

TESTING

Bugs are closely linked with programming. The best way to remove these bugs is to perform extensive testing. The testing that has been performed to check the boundaries and quality of this module starts with unit testing and is followed up by integration and system testing.

- a. As a part of unit test:
 - a. To test the hardware each component was subjected to varied amount of power to test the deviation in its performance.
 - b. Each component was subjected to random inputs and its outputs were mapped
 - c. Each software module and program was probed with random inputs and its boundary conditions were checked
 - d. After debugging, various tests were repeated to check if the new code has created any new errors.
 - e. Debug points were inserted in the code to check if the code is running as desired.
 - f. Debugging the Latencies in the software to match up to the practical expectations of the user.
- b. Integration testing consisted of:
 - a. Check the sent output and compare it with the input received by the other device
 - b. Checking the output of one function and check its behavior when called by different functions and testing the returned value.
 - c. Test the reliability of connection and test the effect of noise and other interference on it
 - d. Keeping track of the Latencies and processing speed of various controllers to achieve timing synchronization between them, as this is a time sensitive system.

1. Test for RF transmitters and receivers (Also covers testing for integration between the components):

- This test showed the result that the performance of the receiver and the transmitter were affected a lot and the distance at which they were communicating correctly reduced from 40 feet to just 10 feet. This made us realize that extra power supply needs to be used for the transmitter and receivers for them to communicate properly.
- The test also burnt up a couple of ATtiny microcontrollers while using high power signal from the receiver unit to the pin of the controller. So a standard voltage of 5-9 volts is maintained.
- Testing the various baud-rate of transmitting and receiving the data gave us an idea of achieving an handoff between high transmission rate and correctness of message reception over a certain distance. As we increased the baud-rate, more errors were incorporated in our reception. We finalized on a baud rate of 2000 bits per second, which let us communicate at a distance of 30-35 feet.
- The use of external antenna also helps in increasing the range by 10%.

The image below Fig1.1 maps the output of the receiver at close proximity with the transmitter working at 2000 bits per second:

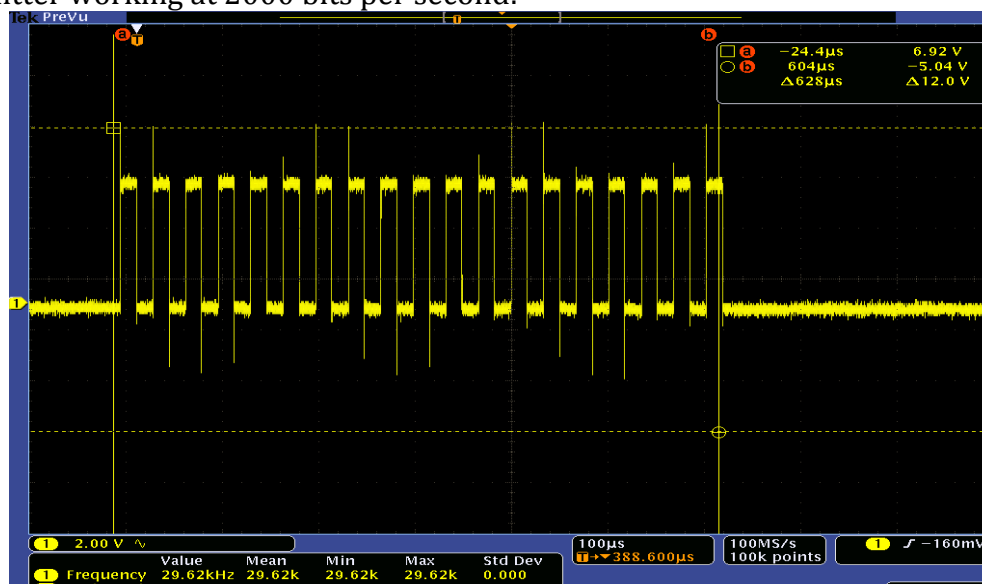


Fig1.1 RF signal on the receiver

The image below Fig1.2 maps the output of the receiver at a distance greater than 50 feet with the transmitter working at 2000 bits per second:

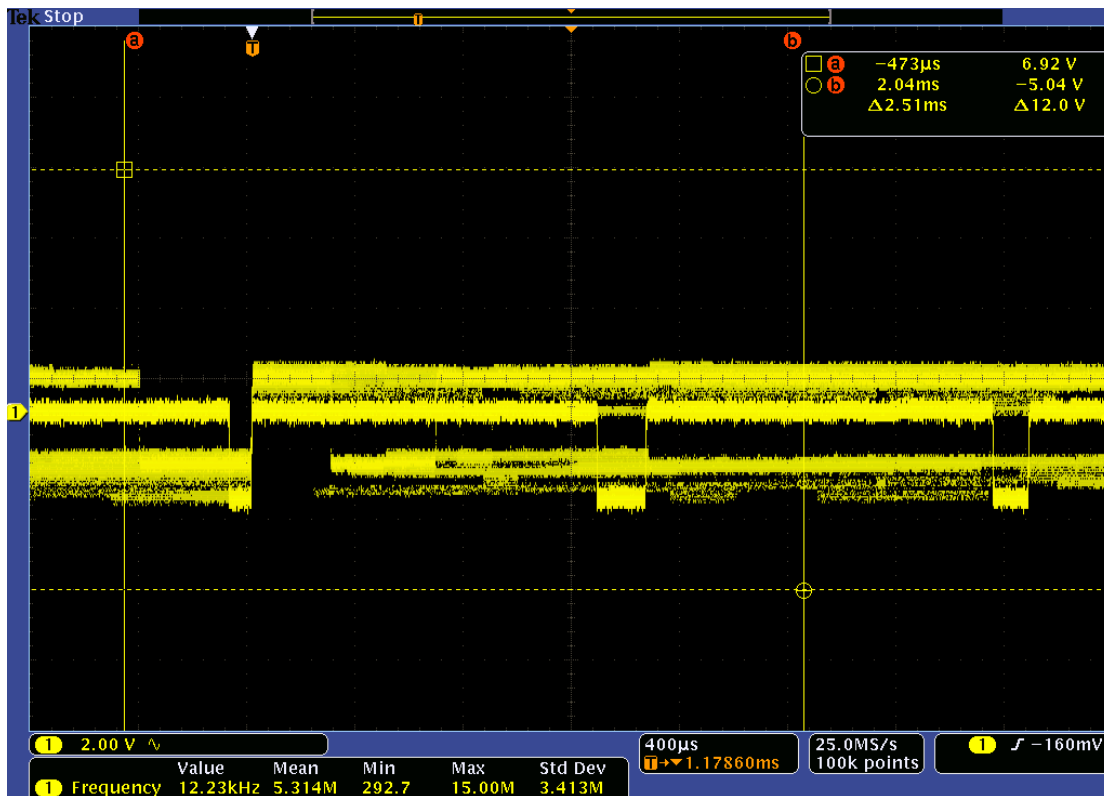
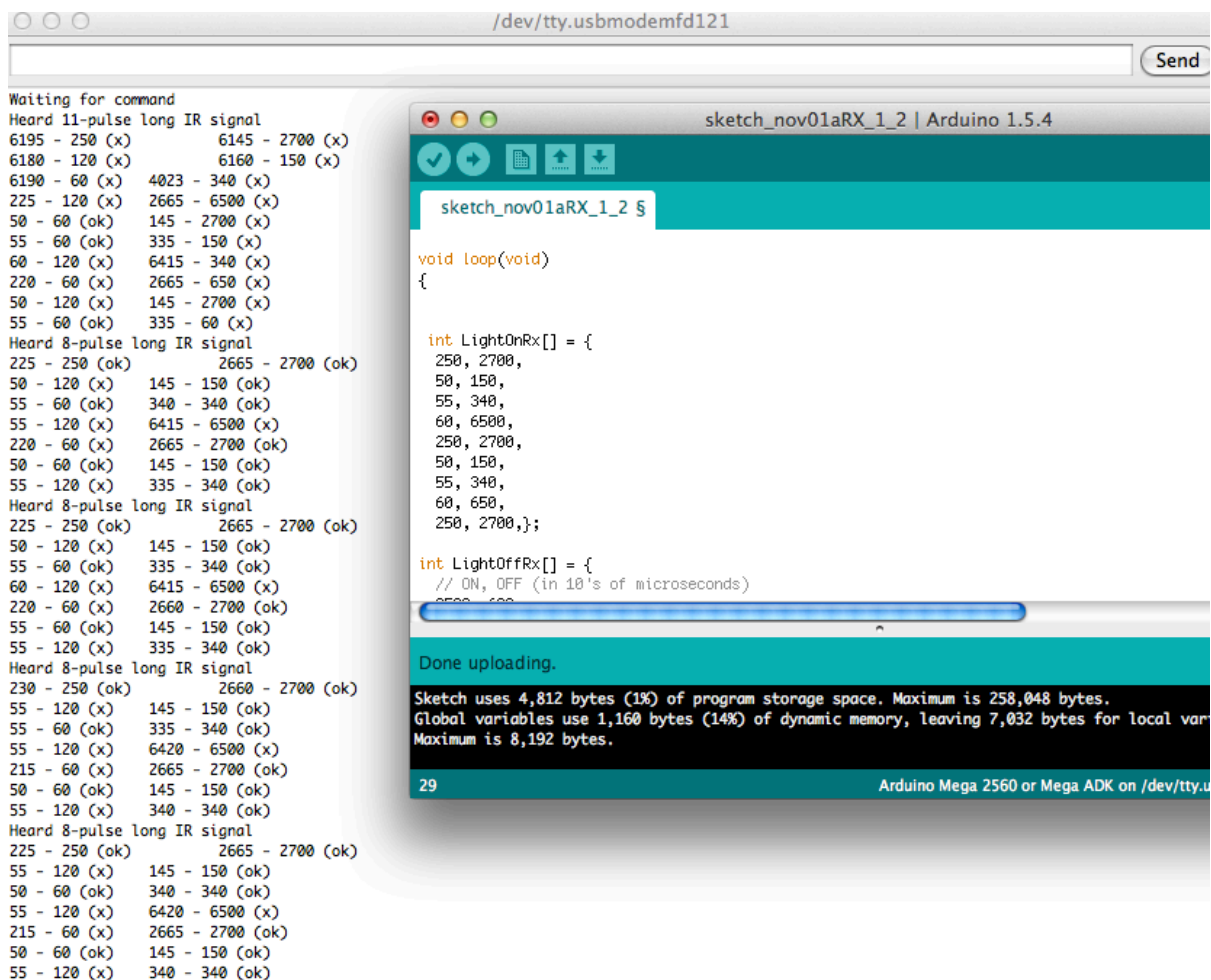


Fig 1.2 noise on the RF receiver

2. Next we tested the IR led that is used to control the music player and we were looking into the following aspects:

- Debug points to view capturing timing data
- Counters verify that no transmitted bits were missed
- Debug points in arduino to affirm transfer accuracy arduino Mega to ATtiny
- Debugging to make Arduino Mega and ATtiny compatible by considering their operating speed and timings.
- Debug points to affirm transfer accuracy between arduino.
- Test the response of arduino is two commands are executed at the same time
- Debugging the Latencies in the software to match up to the practical expectations of the user
- Distance from which the music player can be accurately controlled (not performed coz of lack of hardware)
- We tested the timing an the simulator and the image Fig1.3 is below:



The screenshot shows the Arduino IDE interface. On the left, the serial monitor window displays the following data:

```
Waiting for command
Heard 11-pulse long IR signal
6195 - 250 (x)      6145 - 2700 (x)
6180 - 120 (x)      6160 - 150 (x)
6190 - 60 (x)      4023 - 340 (x)
225 - 120 (x)      2665 - 6500 (x)
50 - 60 (ok)       145 - 2700 (x)
55 - 60 (ok)       335 - 150 (x)
60 - 120 (x)       6415 - 340 (x)
220 - 60 (x)       2665 - 650 (x)
50 - 120 (x)       145 - 2700 (x)
55 - 60 (ok)       335 - 60 (x)
Heard 8-pulse long IR signal
225 - 250 (ok)      2665 - 2700 (ok)
50 - 120 (x)       145 - 150 (ok)
55 - 60 (ok)       340 - 340 (ok)
55 - 120 (x)       6415 - 6500 (x)
220 - 60 (x)       2665 - 2700 (ok)
50 - 60 (ok)       145 - 150 (ok)
55 - 120 (x)       335 - 340 (ok)
Heard 8-pulse long IR signal
225 - 250 (ok)      2665 - 2700 (ok)
50 - 120 (x)       145 - 150 (ok)
55 - 60 (ok)       335 - 340 (ok)
60 - 120 (x)       6415 - 6500 (x)
220 - 60 (x)       2665 - 2700 (ok)
55 - 60 (ok)       145 - 150 (ok)
55 - 120 (x)       335 - 340 (ok)
Heard 8-pulse long IR signal
230 - 250 (ok)      2660 - 2700 (ok)
55 - 120 (x)       145 - 150 (ok)
55 - 60 (ok)       335 - 340 (ok)
55 - 120 (x)       6420 - 6500 (x)
215 - 60 (x)       2665 - 2700 (ok)
50 - 60 (ok)       145 - 150 (ok)
55 - 120 (x)       340 - 340 (ok)
Heard 8-pulse long IR signal
225 - 250 (ok)      2665 - 2700 (ok)
55 - 120 (x)       145 - 150 (ok)
50 - 60 (ok)       340 - 340 (ok)
55 - 120 (x)       6420 - 6500 (x)
215 - 60 (x)       2665 - 2700 (ok)
50 - 60 (ok)       145 - 150 (ok)
55 - 120 (x)       340 - 340 (ok)
```

On the right, the code editor shows the following C++ code:

```
sketch_nov01aRX_1_2 $
void loop(void)
{
    int LightOnRx[] = {
        250, 2700,
        50, 150,
        55, 340,
        60, 6500,
        250, 2700,
        50, 150,
        55, 340,
        60, 650,
        250, 2700,};

    int LightOffRx[] = {
        // ON, OFF (in 10's of microseconds)
        250, 650,};
}
```

Below the code editor, a status bar indicates: "Done uploading. Sketch uses 4,812 bytes (1%) of program storage space. Maximum is 258,048 bytes. Global variables use 1,160 bytes (14%) of dynamic memory, leaving 7,032 bytes for local variables. Maximum is 8,192 bytes. 29 Arduino Mega 2560 or Mega ADK on /dev/tty.u"

Fig1.3 checked for correctness of IR signal (values on arduino serial port monitor)

3. The Light dependent Resistor (LDR) was tested:

- The effect of power variation was checked and the results showed that the Analog to Digital converter in the arduino gives different values over different power. To make the reading more independent, a separate source of power was dedicated to arduino.
- The arduino reading over different intensity of light was monitored and recorded which was later compared to the Oscilloscope readings. The results were very accurate and the speed at which the results were updated was impressive (10milli second). The figure fig1.4 shows the arduino reading of different intensity of light on the LDR. Fig1.5 shows the corresponding Oscilloscope readings.
- The test result showed that arduino readings on its analog pin of more than 300 means complete darkness and at maximum brightness the readings were around 20. This enabled us to map these readings as shown in Fig 1.6

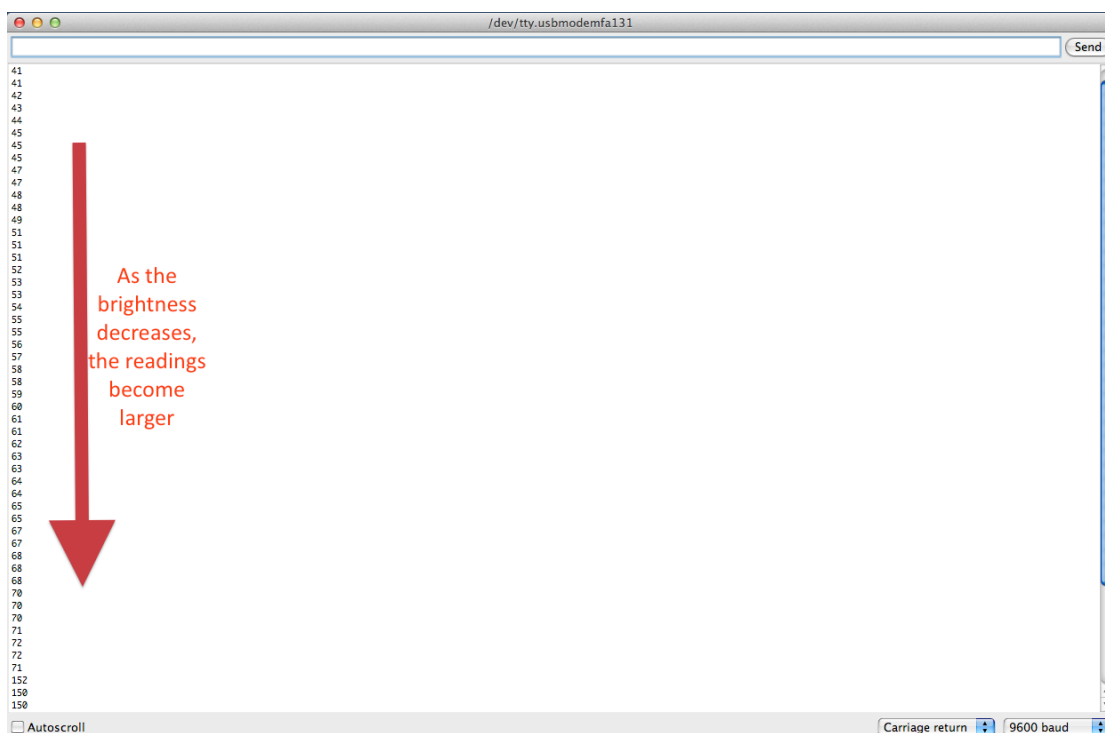
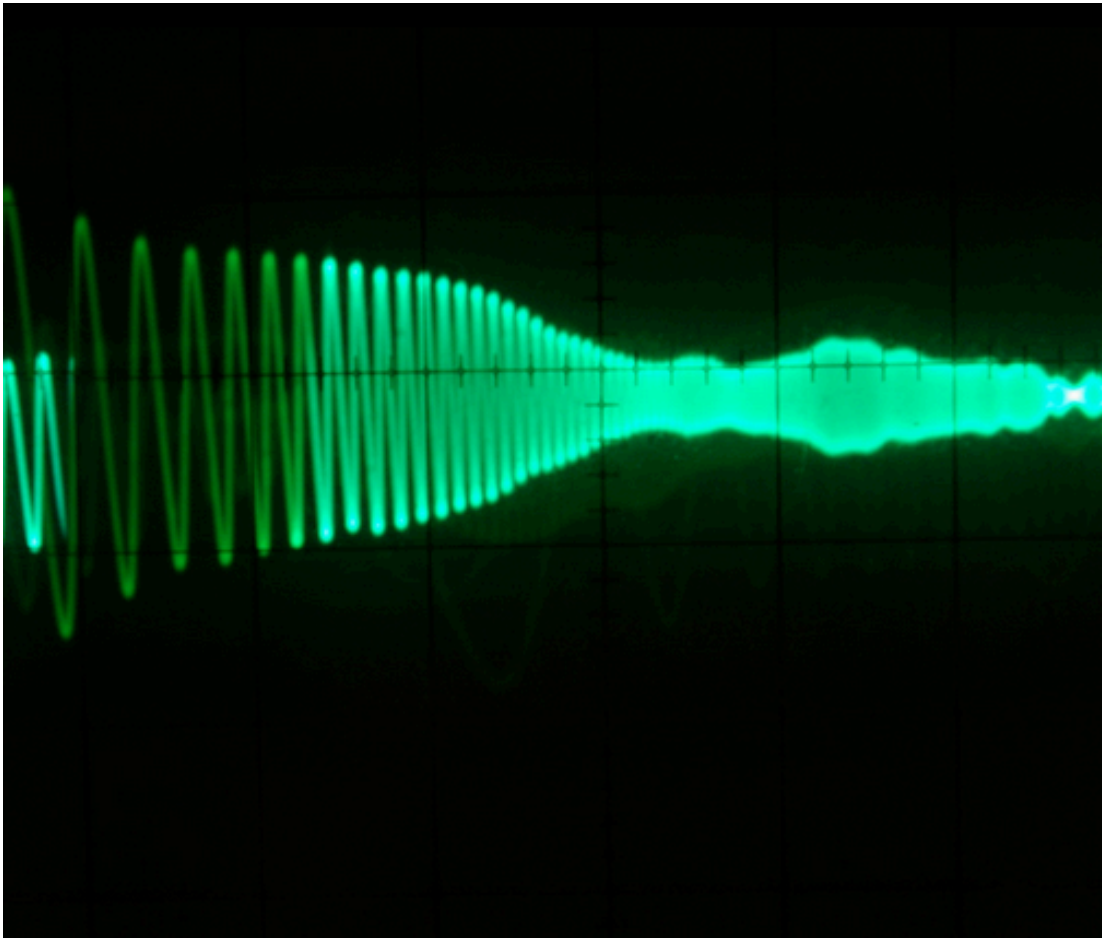


Fig 1.4 raw readings (values on arduino serial port monitor)



ffg 1.5 oscilloscope reading for a dimming light

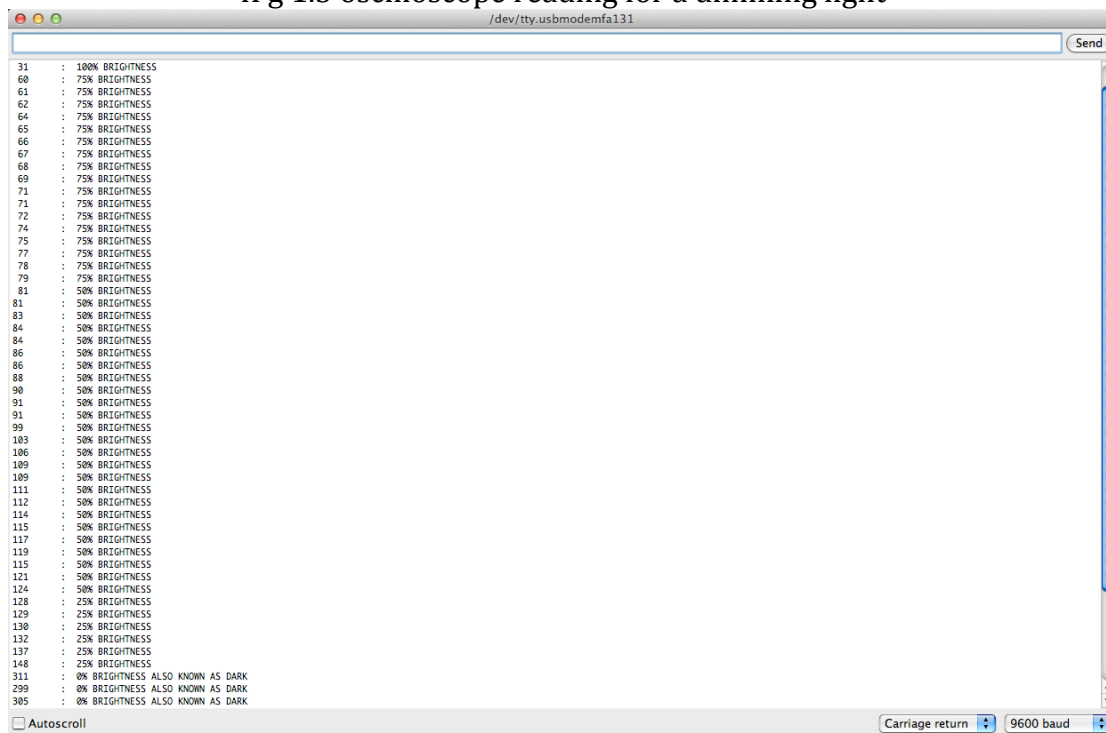


Fig 1.6 mapped brightness to readings (values on arduino serial port monitor)

4. The Microphone sensor was tested:

- The microphone was extensively tested over a various conditions and using different music. In fact this sensor was developed while testing because it was required to give an accurate for vast range of inputs.
- The whole process of testing was a learning process and the results on the oscilloscope were taken as reference and the controller was programmed. The test results indicated that that after a certain volume, the amplitude of the wave becomes constant, but the frequency of encountering waves carrying high amplitude increases with the volume. Taking advantage of this knowledge, we have developed a program to use the root mean square of the average over a certain time period deduced from the tests. This gave us 80% accuracy in our test results.
- Fig 1.7 and Fig 1.8 show us the difference in accuracy of the sensor when the volume is gradually increased. Fig 1.7 uses a system taking just the RMS value of the readings where as Fig 1.8 gives the output depending on our own developed signal-processing algorithms.
- Fig 1.9 shows how we have mapped different waves into volume.

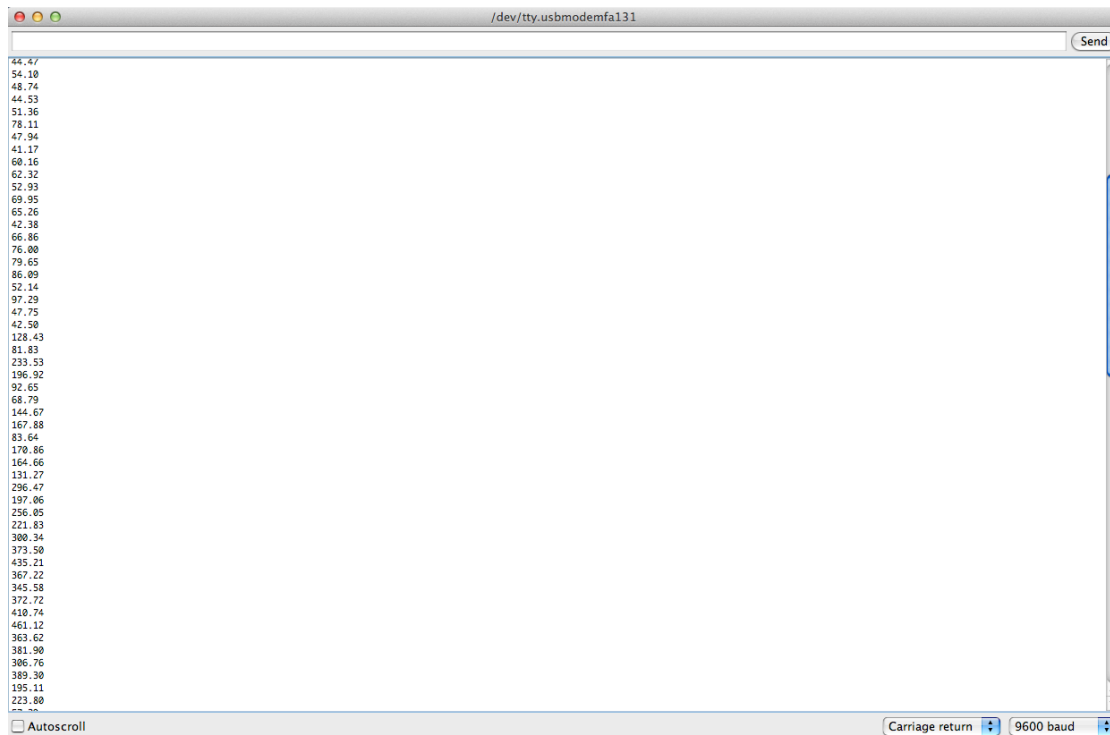


Fig 1.7 readings with just RMS (high error)(values on arduino serial port monitor)

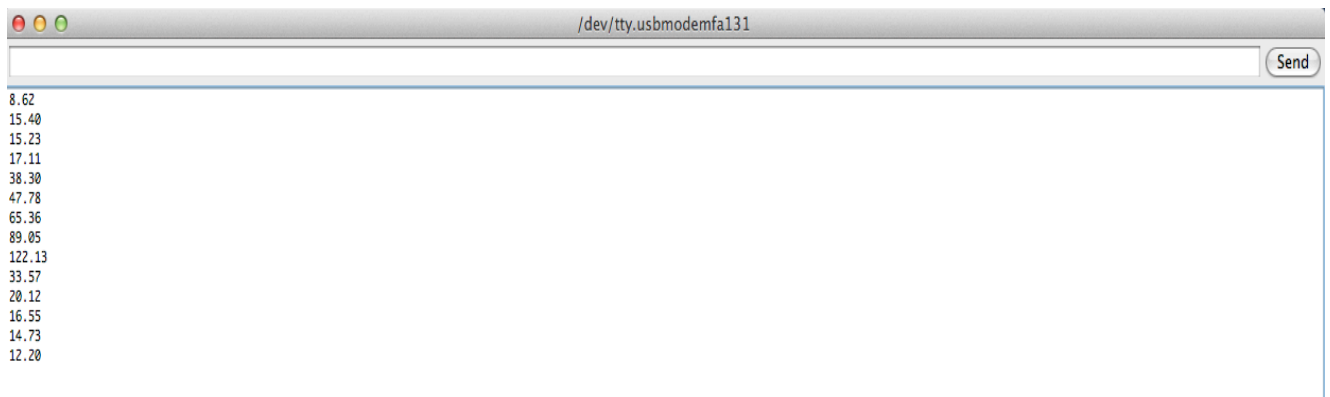
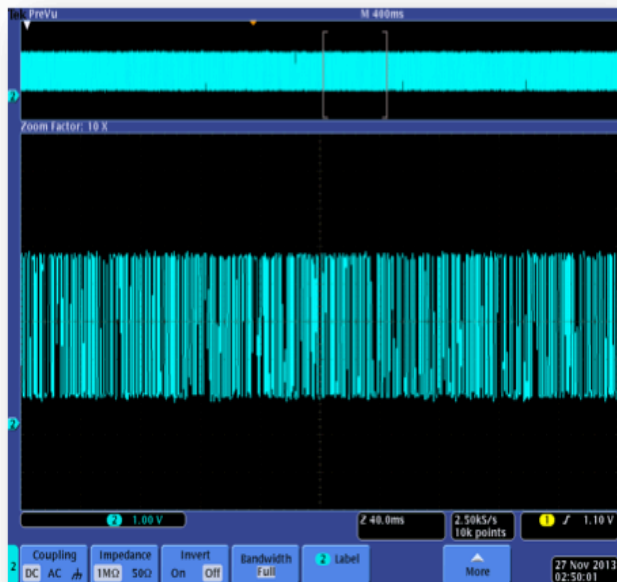


Fig 1.8 reading after software processing (low error) (values on arduino serial port monitor)

100% Volume



75% Volume



25% Volume



50% Volume



Fig 1.9 How the waves are mapped to different volume levels

2) Testing of the integrity of the software was mainly performed by analyzing the response of the program to different commands.

- For testing and debugging, after every unit in the program the values of the variables are checked. This is done by dry-running the program and checking it and debugging it using the arduino serial port monitoring.
- This helps us to isolate the bugs and record the response of each unit while working alone and also while integrated with the whole system.
- All the debugging and Testing of the program for the controller were done using this method.
- The main advantage of using the serial monitoring port is that we can detect the course of the software at run-time.
- Fig 2.1 and Fig2.2 shows the Serial-port-Monitoring of the program while it is being run. In Fig 2.1 command AA#(to turn on the light at 100% brightness), AB#(to turn off the light) & AE#(to turn on the light at 75% brightness) has been shown. In Fig 2.2 command BA#(to set the volume at 100%) and BB#(to set the volume at zero %), the two boundary cases have been shown.
- Fig 2.3 shows Serial.print command on arduino ide and the corresponding serial port monitoring for testing to give a better explanation of how the testing is done.
- FIG 2.4 shows how the parts are integrated and tested. It shows the integration testing of central hub and light-controller node
- There are more test images in the folder named “Test contd.”.

returned:25%I received:
AC#Turn light ON at 25%
comand is:
2
ACB
Got: A A [return_fbAfb1A
100% brightness/ volume
25alarm off
Got: 90 90 The Light is glowing at:
returned:25%I received:
AF#Feedback
comand is:
5
AFB
return_fbAFB
return_fbAFB
return_fbAFB
return_fbAFB
Got: A B [return_fbBfb1B
0% brightness/ volume
23alarm off
Got: 90 90 The Light is glowing at:
returned:23%I received:
AE#Turn light ON at 75%
comand is:
4
AEB
return_fbAEB
return_fbAEB
return_fbAEB
return_fbAEB
return_fbAEB
return_fbAEB
return_fbAEB
Got: A E [return_fbEfb1E
69alarm off
Got: 90 90 The Light is glowing at:
returned:69%

This is the encoded command received from the server

This is after decoding the command

This is the feedback received from the device

Feedback is 23% brightness

Feedback is 69% brightness

☒ Autoscroll

Fig 2.1 commands to change the brightness (values on arduino serial port monitor)

```
off
0 90 The music is playing at:
ed:2%I received:
dback
is:
```

**Feedback is 2%
volume**

```
B 0return_fb8fb1B
ghtness/ volume
off
0 90 The Light is glowing at:
ed:0%I received:
dback
is:
```

```
B 0return_fb8fb1B
ghtness/ volume
off
ght is glowing at:
ed:0%I received:
n OFF the music player
is:
```

```
_fb0fb10
off
sic is playing at:
ed:2%I received:
rease volume to 100%
is:
```

```
B 0return_fb8fb1B
ghtness/ volume
off
5 66 The music is playing at:
ed:0%
scroll
```

**Feedback is 0%
volume**

Fig 2.2 commands to change the volume(values on arduino serial port monitor)

