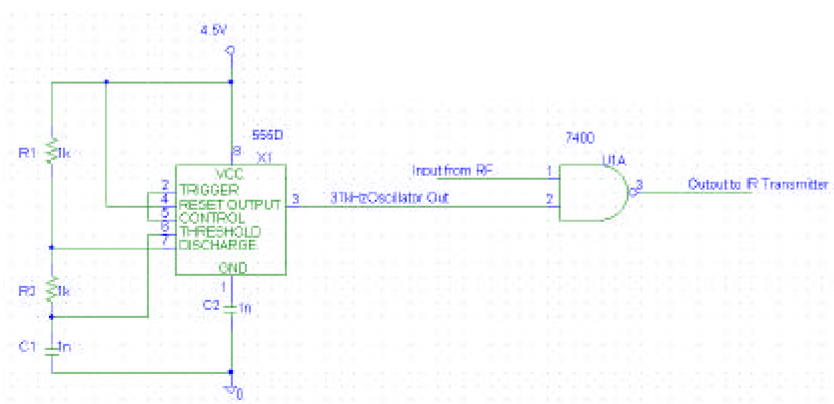

Figure A4.1: IR Module Circuit