AUTO-TUNING ANTENNA

By

John Chen
Jennifer Tighe

Senior Project

ELECTRICAL ENGINEERING DEPARTMENT

California Polytechnic State University

December 2011

TABLE OF CONTENTS

TABLE OF FIGURES

TABLE OF TABLES

Acknowledgements

We would like to thank our family and friends who have supported us throughout our time at Cal Poly. Thank you also to Dr. Dean Arakaki for all his time spent advising us and thank you to Professor Bryan Mealy for his programming support.

Abstract

Radio transmitters and antennas have differing impedances at the output and feedpoint, depending on frequency. Modern antenna tuning methods involve manual or automatic resistance and reactance network adjustment to match transmitter and antenna impedance. This reduces power reflections to the transmitter, but introduces losses and prevents full transmitter power from being radiated at the antenna. This project utilizes a variable length antenna element to tune and match transmitter and antenna impedances, allowing efficient power transfer to the antenna. The antenna length varies from 0 to 30.5 inches. The method involves an RF signal with a maximum power of 2 watts and a standing wave ratio (SWR) analyzer for measuring reflections. An Atmega32 is used to analyze SWR analyzer data and control the motor driver and liquid crystal display (LCD). The motor driver controls antenna extension and retraction. The LCD displays the SWR reading. The tuning process is an automated easy-to-use system for radio operators.

I. Introduction

The goal of this project is to improve current antenna tuning methods. One method involves placing an in-line standing wave ratio (SWR) analyzer and manually adjusting antenna length based on measured SWR values, as in Figure I-Ia. A second method is to implement a matching network between the transmitter and antenna, as in Figure I-Ib. This is not desirable because additional circuitry inherently increases power consumption thus reducing the power available for transmitting the signal. Additionally, a matching network is dependent on transmit frequency, i.e. matching networks are required for each frequency.

A system that automatically tunes a monopole antenna is designed to improve the current method of tuning with matching networks. The system senses the standing wave ratio at the antenna feed point and conveys this value to the Atmega32 microcontroller programmed to optimize antenna length for each transmit frequency. The antenna is used in the 144-148 MHz amateur radio band, but has the capability of tuning from 120 to 160 MHz. This corresponds to a 46.9 to 62.5 cm length for a quarter-wavelength antenna. The antenna is tuned to an SWR of 2:1 or less; 1:1 is ideal. Using an auto-tuning system combined with an antenna length of at least a quarter wavelength eliminates the need for a matching network. This increases the effective transmit power, efficiency, and is more convenient compared to manual tuning methods. An auto-tuned antenna allows use of a single antenna for transmission on multiple bands.

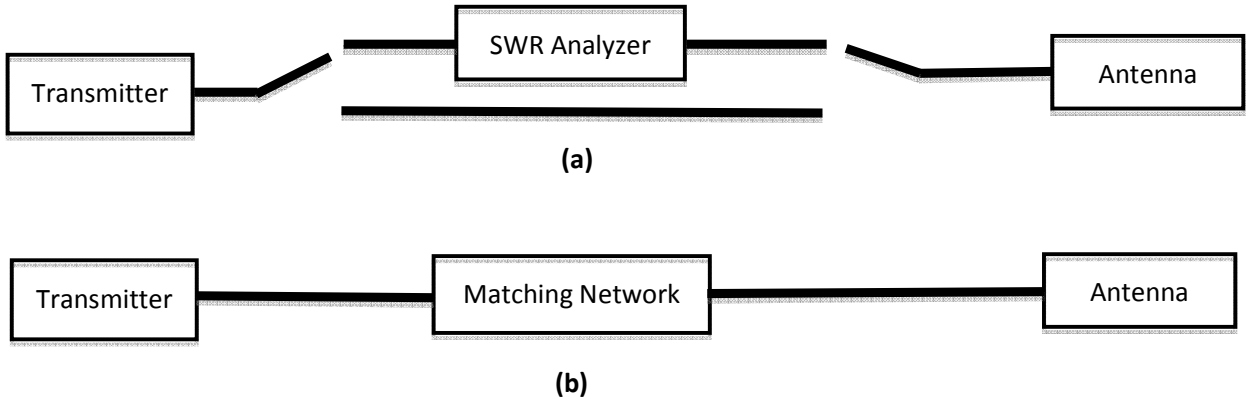


Figure I-I: Antenna Tuning Systems

II. Background

The tuning method used in this project, an adjustable length antenna element, is possible because antenna elements are resonant at transmitting wavelength fractions. This is due to standing waves that develop and resulting voltage and current distribution across the antenna. At these lengths, more efficient power transfer between the radio transmitter and antenna can be achieved and power reflections back to the transmitter are minimized. Traditionally, amateur radio operators use matching networks between the transmitter and antenna with variable resistances and reactances. This tuning method introduces system losses and does not allow the full transmit power to be radiated from the antenna. By tuning the antenna element length, a match can be achieved without the use of lossy components. The ability to detect a good match is possible with a Standing Wave Ratio (SWR) analyzer. The SWR analyzer sends data to a microcontroller, which controls a telescoping antenna motor to optimize the length for a given operating frequency.

The Atmega32 microcontroller was chosen for this project because of its high-speed capability, low cost, and versatile features. It uses Harvard Architecture, which enables instruction execution at each clock cycle. Additionally, maximum clock frequency is 16MHz. This microcontroller has an onboard flash memory that enables program storage when the chip is powered down. The Atmega32 has four input and output ports capable of interfacing with external devices. Each port has 8 pins, where each pin can be programmed as an input or output. The block diagram, Figure II-I, shows the chip configuration and alternative functions for each port. For example, PORTB can be programmed to interface in SPI (Serial Peripheral Interface) mode, which allows devices to communicate as master/slave. The pin descriptions for the Atmega32 appear in Figure II-II. The Atmega32 can be integrated with the STK500 development board to enable simple integration with components. The speed and functionality meet requirements for controlling the motor and providing communications with the SWR Analyzer.

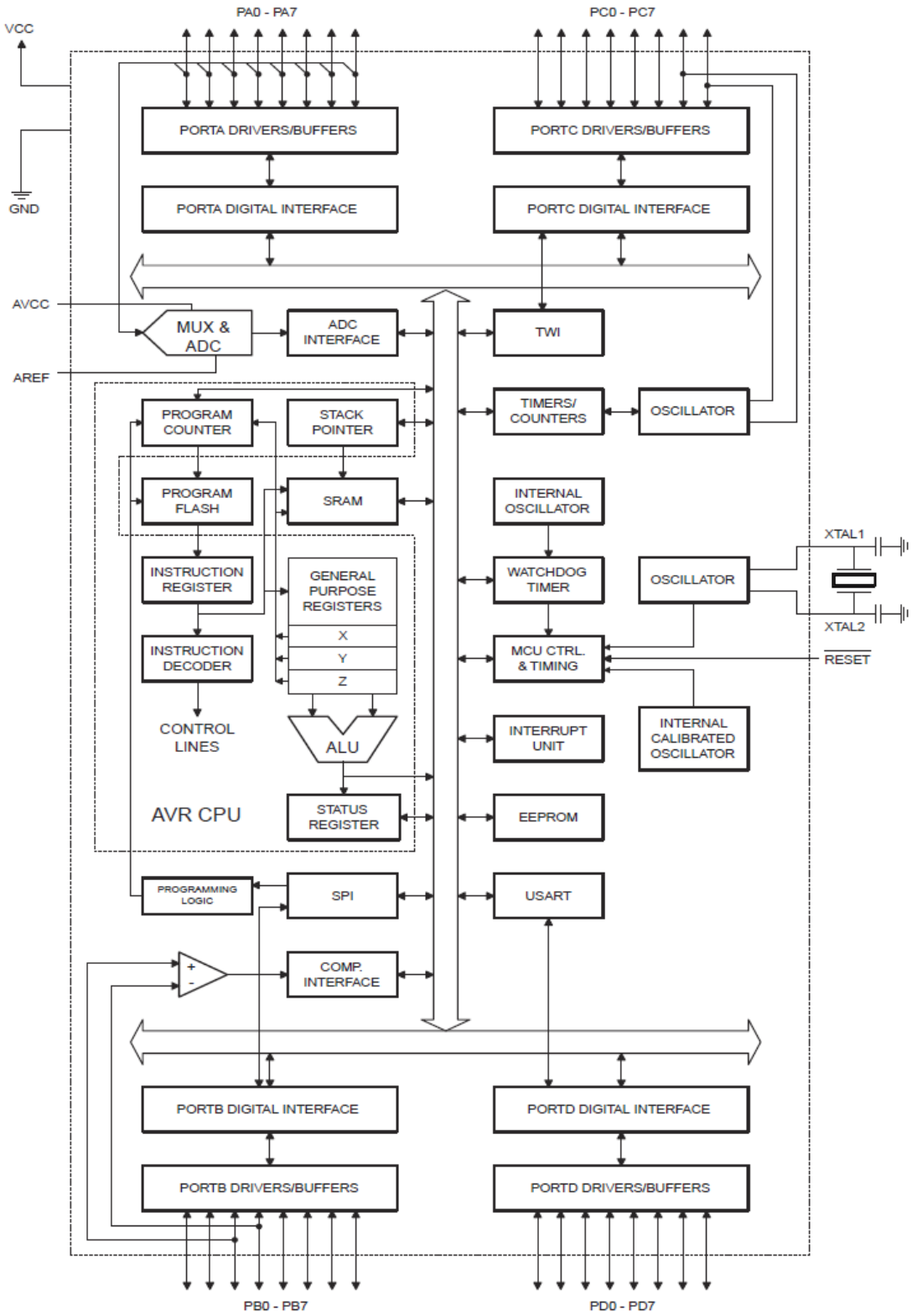


Figure II-I: Atmega32 Block Diagram

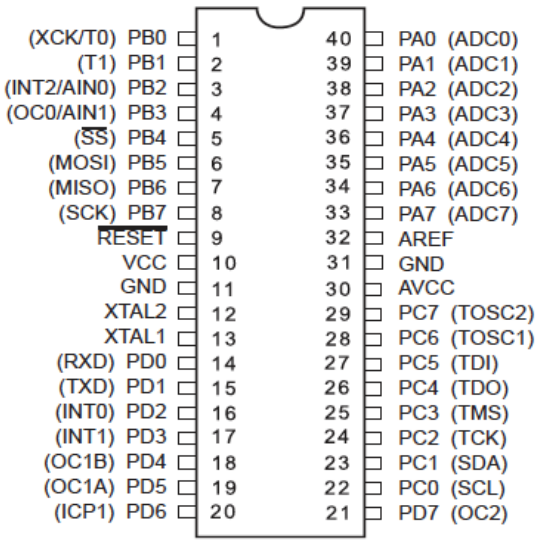


Figure II-II: Atmega32 Pin Descriptions

A motor is required for automated antenna extension and retraction. Brushed, brushless, and stepper motors are among conventional DC electric motors [1]. Brushed DC electric motors use stationary magnets and rotating electric magnets supplied directly from DC power [2]. Brushless DC electric motors require a motor controller to convert DC to AC and regulate voltage and frequency to control speed.AC power is used to rotate a permanent magnet while an electric magnet remains stationary [3]. A stepper motor is a brushless electric motor that steps through rotation. This enables precise motor positioning, but requires an external control circuit [4]. Table II-I compares brushed and brushless DC electric motors.

Table II-I: Motor Pros and Cons

Motor Type	Pros	Cons
Brushed DC Electric	• Low initial cost • High reliability • Simple control of motor speed	• High maintenance • Limited lifespan for high intensity uses
Brushless DC Electric	• Long lifespan • Low maintenance • High efficiency	• High initial cost • More complicated motor speed controllers

The antenna's motor has been examined and determined to be a brushed DC electric motor. The motor is DC powered and does not include speed controllers. A stepper motor is ideal for high precision, but the high cost and complicated installation procedure makes it an unsuitable choice. The brushed DC motor is suitable if it remains at constant speed, which is achievable by using a constant voltage for power.

III. Requirements

The main requirement is tuning the antenna such that an SWR of less than 2:1 is achieved at the operating frequency. This value was chosen because it prevents the transmitter from being damaged from reflected waves. The system must also be automated and only require the user to apply power from the transmitter and press a tuning button on the enclosure. The operating frequency range is dependent on the physical telescoping antenna characteristics because the element length directly correlates with possible transmission frequencies.

IV. Design

A. Overview

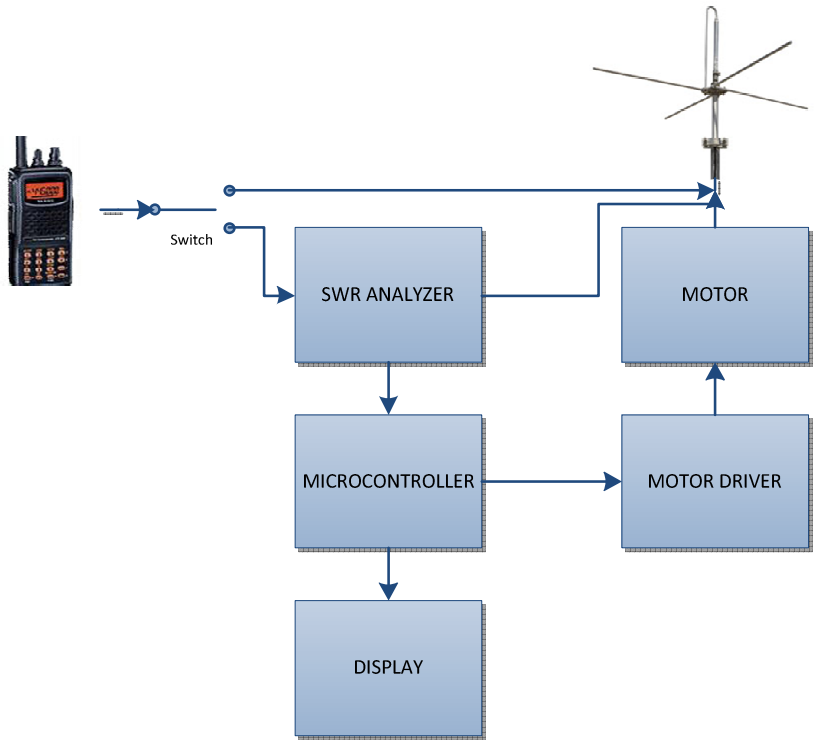


Figure IV-I: System Block Diagram

The Auto-Tuning Antenna system includes the components shown in Figure IV-I. A radio is connected to the SWR Analyzer, which is placed in-line between the radio and antenna. The SWR Analyzer outputs two DC values to the microcontroller, corresponding to forward and reverse power. The microcontroller continuously samples these values and calculates the SWR ratio while the antenna is extended. When the program determines optimal SWR, the microcontroller stops antenna adjustment and displays the SWR on the LCD. Since the motor is an inductive device, it stores energy and releases it when power is disconnected, which requires protection circuitry to prevent power supply and microcontroller damage. The motor driver is included to provide back-EMF protection and regulate surges.

B. SWR Analyzer

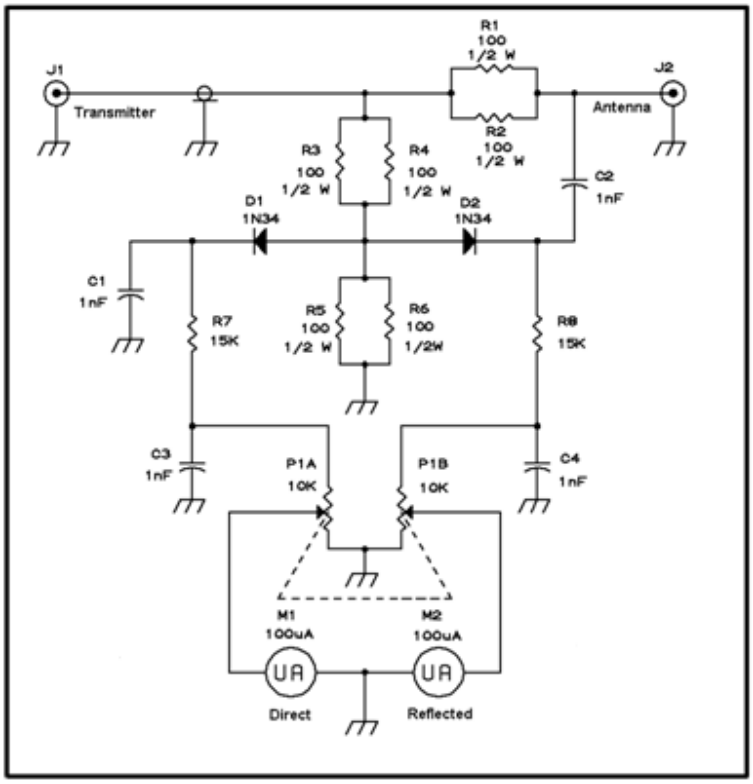


Figure IV-II: SWR Analyzer Schematic

The SWR analyzer circuit, Figure IV-II, was designed by an amateur radio operator [5]. The circuit utilizes a resistive bridge that operates up to a frequency of about 500 MHz. A disadvantage isthe circuitcan only handle transmitter power up to 2 watts. This circuit was chosen because it requires low-cost components and high power input was not necessary due to the 300 mW transmitter being used. The SWR analyzer is connected only while the antenna is tuning. The radio is connected directly to the antenna during communication because the analyzer absorbs power. This power absorption serves as protection to the transmitter while tuning because it reduces reflections back to the transmitter during mismatch.

The circuit uses two 100 Ω, 100 watt resistorsconnected in parallel to provide a 50 Ω comparision. If the load attached at the antenna port is a perfectly matched 50 Ω load, the voltages at D2 anode and cathode are identical, hence 0V. However, the voltage at D1 anode relative to the cathode is about 2.5 V_{RMS} given a 5 V_{RMS} input signal. This means there is a 3.5 V_{pk}which producesa rectified DC voltage of 3.5 V at theD1 cathode, resulting in a reading of about 1.4 V at the forward output, after passing through the resistor divider and 0V at the SWR analyzer reverse output. A mismatched load at the antenna port

causes a voltage differential on both sides of D2 and creates a DC voltage on the circuit's right side where the reflected voltage is measured. The relationship between SWR and voltage is shown in Equation IV-I below.

C. Software

The Atmega32 is the microcontroller used on the STK500 development board, which determines optimal antenna length and executes instructions to adjust the antenna to its optimal length. Basic program operation involves reading voltages from the SWR analyzer, optimizing antenna length, and displaying the calculated SWR on the LCD.

The program must read SWR analyzer forward (V_f) and reverse (V_r) voltages during signal transmission. These values are read into the Atmega32 as analog voltages and are used to calculate the standing wave ratio [6]. The analog voltages are passed through the Atmega32 built-in analog-to-digital converter. Once these values are digitized, SWR is calculated, according to Equation IV-I.

$$SWR = \frac{1 + \frac{V_r}{V_f}}{1 - \frac{V_r}{V_f}} \qquad \text{(IV-I)}$$

Figure IV-III describes optimization methods used to adjust antenna length for maximum transmission. The method utilizes two functions, a coarse optimization that approximates the optimal antenna length and a fine optimization function that hones in on the optimal SWR and antenna length.

The coarse optimization function acquires SWR measurements at wide intervals and determines a range where the optimal antenna length exists. [Note: The antenna requires6250ms to extend from 0 to 30.5 inches. The program uses time delays to adjust the length by equal distances.] The program records SWR measurements to an array at 250ms intervals until the antenna is fully extended. The minimum array value and corresponding antenna length are identified and antenna length is adjusted to this value. The coarse optimization function uses a delay to return the antenna to the minimum SWR length. This method approximates the minimum SWR location. The fine optimization function records 10 measurements in an array within a 250ms range, which is centered on the SWR value found in the coarse optimization function. Measurements are recorded at 25ms intervals. The program determines the minimum array value and corresponding length. This value is the optimal SWR value and the antenna is returned to the corresponding length. A comparison method is used to return the antenna to the length where the minimum SWR occurred. This method is more accurate than the coarse optimization's method. The program returns the antenna to the beginning of the 250ms range (the

range determined by the coarse optimization function's optimal SWR value) and begins increasing antenna length while making SWR measurements. At each iteration, the SWR measurement is compared to the minimum SWR value. Once the measurement is equal to the minimum SWR, the antenna is positioned at its optimal length. The program concludes by displaying the SWR reading on the LCD. If the SWR measurement after tuning does not reach the minimum SWR value, the program will display an error message on the LCD. The error message is implemented for atypical situations, such as the transmitter turning off during tuning.

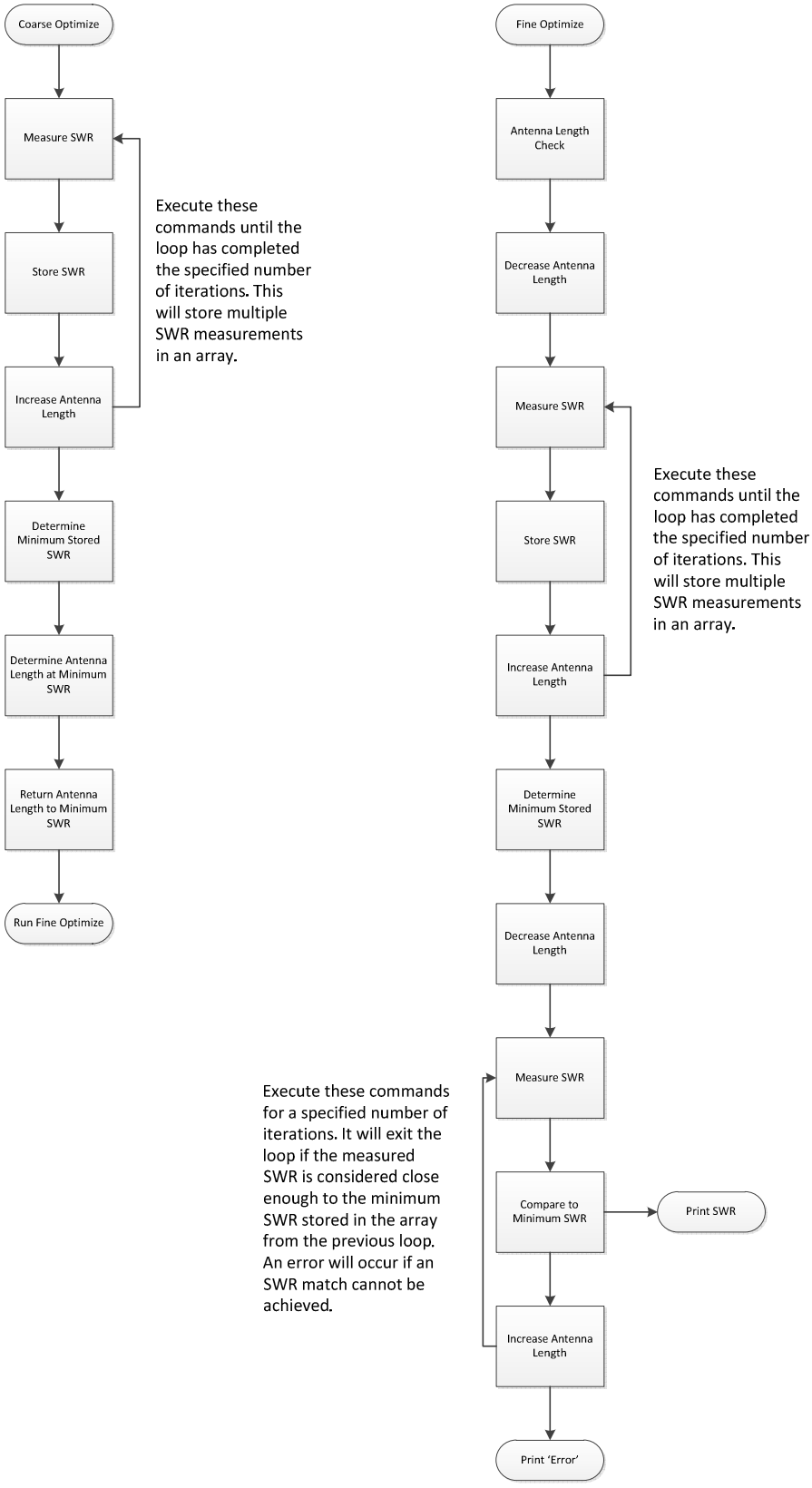


Figure IV-III: SWR Optimization Algorithm

D. Motor Control

The antenna used in this system is a universal motorized car antenna made by Antenna Works, model 44-PW22. Specifications for this assembly were not found, but research into motors addresses the issue of current loading. The microcontroller is used to control the motor but it cannot provide the necessary current for the motor. The L298 2A Dual H-Bridge Motor Driver, Figure IV-IV, was chosen to solve this issue. The motor driver requires a DC power source and control pins from the microcontroller. The external power source enables the motor driver to control the DC motor. The motor driver pins and power requirements are described in Table IV-I and Table IV-II, respectively.

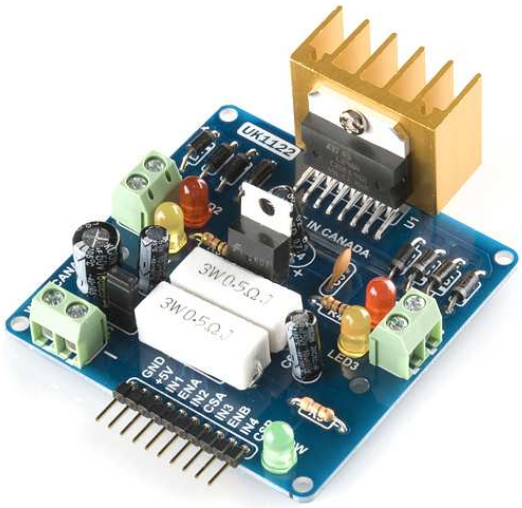


Figure IV-IV: L298 Motor Driver

Table IV-I: L298 Motor Driver Pin Descriptions

Motor Pin	Description
GND	Optional internal connection to MOT DC via J1 jumper
+5V	Optional internal connection to MOT DC via J1 jumper
IN1	Controls direction of motor 1
ENA	Enable output to motor 1
IN2	Controls direction of motor 1
CSA	Current sense output for motor 1
IN3	Controls direction of motor 2
ENB	Enable output to motor 2
IN4	Controls direction of motor 2
CSB	Current sense output for motor 2

Table IV-II: L298 Motor Driver Outputs &Power Requirements

Port	Description
MOT DC	Requires a 6 – 35 DC voltage
MOT 1	Motor 1 output
MOT2	Motor 2 output

Three functions were created in the program for motor control. The first function commands the motor driver to rotate the motor in the clockwise direction. This is achieved by enabling the motor output by sending ENA/ENB and IN1/IN3 pins a high signal and IN2/IN4 a low signal. Inverting the signals for IN1/IN3 and IN2/IN4 rotates the motor in the counterclockwise direction. Setting ENA/ENB to low disables the motor driver output. The functions and controls for the motor driver are summarized in Table IV-III and Table IV-IV, respectively.

Table IV-III: Motor Driver Functions

Function	Description
motorOFF()	Turns motor driver off, which effectively removes power from the motor causing it to stop.
motorFWD()	Rotates the motor in the clockwise direction. The motor will stay on until motorOFF() is called.
motorREV()	Rotates the motor in the counterclockwise direction (reverses the motor lead voltage). The motor will stay on until motorOFF() is called.

Table IV-IV: Logic Levels for Motor B Control.

Motor Direction	Logic Levels
Forward	ENB - High IN3 - High IN4 - Low
Reverse	ENB - High IN3 - Low IN4 - High
Off	ENB - Low IN3 - Don't Care IN4 - Don't Care

E. Display

The PmodCLD LCD, Figure IV-V, is a 16x2 character LCD used to display the measured SWR. It requires 3.3V or 5V for input power. There are eight data pins and three control pins necessary for the microcontroller to send information. Table IV-Vdescribes each pin.

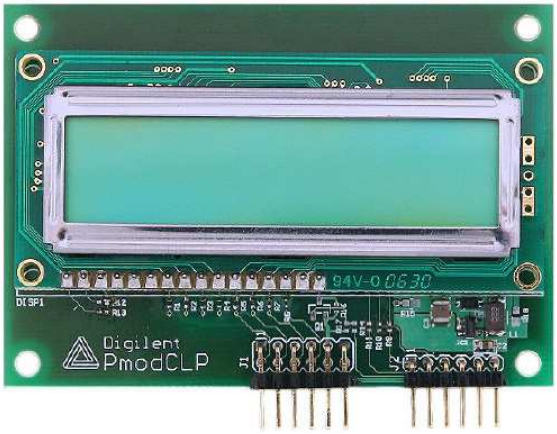


Figure IV-V: PmodCLP LCD

Table IV-V: PmodCLP LCD Pin Descriptions

Pin Number	Signal Description
J1.5, J1.11, J2.5	Signal Ground
J1.6, J1.12, J2.6	Power Supply (5V)
J2.1	Register Select (RS)
J2.2	Read/Write Signal (R/W)
J2.3	Read/Write Strobe (E)
J2.4	Not Connected
J1.1,2,3,4, J1.7,8,9,10	Bi-Directional 8-bit Data Bus

Three functions were written for the LCD: initLCD (), clearLCD(), and printASCII(). The LCD requires a specific startup sequence [7], which is programmed in the initLCD() function. This function configures setup parameters such as cursor location and direction. The second function, clearLCD(), enables the programmer to clear the screen before writing new characters. The last function, printASCII(),requires an ASCII character input parameter and sends this character to the LCD.

V. Test Plans

The SWR analyzer board is tested using a Yaesu VX-5R amateur radio operating on 146.5 MHz at a power level of 300 mW. The SWR value is calculated manually from the SWRDC Output, measured between the middle pin of the potentiometers and ground, and also read from the LCD Display. The measurements are read from 0 inches to the fully extended length of 30.5 inches at 0.5 inch increments. These values are compared to the commercial MFJ MFJ-269 SWR Analyzer to verify SWR analyzer board accuracy, with results shown in Figure VII-I.

Automatic tuning ability is tested using the same Yaesu VX-5R amateur radio operating on 146.5 MHz with a power level of 300 mW. The transmitter is turned on and the microcontroller is reset to start the tuning process. Once the microcontroller completes tuning, the final SWR value is displayed on the LCD display with multiple trials to check for accuracy, as shown in Figure VII-II.

VI. Development and Construction

A. SWR Analyzer

The SWR analyzer was implemented using a breadboard for circuit verification, Figure VI-I. The circuit was built according to Figure IV-II. The standing wave ratio was calculated using Equation IV-I. This method, Figure VI-I, is inaccurate because of long lead lengths, which add inductance to the circuit.

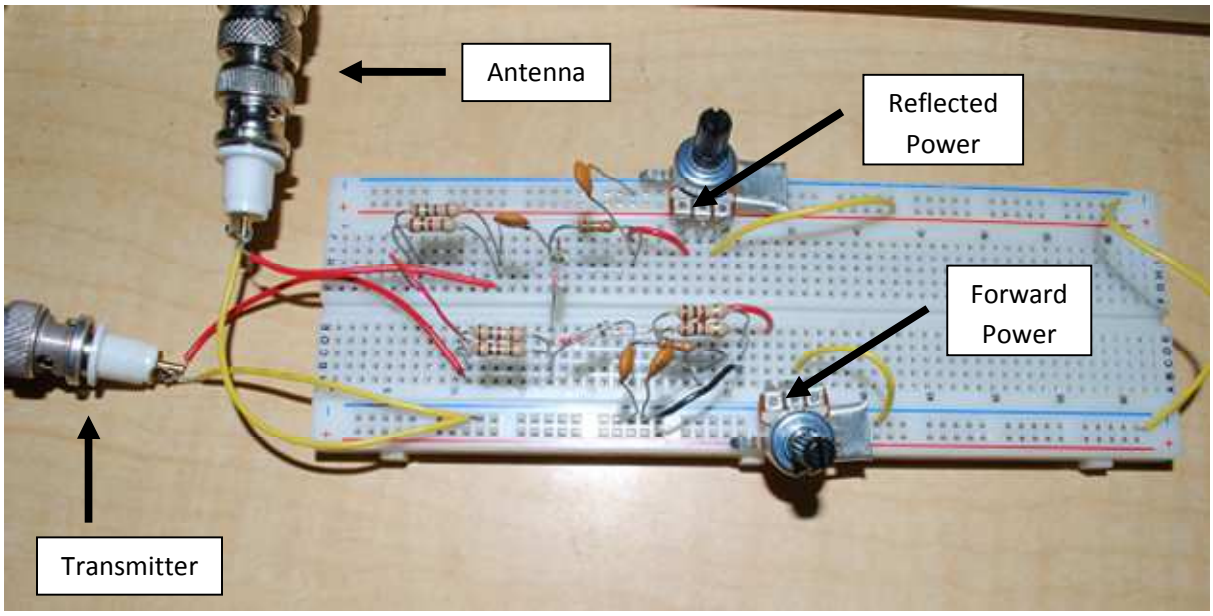


Figure VI-I: Breadboard implementation of the SWR analyzer.

A PCB, with the majority of components surface mounted, will be used as the final SWR analyzer. Small components allow shorter circuit traces. This reduces signal interference and characteristic impedance variations, due to inductances created, and allow the analyzer to output accurate values.

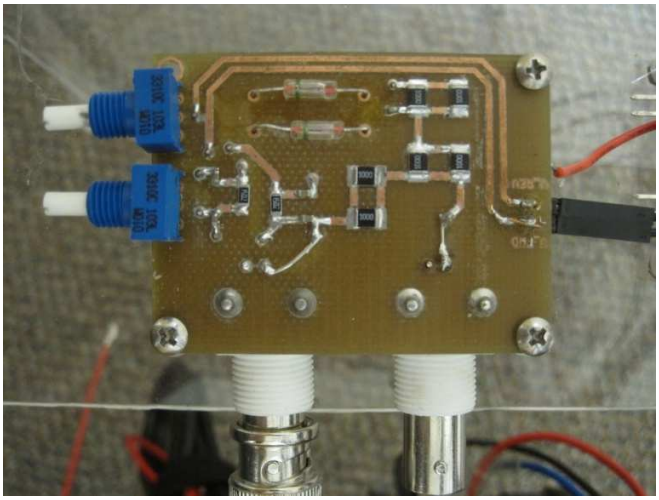


Figure VI-II: SWR Analyzer PCB

B. Atmega32 Interfacing

The SWR analyzer, motor driver, and LCD connect to the Atmega32 input/output ports. Table VI-I describes the microcontroller ports that each device communicates through. Atmega32 PORTA internally connects to a built-in analog-to-digital converter. Forward and reverse voltages from the SWR analyzer must be digitized to enable microcontroller calculations. The LCD and motor driver can connect to microcontroller general input/output ports.

Table VI-I: Microcontroller Port Designations

PORT	Interfacing Description
A	SWR Analyzer circuit forward and reverse voltages
B	EMPTY
C	LCD data pins
D	LCD control pins and Motor Driver

Jumper wires are used to connect the Atmega32 to the SWR analyzer, motor driver, and LCD according to Table VI-II. The STK500 provides power to the Atmega32 via an AC adapter that connects to 120V, 60Hz. The connections are shown in Figure VI-III.

Table VI-II: Atmega32 Device Connection

Pin Number	Connection	Data Direction
Vcc, AVCC	+5.0V	-
GND	Signal Ground	-
AREF	EMPTY	-
XTAL1, XTAL2	EMPTY	-
PA0	SWR Analyzer Forward Voltage (V_f)	Input
PA1	SWR Analyzer Reverse Voltage (V_r)	Input
PA2 – PA7	EMPTY	-
PB0 – PB7	EMPTY	-
PC0	LCD Data Bus J1.1	Output
PC1	LCD Data Bus J1.2	Output
PC2	LCD Data Bus J1.3	Output
PC3	LCD Data Bus J1.4	Output
PC4	LCD Data Bus J1.7	Output
PC5	LCD Data Bus J1.8	Output
PC6	LCD Data Bus J1.9	Output
PC7	LCD Data Bus J1.10	Output
PD0	LCD Control J2.1	Output
PD1	LCD Control J2.2	Output

PD2	LCD Control J2.3	Output
PD3	Motor Driver IN3	Output
PD4	Motor Driver ENB	Output
PD5	Motor Driver IN4	Output
PD6	Motor Driver CSB	Output
PD7	EMPTY	-

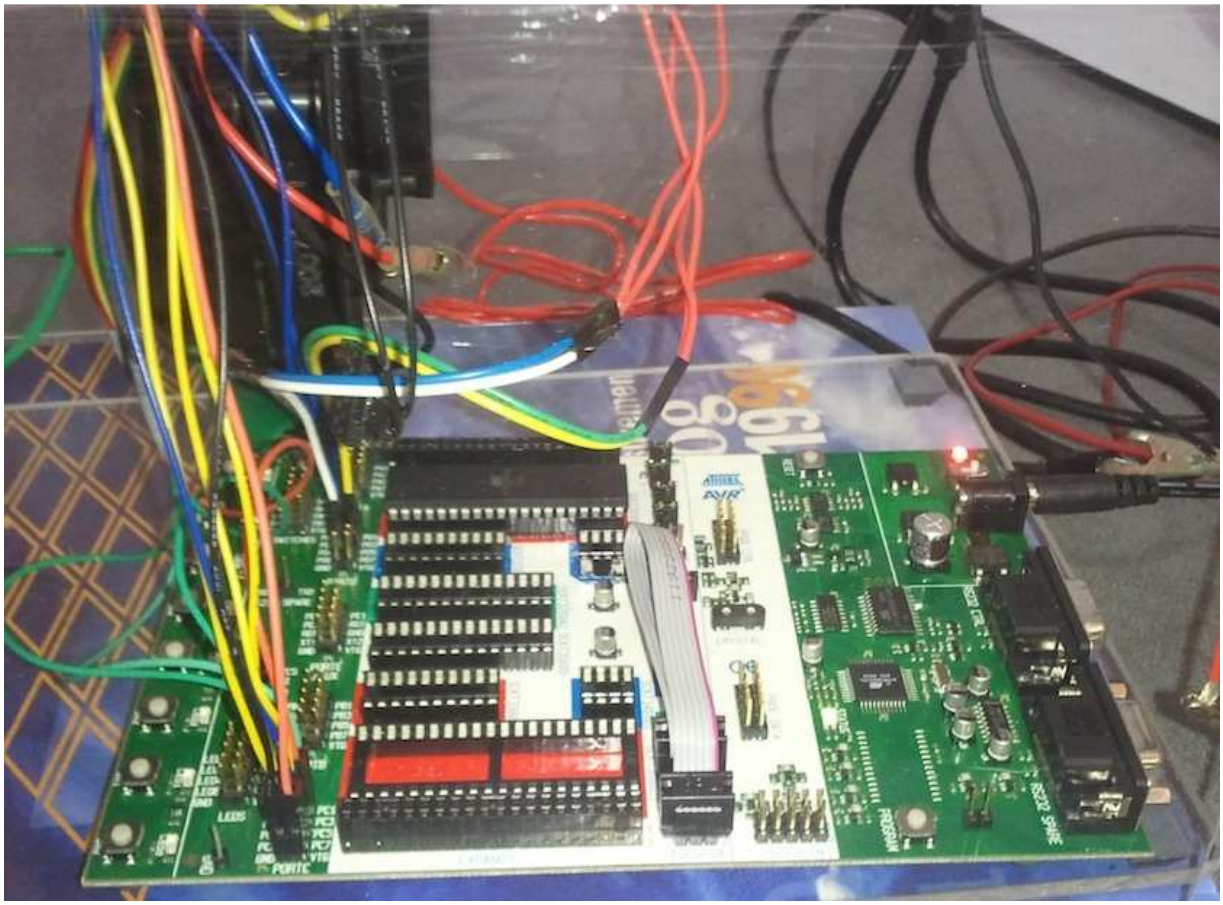


Figure VI-III: STK500 Development Board with ATmega32 Microcontroller

C. Enclosure and Ground Plane

The system enclosure supports the telescoping antenna and motor in an upright position, as well as mounting locations for the LCD and switches. The final enclosure design is acrylic and measures 11 inches wide, 11 inches deep, and 14 inches tall, as shown in Figure VI-IV. Epoxy was used to bond the sheets together and secure the shelf inside the enclosure.

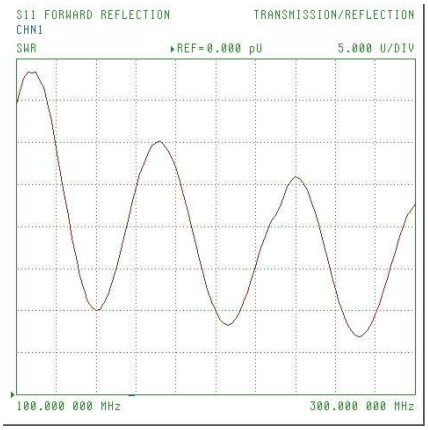
The ground plane was necessary to increase monopole antenna efficiency. In an end-fed monopole arrangement, a ground plane is necessary to provide a "virtual image" of the missing half when compared to a dipole antenna setup. The motor and antenna assembly is intended for an automobile antenna, the car body acts as a ground plane. Since the original plan was to have the antenna operate in the 1.25 m and 70 cm amateur radio bands, the ground plane elements were cut accordingly. This resulted in $\lambda/4$ lengths of about 17.2 and 33.6 cm, 1.25 m and 70 cm bands [8]. The rods were arranged in a circle and alternated between17.2 and 33.6 cm lengths with 4 elements of each length used. The rods were then connected at the center by a copper wire circle and this was connected to the telescoping antenna base with wire, as shown in Figure VI-IV.

The motor, antenna, ground plane, and enclosure were tested together with a frequency sweep, without the microcontroller or SWR analyzer. The frequency sweep was performed from 100-300 MHz and the SWR graph, in Figure VI-V, was observed. A trough in the SWR was apparent and the location of this trough increased in frequency as antenna length decreased from 30.5 to 0 inches. When the antenna was fully retracted, the trough in the SWR reached about 250 MHz and frequencies above that were not affected by change in antenna length. The original plan was to operate in the UHF amateur radio band (420-450 MHz); however this discovery made operation in this band not possible without the addition of a matching network.

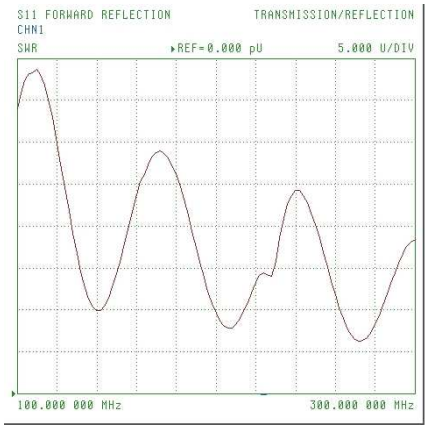
The ground rods were kept at the same length for the change to operation in the 2 m amateur band. This was due to good performance at this frequency and for practical reasons of having shorter ground rods, which made transporting the system more convenient.

Figure VI-IV: Enclosure with Ground Plane

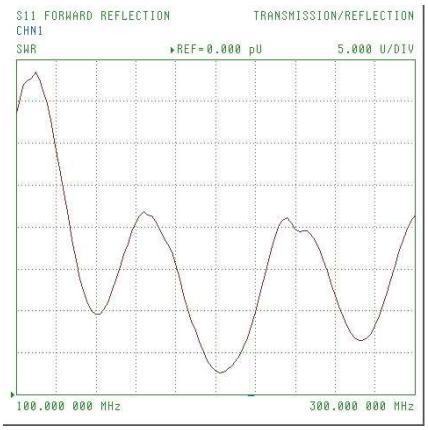
0 inches



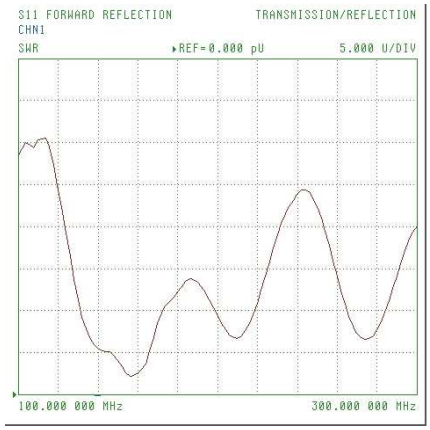
6 inches



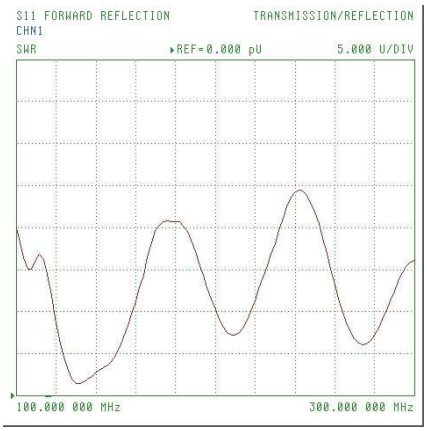
12 inches



18 inches



24 inches



30 inches

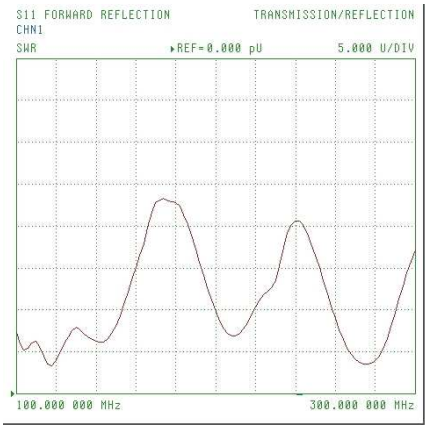


Figure VI-V: SWR Change with Element Extension, 5U/Vertical Div.

D. Final Assembly

The display and switch mounting, along with the PCBs, was done using a Dremel rotary tool and a hand drill. The switches were panel-mount switches that snapped into an appropriately sized slot using tabs. The LCD display and PCBs were mounted using nylon standoffs and washers, along with #2 and #4 sized machine screws. The final assembly requires power from a 12V DC source, which makes it possible to use the system portably using a common 12V lead-acid battery. Shown below in Figure VI-VIis the completed enclosure with the switches and display mounted at the front. The back and enclosure bottom were left open for convenience to wire the components together.

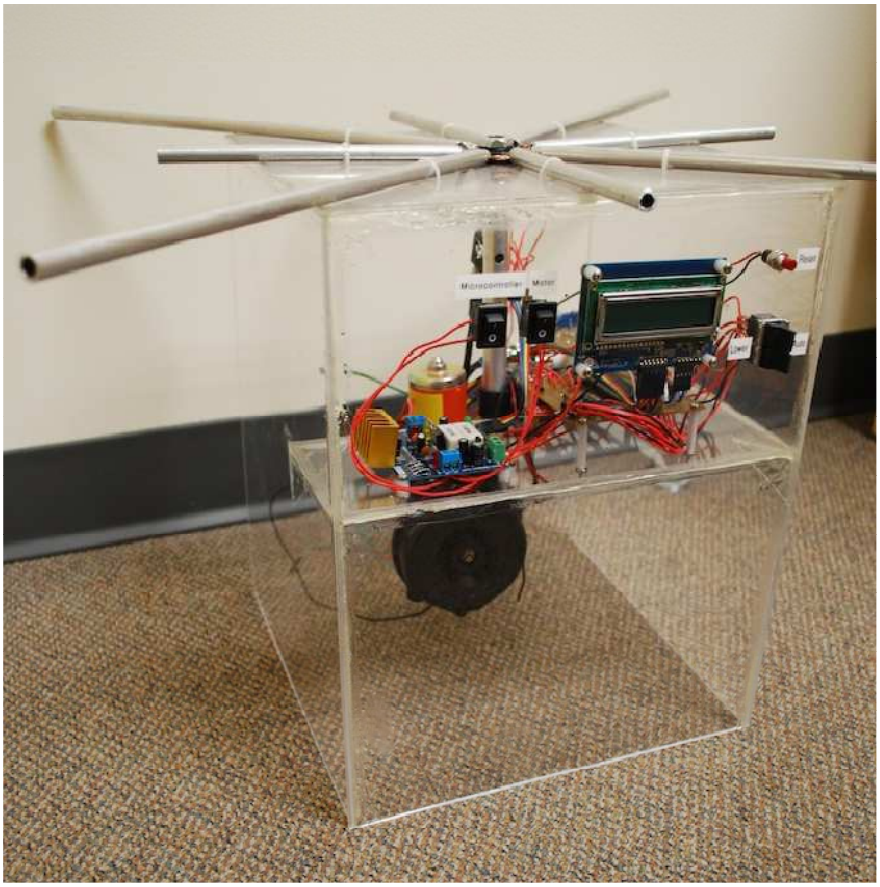


Figure VI-VI: Final Assembly

VII. Test Results

The SWR analyzer comparison test to theMFJ-269 SWR analyzer is shown in Figure VII-I. The results show that the correlation above an SWR Ratio of about 9:1 tends to drift, however the inaccuracy at this high of an SWR ratio is not important because this is not a region where the antenna is tuning. There is a very close relationship at SWRs of 9:1 or less, which verifies SWR Board accuracy.

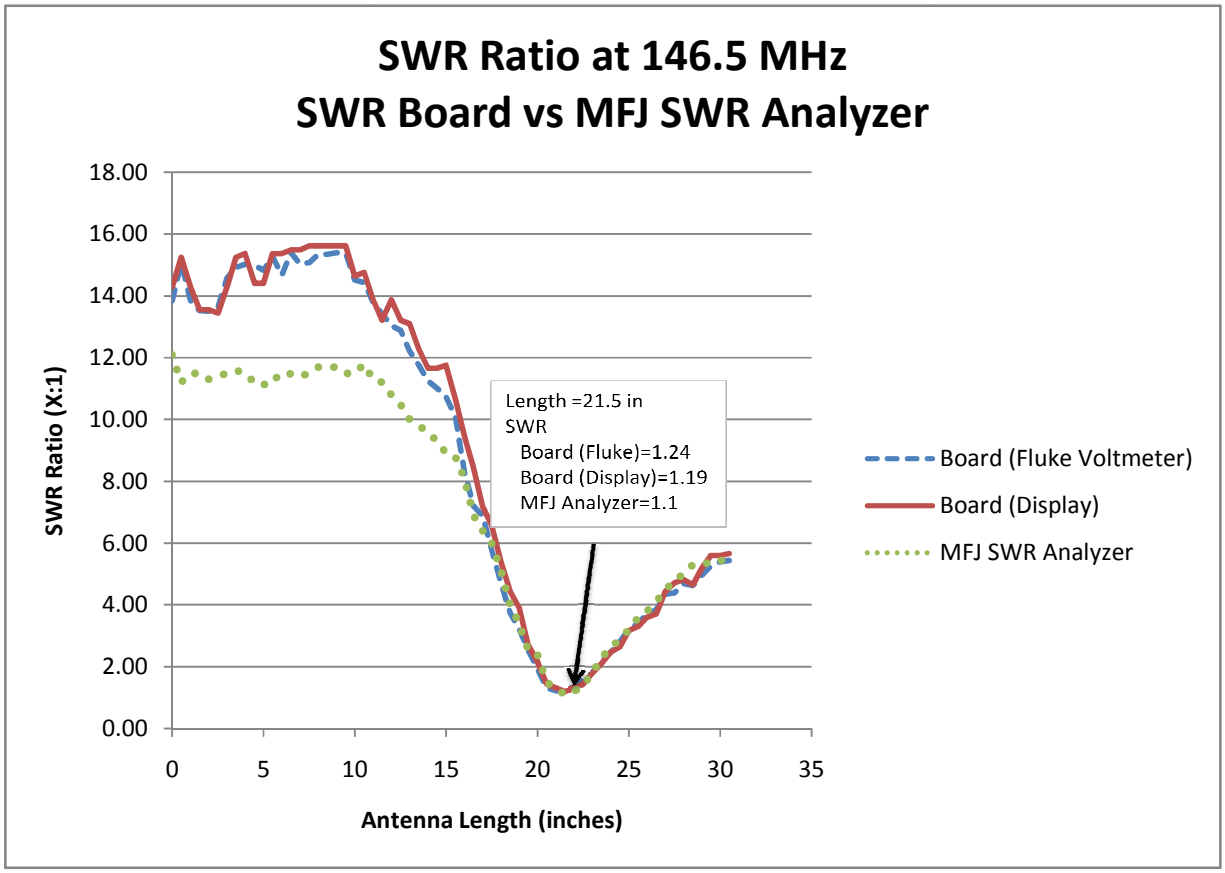


Figure VII-I: SWR comparison between MFJ SWR analyzer, microcontroller SWR display, and SWR analyzer PCB

The integrated auto-tuning antenna system was tested by executing the automatic tuning sequence for several trials to check for consistency in the final SWR value. Figure VII-II shows the test results, which are typically less than a 2:1 SWR ratio.

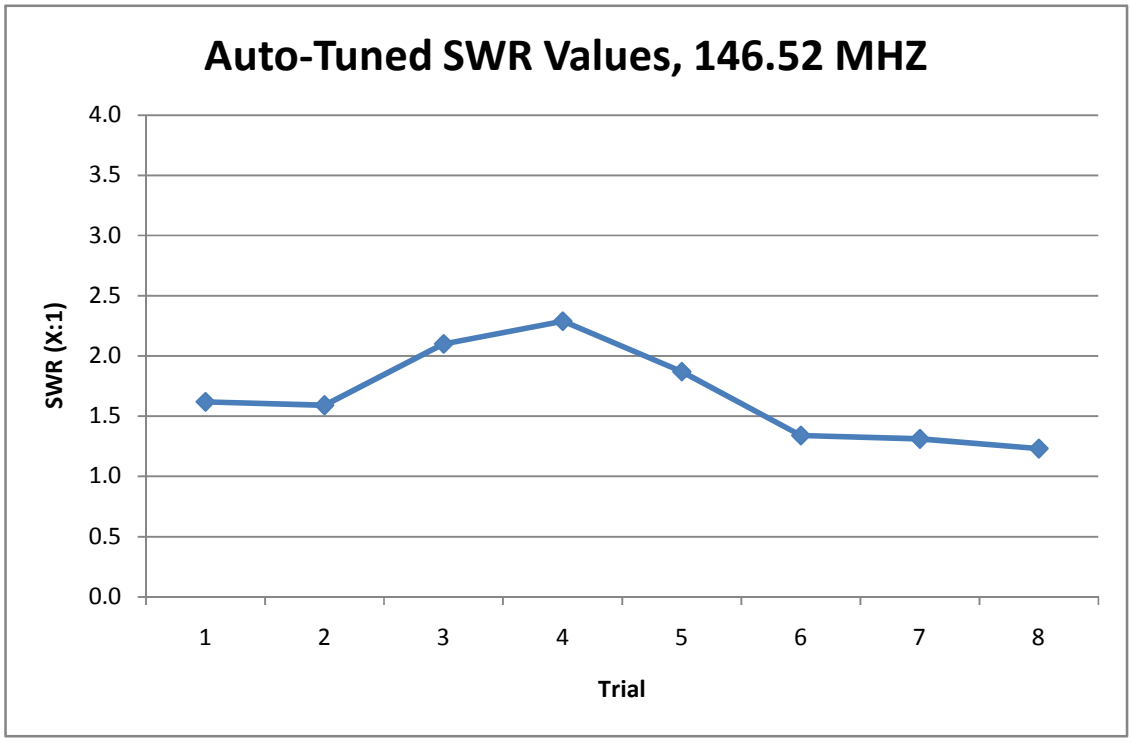


Figure VII-II: Integrated System Auto-Tuned SWR Values at 146.52 MHz

VIII. Improvement and Further Development Possibilities

The Auto-tuning antenna provides an improved tuning method for amateur radio operators. This antenna operates at a transmitting frequency of 146.5 MHz. To become a more functional product, the system must incorporate additional amateur radio frequency ranges. This might be possible by attempting to disassemble the antenna to check for any added in-line components between the connector and feedpoint. This approach would be difficult because of manufacturing methods used that make the antenna complicated to disassemble. A trial-and-error approach with a frequency sweep of several off-the-shelf antennas could also be used. Finally, a standalone telescoping antenna, motor and an enclosure can be purchased separately and this would allow an antenna with more length range to be used.

Implementing the Atmega32 on a PCB minimizes device volume. This was attempted during the course of the project, but complications arose regarding a system clock. The program requires an external crystal across the Atmega32 XTAL1 and XTAL2 pins. After the board was constructed, the software was installed on the microcontroller, but the program did not execute as expected. The crystal was probed using an oscilloscope but no recognizable signal was captured. It was concluded that the crystal failed. Insufficient time and lack of a second crystal prevented further investigation into the problem. With more time, this PCB would yield a more compact system.

Designing a PCB to supply power to individual components would eliminate a large power supply. The motor driver requires +6V and up to 3A when driving the motor. The Atmega32 requires +5V and less than 2mA while in active mode. A circuit with voltage regulators could provide power to both devices. Parallel voltage regulators must be used to achieve motor driver current requirements.

IX. Conclusion

The project was successful in achieving the goal of automatically adjusting antenna length to provide a sufficiently low SWR of about 2:1 or lower for radio transmission. After fixing intermittent connections with the BNC connectors, the fabricated SWR analyzer values correlated with the commercial SWR analyzer values. After implementing several optimization algorithms, a consistent algorithm was developed that is not susceptible to noise fluctuations in voltage. Unfortunately, the antenna assembly did not function within the frequency range that was originally desired, but the system operates within the 2 m amateur band.

X. Bibliography

[1] DC Motor. *Wikipedia*. [Online] 2011 13-November. [Cited: 2011 15-Novemeber.] http://en.wikipedia.org/wiki/DC_motor.

[2] Brushed DC Electric Motor. *Wikipedia*. [Online] 2011 13-November. [Cited: 2011 15-November.] http://en.wikipedia.org/wiki/Brushed_DC_electric_motor.

[3] Brushless DC Electric Motor. *Wikipedia*. [Online] 2011 9-November. [Cited: 2011 15-November.] http://en.wikiepdia.org/wiki/Brushless_DC_electric_motor.

[4] Stepper Motor. *Wikipedia*. [Online] 2011 14-November. [Cited: 2011 15-November.] http://en.wikipedia.org/wiki/Stepper_motor.

[5] Mornhinweg, Manfred. Wideband SWR Meter. *Homo Ludens*. [Online] http://ludens.cl/Electron/swr/swr.html.

[6] Standing Wave Ratio. *Wikipedia*. [Online] 2011 4-April. http://en.wikipedia.org/wiki/Standing_wave_ratio.

[7] DIGILENT. *PmodCLP Parallel LCD Display Module Reference Manual*. [PDF] Pullman : Digilent, 2008. 502-142.

[8] Carr, Joseph J. *Practical Antenna Handbook*. 2001.

[9] How To Select A DC Motor. *MICROMO*. [Online] 2007. http://www.micromo.com/how-to-select-a-dc-motor.apsx#

[10] ATMEL. *ATmega32 Datasheet*. [PDF] San Jose : ATMEL, 2010. 2503.

[11] HVW Technologoies. *L298 Motor Controller Kit*. [PDF] Calgary : HVW Technologies. V3.1.

[12] STMicroelectronics. *L298 Dual Full-Bridge Driver*. [PDF] Italy : STMicroelectronics, 2000. L298.

[13] Cana Kit Corporation. *L298 H-Bridge Dual Bidirectional Motor Driver (2 x 2A)*. [PDF] North Vancouver : Cana Kit Corporation, 2010. CK1122.

Appendices

A. Specifications and Requirements

SWR Analyzer

Power Source	Provided from RF input
Input RF Power	2 Watts maximum
Output Signal Voltage	0-1.4 Volts DC
Connectors	Transmitter Input, BNC Female Antenna Output, BNC Female Forward and Reverse Voltage, 2 Pin Header
Dimensions	1.5 x 2.25 inches

Antenna

Input Voltage	4-13.8 Volts DC
Input Current	0.5 Amps
Extended Length	30.5 Inches
Dimensions	4 x 2.5 x 12 inches

L298 Motor Driver

Input Voltage	6-35 Volts DC
Output Current	2 Amps per channel
Dimensions	2.5 x 3 inches (including heat sink)
Connectors	DC Power Input, Terminal Block Motor 1 and 2 Output, Terminal Block Microcontroller Signals, 10 Pin Header

STK500

Input Voltage	10-15 Volts DC
Programming Interface	RS232; ISP
Recommended Programming Software	AVR Studio 3.2 or higher
Device Support	ATtiny11, ATtiny12, ATtiny15, ATtiny22, ATtiny28, AT90S1200, AT90S2313, AT90S2323, AT90S2333, AT90S2343, AT90S4414, AT90S4433, AT90S4434, AT90S8515, AT90S8535, ATmega8, ATmega16, ATmega161, ATmega163, ATmega323, ATmega103, ATmega128, ATmega32

Atmega32

Input Voltage	4.5-5.5 Volts DC
Speed Grades	0-16 MHz
Power Consumption	Active: 1.1 mA Idle Mode: 0.35 mA Power-down Mode: < 1 µA
Memory	32 Kbytes of In-System Self-programmable Flash program memory 1024 Bytes of EEPROM 2 Kbytes of Internal SRAM
Data Retention	20 years at 85°C/100 years at 25°C
Peripheral Features	Two 8-bit Timer/Counters One 16-bit Timer/Counter Real Time Counter with Separate Oscillator Four PWM Channels 8-Channel 10-bit ADC Byte-oriented Two-wire Serial Interface Serial USART SPI Serial Interface Programmable Watchdog Timer On-chip Analog Comparator
I/O	32 Programmable IO Lines

B. Parts List and Costs

SWR Analyzer:

PART NUMBER	PART DESCRIPTION	QUANTITY	UNIT COST
541-100AACT	100Ω, ½ W Resistor	6	$0.29
311-15.0KFRCT	15kΩ, ¼ W Resistor	2	$0.03
3310C-001-103L	10kΩ Potentiometer	2	$2.56
A97553	BNC Connector	2	$2.30
	1N34 Diodes	2	$0.45
587-1069-1	1nF Capacitor	4	$0.33
		TOTAL:	**$13.74**

Hardware:

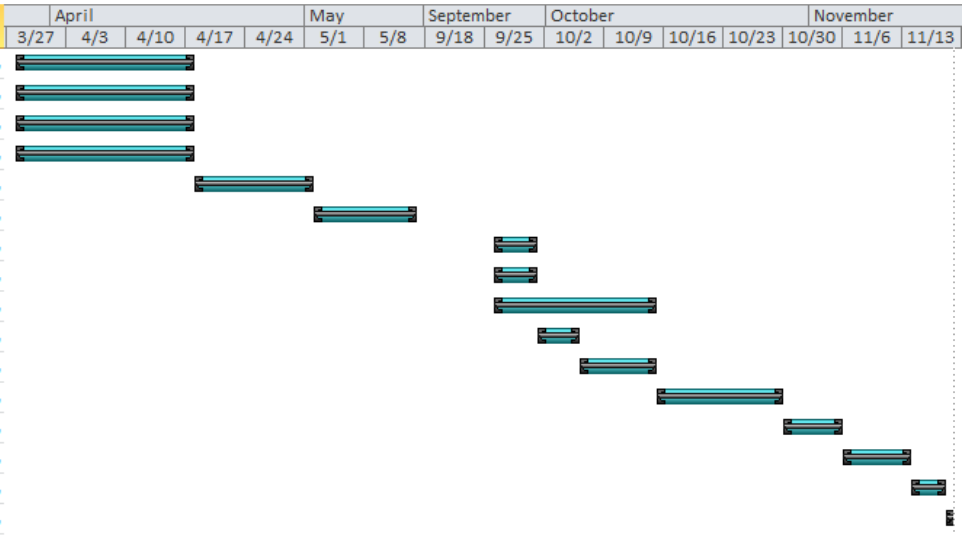
PART NUMBER	PART DESCRIPTION	QUANTITY	UNIT COST
-	Motorized Antenna	1	$29.99
ATSTK500	Atmel AVR STK500 Development Board	1	Donated ($79)
ATMEGA32A-PU	Atmega32 Processor	1	$8.28
PMOD-CLP	PmodCLP LCD	1	$25.00
ROB-09670	Motor Driver 2A Dual L298 H-Bridge	1	$34.95
PRT-10362	Jumper Wire - 0.1”, 2-pin, 4”	8	Donated ($0.95)
PRT-10374	Jumper Wire - 0.1", 4-pin, 12"	1	$1.95
PRT-10366	Jumper Wire - 0.1", 6-pin, 4"	2	$0.95
PRT-10364	Jumper Wire - 0.1", 4-pin, 4"	1	$0.95
3-6437630-7	Switch	2	Sample
PRASA1-16L-BB0BW	On-Off Switch	2	Sample
	Pushbutton Switch	1	$1.00
	BNC Male Connector	3	$1.29
	RG-58/U Coaxial Cable	3 feet	$0.46
		TOTAL:	**$109.27**

Enclosure Hardware:

PART NUMBER	PART DESCRIPTION	QUANTITY	UNIT COST
	Acrylic Sheets	5	$3.98
	Aluminum Rods		Donated, scrap metal
	Copper Wire		Donated
	Epoxy	2	$5.00
	Screws, standoffs, nuts, washers		$13.52
		TOTAL:	**$43.42**

C. Schedule – Time Estimates

	Task Name	Start	Finish	Task Lead	% Complete
1	Identify Suitable Antenna	Mon 3/28/11	Sun 4/17/11	John	100%
2	Design SWR Sensor	Mon 3/28/11	Sun 4/17/11	John	100%
3	Identify Suitable Motor	Mon 3/28/11	Sun 4/17/11	Jen	100%
4	Indentify Suitable Microcontroller	Mon 3/28/11	Sun 4/17/11	Jen	100%
5	Create Design & Integration Plan	Sun 4/17/11	Sun 5/1/11	John & Jen	100%
6	Obtain Components	Mon 5/2/11	Fri 5/13/11	John & Jen	100%
7	Build SWR Analyzer	Mon 9/26/11	Fri 9/30/11	John	100%
8	Interface Motor Driver w/ Microcontroller	Mon 9/26/11	Fri 9/30/11	Jen	100%
9	Order PCB for Microcontroller Circuitry	Mon 9/26/11	Fri 10/14/11	Jen	100%
10	Characterize SWR Analyzer	Sat 10/1/11	Wed 10/5/11	John	100%
11	Interface SWR w/ Microcontroller	Thu 10/6/11	Fri 10/14/11	Jen	100%
12	Program Microcontroller	Sat 10/15/11	Sat 10/29/11	Jen	100%
13	Test/Debug	Sun 10/30/11	Sat 11/5/11	John & Jen	100%
14	Final Construction	Sun 11/6/11	Sun 11/13/11	John & Jen	100%
15	Improvements	Mon 11/14/11	Thu 11/17/11	John & Jen	100%
16	Demo Project	Fri 11/18/11	Fri 11/18/11	John & Jen	100%



D. PC Board Layout

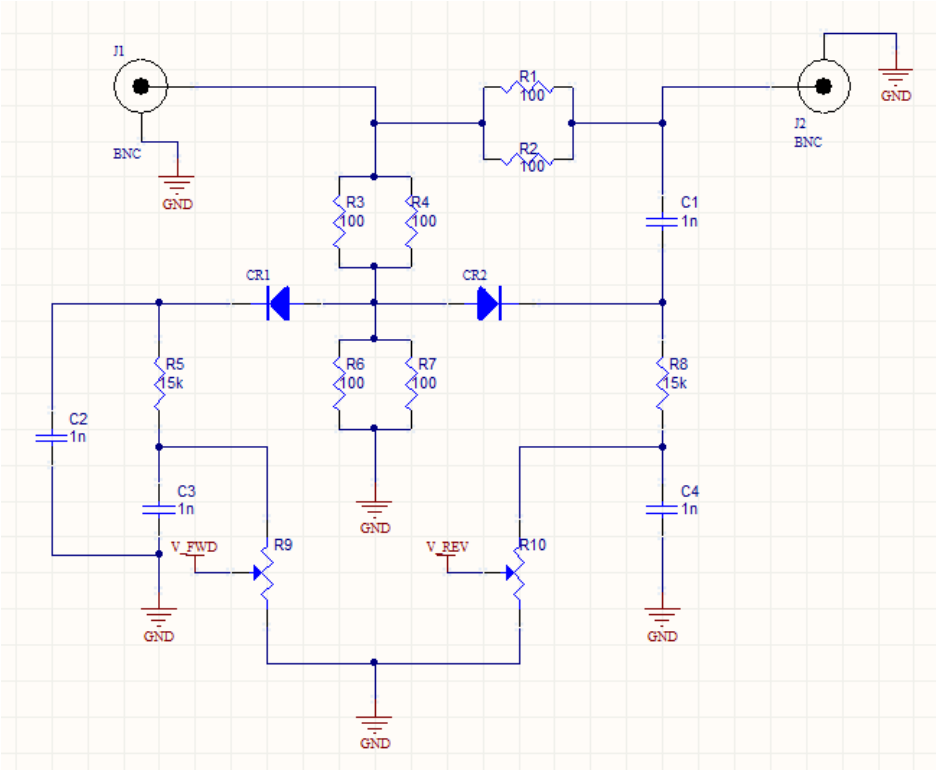


Figure D-I: SWR Analyzer Schematic (PCB)

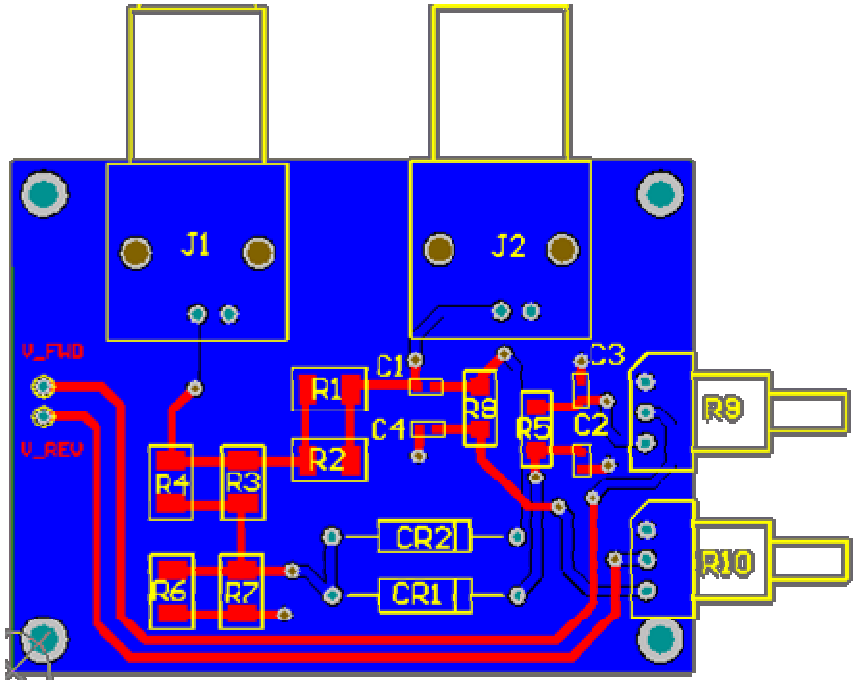


Figure D-II: SWR Analyzer PCB Layout

E. Software

Main.c

```
#define F_CPU        3686400
#include "LCD.h"
#include "initio.h"
#include "ADC.h"
#include "Motor.h"
#include "SWR.h"

int main(void)
{
        initio();
        initLCD();
        initADC();

        printASCII('M');
        printASCII('e');
        printASCII('a');
        printASCII('s');
        printASCII('u');
        printASCII('r');
        printASCII('i');
        printASCII('n');
        printASCII('g');
        printASCII('.');
        printASCII('.');
        printASCII('.');

        optimizeSWR();

        return 0;
}
```

Initio.c

```
#include "initio.h"
#include<avr/io.h>
#include<stdint.h>
//-------------------------------------------------------------------
//- Function: initIO()
//-
//- return value: none
//- parameters: none
//-
//- Description:      This function initializes the input and output
//-                   ports used.
//-------------------------------------------------------------------
void initio(void)
{
        //Inputs
```

```
	DDRA = 0x00;	//SWR ANALYZER
			//PA0 - V_FWD
			//PA1 - V_REV

	//Outputs
	DDRC = 0xFF;	//LCD DATA PINS
			//PC0 - LCD J1.1
			//PC1 - LCD J1.2
			//PC2 - LCD J1.3
			//PC3 - LCD J1.4
			//PC4 - LCD J1.7
			//PC5 - LCD J1.8
			//PC6 - LCD J1.9
			//PC7 - LCD J1.10

	DDRD = 0xFF;	//LCD CONTROL PINS
			//PD0 - LCD J2.1 RS
			//PD1 - LCD J2.2 R/W
			//PD2 - LCD J2.3 E

			//MOTOR DRIVER
			//PD3 - MTR IN3
			//PD4 - MTR ENB
			//PD5 - MTR IN4
			//PD6 - MTR CSB

	PORTA = 0x00;	//Clear PORTA
	PORTC = 0x00;	//Clear PORTC
	PORTD = 0x00;  //Clear PORTD
}
```

Initio.h

```
/*****************************************************************************
*	Filename: initio.h
*
*	Description:	 Contains Function and task prototypes for IO initialization
*			as well as relevant #defines used in the IO initialization.
*****************************************************************************/

#include<stdint.h>
void initio(void);
```

LCD.c

```
/*****************************************************************************
*	Filename: LCD.c
*
*
*	Description:	This file contains functions to drive the PMOD CLP LCD,
*			which include initializing, clearing, and printing to the
*			screen.
*****************************************************************************/
```

```
#define F_CPU 3686400
#include "LCD.h"
#include <avr/io.h>
#include <stdint.h>
#include <util/delay.h>

//-------------------------------------------------------------------------
//- Function: initLCD()
//-
//- Return value: none
//- Parameters: none
//-
//- Description: This function initializes the LCD.
//-------------------------------------------------------------------------
void initLCD(void)
{
	//Function Set
	_delay_ms(20);

	PORTD = (1<<PORTD2);
	PORTC = 0x38;
	PORTD = (0<<PORTD2);

	//Display On
	_delay_us(37);

	PORTD = (1<<PORTD2);
	PORTC = 0x0F;
	PORTD = (0<<PORTD2);

	//Display Clear
	_delay_us(37);

	PORTD = (1<<PORTD2);
	PORTC = 0x01;
	PORTD = (0<<PORTD2);

	//Entry Mode
	_delay_ms(1.52);

	PORTD = (1<<PORTD2);
	PORTC = 0x06;
	PORTD = (0<<PORTD2);

	//Write Ascii
	_delay_ms(1.52);
}

//-------------------------------------------------------------------------
//- Function: clearLCD()
//-
//- Return value: none
//- Parameters: none
//-
//- Description: This function clears the LCD when called.
//-------------------------------------------------------------------------
void clearLCD()
{
```

```
		PORTD = (1<<PORTD2);
		PORTC = 0x01;
		PORTD = (0<<PORTD2);
		_delay_ms(1.52);
}

//-------------------------------------------------------------------------
//- Function: printASCII()
//-
//- Return value: none
//- Parameters: character
//-
//- Description: This function takes in an ASCII character and displays it
//-		on the LCD.
//-------------------------------------------------------------------------
 void printASCII(uint8_t character)
{
		PORTD = (1<<PORTD0);
		PORTD = (1<<PORTD2) | (1<<PORTD0);
		PORTC = character;
		PORTD = (1<<PORTD0);
		_delay_us(37);

}
```

LCD.h

```
#include<stdint.h>
//-------------------------------------------------------------------------
//- Function: initLCD()
//-
//- Return value: none
//- Parameters: none
//-
//- Description: This function initializes the LCD.
//-------------------------------------------------------------------------
void initLCD(void);

//-------------------------------------------------------------------------
//- Function: clearLCD()
//-
//- Return value: none
//- Parameters: none
//-
//- Description: This function clears the LCD when called.
//-------------------------------------------------------------------------
void clearLCD();

//-------------------------------------------------------------------------
//- Function: printASCII()
//-
//- Return value: none
//- Parameters: character
//-
//- Description:	This function takes in an ASCII character and displays it
//-			on the LCD.
//-------------------------------------------------------------------------
void printASCII(uint8_t character);
```

ADC.c

```
/*************************************************************************
*           Filename: ADC.c
*
*
*           Description:        This file contains functions to initialize the built-in
*                               analog-to-digital converter on the ATmega32 and read the
*                               voltage values on specified pins.
*************************************************************************/

#include "ADC.h"
#include <avr/io.h>
#include <stdint.h>

//-------------------------------------------------------------------------
//- Function: initADC()
//-
//- return value: none
//- parameters: none
//-
//- Description:      This function initializes the ADC on PORTA of the ATmega32.
//-                   it sets the reference voltage to use the built-in internal
//-                   voltage, enables the ADC and sets the prescaler.
//-------------------------------------------------------------------------
void initADC(void)
{
            //Set Reference Voltage to use internal 2.56V
            ADMUX = (1<<REFS1)|(1<<REFS0);

            //Enable the ADC and set prescaler to clk/128
            ADCSRA = (1<<ADEN)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0);
}

//-------------------------------------------------------------------------
//- Function: Read_ADC()
//-
//- return value: voltage on specified channel
//- parameters: channel to read the voltage on
//-
//- Description:      This function takes in the desired channel on which to read
//-                   the voltage and converts it from an analog reading to a
//-                   digital reading. It then returns the reading and disables
//-                   the ADC.
//-------------------------------------------------------------------------
double Read_ADC(uint8_t channel)
{
            double return_val;

            //enable ADC
            ADCSRA |= (1<<ADEN);
```

```
	//set the channel we are pulling
	ADMUX = channel;

	//start conversion
	ADCSRA |= (1<<ADSC);

	//wait for conversion to finish
	while(!(ADCSRA & (1<<ADIF)));

	return_val = ADCL|(ADCH<<SHIFT_8);

	//disable ADC
	ADCSRA &= ADC_OFF;

	return return_val;
}
```

ADC.h

```
/*************************************************************************
*	Filename: ADC.h
*
*
*	Description:	Contains Function and task prototypes for IO initialization
*			as well as relevant #defines used in the IO initialization.
*************************************************************************/

#include <stdint.h>
#define SHIFT_8	8
#define ADC_OFF	0x7F
//-------------------------------------------------------------------------
//- Function: initADC()
//-
//- return value: none
//- parameters: none
//-
//- Description:	This function initializes the ADC on PORTA of the ATmega32.
//-			it sets the reference voltage to use the built-in internal
//-			voltage, enables the ADC and sets the prescaler.
//-------------------------------------------------------------------------
void initADC(void);

//-------------------------------------------------------------------------
//- Function: Read_ADC()
//-
//- return value: voltage on specified channel
//- parameters: channel to read the voltage on
//-
//- Description:	This function takes in the desired channel on which to read
//-			the voltage and converts it from an analog reading to a
//-			digital reading. It then returns the reading and disables
```

```
//-                     the ADC.
//---------------------------------------------------------------------
doubleRead_ADC(uint8_t channel);
```

Motor.c

```
/*****************************************************************************
*       Filename: Motor.c
*
*       Description: This file contains the functions necessary to control the
*        operation of the motor driver, which drives the antenna
*        motor, thus adjusting antenna length.
*****************************************************************************/

#include "Motor.h"
#include <avr/io.h>
#include <stdint.h>

//---------------------------------------------------------------------
//- Function: motorOFF()
//-
//- return value: none
//- parameters: none
//-
//- Description:    This function disables the output of the motor driver, thus
//-                 turning off Motor A and Motor B. Note: PD0-IN3, PD2-ENB,
//-                  PD4-IN4, PD6-CSB.
//---------------------------------------------------------------------
void motorOFF()
{
        PORTD = (0<<PORTD3)|(0<<PORTD4)|(0<<PORTD5)|(0<<PORTD6);
}

//---------------------------------------------------------------------
//- Function: motorFWD()
//-
//- return value: none
//- parameters: delay
//-
//- Description:    This function enables the output to Motor B and sets IN3 and
//-                 IN4 to turn the motor in the forward direction.
//---------------------------------------------------------------------
void motorFWD()
{
        PORTD = (1<<PORTD3)|(1<<PORTD4)|(0<<PORTD5)|(0<<PORTD6);
//      _delay_ms(delay);
//      motorOFF();
}

//---------------------------------------------------------------------
//- Function: motorREV()
```

```
//-
//- return value: none
//- parameters: delay
//-
//- Description:	This function enables the output to Motor B and sets IN3 and
//-			IN4 to turn the motor in the reverse direction.
//--------------------------------------------------------------------------
void motorREV()
{
		PORTD = (0<<PORTD3)|(1<<PORTD4)|(1<<PORTD5)|(0<<PORTD6);
//		_delay_ms(delay);
//		motorOFF();
}
```

Motor.h

```
#include<stdint.h>

//--------------------------------------------------------------------------
//- Function: motorOFF()
//-
//- return value: none
//- parameters: none
//-
//- Description: This function disables the output of the motor driver, thus
//-			turning off Motor A and Motor B. Note: PD0-IN3, PD2-ENB, PD4-IN4, PD6-CSB.
//--------------------------------------------------------------------------
void motorOFF();

//--------------------------------------------------------------------------
//- Function: motorFWD()
//-
//- return value: none
//- parameters: none
//-
//- Description:	This function enables the output to Motor B and sets IN3 and
//-		IN4 to turn the motor in the forward direction.
//--------------------------------------------------------------------------
void motorFWD();

//--------------------------------------------------------------------------
//- Function: motorREV()
//-
//- return value: none
//- parameters: none
//-
//- Description:	This function enables the output to Motor B and sets IN3 and
//-			IN4 to turn the motor in the reverse direction.
//--------------------------------------------------------------------------
void motorREV();
```

SWR.c

```
/*****************************************************************************
*       Filename: SWR.c
*
*       Description:    This file includes functions to measure the standing-wave
*                       ratio (SWR) and optimize the SWR by adjusting the antenna
*                       length.
*****************************************************************************/

#include "SWR.h"
#include "ADC.h"
#include "LCD.h"
#include "Motor.h"

//-----------------------------------------------------------------------------
//- Function: calculateSWR()
//-
//- return value: swr
//- parameters: forward voltage, reverse voltage
//-
//- Description:  This function takes in the voltages measured on PA0 and PA1
//-               and uses them to calculate the integer value of the SWR.
//-----------------------------------------------------------------------------
double calculateSWR(double fwd, double rev)
{
        double swr = 0;
        swr = (1+(rev/fwd))/(1-(rev/fwd));
        return swr;
}

//-----------------------------------------------------------------------------
//- Function: calculateDecimal()
//-
//- return value: decimal
//- parameters: forward voltage, reverse voltage
//-
//- Description: This function takes in the voltages measured on PA0 and PA1
//-               and uses them to calculate the decimal value of the SWR.
//-----------------------------------------------------------------------------
uint16_t calculateDecimal(double swr)
{
        uint16_t decimal = 0;
        decimal = (swr - (uint16_t)swr)*100;
        return decimal;
}

//-----------------------------------------------------------------------------
//- Function: optimizeSWR()
//-
//- return value: none
//- parameters: none
```

```
//-
//- Description:      This function adjusts the antenna length (through programming
//-                   the motor driver) and measures the SWR at multiple locations
//-                   along the antenna. The minimum SWR is determined and the
//-                   antenna is adjusted to where the minimum SWR value occurred.
//-------------------------------------------------------------------------------
void optimizeSWR()
{
        double curr_swr  = 0;
        double v_fwd     = 0;
        double v_rev     = 0;
        uint16_t  decimal   = 0;
        uint16_t  length    = 0;
        uint16_t  delay     = 250;
        double    array[25] = {0};
        double    min       = 0;
        uint16_t  i         = 0;
        uint16_t  j         = 0;

        // Iterate through antenna lengths, measure SWR,
        // and store in an array.
        for(i=0; i<26; i++)
        {
                v_fwd = Read_ADC(ADC0);
                v_rev = Read_ADC(ADC1);
                array[i] = calculateSWR(v_fwd, v_rev);

                motorFWD();
                _delay_ms(delay);
                motorOFF();
        }

        min = array[0];

        // Determine the minimum value in the array.
        for (i=0; i<26; i++)
        {
        if (array[i]<min)
                {
                min = array[i];
                        j = i;
                }
         }

        length = 25-j;

        motorREV();
        for(i=0; i<25-j; i++)
        {
                _delay_ms(delay);
        }
```

```
	motorOFF();

	// Improve the SWR by making finer length adjustments
	fineOptimize(length, curr_swr);
}

//------------------------------------------------------------------------------
//- Function: fineOptimize()
//-
//- return value: none
//- parameters: coarse swr, length
//-
//- Description: This function finds the optimum antenna length by measuring
//-			SWR at smaller iterations than optimizeSWR(). The minimum
//-			 SWR is determined and the antenna is adjusted back to the
//-			location where the minimum SWR occurred.
//------------------------------------------------------------------------------
void fineOptimize(uint16_t length, double coarse_swr)
{
	double curr_swr  = 0;
	double v_fwd		= 0;
	double v_rev		= 0;
	uint16_t  decimal   = 0;
	uint16_t  delay		= 25;
	double    array[10] = {0};
	double    min	= 0;
	double    error	= 0.1;
	uint16_t  i	= 0;
	uint16_t  j	= 0;

	length = length * 5; // Current antenna position

	// If the position of the antenna is more than 125ms from
	// the maximum length of the antenna, search for the optimal
	// swr within a 250ms range.
	if(length <= 246)
	{
		motorREV();
		_delay_ms(125);
		motorOFF();

		length = length - 5;

		v_fwd = Read_ADC(ADC0);
		v_rev = Read_ADC(ADC1);
		array[0] = calculateSWR(v_fwd, v_rev);

		for(i=1; i<10; i++)
		{
			motorFWD();
			_delay_ms(delay);
```

```
		motorOFF();

		v_fwd = Read_ADC(ADC0);
		v_rev = Read_ADC(ADC1);
		array[i] = calculateSWR(v_fwd, v_rev);
}

length = length + 10;

min = array[0];

for (i=0; i<10; i++)
{
		if (array[i]<min)
		{
			min = array[i];
			j = 10 - i; //antenna length at minimum swr
		}
}

if(coarse_swr< min)
{
		min = coarse_swr;
		j = length-5; //antenna length at coarse swr
}

//decrease antenna to the start of the range
motorREV();
_delay_ms(10*delay);
motorOFF();

for(i=0; i<10; i++)
{
		//measure swr
		v_fwd = Read_ADC(ADC0);
		v_rev = Read_ADC(ADC1);
		curr_swr = calculateSWR(v_fwd, v_rev);

		if((curr_swr>= (min-error)) && (curr_swr<= (min+error)))
		{
			decimal = calculateDecimal(curr_swr);
			printSWR(curr_swr, decimal);
			i=9;
		}

		else if((curr_swr>= (min-(2*error))) && (curr_swr<= (min+(2*error))))
		{
			decimal = calculateDecimal(curr_swr);
			printSWR(curr_swr, decimal);
			i=9;
		}
```

```
                        else if((curr_swr>= (min-(8*error))) && (curr_swr<= (min+(8*error))))
                        {
                                decimal = calculateDecimal(curr_swr);
                                printSWR(curr_swr, decimal);
                                i=9;
                        }

                        else if(i==9)
                        {
                                clearLCD();

                                printASCII('E');
                                printASCII('R');
                                printASCII('R');
                                printASCII('O');
                                printASCII('R');
                        }

                        //Iterate antenna length
                        motorFWD();
                        _delay_ms(delay);
                        motorOFF();
                }
        }

        // If the antenna is within 125ms from the maximum length, measure
        // and print the swr at the current antenna length.
        else
        {
                v_fwd = Read_ADC(ADC0);
                v_rev = Read_ADC(ADC1);

                curr_swr = calculateSWR(v_fwd, v_rev);
                decimal = calculateDecimal(curr_swr);

                printSWR(curr_swr, decimal);
        }
}


//-------------------------------------------------------------------------
//- Function: printSWR()
//-
//- return value: none
//- parameters: swr, decimal
//-
//- Description: This function takes in the swr and decimal value and prints
//-              it to the LCD.
//-------------------------------------------------------------------------
void printSWR(uint16_t swr, uint16_t decimal)
```

```
{
	uint8_t i = 0;
	char string[10];

	clearLCD();
	_delay_ms(1.52);

	printASCII('S');
	printASCII('W');
	printASCII('R');
	printASCII(' ');
	printASCII('=');
	printASCII(' ');

	itoa(swr, string, 10);

	for(i=0; i<strlen(string); i++)
	{
		printASCII(string[i]);
	}

	itoa(decimal, string, 10);

	printASCII('.');

	for(i=0; i<strlen(string); i++)
	{
		printASCII(string[i]);
	}
	printASCII(':');
	printASCII('1');
}
```

SWR.h

```
#define F_CPU	3686400

#include<avr/io.h>
#include<stdint.h>
#include<util/delay.h>
#include<string.h>
#include<stdlib.h>

#define ADC0 0xC0 // Channel 0 - PA0
#define ADC1 0xC1 // Channel 1 - PA1

//-------------------------------------------------------------------------
//- Function: calculateSWR()
//-
//- return value: swr
//- parameters: forward voltage, reverse voltage
//-
//- Description: This function takes in the voltages measured on PA0 and PA1
//-		and uses them to calculate the integer value of the SWR.
```

```
//---------------------------------------------------------------------------
double calculateSWR(double fwd, double rev);

//---------------------------------------------------------------------------
//- Function: calculateDecimal()
//-
//- return value: decimal
//- parameters: forward voltage, reverse voltage
//-
//- Description: This function takes in the voltages measured on PA0 and PA1
//-              and uses them to calculate the decimal value of the SWR.
//---------------------------------------------------------------------------
uint16_t calculateDecimal(double swr);

//---------------------------------------------------------------------------
//- Function: optimizeSWR()
//-
//- return value: none
//- parameters: none
//-
//- Description:  This function adjusts the antenna length (through programming
//-              the motor driver) and measures the SWR at multiple locations
//-              along the antenna. The minimum SWR is determined and the
//-              antenna is adjusted to where the minimum SWR value occurred.
//---------------------------------------------------------------------------
void optimizeSWR();

//---------------------------------------------------------------------------
//- Function: fineOptimize()
//-
//- return value: none
//- parameters: coarse swr, length
//-
//- Description: This function finds the optimum antenna length by measuring
//-              SWR at smaller iterations than optimizeSWR(). The minimum
//-              SWR is determined and the antenna is adjusted back to the
//-              location where the minimum SWR occurred.
//---------------------------------------------------------------------------
void fineOptimize(uint16_t length, double coarse_swr);

//---------------------------------------------------------------------------
//- Function: printSWR()
//-
//- return value: none
//- parameters: swr, decimal
//-
//- Description: This function takes in the swr and decimal value and prints
//-              it to the LCD.
//---------------------------------------------------------------------------
void printSWR(uint16_t swr, uint16_t decimal);
```

F. System Schematic