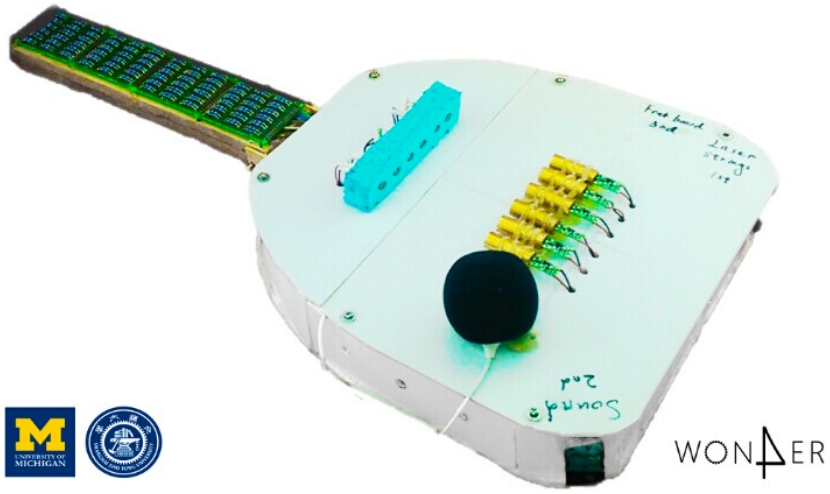
JOINT INSTITUTE
交大密西根学院

WONDER

Aug 4, 2014

Portable Laser Guitar



WonderFour

LIU Xieyang
PENG Junda
YU Tong
YU Zhiwei

I. Executive Summary

The guitar is popular nowadays, but it is too big and heavy to carry. To tackle this problem, our team came up with a completely new idea of making a “Portable Laser Guitar”. We replaced the strings with laser beams and the body with a small amplifier. We also designed a retractable fret board to reduce the space it takes. The sensors installed on the guitar will automatically detect your finger styles on both hands and send the information to the Arduino controlling board. The Arduino controlling board will then drive the amplifier to play sounds accordingly. All these designs are made to make the guitar as portable as possible, yet provide users with the same fantastic playing experience. To complete the project, we first came up with some preliminary designs and consulted with the professors. We then purchased the materials and built the first prototype. We examined the first prototype and built the second one. Finally we attended the symposium and delivered a presentation on the project. To make sure the guitar is fully operational, we first gathered the signals from the sensors on the fret board and the body respectively. We did every possible finger styles on the fret board, and made sure that each different finger position corresponds to a unique standard guitar pitch. We then debugged the program to make sure that only when the laser beams are blocked would the guitar make a sound, so that it worked just like an acoustic guitar. We also tried different kinds of laser generators and found the most suitable one for our project. The laser generators generate green laser beams, and are both powerful and stable. The final product we built is fully operational and has all the functions that we have wished for. When the laser beams are blocked by our fingers, the guitar will make a sound, just like an acoustic guitar will make a sound when the strings are strummed. We can easily retract the fret board into the body when we are not playing the guitar. The electronic audio system stores all the standard tone sources recorded from an acoustic guitar. It makes sure that the sound it makes is always pitched at the standard level. With our Portable Laser Guitar, musicians will no longer waste time on changing broken strings or tuning, and will have more time for exercising and creating great works at almost anywhere. Guitar performances will become much more fantastic and attractive than they have ever been. The green laser strings represent modern

technology and will attract many children and teenagers. We no longer have to consume a large amount of trees to build guitars, which will benefit the environment fundamentally and change the entire guitar industry. We believe that after some improvements and modifications, our Portable Laser Guitar can be brought to massive production and will win an appreciable market share.

II. Acknowledgements

We would like to express our deepest gratitude to all the people, organizations, and agencies that once provided help and support to our project.

First, we would like to give a special thanks to Dr. Roberto Dugnani and Mr. Nick Welch-Bolen, who are the lecturers of Vg101 Introduction to Engineering at UM-SJTU Joint Institute. Without their professional guidance and suggestions, great patience, and abiding support, we could never have done such a wonderful job in this project.

We would like to thank HUANG Jiannan, CAI Yusheng, and ZHAO Yuyue, who are the Teaching Assistants for Vg101 Introduction to Engineering. Because of their practical advice and great kindness, we were able to make our project better and better.

We would also like to give a special thanks to ZOU Yusheng, who is a senior from University of Michigan. He joined the 2014 JI Summer Program in Shanghai and established great friendship with LIU Xieyang, the leader of the team. ZOU kindly provided some very novel ideas and concepts in the design of our Portable Laser Guitar. Without his support, we maybe could never build such a beautiful product.

Finally, we would like to thank all the members on our team. We spent lots of sleepless nights together along the way. We all spared no efforts to make the project a better one, even better than our imagination in the first place. We are often stunned at what we have achieved, greatly moved by our own capability and perseverance, and thus deeply convinced that we will finally make it. We see ourselves as heroes in the completion of this project.

WonderFour

Aug 1, 2014

CONTENTS

I. EXECUTIVE SUMMARY 2

II. ACKNOWLEDGEMENTS 4

III. PROBLEM 6

IV. NEEDS 8

V. SOLUTION … [PORTABLE LASER GUITAR] 9

VI. OBJECTIVES 11

VII. TASKS 13

VIII. DISCUSSION 34

IX. CONCLUSION 35

X. SCHEDULE 36

XI. BILL OF MATERIALS 37

XII. KEY PERSONNEL 40

XIII. REFERENCES 41

XIV. APPENDIX A: PROGRAMMING CODES 44

XV. APPENDIX B: DETAILED SOURCES OF PURCHASED MATERIALS 104

XVI. APPENDIX C: GRAPHS AND VIDEOS 108

XVII. APPENDIX D: COMPLETE SCHEDULE 111

XVIII. APPENDIX E: SYMPOSIUM SLIDES 114

III. Problem

Many famous guitar masters carry their guitars for world tours every time. From time to time, they complain to interviewers about the heavy weight of the guitars and the inconvenience of carrying them around. Although manufacturers have been changing the appearance of their guitars all the time, problem like this still exists today and have bothered many musicians.

For acoustic guitars, their strings break very often, especially during tuning, as shown in Figure 1. If a string unfortunately breaks during performances, the performer will be embarrassed and the audience will be disappointed.

Figure 1: Common Problem when playing the guitar: Strings break under too much tension.[1]

Figure 2: Common Problem when playing the guitar: Strings have to be tuned each time before performance.[2]

Since the tension in the strings will inevitably change after some time, musicians today always have to spend time tuning before each performance, as shown in Figure 2. Also for learners, tuning their guitar is generally not an easy job.

Figure 3: Common Problem when playing the guitar: Long time of performance will hurt fingers.[3]

For crazy guitar fans, long time performance will hurt their fingers, as shown in Figure 3. They will have to stop even if they really want to continue.

In summary, although the guitar is one of the most popular musical instruments in the world, problems like the following still exist today:

- **Much inconvenience in carrying due to big size and heavy weight**
- **A high probability for the strings to break during performance**
- **A lack of appropriate techniques to get rid of the tuning before each performance**
- **A lack of effective methods to prevent the fingers from being hurt after long time of performance**

IV. Needs

Customers want a lighter and smaller guitar so that they can carry it with them. They also need a guitar that is free of string breaking and tuning. This will save them a lot of time. For some crazy guitar fans, they need a guitar that allows then to play for as long as they like without hurting fingers.

To solve these problems in an acoustic guitar, several new techniques are required:

- **A more small and compact design for the body of the guitar**
- **A special material to make strings so that they never breaks**
- **A reliable technology to automatically tune the guitar once and for all**
- **A special design for the strings so that they never hurt fingers**

V. Solution … [Portable Laser Guitar]

A completely new idea of making a Portable Laser Guitar is proposed which offers:

1. **A light and small plastic body which easily fits in bags or cases**
2. **A technology that use laser beams as strings which will not break**
3. **An electronic audio system which ensures that every sound is pitched at a standard level**
4. **A technology that use laser beams as strings which never causes pain on fingers**

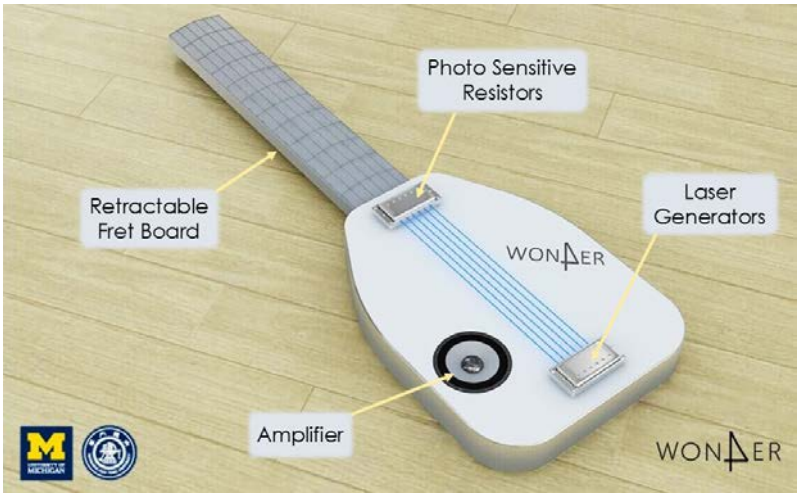


Figure 4: Portable Laser Guitar's concept diagram (View from the exterior)

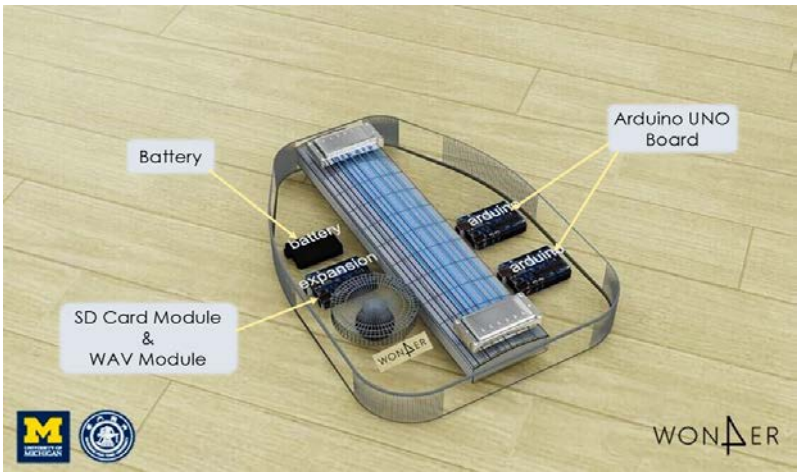


Figure 5: Portable Laser Guitar's concept diagram (View from the interior)

The Portable Laser Guitar is shown schematically in Figure 4 and Figure 5. The metal strings on an acoustic guitar are replaced by the laser beams. The fret board on the guitar can

be retracted into the body when it is not being used. The Portable Laser Guitar has a built-in electronic audio system which makes sound according to user's preference.

The Portable Laser Guitar works in the following way: The green laser generators shoot laser beams onto the photo sensitive resistors. When the user blocks one of the laser strings, the value of the resistance of that particular resistor will change immediately, and an Arduino UNO will monitor this change from one of its digital pins, then the system knows that this string has been played. For the fret board, we designed a sophisticated circuit containing many resistors. When the user shorts out two parallel pins on a particular position (see Tasks) with a conductor bonded on his/her finger, the partial voltage in the circuit changes, and the same Arduino UNO will monitor this change from one of its six analog pins, then the system knows that this fret has been played. The system will then automatically analyze the information from the laser strings and the fret board, and fetch the corresponding audio file from the SD card and play it.

VI. Objectives

- **Objective 1: Search for information on laser musical instruments**

  Any information or precedents on using laser beams to reconstruct musical instruments will be valuable to our own project. We can carefully analyze the strengths and weaknesses of these former projects and learn from their experiences and mistakes. We will be able to avoid the same mistakes they had made and improve our own project.

- **Objective 2: Design our own Portable Laser Guitar**

  Based on the information we gathered in the first objective, we will generate some conceptual designs of our own Portable Laser Guitar. We can then consult with the professors and teaching assistants on our preliminary designs and receive constructive suggestions from them.

- **Objective 3: Modify the preliminary design and search for appropriate materials**

  With the feedbacks from the professors and teaching assistants, along with our own analysis, we will modify our preliminary design. We will also search for available materials to help us make the final design feasible. This objective will provide us with much practical information which will help us make our detailed design including precise structure and circuit diagram.

- **Objective 4: Make the detailed design and purchase the materials**

  We will make our detailed design including information on the precise structure, the circuit diagram, and the electronic audio system. We will then purchase all the materials required to build this Portable Laser Guitar. We will purchase the same type with different technical parameters for some vital components to choose the best suitable one.

- **Objective 5: Record the standard pitches from an acoustic guitar and process the audio files**

  To make our guitar sound like a real guitar, we will record the standard pitches from an acoustic guitar. We will then process the recorded audio files into specific formats suitable for playing by our electronic audio system.

- **Objective 6: Write the controlling programs and build the first prototype**

  We will first write the controlling program for our Portable Laser Guitar. With all the materials needed, we will then build the first prototype according to the detailed design. This will allow us to test all the designed functions of our Portable Laser Guitar.

- **Objective 7: Test the first prototype and debug the controlling program**

  We will make every possible finger styles on the fret board to make sure the guitar will make a unique sound for each different finger position. We will test the component with laser beams to make sure only when the laser beams are blocked will the guitar make a sound. We will debug the controlling program based on the results of the tests.

- **Objective 8: Modify the detailed design and assemble the final prototype**

  Enlightened by the information gathered from the first prototype, we will modify our detailed design and our controlling algorithm. We will then assemble the final prototype based on these modifications and improvements.

- **Objective 9: Report**

  We will attend the symposium and deliver a presentation on our final prototype. We will write a detailed final report based on our own experiences. We will also attend the design expo to demonstrate our Portable Laser Guitar.

VII. Tasks

The task flow diagram for the Portable Laser Guitar is shown below in Figure 6:

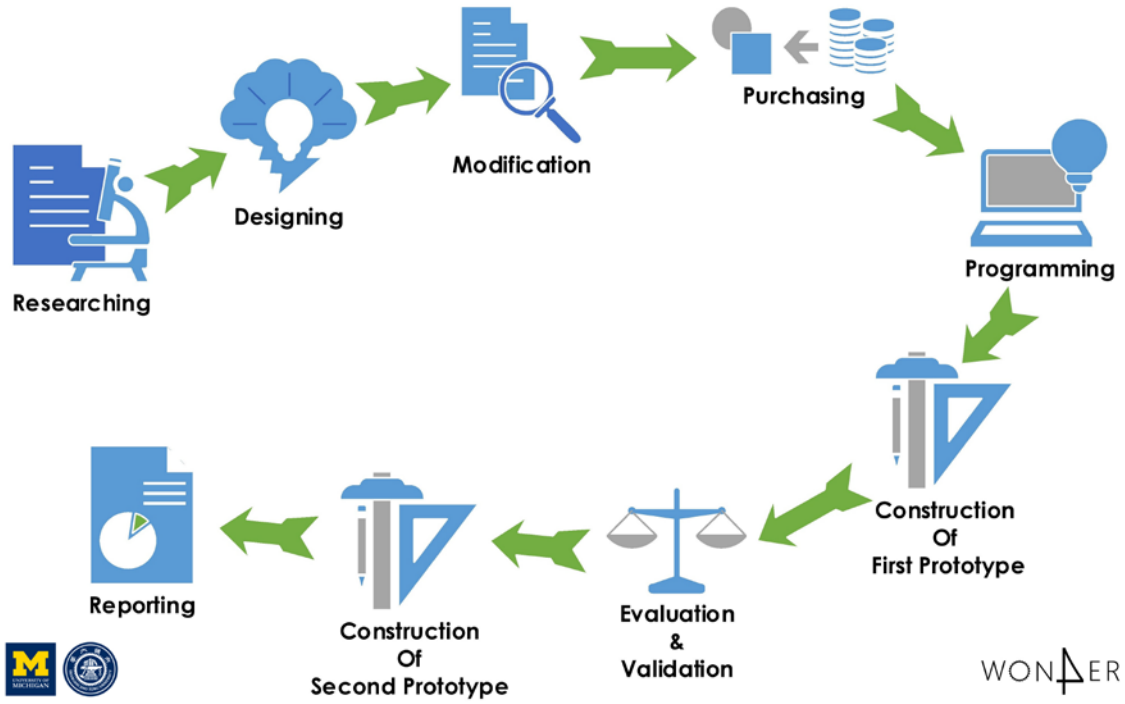


Figure 6: Task Flow Diagram of the Portable Laser Guitar Project

- **Task 1: Search for information on laser musical instruments**

We first searched for information on laser musical instruments on the Internet. We found that some of the instruments like harps become extremely cool and attractive when they are reconstructed with laser beams (see Figure 7).

Figure 7: Laser harp[4]

Figure 8: Basic structure of a laser harp[5]

In Figure 8, the basic structure of a laser harp is shown. It uses laser generators which are fixed on the top to shoot laser beams onto the base of the harp. The tiny ultrasonic range sensors installed next to the laser generators will automatically detect the distance between the fingers and the top of the harp. The controlling program will automatically convert the distance signal into sound signals. The speaker will amplify the sound signal and therefore play music.

We carefully analyzed the strengths and weaknesses of the former projects, and learned from their experiences and mistakes.

After the completion of Task 1:

- **We analyzed the strengths and weaknesses of similar projects in the past**
- **We gained knowledge of the basic working principles of replacing traditional material strings with laser beams**

- **Task 2: Design our own Portable Laser Guitar**

Based on the knowledge we learned in the Task 1, we generated some conceptual designs of our own Portable Laser Guitar.

Figure 9 shows one of our preliminary designs. We used Auto CAD (a popular computer aided design software) to help us generate a vivid design graph. PENG Junda and LIU Xieyang spent a weekend learning how to use the software from scratch.

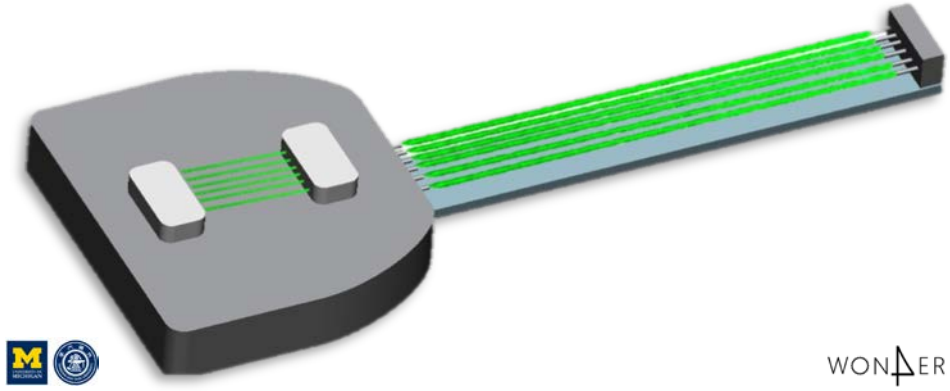


Figure 9: Preliminary design of our project

Since we only needed to sketch a preliminary design to pitch the professors for the time being, we focused on determining the appearance and the general working principle of the Portable Laser Guitar. We came up with many beautiful designs, and finally decided to use the one shown in Figure 9 in the pitch. Note that the fret board could be retracted into the body in our design. This could save a lot of space for the users.

Figure 10: A general type of laser range finder[6]

For the general working principle, we originated two ideas. The first was to use laser range finders to determine the positions of the fingers on the fret board. This method was not too difficult to implement, but we did not know about the accuracy of the finders. Besides, the laser range finders are generally too big to be fixed on a guitar (see Figure 10). The second was to use cameras to dynamically capture images of fingers on the fret board, and then use Open CV[7] technology to process image analyses. This may be accurate enough, but we were not sure we can master Open CV in such a short time. Finally, we decided to bring up both the working principles during the pitch and listen to the professors' advice. The slides for the pitch is shown below:

Figure 11: Slides used to pitch the professors

After the completion of Task 2:

- **We had some preliminary designs for the Portable Laser Guitar**
- **We gained the professors' approval to carry out the Portable Laser Guitar project**
- **We received some practical advice from the professors regarding both the working principles we had put forward**

- **Task 3: Modify the preliminary design and search for appropriate materials**

With the feedbacks from the professors and teaching assistants, along with our own analysis, we modified our preliminary design. We decided to continue using the appearance in Task 2. Before determining the working principle of the guitar, we had to search information of the materials involved and learn their technical parameter.

We first found the smallest laser range sensors available on Taobao (see Figure 12).

Figure 12: Smallest laser range sensor available[8]

This module is 45 mm in length and 19 mm in width. It shoot out laser beams and receive the laser that bounces back. It will calculate the time elapsed during the journey of the laser and thus convert the time information into distance information. However, based on our careful calculation, the madule was still too big for our guitar. It was impossible for us to fix 6 modules like this in the same line without increasing the width of the fret board. Besides, according to user feedback, this module was not as stable as we think, which means it did not always receive the signal it sent. So we decided to give up this method. For the camera method, we also found that currently Open CV cannot work very well with any type of Arduinos. So we decided to find a third way.

Based on the information we gathered during the hunt for available materials, we decided to design a touchable fret board consists of resistors and fiberglass panels. In the next task, we would focus on the design of the fret board and the circuit of the system.

We also made a shopping list for all the materials that are needed (see Figure 13).

Screws & nuts set	Arduino expansion board
Wood board	WAV moldule for Arduino board
ABS plastic board	SD card module for Arduino board
DC adjustable regulator for Arduino (reducing)	SD card
DC adjustable regulator for Arduino (booster)	Audio connector
Switchers	Adapter
Mini amplifier	Hot glue
Fiberglass panel	Hexagon plastic socket
Resistor	Electrical tape
Photo resistor	Battery box
Laser generator	Battery
Touch switcher	Dupont line
Arduino	502 glue

Figure 13: Shopping list

After the completion of Task 3:

- **We worked out the final solution to pinpointing finger positions on the fret board**
- **We had a shopping list for the materials required**

- **Task 4: Make the detailed design and purchase the materials**

The design for the positions of different components is shown in Figure 14 and Figure 15. All of the design graphs we posted are of very high quality. If you are reading a digital version of our report, you can arbitrarily zoom in to find more details.

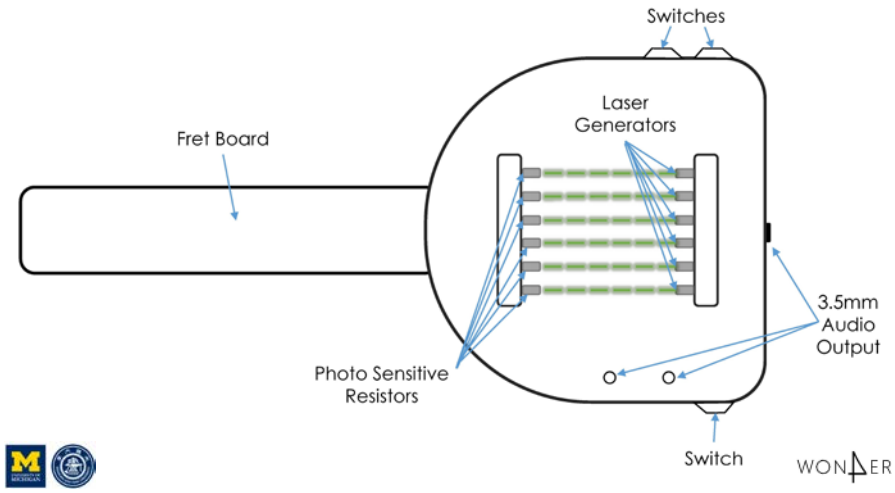


Figure 14: Detailed positions of different components (1)

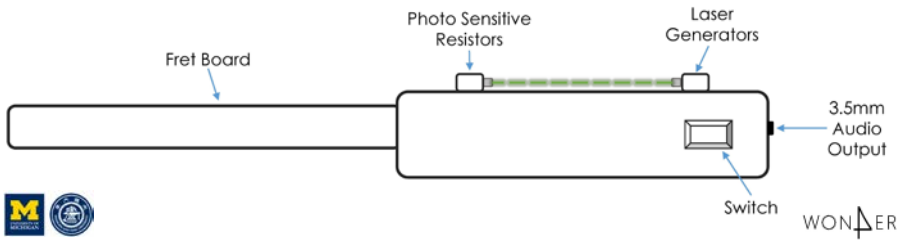


Figure 15: Detailed positions of different components (2)

We also determined the size of body along with that of different components. They are shown in Figure 16 and Figure 17. Again, all of the design graphs we posted are of very high quality. If you are reading a digital version of our report, you can arbitrarily zoom in to have a closer look on our design.

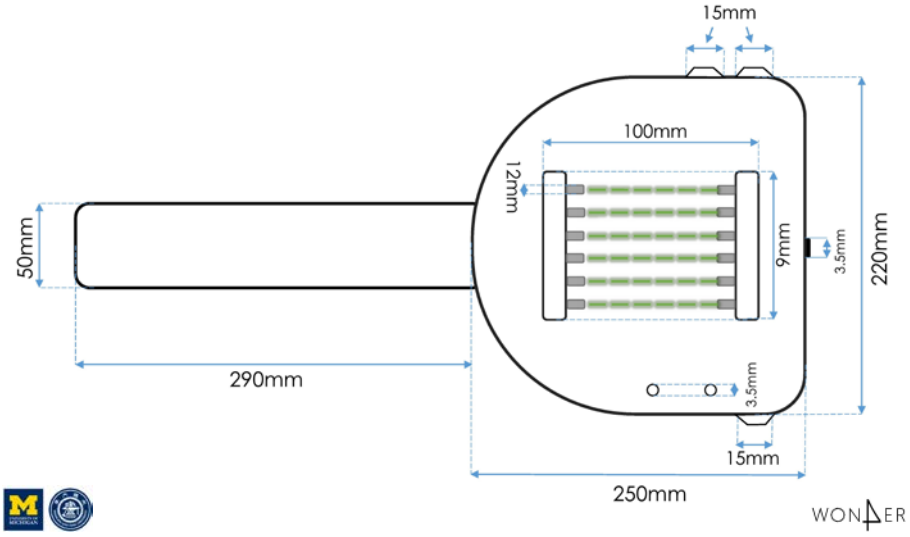


Figure 16: Size of the body along with different components (1)

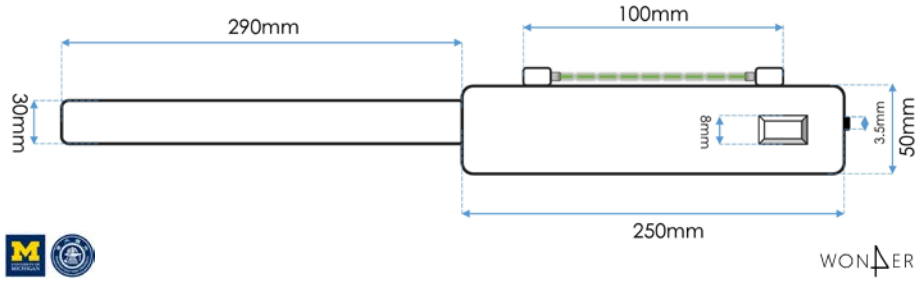


Figure 17: Size of the body along with different components (2)

We then purchased all the materials required to build this Portable Laser Guitar. Note that we purchased the same type with different technical parameters for some vital components to choose the best suitable one.

Figure 18 to Figure 27 show some of the vital components we purchased.

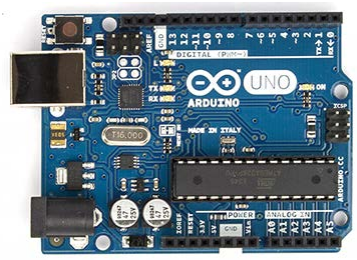


Figure 18: Arduino UNO[9]

Figure 19: DuPont lines[10]

Figure 20: SD card module[11]

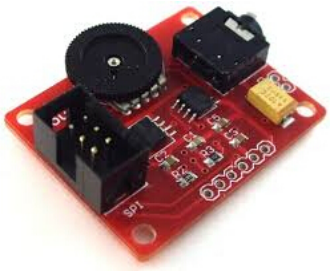


Figure 21: WAV module[12]

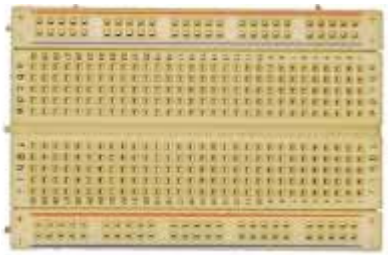


Figure 22: Bread board[13]

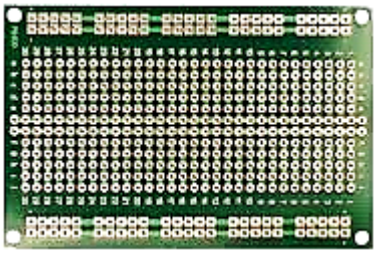


Figure 23: Fiberglass panel[14]

Figure 24: ABS plastic board[15]

Figure 25: DC module (reducing voltage)[16]

Figure 26: DC module (boosting voltage)[17]

Figure 27: Li-ion battery[18]

After the completion of Task 4:

- **We worked out the detailed design of our Portable Laser Guitar**
- **We had all the materials purchased**
- **We were ready to build the first prototype**

- **Task 5: Record the standard pitches from an acoustic guitar and process the audio files**

To make our guitar sound like a real guitar, we recorded all the standard pitches from an acoustic guitar (see Figure 28).

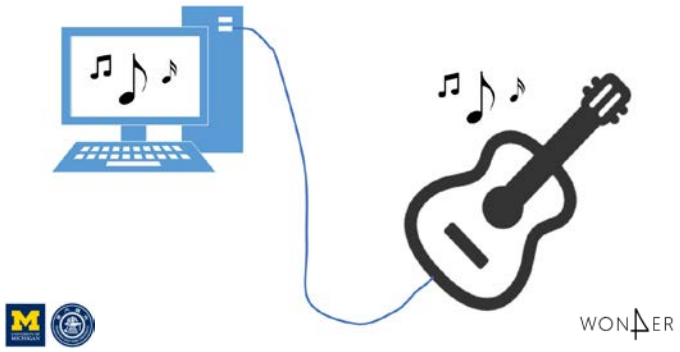


Figure 28: Recording standard pitches from an acoustic guitar

We then processed the recorded audio files into specific formats suitable for playing by our electronic audio system. We used Adobe Audition[19] to process the audio files. The sample format is "WAVE PCM", the sample type is "44100Hz, mono, 16-bit", the format setting is "Wave uncompressed" (see Figure 29 and Figure 30).

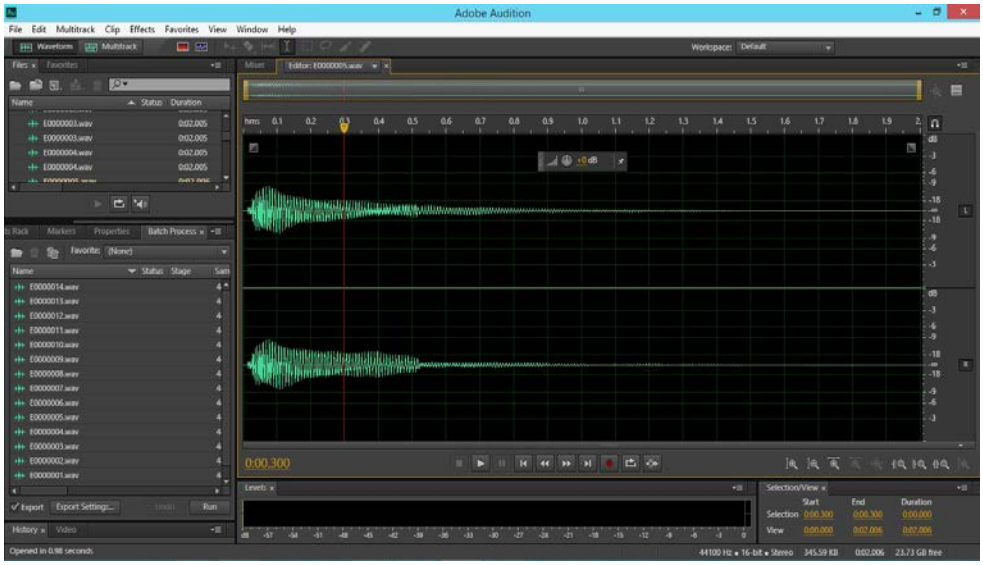


Figure 29: Using Audition to process the audio files

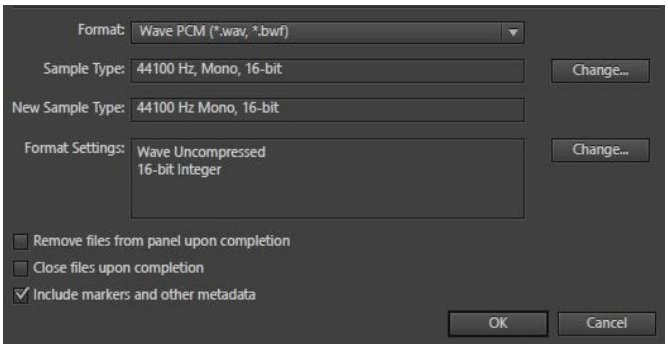


Figure 30: Export settings in Audition

After the completion of Task 5:

- **We had all the standard audio files recorded from an acoustic guitar**
- **We were ready to build the circuits and write the controlling program**

- **Task 6: Write the controlling programs and build the first prototype**

Before writing the controlling program, we had to design the circuit for the Portable Laser Guitar. The final circuit design is shown below in Figure 31. We used Fritzing[20] (an open-source software focused on the design of circuit schematics) to generate the whole circuit schematics.

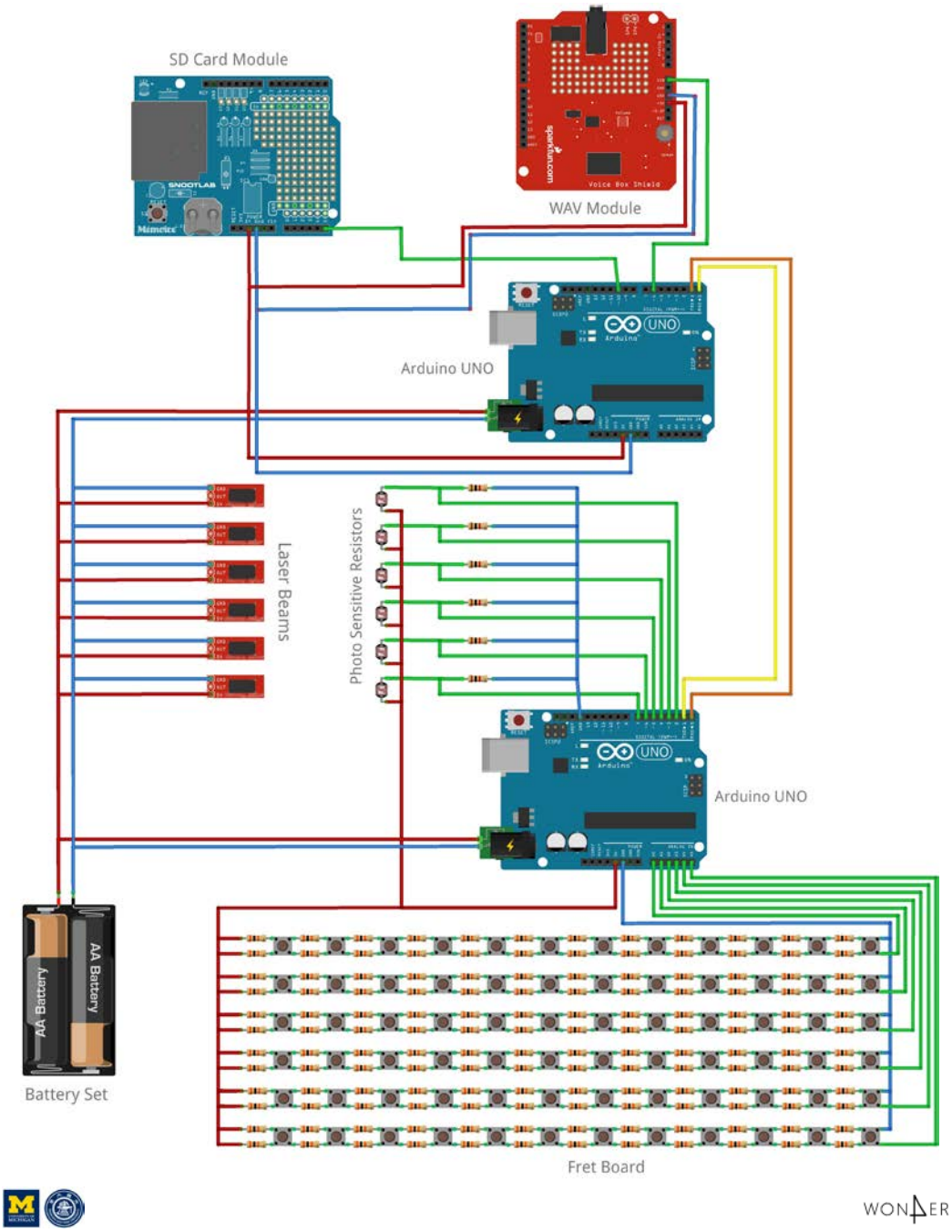


Figure 31: Circuit schematics for the Portable Laser Guitar Project

For the convenience of programming, we also built a testing fret board according to the circuit schematics shown above (see Figure 32).

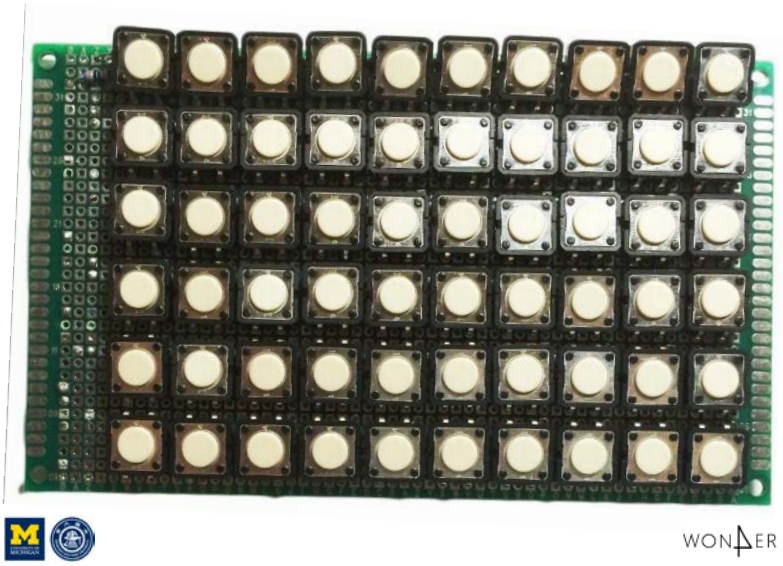


Figure 32: Testing fret board built for the sake of programming

With the testing fret board ready and the laser generators and the photo sensitive resistors ready, we then wrote the controlling program for the Portable Laser Guitar. Figure 33 shows basic working principle.

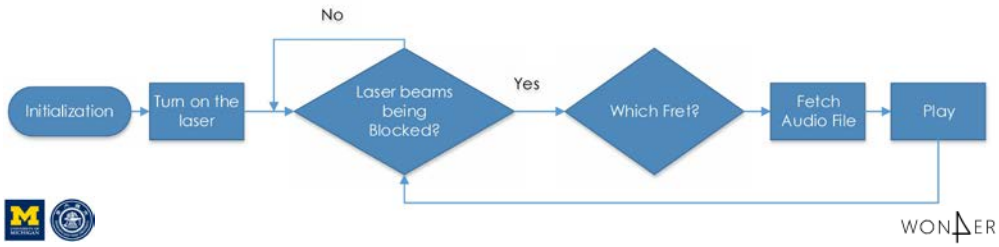


Figure 33: Basic working principle of the controlling program

The controlling program works basically in the following way: The green laser generators shoot laser beams onto the photo sensitive resistors. When the user blocks one of the laser strings, the value of the resistance of that particular resistor will change immediately, and an Arduino UNO will monitor this change from one of its digital pins, then the system knows that this string has been played. For the fret board, we designed a sophisticated circuit containing many resistors. When the user shorts out two parallel pins on a particular position with a conductor bonded on his/her finger, the partial voltage in the circuit changes, and the same Arduino UNO will monitor this change

from one of its six analog pins, then the system knows that this fret has been played. The system will then automatically analyze the information from the laser strings and the fret board, and another Arduino UNO will fetch the corresponding audio file from the SD card and play it.

Figure 34 shows the programs for each Arduino UNO respectively. The complete program is shown in Appendix A

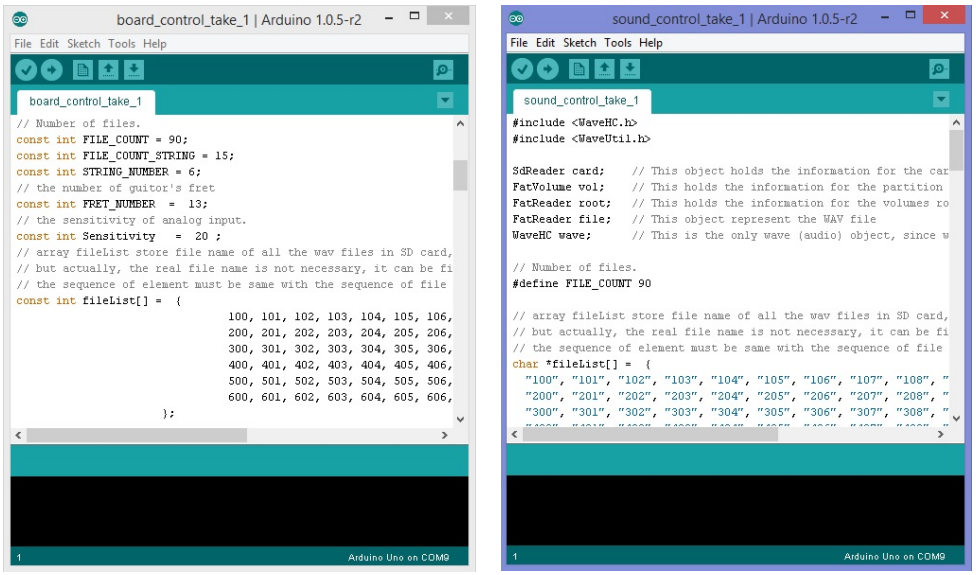


Figure 34: Programs for each Arduino UNO respectively

Combining the structure design finished in Task 6 and program design just finished in this task, we assembled our first prototype (see Figure 35).

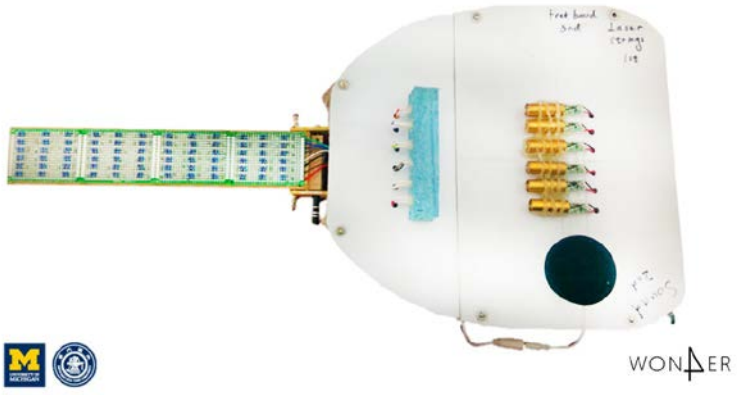


Figure 35: The first prototype of the Portable Laser Guitar

After the completion of Task 6:

- **We finished programming for the Portable Laser Guitar**
- **We built the first prototype**
- **We were ready to carry out tests on our first prototype**

- **Task 7: Test the first prototype and debug the controlling program**

First is the fret board. Our Portable Laser Guitar does not require users to learn new finger styles. They could just do those finger styles they like when playing an acoustic guitar on our guitar. It is really easy to use and fast to master our Portable Laser Guitar.

We made every possible finger styles on the fret board to make sure the guitar would make a unique sound for each different finger position. During the test, we found that some of the resistors refused to work because they got a cold solder joint. We decided to fix those problems when building the second prototype. Figure 36 shows the circuit on the fret board.

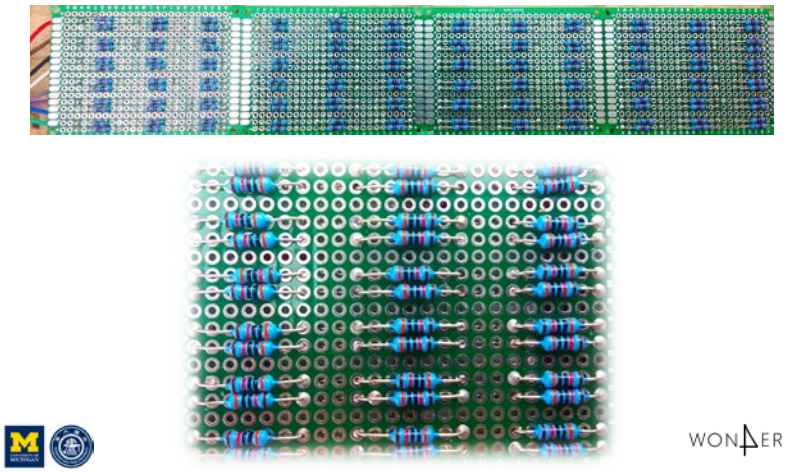


Figure 36: Fret board on the first prototype

Then we tested the component with laser beams and photo sensitive resistors to make sure only when the laser beams are blocked will the guitar make a sound. We did not find any problems regarding this part. Figure 37 shows the fixed photo sensitive resistors. Figure 38 shows the laser generators. Figure 39 shows how we were testing.

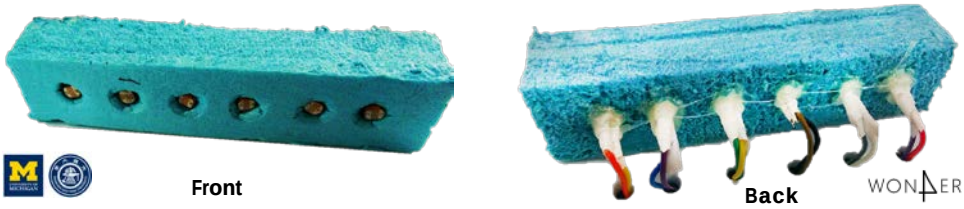


Figure 37: Fixed photo sensitive resistors

Figure 38: Laser Generators

Figure 39: Testing laser strings

We debugged the controlling program based on the results of the tests. Figure 40 shows the revised program. Again, the complete program is shown in Appendix A

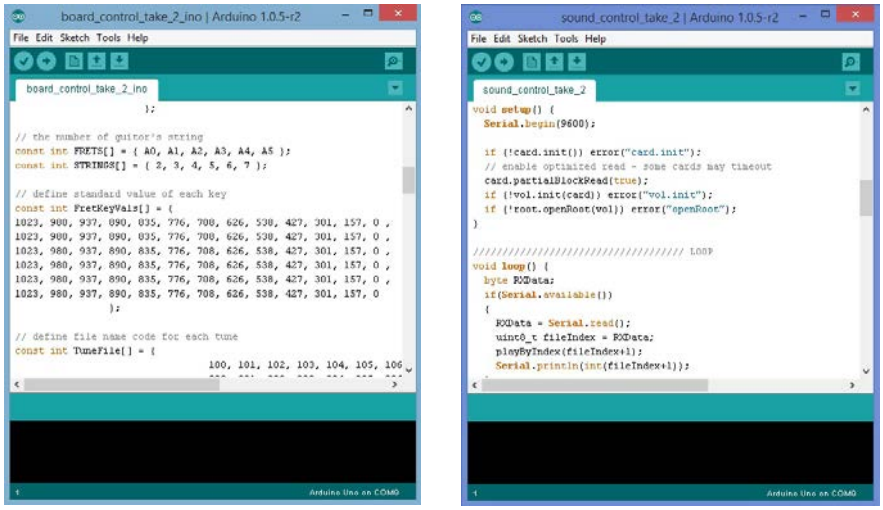


Figure 40: Revised program

After the completion of Task 7:

- **We revised programming for the Portable Laser Guitar**
- **We analyzed the strengths and weaknesses of the first prototype**
- **We were ready to build the second (final) prototype**

- **Task 8: Modify the detailed design and assemble the final prototype**

Enlightened by the information gathered from the first prototype, we modified our detailed design and our controlling algorithm.

For the appearance, we decided to reduce thickness by 30%.

For the controlling program, we decided to use "delay (297)", the exact length of an audio file processed in Task 5, instead of "delay (100)". This change would significantly improve the performance of our Portable Laser Guitar.

We then assembled the final prototype based on these modifications and improvements.

The final prototype is shown in Figure 41 and Figure 42. More graphs and demo videos are shown in Appendix C.

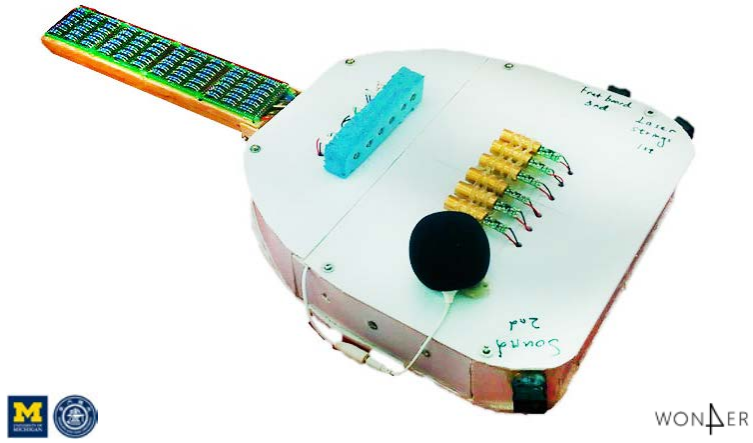


Figure 41: Final prototype

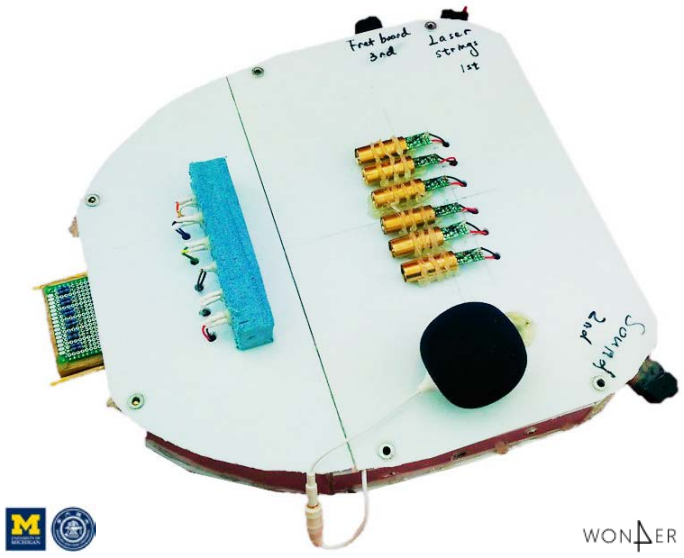


Figure 42: Final prototype (fret board retracted)

After the completion of Task 8:

- **We built the final prototype**

- **Task 9: Report**

We attended the symposium and delivered a presentation on our final prototype. For the symposium, we used Prezi to prepare a wonderful presentation. The sides are shown in Appendix E. Figure 43 is a photo we took with Professor Dugnani.

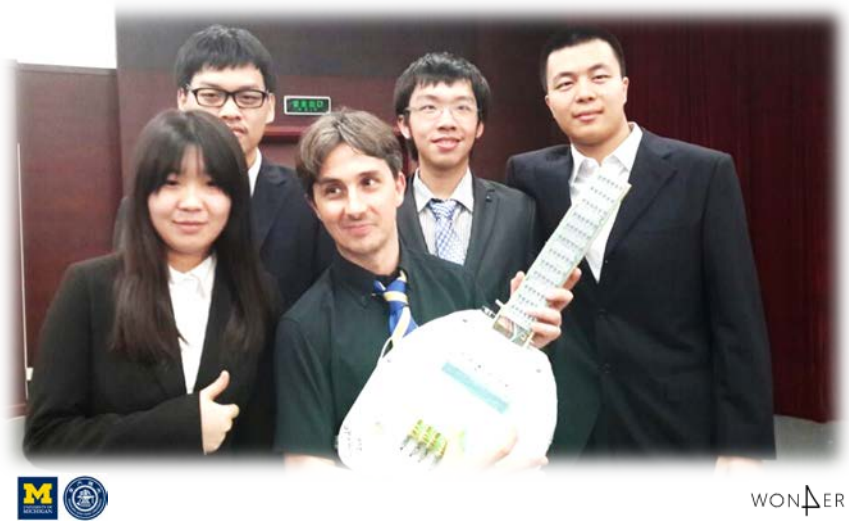


Figure 43: Photo with Professor Dugnani

We wrote a detailed final report based on our own experiences.

We also attended the design expo to demonstrate our portable laser guitar.

After the completion of Task 9:

- **We finished the project**
- **We achieved great success in this project**

VIII. Discussion

(I) What is good about our project?

The laser strings will never break or hurt fingers. The green laser beams also make playing the guitar even cooler, especially at night. The fret board can be retracted when not in use, which saves much space. The electronic audio system stores all the standard tone sources from an acoustic guitar. It will ensure that the sound it makes is always pitched at the standard level.

(II) What is less than ideal?

Although we managed to make the guitar smaller than an acoustic guitar, we didn't do very well in reducing its weight. Since it is a prototype, we used some heavy materials to ensure the strength of the structure. Real guitars are popular for different chords it plays, while our guitar can only play one pitch at a time, since we installed only one WAV Module. Now, we have to bind a short piece of conductor on our fingers to short out particular pins on the fret board. This is more or less tiring and will waste some time.

(III) What we might do to improve?

If we could have more time for our project, we would like to rebuild the prototype with lighter materials. We will also install 6 stand-alone WAV Module to grant our guitar the ability to play different chords. For the fret board, we will try using capacitive screens to detect the finger styles. This will significantly raise the accuracy of the performances.

(IV) Is our final solution feasible?

Our final solution is feasible, and the prototype functions very well. With our Portable Laser Guitar, musicians will no longer waste time on changing broken strings or tuning, and will have more time for exercising and creating great works at almost anywhere. Guitar performances will become much more fantastic and attractive than they have ever been. The green laser strings represent modern technology and will attract many children and teenagers. We no longer have to consume a large amount of trees to build guitars, which will benefit the environment fundamentally and change the entire guitar industry. We believe that after some improvements and modifications, our Portable Laser Guitar can be brought to massive production and will win an appreciable market share.

IX. Conclusion

The objective of our project is to make the guitar cooler and portable. To achieve this objective, our team came up with a completely new idea of making a “Portable Laser Guitar”. We replaced the strings with laser beams and the body with a small amplifier. The sensors installed on the device automatically detects your finger style on both hands. The Arduino controlling board then drives the amplifier to play sounds according to the different finger styles. All these designs are made to make the guitar as portable as possible, yet provide users with the same fantastic playing experience. The final product we built is functional and has all the functions that we have wished for. When the laser beams are blocked by our fingers, the guitar will make a sound. We can easily retract the fret board into the body when we are not playing. The electronic audio system stores all the standard tone sources from an acoustic guitar and makes sure that the sound it makes is always pitched at the standard level.

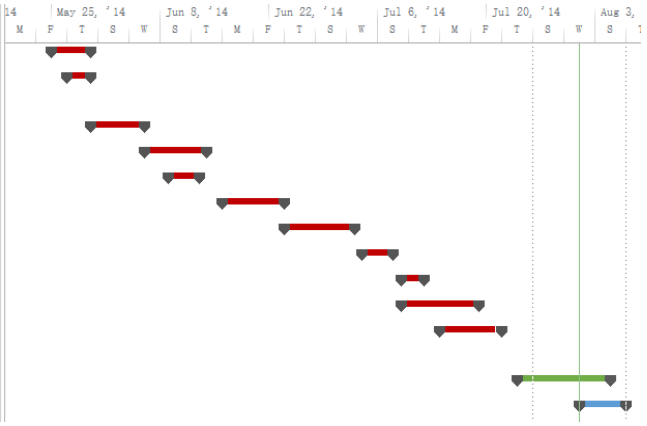
During this project, our group leaned much from both the technical and non-technical realm. The most important achievement we made is that we actually gave birth to this Portable Laser Guitar, considering that at the beginning, no one, even ourselves, were convinced that we could ever make it. We did not start with much confidence or technical knowledge. We learned how to use Arduinos, various sensors and shields from scratch, and then designed our own circuit and built our own electronic audio system. We gradually built our confidence through the whole process and learned the significance of team work and perseverance. We treated ourselves as perfectionists and sacrificed no matter what we need to complete the project.

We think that our design is novel and practical. The green laser strings will never break or hurt fingers. They also make playing the guitar even cooler, especially at night. When the fret board is retracted, the guitar becomes much smaller. The electronic audio system makes sound according to user preferences. We believe that after some improvements and modifications, our product can be brought to massive production and will win an appreciable market share.

X. Schedule

The Schedule of our Portable Laser Guitar Project is shown below:

Task Name	Duratic	Start	Finish	Responsible person
Initiate the project	5 days	May 25 '14	May 29 '14	All
Attend the first Professor Consultation	3 days	May 27 '14	May 29 '14	All
Deliver the first pitch	5 days	May 30 '14	Jun 5 '14	All
Make the conceptual design	6 days	Jun 6 '14	Jun 13 '14	All
Deliver the second pitch	4 days	Jun 9 '14	Jun 12 '14	All
Purchase the materials	6 days	Jun 16 '14	Jun 23 '14	All
Build the first prototype	7 days	Jun 24 '14	Jul 2 '14	All
Write the progress report	2 days	Jul 4 '14	Jul 7 '14	All
Test the first prototype	3 days	Jul 9 '14	Jul 11 '14	All
Build the second prototpye	8 days	Jul 9 '14	Jul 18 '14	All
Attend the symposium and deliver the oral presentation	6 days	Jul 14 '14	Jul 21 '14	All
Write the final report	8 days	Jul 24 '14	Aug 4 '14	All
Attend the final Expo	4 days	Aug 1 '14	Aug 6 '14	All



XI. Bill of Materials

Items	Part Description	Purchased From	Quantity	Unit Price(¥)	Total Price(¥)
Screws & nuts set	Nuts: GB/T 5782-2000，GB845-85，GB/T 5783-2000	Taobao**	1	23.8	23.8
Wood board	20cm *30cm *2cm	Taobao**	1	3.5	3.5
ABS plastic board	30cm *20cm *3mm	Taobao**	4	6.84	27.36
ABS plastic board	30cm *20cm *4mm	Taobao**	4	13.35	53.4
DC adjustable regulator for Arduino (reducing voltage)	LM2596S-ADJ 3A 10W 45mm *20mm *14mm	Taobao**	2	2.79	5.58
DC adjustable regulator for Arduino (boosting voltage)	XL6009 4A 43mm *30mm *12mm	Taobao**	2	5.1	10.2
Switchers	KCD1-104 5A	Taobao**	4	0.55	2.2
Mini amplifier	Socket: 3.5mm 3W 55mm *55mm *55mm	Taobao**	1	15.7	15.7
Fiberglass panel	60mm *80mm *1.6mm double-faced	Taobao**	1	5.45	5.45
Fiberglass panel	50mm *70mm *1.6mm double-faced	Taobao**	4	0.55	2.2
Resistor	0.25W 10kΩ	Taobao**	72	0.01	0.72
Resistor	0.25W 1kΩ	Taobao**	6	0.01	0.06
Photo resistor	5561 photo resistor 25mm 8kΩ	Taobao**	9	0.15	1.35

Thimble	Copper raduis:18mm	Taobao**	10	0.5	5
Laser Generator	Green laser beam 532nm 10mW 3V 12×52(33+19) mm	Taobao**	6	30	180
Laser generator	Linearly laser beam, 650nm, 30mA, 2.5-5V, red light	Taobao**	8	7.2	57.6
Touch switcher	6mm *6mm *5mm	Taobao**	72	0.75	54
Arduino	UNO board	Taobao**	2	50.5	101
Arduino expansion board	Interface: IPC-6/SPI	Taobao**	1	28	28
WAV module for Arduino board	Use with the Arduino IDC expansion board	Taobao**	2	58	116
SD card module for Arduino board	Interface:IPC-6	Taobao**	1	38	38
SD card	Kingston 16G	Taobao**	1	50	50
Audio connector	Interface: 3.5mm	Taobao**	1	19.1	19.1
Adapter	IDC-6/SPI 30cm	Taobao**	3	6	18
Hot glue	7mm	Taobao**	10	0.5	5
Photo sensitive transistor	3DU5C 10V 0.5-1mA 30mW	Taobao**	10	1.38	13.8
Hexagon copper socket	M3*6	Taobao**	50	6	300
Hexagon plastic socket	M3*20	Taobao**	8	0.19	1.52
Hexagon plastic socket	M3*30	Taobao**	8	0.26	2.08
Hexagon plastic socket	M3*40	Taobao**	8	0.35	2.8

Electrical tape	PVC NET1-00101	Taobao**	2	3	6
Battery box	68mm *33mm *18mm for double A battery	Taobao**	3	0.8	2.4
Battery(GP2200mAh)	ACE 2200mAh 3S 11.1V 20C 25C 40C Li battery	Taobao**	1	130	130
Battery	Double A battery	Taobao**	8	4	32
DuPont line	Male to female	Taobao**	2	4	8
DuPont line	With two female ends	Taobao**	2	3.9	7.8
DuPont line	With two male ends	Taobao**	2	3.97	7.94
502 glue	Brand: JINGU 502 super glue	Taobao**	2	0.99	1.98
Digital ranging sensor	45mm *19mm *28mm Accuracy: 10cm DC: 5V	Taobao**	1	114	114
Bread board	85mm *55mm	Taobao**	1	5.35	5.35
Laser generator	Punctiform green laser beam wave:532nm 10mW 3V 12×52(33+19) mm	Taobao**	6	30	180
Conductive rubber		Taobao**	1	3	3
Ultrasonic ranging sensor	DC:5V 0.2mA Accurancy:0.3cm	Taobao**	1	16.5	16.5
Resistor	0.25W 20kΩ	Taobao**	100	0.01	1
Resistor	0.25W 30kΩ	Taobao**	100	0.01	1
Resistor	0.25W 50kΩ	Taobao**	100	0.01	1
Resistor	0.25W 100kΩ	Taobao**	100	0.01	1
Spray paint	Brown metal painting	Taobao**	1	5.5	5.5
total					1667.89

**http://www.taobao.com/

XII. Key Personnel

XIII. References

1. The Guide to Buying Guitar Strings. Guitar Strings Order. 2014. Photograph. Web. Sun 27th July 2014. . Available from: <http://www.guitarstringsorder.net/buying-guitar-strings/ >

2. Summitt, Krista. The Guide to Buying Guitar Strings: Guitar Strings Order. 2014. Web. Photograph. Sun 27th July 2014. . Available from: <http://www.guitarstringsorder.net/buying-guitar-strings/>

3. Puppy, Spasmic. Zombies ate my games. 2013. Photograph. Web. Sun 27th July 2014. . Available from: <http://zombiesatemygames.com/2013/01/rocksmith/>

4. Glenn J. Hill. The Finest Custom Laser & Light Beam Harps in the world. 2014. Web. Photograph. Fri 1st Aug 2014. Available from: <http://www.mountainglenharps.com/laser_harps.htm>

5. Ross Dave. Make the laser harp of the intention by oneself. June 21, 2012. Web. Photograph. Fri 1st Aug 2014. Available from: <http://ele-tech.com/html/make-the-laser-harp-of-the-intention-by-oneself.html >

6. Bushnell Sport 600 Laser Rangefinder. 29th Mar 2013. Web. Photograph. Fri 1st Aug 2014. Available from: < http://www.scoperangefinder.com/rangefinders/bushnell-sport-600-laser-rangefinder>

7. OpenCV. 25th Apr 2014. Web. Code. Fri 1st Aug 2014. Available from: < http://opencv.org/>

8. 2014. Web. Photograph. Fri 1st Aug 2014. Available from: <http://item.taobao.com/item.htm?spm=a230r.1.0.0.mvutFd&id=14898953905>

9. Arduino UNO. 2014. Web. Photograph. Sat 2nd Aug 2014. Available from: < http://www.arduino.cc/en/Main/ArduinoBoardUno/>

10. DuPont lines. 2014. Web. Photograph. Sat 2nd Aug 2014. Available from: < http://www.aliexpress.com/item/20-PCS-LOT-Mini-Color-Cable-2-54mm-to-2-54mm-40pcs-in-1-Row-Dupont/562328228.html>

11. SD Card Module. 2014. Web. Photograph. Sat Aug 2nd 2014. Available from: < http://www.emartee.com/product/41392/>

12. WAV Module. 2014. Web. Photograph. Sat Aug 2nd 2014. Available from: <http://www.emartee.com/product/41973/>

13. Bread Board Protoboard Combo. 2014. Web. Photograph. Sat Aug 2nd 2014. Available from: <http://www.wrighthobbies.com/product.php?productid=131>

14. 8x12 cm Double Side Prototype PCB Panel Universal Circuit Board Fr 4 Glass Fiber. 2014. Web. Photograph. Sat Aug 2nd 2014. Available from: < http://www.aliexpress.com/item/8x12-cm-Double-Side-Prototype-PCB-Panel-Universal-Circuit-Board-Fr-4-Glass-Fiber/1477414890.html>

15. Model DIY material abs model transformation plate abs board PVC plastic board model. 2014. Web. Photograph. Sat Aug 2nd 2014. Available from :< http://www.aliexpress.com/item/Model-diy-material-abs-model-transformation-plate-abs-board-pvc-plastic-board-model/828452514.html>

16. LM2596 DC-DC Buck Converter Step-Down Power Module Output 1.25V-35V. 2014. Web. Photograph. Sat Aug 2nd 2014. Available from: <http://imall.iteadstudio.com/im130731002.html>

17. 100 PCS/LOT DC Step Up Voltage Regulator LM2577 DC Booster Converter DC 3-24V to 4-26.5V Max 3A Boost Voltage Regulator #090611. 2014. Web. Photograph. Sat Aug 2nd 2014. Available from:

<http://www.aliexpress.com/item/100-PCS-LOT-DC-Step-Up-Voltage-Regulator-LM2577-DC-Booster-Converter-DC-3-24V-to/1007974895.html>

18. Li-ion Polymer Battery for Model Airplane. 2014. Web. Photograph. Sat Aug 2nd 2014. Available from: <http://www.dx.com/p/wild-scorpion-11-1v-3500mah-3-cells-li-ion-polymer-battery-for-model-airplane-red-black-silver-184633#.U9yUY6O5U60>

19. Audition. 2014. Web. Sounds. Sat Aug 2nd 2014. Available from: <http://www.adobe.com/cn/products/audition.html>

20. Fritzing. 2014. Web. Software. Sat Aug 2nd 2014. Available from: < http://fritzing.org/home/>

21. Google Code. 14 Apr 2014. Code. Google Inc. Arduino Library for the Adafruit Wave Shield. Web. 26 July 2014. <https://code.google.com/p/wavehc/>

22. Google Code. 14 Apr 2014. Code. Google Inc. Arduino Library for the Adafruit Wave Shield. Web. 26 July 2014. <https://code.google.com/p/wavehc/downloads/detail?name=wavehc20110919.zip&can=2&q=>

23. Google Code. 14 Apr 2014. Code. Google Inc. Arduino Library for the Adafruit Wave Shield. Web. 26 July 2014. <https://code.google.com/p/wavehc/downloads/detail?name=wavehc20110919.zip&can=2&q=>

24. Google Code. 14 Apr 2014. Code. Google Inc. Arduino Library for the Adafruit Wave Shield. Web. 26 July 2014. <https://code.google.com/p/wavehc/downloads/detail?name=wavehc20110919.zip&can=2&q=>

XIV. Appendix A: Programming Codes

Controlling Programs

Board control take 1.ino

We wrote this program for the Arduino UNO that receives signals from the fret board and the photo sensitive resistors.

```
// Number of files.
const int FILE_COUNT = 90;
const int FILE_COUNT_STRING = 15;
const int STRING_NUMBER     = 6;
// the number of guitar's fret
const int FRET_NUMBER  =  13;
// the sensitivity of analog input.
const int Sensitivity   =  20 ;
// array fileList store file name of all the wav files in SD card,
// but actually, the real file name is not necessary, it can be file code there.
// the sequence of element must be same with the sequence of file in FAT system
const int fileList[] =  {
                            100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110,
111, 112, 113, 114,
                            200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210,
211, 212, 213, 214,
                            300, 301, 302, 303, 304, 305, 306, 307, 308, 309, 310,
311, 312, 313, 314,
                            400, 401, 402, 403, 404, 405, 406, 407, 408, 409, 410,
411, 412, 413, 414,
                            500, 501, 502, 503, 504, 505, 506, 507, 508, 509, 510,
511, 512, 513, 514,
                            600, 601, 602, 603, 604, 605, 606, 607, 608, 609, 610,
611, 612, 613, 614
                       };

// the number of guitar's string
const int FRETS[] = { A0, A1, A2, A3, A4, A5 };
const int STRINGS[] = { 2, 3, 4, 5, 6, 7 };

// define standard value of each key
const int FretKeyVals[] = {
1023, 980, 937, 890, 835, 776, 708, 626, 538, 427, 301, 157, 0 ,
1023, 980, 937, 890, 835, 776, 708, 626, 538, 427, 301, 157, 0 ,
1023, 980, 937, 890, 835, 776, 708, 626, 538, 427, 301, 157, 0 ,
1023, 980, 937, 890, 835, 776, 708, 626, 538, 427, 301, 157, 0 ,
1023, 980, 937, 890, 835, 776, 708, 626, 538, 427, 301, 157, 0 ,
1023, 980, 937, 890, 835, 776, 708, 626, 538, 427, 301, 157, 0
             };

// define file name code for each tune
```

```
const int TuneFile[] = {
                              100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110,
111, 112,
                              200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210,
211, 212,
                              300, 301, 302, 303, 304, 305, 306, 307, 308, 309, 310,
311, 312,
                              400, 401, 402, 403, 404, 405, 406, 407, 408, 409, 410,
411, 412,
                              500, 501, 502, 503, 504, 505, 506, 507, 508, 509, 510,
511, 512,
                              600, 601, 602, 603, 604, 605, 606, 607, 608, 609, 610,
611, 612
             };

int sensorValue = 0;        // value read from analog input;
int stringValue = LOW;     // value read from digital input;
                            //value "LOW" indicates the string has been played,
and "HIGH" indicates it has not been played

/*
 * Define macro to put error messages in flash memory
 */
////////////////////////////////// SETUP
void setup() {
  Serial.begin(9600);

  for (int i=0; i<STRING_NUMBER; i++)
    pinMode(STRINGS[i], INPUT);
}

  int ctrl = 10;
////////////////////////////////// LOOP
void loop() {
  //playByIndex(30);
  //playByIndex(40);
  //delay(1000);
  //playAllByIndex();
  //playByName("510");
  //delay(2000);

  int i=0, j=0;
  for(i=0; i<STRING_NUMBER; i++)
  {

    sensorValue = analogRead(FRETS[i]);
    stringValue = digitalRead(STRINGS[i]);
//stringValue = LOW;
    if( stringValue== HIGH )
    {

      // string "i" is played, then find is there any fret is pressed
      for(j=0; j<FRET_NUMBER; j++)
      {
        // compare key value read from analog input with standard key value, to
find which key is pressed
        if( abs(sensorValue - FretKeyVals[i*FRET_NUMBER+j]) <= Sensitivity )
            break;
```

```
      }

     // the value of variable "i" indicates which string is pressed
     // the value of variable "j" indicates which fret is pressed. If the value of
"j" is 0, no fret is pressed.
     // from TuneFile[i*FRET_NUMBER+j], you can get the tune file name which will
be played

      // print the results to the serial monitor:
     // Serial.print("String No = " );
     // Serial.print(i);
     // Serial.print("\t stringValue = ");
     // Serial.print(stringValue);
     // Serial.print("\t sersorValue = ");
     // Serial.print(sensorValue);
     // Serial.print("\t Fret No = ");
     // Serial.print(j);
     // Serial.print("\t FretKeyVals Value = ");
     // Serial.print(FretKeyVals[i*FRET_NUMBER+j]);
      if(j<FRET_NUMBER)
      {
      //  Serial.print("\t Tune File Name = ");
     //  Serial.println(TuneFile[i*FRET_NUMBER+j]);

        playByName(TuneFile[i*FRET_NUMBER+j]);
      }
     // else
     //   Serial.println("");
    }
  //  else
   // {
   //   Serial.print("String No = " );
   //   Serial.print(i);
    //  //  Serial.print("\t stringValue = ");
    //  //  Serial.println(stringValue);
    //}

 }

  // after the last reading:
 // Serial.println("");
   delay(298);
}

/*
 * Play file by file index in FAT file system
 * the index of first file is 0
 */
void playByIndex(uint8_t fileIndex) {
 // Serial.println(fileIndex);
  byte TXData = fileIndex;
  Serial.write(TXData);
}

void playAllByIndex(void) {
  for (uint8_t i = 0; i < FILE_COUNT; i++) {
    playByIndex(i);
  }
```

```
}

/*
 * Play file by name and print latency in ms
 */
void playByName(int fileName) {
  int i=0;
  i = (fileName / 100 - 1)*FILE_COUNT_STRING + fileName % 100;

  if(i<FILE_COUNT)
    playByIndex(i);
}
```

Sound control take 1.ino

We wrote this program for the Arduino UNO that controls the WAV Module. It is written based on the examples in the WaveHC library[21].

```
#include <WaveHC.h>
#include <WaveUtil.h>

SdReader card;    // This object holds the information for the card
FatVolume vol;    // This holds the information for the partition on the card
FatReader root;   // This holds the information for the volumes root directory
FatReader file;   // This object represent the WAV file
WaveHC wave;      // This is the only wave (audio) object, since we will only play
one at a time

// Number of files.
#define FILE_COUNT 90

// array fileList store file name of all the wav files in SD card,
// but actually, the real file name is not necessary, it can be file code there.
// the sequence of element must be same with the sequence of file in FAT system
char *fileList[] =  {
  "100", "101", "102", "103", "104", "105", "106", "107", "108", "109", "110",
"111", "112", "113", "114",
  "200", "201", "202", "203", "204", "205", "206", "207", "208", "209", "210",
"211", "212", "213", "214",
  "300", "301", "302", "303", "304", "305", "306", "307", "308", "309", "310",
"311", "312", "313", "314",
  "400", "401", "402", "403", "404", "405", "406", "407", "408", "409", "410",
"411", "412", "413", "414",
  "500", "501", "502", "503", "504", "505", "506", "507", "508", "509", "510",
"511", "512", "513", "514",
  "600", "601", "602", "603", "604", "605", "606", "607", "608", "609", "610",
"611", "612", "613", "614"
};

// index of DTMF files in the root directory
uint16_t fileIndex[FILE_COUNT];
```

```
/*
 * Define macro to put error messages in flash memory
 */
#define error(msg) error_P(PSTR(msg))

//////////////////////////////////// SETUP
void setup() {
  Serial.begin(9600);

  if (!card.init()) error("card.init");
  // enable optimized read - some cards may timeout
  card.partialBlockRead(true);
  if (!vol.init(card)) error("vol.init");
  if (!root.openRoot(vol)) error("openRoot");
}

//////////////////////////////////// LOOP
void loop() {
  byte RXData;
  if(Serial.available())
  {
    RXData = Serial.read();
    uint8_t fileIndex = RXData;
    playByIndex(fileIndex+1);
    Serial.println(int(fileIndex+1));
  }
}

//////////////////////////////////// HELPERS
/*
 * print error message and halt
 */
void error_P(const char *str) {
  PgmPrint("Error: ");
  SerialPrint_P(str);
  sdErrorCheck();
  while(1);
}
/*
 * print error message and halt if SD I/O error, great for debugging!
 */
void sdErrorCheck(void) {
  if (!card.errorCode()) return;
  PgmPrint("\r\nSD I/O error: ");
 // Serial.print(card.errorCode(), HEX);
  PgmPrint(", ");
//  Serial.println(card.errorData(), HEX);
  while(1);
}

/*
 * Play file by file index in FAT file system
 * the index of first file is 0
 */
void playByIndex(uint8_t fileIndex) {
 // Serial.print(fileIndex);
  //Serial.print("(");
```

```
  //Serial.print((char)fileIndex);
  //Serial.print(")");
  if(fileIndex<FILE_COUNT) {
   // Serial.print(":   ");
    if (!file.open(root, fileIndex+3)) {
      error("open by index");
    }

    // create and play Wave
    if (!wave.create(file)) error("wave.create");
    wave.play();

    uint8_t n = 0;
    while (wave.isplaying) {// playing occurs in interrupts, so we print dots in
realtime
      putstring(".");
      if (!(++n % 32))//Serial.println();
      delay(100);
    }
    // check for play errors
    sdErrorCheck();
  }
 // Serial.println();
}

void playAllByIndex(void) {
  for (uint8_t i = 0; i < FILE_COUNT; i++) {
    playByIndex(i);
  }
  PgmPrintln("Done");
}

/*
 * Play file by name and print latency in ms
 */
void playByName(char *fileName) {
  uint8_t i;
  for(i=0; i<FILE_COUNT; i++)
    if(strcmp(fileName, fileList[i])==0) break;

  if(i<FILE_COUNT)
    playByIndex(i);
}
```

WaveHC Library

Without the library file, we could never control the WAV Module to play music. We downloaded the original codes of the WaveHC library from

https://code.google.com/p/wavehc/downloads/detail?name=wavehc20110919.zip&can=2&q=

(You probably have to use a VPN to open this url in The People's Republic of China, as we did previously). We then found that the original codes didn't work with our module. So we examine the codes row by row, and made some significant modifications to make the codes usable for our WAV Module. Some of the modified codes in the WaveHC library are shown below:

WavePinDefs.h

In this .h file, we mainly made some modifications to the declarations of pins. We now use pin 6, 7, 8, 9 on the Arduino UNO instead of the original[22] 2, 3, 4, 5 (You can observe these modifications since we just rewrite the codes for declaring the pins, and left the comments in the original codes unchanged).

```
/**
 * \file
 * Pin definitions
 */
#include <ArduinoPins.h>
#ifndef WavePinDefs_h
#define WavePinDefs_h

//SPI pin definitions

/** SPI slave select pin. Warning: SS may be redefined as another pin
 but the hardware SS_PIN must be set to output mode before any calls to
 WaveHC functions. The SS_PIN can then be used as a general output pin */
#define SS   SS_PIN

/** SPI master output, slave input pin. */
#define MOSI MOSI_PIN

/** SPI master input, slave output pin. */
#define MISO MISO_PIN

/** SPI serial clock pin. */
#define SCK  SCK_PIN

//------------------------------------------------------------------------------
// DAC pin definitions

// LDAC may be connected to ground to save a pin
/** Set USE_MCP_DAC_LDAC to 0 if LDAC is grounded. */
#define USE_MCP_DAC_LDAC 1

// use arduino pins 2, 3, 4, 5 for DAC

// pin 2 is DAC chip select
```

```
/** Data direction register for DAC chip select. */
//#define MCP_DAC_CS_DDR  PIN2_DDRREG
/** Port register for DAC chip select. */
//#define MCP_DAC_CS_PORT PIN2_PORTREG
/** Port bit number for DAC chip select. */
//#define MCP_DAC_CS_BIT  PIN2_BITNUM

#define MCP_DAC_CS_DDR  PIN9_DDRREG
#define MCP_DAC_CS_PORT PIN9_PORTREG
#define MCP_DAC_CS_BIT  PIN9_BITNUM

// pin 3 is DAC serial clock
/** Data direction register for DAC clock. */
//#define MCP_DAC_SCK_DDR  PIN3_DDRREG
/** Port register for DAC clock. */
//#define MCP_DAC_SCK_PORT PIN3_PORTREG
/** Port bit number for DAC clock. */
//#define MCP_DAC_SCK_BIT  PIN3_BITNUM

#define MCP_DAC_SCK_DDR  PIN7_DDRREG
#define MCP_DAC_SCK_PORT PIN7_PORTREG
#define MCP_DAC_SCK_BIT  PIN7_BITNUM

// pin 4 is DAC serial data in

/** Data direction register for DAC serial in. */
//#define MCP_DAC_SDI_DDR  PIN4_DDRREG
/** Port register for DAC clock. */
//#define MCP_DAC_SDI_PORT PIN4_PORTREG
/** Port bit number for DAC clock. */
//#define MCP_DAC_SDI_BIT  PIN4_BITNUM

#define MCP_DAC_SDI_DDR  PIN6_DDRREG
#define MCP_DAC_SDI_PORT PIN6_PORTREG
#define MCP_DAC_SDI_BIT  PIN6_BITNUM

// pin 5 is LDAC if used
#if USE_MCP_DAC_LDAC
/** Data direction register for Latch DAC Input. */
//#define MCP_DAC_LDAC_DDR  PIN5_DDRREG
/** Port register for Latch DAC Input. */
//#define MCP_DAC_LDAC_PORT PIN5_PORTREG
/** Port bit number for Latch DAC Input. */
//#define MCP_DAC_LDAC_BIT  PIN5_BITNUM

#define MCP_DAC_LDAC_DDR  PIN8_DDRREG
#define MCP_DAC_LDAC_PORT PIN8_PORTREG
#define MCP_DAC_LDAC_BIT  PIN8_BITNUM


#endif // USE_MCP_DAC_LDAC

#endif // WavePinDefs_h
```

WaveHC.h

In this WaveHC.h file, we mainly made some modifications to the maximum sample rate limit of the module. We raised the maximum sample rate of the audio file that can be played from 44100 Hz[23] to 88200 Hz. We did so to make the sound of the guitar more fluent and sounds like a real one. Such modifications are significant to the success of our project.

```
/*
 This library is a highly modified version of Ladyada's Wave Shield library.
 I have made many changes that may have introduced bugs.
*/
#ifndef WaveHC_h
#define WaveHC_h
#include <FatReader.h>
/**
 * \file
 * WaveHC class
 */
/**
 * If nonzero, optimize the player for contiguous files.  It takes
 * longer to open a file but can play contiguous files at higher rates.
 * Disable if you need minimum latency for open.  Also see open by index.
 */
#define OPTIMIZE_CONTIGUOUS 1
 /**
  * Software volume control should be compatible with Ladyada's library.
  * Uses shift to decrease volume by 6 dB per step. See DAC ISR in WaveHC.cpp.
  * Must be set after call to WaveHC::create().
  * Decreases MAX_CLOCK_RATE to 22050.
  */
#define DVOLUME 0
/**
 * Set behavior for files that exceed MAX_CLOCK_RATE or MAX_BYTE_RATE.
 * If RATE_ERROR_LEVEL = 2, rate too high errors are fatal.
 * If RATE_ERROR_LEVEL = 1, rate too high errors are warnings.
 * If RATE_ERROR_LEVEL = 0, rate too high errors are ignored.
 */
#define RATE_ERROR_LEVEL 2
//------------------------------------------------------------------------------
// Set the size for wave data buffers.  Must be 256 or 512.
#if defined(__AVR_ATmega168P__) || defined(__AVR_ATmega168__)

/** Buffer length for for 168 Arduino. */
#define PLAYBUFFLEN 256UL
#else // __AVR_ATmega168P__

/** Buffer length for Arduinos other than 168. */
#define PLAYBUFFLEN 512UL
#endif //__AVR_ATmega168P__

// Define max allowed SD read rate in bytes/sec.
#if PLAYBUFFLEN == 512UL && OPTIMIZE_CONTIGUOUS
```

```
/** Maximum SD read rate for 512 byte buffer and contiguous file */
#define MAX_BYTE_RATE 88200
#else // MAX_BYTE_RATE
/** Maximum SD read rate for 256 byte buffer or fragmented file */
#define MAX_BYTE_RATE 44100
#endif // MAX_BYTE_RATE

// Define maximum clock rate for DAC.
#if !DVOLUME
/** maximum DAC clock rate 44100 */
#define MAX_CLOCK_RATE 88200
#else // DVOLUME
/** Decreased clock rate if volume control is used */
#define MAX_CLOCK_RATE 44100
#endif //DVOLUME

//------------------------------------------------------------------------------
/**
 * \class WaveHC
 * \brief Wave file player.
 *
 * Play wave files from FAT16 and FAT32 file systems
 * on SD and SDHC flash memory cards.
 *
 */
class WaveHC {
public:
  /** Wave file number of channels. Mono = 1, Stereo = 2 */
  uint8_t Channels;
  /** Wave file sample rate. Must be not greater than 44100/sec. */
  uint32_t dwSamplesPerSec;
  /** Wave file bits per sample.  Must be 8 or 16. */
  uint8_t BitsPerSample;
  /** Remaining bytes to be played in Wave file data chunk. */
  uint32_t remainingBytesInChunk;
  /** Has the value true if a wave file is playing else false. */
  volatile uint8_t isplaying;
  /** Number of times data was not .   Available from the SD in the DAC ISR */
  uint32_t errors;

#if DVOLUME
  /** Software volume control. Reduce volume by 6 dB per step. See DAC ISR. */
  uint8_t volume;
#endif // DVOLUME
  /** FatReader instance for current wave file. */
  FatReader* fd;

  WaveHC(void);
  uint8_t create(FatReader &f);
  /** Return the size of the WAV file */
  uint32_t getSize(void) {return fd->fileSize();}
  uint8_t isPaused(void);
  void pause(void);
  void play(void);
  int16_t readWaveData(uint8_t *buff, uint16_t len);
  void resume(void);
  void seek(uint32_t pos);
  void setSampleRate(uint32_t samplerate);
```

```
  void stop(void);
};

#endif //WaveHC_h
```

WaveHC.cpp

In this WaveHC.cpp[24] file, we mainly made some modifications according to the changes in the WaveHC.h file.

```
#include <string.h>
#include <avr/interrupt.h>
#include <mcpDac.h>
#include <WaveHC.h>
#include <WaveUtil.h>

// verify program assumptions
#if PLAYBUFFLEN != 256 && PLAYBUFFLEN != 512
#error PLAYBUFFLEN must be 256 or 512
#endif // PLAYBUFFLEN

WaveHC *playing = 0;

uint8_t buffer1[PLAYBUFFLEN];
uint8_t buffer2[PLAYBUFFLEN];
uint8_t *playend;      // end position for current buffer
uint8_t *playpos;      // position of next sample
uint8_t *sdbuff;       // SD fill buffer
uint8_t *sdend;        // end of data in sd buffer

// status of sd
#define SD_READY 1     // buffer is ready to be played
#define SD_FILLING 2   // buffer is being filled from DS
#define SD_END_FILE 3  // reached end of file
uint8_t sdstatus = 0;

//------------------------------------------------------------------------------
// timer interrupt for DAC
ISR(TIMER1_COMPA_vect) {
  if (!playing) return;

  if (playpos >= playend) {
    if (sdstatus == SD_READY) {

      // swap double buffers
      playpos = sdbuff;
      playend = sdend;
      sdbuff = sdbuff != buffer1 ? buffer1 : buffer2;

      sdstatus = SD_FILLING;
      // interrupt to call SD reader
```

```
            TIMSK1 |= _BV(OCIE1B);
    }
    else if (sdstatus == SD_END_FILE) {
      playing->stop();
      return;
    }
    else {
      // count overrun error if not at end of file
      if (playing->remainingBytesInChunk) {
        playing->errors++;
      }
      return;
    }
  }

  uint8_t dh, dl;
  if (playing->BitsPerSample == 16) {

    // 16-bit is signed
    dh = 0X80 ^ playpos[1];
    dl = playpos[0];
    playpos += 2;
  }
  else {

    // 8-bit is unsigned
    dh = playpos[0];
    dl = 0;
    playpos++;
  }

#if DVOLUME
  uint16_t tmp = (dh << 8) | dl;
  tmp >>= playing->volume;
  dh = tmp >> 8;
  dl = tmp;
#endif //DVOLUME

  // dac chip select low
  mcpDacCsLow();

  // send DAC config bits
  mcpDacSdiLow();
  mcpDacSckPulse();  // DAC A
  mcpDacSckPulse();  // unbuffered
  mcpDacSdiHigh();
  mcpDacSckPulse();  // 1X gain
  mcpDacSckPulse();  // no SHDN

  // send high 8 bits
  mcpDacSendBit(dh,  7);
  mcpDacSendBit(dh,  6);
  mcpDacSendBit(dh,  5);
  mcpDacSendBit(dh,  4);
  mcpDacSendBit(dh,  3);
  mcpDacSendBit(dh,  2);
  mcpDacSendBit(dh,  1);
  mcpDacSendBit(dh,  0);
```

```
  // send low 4 bits
  mcpDacSendBit(dl,  7);
  mcpDacSendBit(dl,  6);
  mcpDacSendBit(dl,  5);
  mcpDacSendBit(dl,  4);

  // chip select high - done
  mcpDacCsHigh();

}
//------------------------------------------------------------------------------
// this is the interrupt that fills the playbuffer

ISR(TIMER1_COMPB_vect) {

  // turn off calling interrupt
  TIMSK1 &= ~_BV(OCIE1B);

  if (sdstatus != SD_FILLING) return;

  // enable interrupts while reading the SD
  sei();

  int16_t read = playing->readWaveData(sdbuff, PLAYBUFFLEN);

  cli();
  if (read > 0) {
    sdend = sdbuff + read;
    sdstatus = SD_READY;
  }
  else {
    sdend = sdbuff;
    sdstatus = SD_END_FILE;
  }
}
//------------------------------------------------------------------------------
/** create an instance of WaveHC. */
WaveHC::WaveHC(void) {
  fd = 0;
}
//------------------------------------------------------------------------------
/**
 * Read a wave file's metadata and initialize member variables.
 *
 * \param[in] f A open FatReader instance for the wave file.
 *
 * \return The value one, true, is returned for success and
 * the value zero, false, is returned for failure.  Reasons
 * for failure include I/O error, an invalid wave file or a wave
 *  file with features that WaveHC does not support.
 */
uint8_t WaveHC::create(FatReader &f) {
  // 18 byte buffer
  // can use this since Arduino and RIFF are Little Endian
  union {
    struct {
      char     id[4];
```

```
      uint32_t size;
      char     data[4];
    } riff;  // riff chunk
    struct {
      uint16_t compress;
      uint16_t channels;
      uint32_t sampleRate;
      uint32_t bytesPerSecond;
      uint16_t blockAlign;
      uint16_t bitsPerSample;
      uint16_t extraBytes;
    } fmt; // fmt data
  } buf;

#if OPTIMIZE_CONTIGUOUS
  // set optimized read for contiguous files
  f.optimizeContiguous();
#endif // OPTIMIZE_CONTIGUOUS

  // must start with WAVE header
  if (f.read(&buf, 12) != 12
      || strncmp(buf.riff.id, "RIFF", 4)
      || strncmp(buf.riff.data, "WAVE", 4)) {
        return false;
  }

  // next chunk must be fmt
  if (f.read(&buf, 8) != 8
      || strncmp(buf.riff.id, "fmt ", 4)) {
        return false;
  }

  // fmt chunk size must be 16 or 18
  uint16_t size = buf.riff.size;
  if (size == 16 || size == 18) {
    if (f.read(&buf, size) != (int16_t)size) {
      return false;
    }
  }
  else {
    // compressed data - force error
    buf.fmt.compress = 0;
  }

  if (buf.fmt.compress != 1 || (size == 18 && buf.fmt.extraBytes != 0)) {
    putstring_nl("Compression not supported");
    return false;
  }

  Channels = buf.fmt.channels;
  if (Channels > 2) {
    putstring_nl("Not mono/stereo!");
    return false;
  }
  else if (Channels > 1) {
    putstring_nl(" Warning stereo file!");
  }
```

```
  BitsPerSample = buf.fmt.bitsPerSample;
  if (BitsPerSample > 16) {
    putstring_nl("More than 16 bits per sample!");
    return false;
  }

  dwSamplesPerSec = buf.fmt.sampleRate;
  uint32_t clockRate = dwSamplesPerSec*Channels;
  uint32_t byteRate = clockRate*BitsPerSample/8;

#if RATE_ERROR_LEVEL > 0
  if (clockRate > MAX_CLOCK_RATE
      || byteRate > MAX_BYTE_RATE) {
    putstring_nl("Sample rate too high!");
    if (RATE_ERROR_LEVEL > 1) {
      return false;
    }
  }
  else if (byteRate > 88200 && !f.isContiguous()) {
    putstring_nl("High rate fragmented file!");
    if (RATE_ERROR_LEVEL > 1) {
      return false;
    }
  }
#endif // RATE_ERROR_LEVEL > 0

  fd = &f;

  errors = 0;
  isplaying = 0;
  remainingBytesInChunk = 0;

#if DVOLUME
  volume = 0;
#endif //DVOLUME
  // position to data
  return readWaveData(0, 0) < 0 ? false: true;
}
//------------------------------------------------------------------------------
/**
 * Returns true if the player is paused else false.
 */
uint8_t WaveHC::isPaused(void) {
  cli();
  uint8_t rtn = isplaying && !(TIMSK1 & _BV(OCIE1A));
  sei();
  return rtn;
}
//------------------------------------------------------------------------------
/**
 * Pause the player.
 */
void WaveHC::pause(void) {
  cli();
  TIMSK1 &= ~_BV(OCIE1A); //disable DAC interrupt
  sei();
  fd->volume()->rawDevice()->readEnd(); // redo any partial read on resume
}
```

```
//------------------------------------------------------------------------------
/**
 * Play a wave file.
 *
 * WaveHC::create() must be called before a file can be played.
 *
 * Check the member variable WaveHC::isplaying to monitor the status
 * of the player.
 */
void WaveHC::play(void) {
  // setup the interrupt as necessary

  int16_t read;

  playing = this;

  // fill the play buffer
  read = readWaveData(buffer1, PLAYBUFFLEN);
  if (read <= 0) return;
  playpos = buffer1;
  playend = buffer1 + read;

  // fill the second buffer
  read = readWaveData(buffer2, PLAYBUFFLEN);
  if (read < 0) return;
  sdbuff = buffer2;
  sdend = sdbuff + read;
  sdstatus = SD_READY;

  // its official!
  isplaying = 1;

  // Setup mode for DAC ports
  mcpDacInit();

  // Set up timer one
  // Normal operation - no pwm not connected to pins
  TCCR1A = 0;
  // no prescaling, CTC mode
  TCCR1B = _BV(WGM12) | _BV(CS10);
  // Sample rate - play stereo interleaved
  OCR1A =  F_CPU / (dwSamplesPerSec*Channels);
  // SD fill interrupt happens at TCNT1 == 1
  OCR1B = 1;
  // Enable timer interrupt for DAC ISR
  TIMSK1 |= _BV(OCIE1A);
}
//------------------------------------------------------------------------------
/** Read wave data.
 *
 * Not for use in applications.  Must be public so SD read ISR can access it.
 * Insures SD sectors are aligned with buffers.
 */
int16_t WaveHC::readWaveData(uint8_t *buff, uint16_t len) {

  if (remainingBytesInChunk == 0) {
    struct {
      char     id[4];
```

```
      uint32_t size;
    } header;
    while (1) {
      if (fd->read(&header, 8) != 8) return -1;
      if (!strncmp(header.id, "data", 4)) {
        remainingBytesInChunk = header.size;
        break;
      }

      // if not "data" then skip it!
      if (!fd->seekCur(header.size)) {
        return -1;
      }
    }
  }

  // make sure buffers are aligned on SD sectors
  uint16_t maxLen = PLAYBUFFLEN - fd->readPosition() % PLAYBUFFLEN;
  if (len > maxLen) len = maxLen;

  if (len > remainingBytesInChunk) {
    len = remainingBytesInChunk;
  }

  int16_t ret = fd->read(buff, len);
  if (ret > 0) remainingBytesInChunk -= ret;
  return ret;
}
//------------------------------------------------------------------------------
/** Resume a paused player. */
void WaveHC::resume(void) {
  cli();
  // enable DAC interrupt
  if(isplaying) TIMSK1 |= _BV(OCIE1A);
  sei();
}
//------------------------------------------------------------------------------
/**
 * Reposition a wave file.
 *
 * \param[in] pos seek will attempt to position the file near \a pos.
 * \a pos is the byte number from the beginning of file.
 */
void WaveHC::seek(uint32_t pos) {
  // make sure buffer fill interrupt doesn't happen
  cli();
  if (fd) {
    pos -= pos % PLAYBUFFLEN;
    if (pos < PLAYBUFFLEN) pos = PLAYBUFFLEN; //don't play metadata
    uint32_t maxPos = fd->readPosition() + remainingBytesInChunk;
    if (maxPos > fd->fileSize()) maxPos = fd->fileSize();
    if (pos > maxPos) pos = maxPos;
    if (fd->seekSet(pos)) {
      // assumes a lot about the wave file
      remainingBytesInChunk = maxPos - pos;
    }
  }
  sei();
```

```
}
//------------------------------------------------------------------------------
/** Set the player's sample rate.
 *
 * \param[in] samplerate The new sample rate in samples per second.
 * No checks are done on the input parameter.
 */
void WaveHC::setSampleRate(uint32_t samplerate) {
  if (samplerate < 500) samplerate = 500;
  if (samplerate > 50000) samplerate = 50000;
  // from ladayada's library.
  cli();
  while (TCNT0 != 0);

  OCR1A = F_CPU / samplerate;
  sei();
}
//------------------------------------------------------------------------------
/** Stop the player. */
void WaveHC::stop(void) {
  TIMSK1 &= ~_BV(OCIE1A);   // turn off interrupt
  playing->isplaying = 0;
  playing = 0;
}
```

SdInfo.h[24]

```
/* Arduino Sd2Card Library
 * Copyright (C) 2009 by William Greiman
 *
 * This file is part of the Arduino Sd2Card Library
 *
 * This Library is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This Library is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with the Arduino Sd2Card Library.  If not, see
 * <http://www.gnu.org/licenses/>.
 */
#ifndef SdInfo_h
#define SdInfo_h
#include <stdint.h>
//Based on the document:
//
//SD Specifications
//Part 1
//Physical Layer
//Simplified Specification
```

```
//Version 2.00
//September 25, 2006
//
//www.sdcard.org/developers/tech/sdcard/pls/Simplified_Physical_Layer_Spec.pdf
//------------------------------------------------------------------------------
// SD card commands
/** GO_IDLE_STATE - init card in spi mode if CS low */
#define CMD0     0X00
/** SEND_IF_COND - verify SD Memory Card interface operating condition.*/
#define CMD8     0X08
/** SEND_CSD - read the Card Specific Data (CSD register) */
#define CMD9     0X09
 /** SEND_CID - read the card identification information (CID register) */
#define CMD10    0X0A
/** SEND_STATUS - read the card status register */
#define CMD13    0X0D
/** READ_BLOCK - read a single data block from the card */
#define CMD17    0X11
/** WRITE_BLOCK - write a single data block to the card */
#define CMD24    0X18
/** WRITE_MULTIPLE_BLOCK - write blocks of data until a STOP_TRANSMISSION */
#define CMD25    0X19
/** ERASE_WR_BLK_START - sets the address of the first block to be erased */
#define CMD32    0X20
/** ERASE_WR_BLK_END - sets the address of the last block of the continuous
    range to be erased*/
#define CMD33    0X21
/** ERASE - erase all previously selected blocks */
#define CMD38    0X26
/** APP_CMD - escape for application specific command */
#define CMD55    0X37
/** READ_OCR - read the OCR register of a card */
#define CMD58    0X3A
/** SET_WR_BLK_ERASE_COUNT - Set the number of write blocks to be
     pre-erased before writing */
#define ACMD23   0X17
/** SD_SEND_OP_COMD - Sends host capacity support information and
    activates the card's initialization process */
#define ACMD41   0X29
//------------------------------------------------------------------------------
/** status for card in the ready state */
#define R1_READY_STATE 0
/** status for card in the idle state */
#define R1_IDLE_STATE  1
/** status bit for illegal command */
#define R1_ILLEGAL_COMMAND 4
/** start data token for read or write single block*/
#define DATA_START_BLOCK      0XFE
/** stop token for write multiple blocks*/
#define STOP_TRAN_TOKEN       0XFD
/** start data token for write multiple blocks*/
#define WRITE_MULTIPLE_TOKEN  0XFC
/** mask for data response tokens after a write block operation */
#define DATA_RES_MASK         0X1F
/** write data accepted token */
#define DATA_RES_ACCEPTED     0X05
//------------------------------------------------------------------------------
typedef struct CID {
```

```
  //byte 0
  uint8_t mid;//Manufacturer ID
  //byte 1-2
  char oid[2];//OEM/Application ID
  //byte 3-7
  char pnm[5];//Product name
  //byte 8
  unsigned prv_m : 4;// Product revision n.m
  unsigned prv_n : 4;
  //byte 9-12
  uint32_t psn;//Product serial number
  //byte 13
  unsigned mdt_year_high : 4;//Manufacturing date
  unsigned reserved : 4;
  //byte 14
  unsigned mdt_month : 4;
  unsigned mdt_year_low :4;
  //byte 15
  unsigned always1 : 1;
  unsigned crc : 7;
}cid_t;
//------------------------------------------------------------------------------
// CSD for version 1.00 cards
typedef struct CSDV1 {
  //byte 0
  unsigned reserved1 : 6;
  unsigned csd_ver : 2;
  //byte 1
  uint8_t taac;
  //byte 2
  uint8_t nsac;
  //byte 3
  uint8_t tran_speed;
  //byte 4
  uint8_t ccc_high;
  //byte 5
  unsigned read_bl_len : 4;
  unsigned ccc_low : 4;
//  unsigned read_bl_len : 4;
  //byte 6
  unsigned c_size_high : 2;
  unsigned reserved2 : 2;
  unsigned dsr_imp : 1;
  unsigned read_blk_misalign :1;
  unsigned write_blk_misalign : 1;
  unsigned read_bl_partial : 1;
  //byte 7
  uint8_t c_size_mid;
  //byte 8
  unsigned vdd_r_curr_max : 3;
  unsigned vdd_r_curr_min : 3;
  unsigned c_size_low :2;
  //byte 9
  unsigned c_size_mult_high : 2;
  unsigned vdd_w_cur_max : 3;
  unsigned vdd_w_curr_min : 3;
  //byte 10
  unsigned sector_size_high : 6;
```

```
  unsigned erase_blk_en : 1;
  unsigned c_size_mult_low : 1;
  //byte 11
  unsigned wp_grp_size : 7;
  unsigned sector_size_low : 1;
  //byte 12
  unsigned write_bl_len_high : 2;
  unsigned r2w_factor : 3;
  unsigned reserved3 : 2;
  unsigned wp_grp_enable : 1;
  //byte 13
  unsigned reserved4 : 5;
  unsigned write_partial : 1;
  unsigned write_bl_len_low : 2;
  //byte 14
  unsigned reserved5: 2;
  unsigned file_format : 2;
  unsigned tmp_write_protect : 1;
  unsigned perm_write_protect : 1;
  unsigned copy : 1;
  unsigned file_format_grp : 1;
  //byte 15
  unsigned always1 : 1;
  unsigned crc : 7;
}csd1_t;
//-----------------------------------------------------------------------------
// CSD for version 2.00 cards
typedef struct CSDV2 {
  //byte 0
  unsigned reserved1 : 6;
  unsigned csd_ver : 2;
  //byte 1
  uint8_t taac;
  //byte 2
  uint8_t nsac;
  //byte 3
  uint8_t tran_speed;
  //byte 4
  uint8_t ccc_high;
  //byte 5
  unsigned read_bl_len : 4;
  unsigned ccc_low : 4;
  //byte 6
  unsigned reserved2 : 4;
  unsigned dsr_imp : 1;
  unsigned read_blk_misalign :1;
  unsigned write_blk_misalign : 1;
  unsigned read_bl_partial : 1;
  //byte 7
  unsigned reserved3 : 2;
  unsigned c_size_high : 6;
  //byte 8
  uint8_t c_size_mid;
  //byte 9
  uint8_t c_size_low;
  //byte 10
  unsigned sector_size_high : 6;
  unsigned erase_blk_en : 1;
```

```
  unsigned reserved4 : 1;
  //byte 11
  unsigned wp_grp_size : 7;
  unsigned sector_size_low : 1;
  //byte 12
  unsigned write_bl_len_high : 2;
  unsigned r2w_factor : 3;
  unsigned reserved5 : 2;
  unsigned wp_grp_enable : 1;
  //byte 13
  unsigned reserved6 : 5;
  unsigned write_partial : 1;
  unsigned write_bl_len_low : 2;
  //byte 14
  unsigned reserved7: 2;
  unsigned file_format : 2;
  unsigned tmp_write_protect : 1;
  unsigned perm_write_protect : 1;
  unsigned copy : 1;
  unsigned file_format_grp : 1;
  //byte 15
  unsigned always1 : 1;
  unsigned crc : 7;
}csd2_t;
//------------------------------------------------------------------------------
// union of old and new style CSD register
union csd_t {
  csd1_t v1;
  csd2_t v2;
};
#endif //SdInfo_h
```

SdReader.h[24]

```
/* Arduino WaveHC Library
 * Copyright (C) 2008 by William Greiman
 *
 * This file is part of the Arduino FAT16 Library
 *
 * This Library is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This Library is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.

 * You should have received a copy of the GNU General Public License
 * along with the Arduino Fat16 Library.  If not, see
 * <http://www.gnu.org/licenses/>.
 */
#ifndef SdReader_h
```

```
#define SdReader_h
#include <SdInfo.h>
/**
 * \file
 * SdReader class
 */
/**
 * Some SD card are very sensitive to the SPI bus speed for initialization.
 * Try setting SPI_INIT_SLOW nonzero if you have initialization problems.
 *
 * Set SPI_INIT_SLOW nonzero to reduce the SPI bus speed for SD initaizaton
 * to F_CPU/128.  F_CPU/64 is used if
 */
#define SPI_INIT_SLOW 0
/**
 * Default card SPI speed. Change to true for Wave Shield V1.0
 * The SPI speed is 4 Mhz for 'true' and 8 Mhz for 'false'.
 */
#define SPI_DEFAULT_HALF_SPEED false

/** read timeout ms */
#define SD_READ_TIMEOUT    300

// SD card errors
/** timeout error for command CMD0 */
#define SD_CARD_ERROR_CMD0  0X1
/** CMD8 was not accepted - not a valid SD card*/
#define SD_CARD_ERROR_CMD8  0X2
/** card returned an error response for CMD17 (read block) */
#define SD_CARD_ERROR_CMD17 0X3
/** card returned an error response for CMD24 (write block) */
#define SD_CARD_ERROR_CMD24 0X4
/** card returned an error response for CMD58 (read OCR) */
#define SD_CARD_ERROR_CMD58 0X5
/** card's ACMD41 initialization process timeout */
#define SD_CARD_ERROR_ACMD41 0X6
/** card returned a bad CSR version field */
#define SD_CARD_ERROR_BAD_CSD 0X7
/** read CID or CSD failed */
#define SD_CARD_ERROR_READ_REG 0X8
/** bad response echo from CMD8 */
#define SD_CARD_ERROR_CMD8_ECHO 0X09
/** timeout while waiting for start of read data */
#define SD_CARD_ERROR_READ_TIMEOUT 0XD
/** card returned an error token instead of read data */
#define SD_CARD_ERROR_READ 0X10
//
// card types
/** Standard capacity V1 SD card */
#define SD_CARD_TYPE_SD1 1
/** Standard capacity V2 SD card */
#define SD_CARD_TYPE_SD2 2
/** High Capacity SD card */
#define SD_CARD_TYPE_SDHC 3
//------------------------------------------------------------------------------
/**
 * \class SdReader
 * \brief Hardware access class for SD flash cards
```

```
 *
 * Supports raw access to SD and SDHC flash memory cards.
 *
 */
class SdReader  {
  uint32_t block_;
  uint8_t errorCode_;
  uint8_t errorData_;
  uint8_t inBlock_;
  uint16_t offset_;
  uint8_t partialBlockRead_;
  uint8_t response_;
  uint8_t type_;
  uint8_t cardCommand(uint8_t cmd, uint32_t arg);
  void error(uint8_t code){errorCode_ = code;}
  void error(uint8_t code, uint8_t data) {errorCode_ = code; errorData_ = data;}
  uint8_t readRegister(uint8_t cmd, uint8_t *dst);
  void type(uint8_t value) {type_ = value;}
  uint8_t waitNotBusy(uint16_t timeoutMillis);
  uint8_t waitStartBlock(void);
public:
  /** Construct an instance of SdReader. */
  SdReader(void) :  errorCode_(0), inBlock_(0), partialBlockRead_(0), type_(0) {};
  uint32_t cardSize(void);
  /** \return error code for last error */
  uint8_t errorCode(void) {return errorCode_;}
  /** \return error data for last error */
  uint8_t errorData(void) {return errorData_;}
  uint8_t init(uint8_t slow = SPI_DEFAULT_HALF_SPEED);
  /**
   * Enable or disable partial block reads.
   *
   * Enabling partial block reads improves performance by allowing a block
   * to be read over the SPI bus as several sub-blocks.  Errors will occur
   * if the time between reads is too long since the SD card will timeout.
   *
   * Use this for applications like the Adafruit Wave Shield.
   *
   * \param[in] value The value TRUE (non-zero) or FALSE (zero).)
   */
  void partialBlockRead(uint8_t value) {readEnd(); partialBlockRead_ = value;}
  /**
 * Read a 512 byte block from a SD card device.
 *
 * \param[in] block Logical block to be read.
 * \param[out] dst Pointer to the location that will receive the data.
 *
 * \return The value one, true, is returned for success and
 * the value zero, false, is returned for failure.
 */
  uint8_t readBlock(uint32_t block, uint8_t *dst) {
    return readData(block, 0, dst, 512);}
  uint8_t readData(uint32_t block, uint16_t offset, uint8_t *dst, uint16_t count);
  /**
   * Read a cards CID register. The CID contains card identification information
   * such as Manufacturer ID, Product name, Product serial number and
   * Manufacturing date. */
  uint8_t readCID(cid_t &cid) {return readRegister(CMD10, (uint8_t *)&cid);}
```

```
  /**
   * Read a cards CSD register. The CSD contains Card-Specific Data that
   * provides information regarding access to the card contents. */
  uint8_t readCSD(csd_t &csd) {return readRegister(CMD9, (uint8_t *)&csd);}
  void readEnd(void);
  /** Return the card type: SD V1, SD V2 or SDHC */
  uint8_t type() {return type_;}
};
#endif //SdReader_h
```

SdReader.cpp[24]

```
/* Arduino WaveHC Library
 * Copyright (C) 2008 by William Greiman
 *
 * This file is part of the Arduino WaveHC Library
 *
 * This Library is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This Library is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.

 * You should have received a copy of the GNU General Public License
 * along with the Arduino WaveHC Library.  If not, see
 * <http://www.gnu.org/licenses/>.
 */
#if ARDUINO < 100
#include <WProgram.h>
#else  // ARDUINO < 100
#include <Arduino.h>
#endif  // ARDUINO < 100
#include <SdReader.h>
#include <WavePinDefs.h>
//------------------------------------------------------------------------------
// inline SPI functions
/** Send a byte to the card */
inline void spiSend(uint8_t b) {SPDR = b; while(!(SPSR & (1 << SPIF)));}
/** Receive a byte from the card */
inline uint8_t spiRec(void) {spiSend(0XFF); return SPDR;}
/** Set Slave Select high */
inline void spiSSHigh(void) {
  digitalWrite(SS, HIGH);
  // insure SD data out is high Z
  spiSend(0XFF);
}
/** Set Slave Select low */
inline void spiSSLow(void) {digitalWrite(SS, LOW);}
//------------------------------------------------------------------------------
```

```
// card status
/** status for card in the ready state */
#define R1_READY_STATE 0
/** status for card in the idle state */
#define R1_IDLE_STATE  1
/** start data token for read or write */
#define DATA_START_BLOCK      0XFE
/** mask for data response tokens after a write block operation */
#define DATA_RES_MASK         0X1F
/** write data accepted token */
#define DATA_RES_ACCEPTED     0X05
/** write data crc error token */
#define DATA_RES_CRC_ERROR    0X0B
/** write data programming error token */
#define DATA_RES_WRITE_ERROR  0X0D
//------------------------------------------------------------------------------
// send command to card
uint8_t SdReader::cardCommand(uint8_t cmd, uint32_t arg) {
  uint8_t r1;

  // end read if in partialBlockRead mode
  readEnd();

  // select card
  spiSSLow();

  // wait up to 300 ms if busy
  waitNotBusy(300);

  // send command
  spiSend(cmd | 0x40);

  // send argument
  for (int8_t s = 24; s >= 0; s -= 8) spiSend(arg >> s);

  // send CRC
  uint8_t crc = 0XFF;
  if (cmd == CMD0) crc = 0X95; // correct crc for CMD0 with arg 0
  if (cmd == CMD8) crc = 0X87; // correct crc for CMD8 with arg 0X1AA
  spiSend(crc);

  // wait for response
  for (uint8_t retry = 0; ((r1 = spiRec()) & 0X80) && retry != 0XFF; retry++);

  return r1;
}
//------------------------------------------------------------------------------
/**
 * Determine the size of an SD flash memory card.
 * \return The number of 512 byte data blocks in the card
 */
uint32_t SdReader::cardSize(void) {
  csd_t csd;
  if (!readCSD(csd)) return false;
  if (csd.v1.csd_ver == 0) {
    uint8_t read_bl_len = csd.v1.read_bl_len;
    uint16_t c_size = (csd.v1.c_size_high << 10)
                      | (csd.v1.c_size_mid << 2) | csd.v1.c_size_low;
```

```
    uint8_t c_size_mult = (csd.v1.c_size_mult_high << 1)
                          | csd.v1.c_size_mult_low;
    return (uint32_t)(c_size + 1) << (c_size_mult + read_bl_len - 7);
  }
  else if (csd.v2.csd_ver == 1) {
    uint32_t c_size = ((uint32_t)csd.v2.c_size_high << 16)
                      | (csd.v2.c_size_mid << 8) | csd.v2.c_size_low;
    return (c_size + 1) << 10;
  }
  else {
    error(SD_CARD_ERROR_BAD_CSD);
    return 0;
  }
}
//------------------------------------------------------------------------------
/**
 * Initialize a SD flash memory card.
 *
 * \param[in] slow If \a slow is false (zero) the SPI bus will
 * be initialize at a speed of 8 Mhz.  If \a slow is true (nonzero)
 * the SPI bus will be initialize a speed of 4 Mhz. This may be helpful
 * for some SD cards with Version 1.0 of the Adafruit Wave Shield.
 *
 * \return The value one, true, is returned for success and
 * the value zero, false, is returned for failure.
 *
 */
uint8_t SdReader::init(uint8_t slow) {
  uint8_t ocr[4];
  uint8_t r;

  pinMode(SS, OUTPUT);
  digitalWrite(SS, HIGH);
  pinMode(MOSI, OUTPUT);
  pinMode(MISO_PIN, INPUT);
  pinMode(SCK, OUTPUT);

#if SPI_INIT_SLOW
  // Enable SPI, Master, clock rate f_osc/128
  SPCR = (1 << SPE) | (1 << MSTR) | (1 << SPR1) | (1 << SPR0);
#else  // SPI_INIT_SLOW
  // Enable SPI, Master, clock rate f_osc/64
  SPCR = (1 << SPE) | (1 << MSTR) | (1 << SPR1);
#endif  // SPI_INIT_SLOW

  // must supply min of 74 clock cycles with CS high.
  for (uint8_t i = 0; i < 10; i++) spiSend(0XFF);

  // next two lines prevent re-init hang by cards that were in partial read
  spiSSLow();
  for (uint16_t i = 0; i <= 512; i++) spiRec();

  // command to go idle in SPI mode
  for (uint8_t retry = 0; ; retry++) {
    if ((r = cardCommand(CMD0, 0)) ==  R1_IDLE_STATE) break;
    if (retry == 10) {
      error(SD_CARD_ERROR_CMD0, r);
      return false;
```

```
    }
  }
  // check SD version
  r = cardCommand(CMD8, 0x1AA);
  if (r == R1_IDLE_STATE) {
    for(uint8_t i = 0; i < 4; i++) {
      r = spiRec();
    }
    if (r != 0XAA) {
      error(SD_CARD_ERROR_CMD8_ECHO, r);
      return false;
    }
    type(SD_CARD_TYPE_SD2);
  }
  else if (r & R1_ILLEGAL_COMMAND) {
    type(SD_CARD_TYPE_SD1);
  }
  else {
    error(SD_CARD_ERROR_CMD8, r);
  }
  // initialize card and send host supports SDHC if SD2
  for (uint16_t t0 = millis();;) {
    cardCommand(CMD55, 0);
    r = cardCommand(ACMD41, type() == SD_CARD_TYPE_SD2 ? 0X40000000 : 0);
    if (r == R1_READY_STATE) break;

    // timeout after 2 seconds
    if (((uint16_t)millis() - t0) > 2000) {
      error(SD_CARD_ERROR_ACMD41);
      return false;
    }
  }
  // if SD2 read OCR register to check for SDHC card
  if (type() == SD_CARD_TYPE_SD2) {
    if(cardCommand(CMD58, 0)) {
      error(SD_CARD_ERROR_CMD58);
      return false;
    }
    if ((spiRec() & 0XC0) == 0XC0) type(SD_CARD_TYPE_SDHC);

    // discard rest of ocr
    for (uint8_t i = 0; i < 3; i++) spiRec();
  }

  // use max SPI frequency unless slow is true
  SPCR &= ~((1 << SPR1) | (1 << SPR0)); // f_OSC/4

  if (!slow) SPSR |= (1 << SPI2X); // Doubled Clock Frequency: f_OSC/2
  spiSSHigh();
  return true;
}
//------------------------------------------------------------------------------
/**
 * Read part of a 512 byte block from a SD card.
 *
 * \param[in] block Logical block to be read.
 * \param[in] offset Number of bytes to skip at start of block
 * \param[out] dst Pointer to the location that will receive the data.
```

```
 * \param[in] count Number of bytes to read
 * \return The value one, true, is returned for success and
 * the value zero, false, is returned for failure.
 */
uint8_t SdReader::readData(uint32_t block,
                           uint16_t offset, uint8_t *dst, uint16_t count) {
  if (count == 0) return true;
  if ((count + offset) > 512) {
    return false;
  }
  if (!inBlock_ || block != block_ || offset < offset_) {
    block_ = block;

    // use address if not SDHC card
    if (type()!= SD_CARD_TYPE_SDHC) block <<= 9;
    if (cardCommand(CMD17, block)) {
      error(SD_CARD_ERROR_CMD17);
      return false;
    }
    if (!waitStartBlock()) {
      return false;
    }
    offset_ = 0;
    inBlock_ = 1;
  }

  // start first SPI transfer
  SPDR = 0XFF;

  // skip data before offset
  for (;offset_ < offset; offset_++) {
    while(!(SPSR & (1 << SPIF)));
    SPDR = 0XFF;
  }

  // transfer data
  uint16_t n = count - 1;
  for (uint16_t i = 0; i < n; i++) {
    while(!(SPSR & (1 << SPIF)));
    dst[i] = SPDR;
    SPDR = 0XFF;
  }

  // wait for last byte
  while(!(SPSR & (1 << SPIF)));
  dst[n] = SPDR;
  offset_ += count;
  if (!partialBlockRead_ || offset_ >= 512) readEnd();
  return true;
}
//------------------------------------------------------------------------------
/** Skip remaining data in a block when in partial block read mode. */
void SdReader::readEnd(void) {
  if (inBlock_) {
    // skip data and crc
    SPDR = 0XFF;
    while (offset_++ < 513) {
      while(!(SPSR & (1 << SPIF)));
```

```
      SPDR = 0XFF;
    }
    // wait for last crc byte
    while(!(SPSR & (1 << SPIF)));
    spiSSHigh();
    inBlock_ = 0;
  }
}
//------------------------------------------------------------------------------
/** read CID or CSR register */
uint8_t SdReader::readRegister(uint8_t cmd, uint8_t *dst) {
  if (cardCommand(cmd, 0)) {
    error(SD_CARD_ERROR_READ_REG);
    return false;
  }
  if(!waitStartBlock()) return false;

  //transfer data
  for (uint16_t i = 0; i < 16; i++) dst[i] = spiRec();

  spiRec();// get first crc byte
  spiRec();// get second crc byte

  spiSSHigh();
  return true;
}
//------------------------------------------------------------------------------
// wait for card to go not busy
uint8_t SdReader::waitNotBusy(uint16_t timeoutMillis) {
  uint16_t t0 = millis();
  while (spiRec() != 0XFF) {
    if (((uint16_t)millis() - t0) > timeoutMillis) return false;
  }
  return true;
}
//------------------------------------------------------------------------------
/** Wait for start block token */
uint8_t SdReader::waitStartBlock(void) {
  uint8_t r;
  uint16_t t0 = millis();
  while ((r = spiRec()) == 0XFF) {
    if (((uint16_t)millis() - t0) > SD_READ_TIMEOUT) {
      error(SD_CARD_ERROR_READ_TIMEOUT);
      return false;
    }
  }
  if (r == DATA_START_BLOCK) return true;
  error(SD_CARD_ERROR_READ, r);
  return false;
}
```

ArduinoPins.h[24]

```
// Map of Arduino pins to avr bit, ddr, port, pin
// Credit Paul Stoffregen for idea
#ifndef ArduinoPins_h
```

```
#define ArduinoPins_h

#define PIN_BITNUM(pin) (PIN ## pin ## _BITNUM)
#define PIN_PORTREG(pin) (PIN ## pin ## _PORTREG)
#define PIN_DDRREG(pin) (PIN ## pin ## _DDRREG)
#define PIN_PINREG(pin) (PIN ## pin ## _PINREG)
#ifndef _BV
#define _BV(n) (1<<(n))
#endif

#if defined(__AVR_ATmega1280__) || defined(__AVR_ATmega2560__)
// Mega Arduino

// Two Wire (aka I2C) ports
#define SDA_PIN 20
#define SCL_PIN 21

// SPI port
#define SS_PIN 53
#define MOSI_PIN 51
#define MISO_PIN 50
#define SCK_PIN 52

// bit number for all digital pins
#define PIN0_BITNUM 0
#define PIN1_BITNUM 1
#define PIN2_BITNUM 4
#define PIN3_BITNUM 5
#define PIN4_BITNUM 5
#define PIN5_BITNUM 3
#define PIN6_BITNUM 3
#define PIN7_BITNUM 4
#define PIN8_BITNUM 5
#define PIN9_BITNUM 6
#define PIN10_BITNUM 4
#define PIN11_BITNUM 5
#define PIN12_BITNUM 6
#define PIN13_BITNUM 7
#define PIN14_BITNUM 1
#define PIN15_BITNUM 0
#define PIN16_BITNUM 1
#define PIN17_BITNUM 0
#define PIN18_BITNUM 3
#define PIN19_BITNUM 2
#define PIN20_BITNUM 1
#define PIN21_BITNUM 0
#define PIN22_BITNUM 0
#define PIN23_BITNUM 1
#define PIN24_BITNUM 2
#define PIN25_BITNUM 3
#define PIN26_BITNUM 4
#define PIN27_BITNUM 5
#define PIN28_BITNUM 6
#define PIN29_BITNUM 7
#define PIN30_BITNUM 7
#define PIN31_BITNUM 6
#define PIN32_BITNUM 5
#define PIN33_BITNUM 4
```

```
#define PIN34_BITNUM 3
#define PIN35_BITNUM 2
#define PIN36_BITNUM 1
#define PIN37_BITNUM 0
#define PIN38_BITNUM 7
#define PIN39_BITNUM 2
#define PIN40_BITNUM 1
#define PIN41_BITNUM 0
#define PIN42_BITNUM 7
#define PIN43_BITNUM 6
#define PIN44_BITNUM 5
#define PIN45_BITNUM 4
#define PIN46_BITNUM 3
#define PIN47_BITNUM 2
#define PIN48_BITNUM 1
#define PIN49_BITNUM 0
#define PIN50_BITNUM 3
#define PIN51_BITNUM 2
#define PIN52_BITNUM 1
#define PIN53_BITNUM 0
#define PIN54_BITNUM 0
#define PIN55_BITNUM 1
#define PIN56_BITNUM 2
#define PIN57_BITNUM 3
#define PIN58_BITNUM 4
#define PIN59_BITNUM 5
#define PIN60_BITNUM 6
#define PIN61_BITNUM 7
#define PIN62_BITNUM 0
#define PIN63_BITNUM 1
#define PIN64_BITNUM 2
#define PIN65_BITNUM 3
#define PIN66_BITNUM 4
#define PIN67_BITNUM 5
#define PIN68_BITNUM 6
#define PIN69_BITNUM 7

// output register for digital pins
#define PIN0_PORTREG PORTE
#define PIN1_PORTREG PORTE
#define PIN2_PORTREG PORTE
#define PIN3_PORTREG PORTE
#define PIN4_PORTREG PORTG
#define PIN5_PORTREG PORTE
#define PIN6_PORTREG PORTH
#define PIN7_PORTREG PORTH
#define PIN8_PORTREG PORTH
#define PIN9_PORTREG PORTH
#define PIN10_PORTREG PORTB
#define PIN11_PORTREG PORTB
#define PIN12_PORTREG PORTB
#define PIN13_PORTREG PORTB
#define PIN14_PORTREG PORTJ
#define PIN15_PORTREG PORTJ
#define PIN16_PORTREG PORTH
#define PIN17_PORTREG PORTH
#define PIN18_PORTREG PORTD
#define PIN19_PORTREG PORTD
```

```
#define PIN20_PORTREG PORTD
#define PIN21_PORTREG PORTD
#define PIN22_PORTREG PORTA
#define PIN23_PORTREG PORTA
#define PIN24_PORTREG PORTA
#define PIN25_PORTREG PORTA
#define PIN26_PORTREG PORTA
#define PIN27_PORTREG PORTA
#define PIN28_PORTREG PORTA
#define PIN29_PORTREG PORTA
#define PIN30_PORTREG PORTC
#define PIN31_PORTREG PORTC
#define PIN32_PORTREG PORTC
#define PIN33_PORTREG PORTC
#define PIN34_PORTREG PORTC
#define PIN35_PORTREG PORTC
#define PIN36_PORTREG PORTC
#define PIN37_PORTREG PORTC
#define PIN38_PORTREG PORTD
#define PIN39_PORTREG PORTG
#define PIN40_PORTREG PORTG
#define PIN41_PORTREG PORTG
#define PIN42_PORTREG PORTL
#define PIN43_PORTREG PORTL
#define PIN44_PORTREG PORTL
#define PIN45_PORTREG PORTL
#define PIN46_PORTREG PORTL
#define PIN47_PORTREG PORTL
#define PIN48_PORTREG PORTL
#define PIN49_PORTREG PORTL
#define PIN50_PORTREG PORTB
#define PIN51_PORTREG PORTB
#define PIN52_PORTREG PORTB
#define PIN53_PORTREG PORTB
#define PIN54_PORTREG PORTF
#define PIN55_PORTREG PORTF
#define PIN56_PORTREG PORTF
#define PIN57_PORTREG PORTF
#define PIN58_PORTREG PORTF
#define PIN59_PORTREG PORTF
#define PIN60_PORTREG PORTF
#define PIN61_PORTREG PORTF
#define PIN62_PORTREG PORTK
#define PIN63_PORTREG PORTK
#define PIN64_PORTREG PORTK
#define PIN65_PORTREG PORTK
#define PIN66_PORTREG PORTK
#define PIN67_PORTREG PORTK
#define PIN68_PORTREG PORTK
#define PIN69_PORTREG PORTK

// direction control register for digital pins
#define PIN0_DDRREG DDRE
#define PIN1_DDRREG DDRE
#define PIN2_DDRREG DDRE
#define PIN3_DDRREG DDRE
#define PIN4_DDRREG DDRG
#define PIN5_DDRREG DDRE
```

```
#define PIN6_DDRREG DDRH
#define PIN7_DDRREG DDRH
#define PIN8_DDRREG DDRH
#define PIN9_DDRREG DDRH
#define PIN10_DDRREG DDRB
#define PIN11_DDRREG DDRB
#define PIN12_DDRREG DDRB
#define PIN13_DDRREG DDRB
#define PIN14_DDRREG DDRJ
#define PIN15_DDRREG DDRJ
#define PIN16_DDRREG DDRH
#define PIN17_DDRREG DDRH
#define PIN18_DDRREG DDRD
#define PIN19_DDRREG DDRD
#define PIN20_DDRREG DDRD
#define PIN21_DDRREG DDRD
#define PIN22_DDRREG DDRA
#define PIN23_DDRREG DDRA
#define PIN24_DDRREG DDRA
#define PIN25_DDRREG DDRA
#define PIN26_DDRREG DDRA
#define PIN27_DDRREG DDRA
#define PIN28_DDRREG DDRA
#define PIN29_DDRREG DDRA
#define PIN30_DDRREG DDRC
#define PIN31_DDRREG DDRC
#define PIN32_DDRREG DDRC
#define PIN33_DDRREG DDRC
#define PIN34_DDRREG DDRC
#define PIN35_DDRREG DDRC
#define PIN36_DDRREG DDRC
#define PIN37_DDRREG DDRC
#define PIN38_DDRREG DDRD
#define PIN39_DDRREG DDRG
#define PIN40_DDRREG DDRG
#define PIN41_DDRREG DDRG
#define PIN42_DDRREG DDRL
#define PIN43_DDRREG DDRL
#define PIN44_DDRREG DDRL
#define PIN45_DDRREG DDRL
#define PIN46_DDRREG DDRL
#define PIN47_DDRREG DDRL
#define PIN48_DDRREG DDRL
#define PIN49_DDRREG DDRL
#define PIN50_DDRREG DDRB
#define PIN51_DDRREG DDRB
#define PIN52_DDRREG DDRB
#define PIN53_DDRREG DDRB
#define PIN54_DDRREG DDRF
#define PIN55_DDRREG DDRF
#define PIN56_DDRREG DDRF
#define PIN57_DDRREG DDRF
#define PIN58_DDRREG DDRF
#define PIN59_DDRREG DDRF
#define PIN60_DDRREG DDRF
#define PIN61_DDRREG DDRF
#define PIN62_DDRREG DDRK
#define PIN63_DDRREG DDRK
```

```
#define PIN64_DDRREG DDRK
#define PIN65_DDRREG DDRK
#define PIN66_DDRREG DDRK
#define PIN67_DDRREG DDRK
#define PIN68_DDRREG DDRK
#define PIN69_DDRREG DDRK

// input register for digital pins
#define PIN0_PINREG PINE
#define PIN1_PINREG PINE
#define PIN2_PINREG PINE
#define PIN3_PINREG PINE
#define PIN4_PINREG PING
#define PIN5_PINREG PINE
#define PIN6_PINREG PINH
#define PIN7_PINREG PINH
#define PIN8_PINREG PINH
#define PIN9_PINREG PINH
#define PIN10_PINREG PINB
#define PIN11_PINREG PINB
#define PIN12_PINREG PINB
#define PIN13_PINREG PINB
#define PIN14_PINREG PINJ
#define PIN15_PINREG PINJ
#define PIN16_PINREG PINH
#define PIN17_PINREG PINH
#define PIN18_PINREG PIND
#define PIN19_PINREG PIND
#define PIN20_PINREG PIND
#define PIN21_PINREG PIND
#define PIN22_PINREG PINA
#define PIN23_PINREG PINA
#define PIN24_PINREG PINA
#define PIN25_PINREG PINA
#define PIN26_PINREG PINA
#define PIN27_PINREG PINA
#define PIN28_PINREG PINA
#define PIN29_PINREG PINA
#define PIN30_PINREG PINC
#define PIN31_PINREG PINC
#define PIN32_PINREG PINC
#define PIN33_PINREG PINC
#define PIN34_PINREG PINC
#define PIN35_PINREG PINC
#define PIN36_PINREG PINC
#define PIN37_PINREG PINC
#define PIN38_PINREG PIND
#define PIN39_PINREG PING
#define PIN40_PINREG PING
#define PIN41_PINREG PING
#define PIN42_PINREG PINL
#define PIN43_PINREG PINL
#define PIN44_PINREG PINL
#define PIN45_PINREG PINL
#define PIN46_PINREG PINL
#define PIN47_PINREG PINL
#define PIN48_PINREG PINL
#define PIN49_PINREG PINL
```

```
#define PIN50_PINREG PINB
#define PIN51_PINREG PINB
#define PIN52_PINREG PINB
#define PIN53_PINREG PINB
#define PIN54_PINREG PINF
#define PIN55_PINREG PINF
#define PIN56_PINREG PINF
#define PIN57_PINREG PINF
#define PIN58_PINREG PINF
#define PIN59_PINREG PINF
#define PIN60_PINREG PINF
#define PIN61_PINREG PINF
#define PIN62_PINREG PINK
#define PIN63_PINREG PINK
#define PIN64_PINREG PINK
#define PIN65_PINREG PINK
#define PIN66_PINREG PINK
#define PIN67_PINREG PINK
#define PIN68_PINREG PINK
#define PIN69_PINREG PINK

#elif defined (__AVR_ATmega644P__)
// Sanguino

#error Sanguino not defined

#else // defined(__AVR_ATmega1280__) || defined(__AVR_ATmega2560__)
// 168 and 328 Arduinos

// Two Wire (aka I2C) ports
#define SDA_PIN 18
#define SCL_PIN 19

// SPI port
#define SS_PIN 10
#define MOSI_PIN 11
#define MISO_PIN 12
#define SCK_PIN 13

// bit number for digital pins
#define PIN0_BITNUM 0
#define PIN1_BITNUM 1
#define PIN2_BITNUM 2
#define PIN3_BITNUM 3
#define PIN4_BITNUM 4
#define PIN5_BITNUM 5
#define PIN6_BITNUM 6
#define PIN7_BITNUM 7
#define PIN8_BITNUM 0
#define PIN9_BITNUM 1
#define PIN10_BITNUM 2
#define PIN11_BITNUM 3
#define PIN12_BITNUM 4
#define PIN13_BITNUM 5
#define PIN14_BITNUM 0
#define PIN15_BITNUM 1
#define PIN16_BITNUM 2
#define PIN17_BITNUM 3
```

```
#define PIN18_BITNUM 4
#define PIN19_BITNUM 5

// output register for all pins
#define PIN0_PORTREG PORTD
#define PIN1_PORTREG PORTD
#define PIN2_PORTREG PORTD
#define PIN3_PORTREG PORTD
#define PIN4_PORTREG PORTD
#define PIN5_PORTREG PORTD
#define PIN6_PORTREG PORTD
#define PIN7_PORTREG PORTD
#define PIN8_PORTREG PORTB
#define PIN9_PORTREG PORTB
#define PIN10_PORTREG PORTB
#define PIN11_PORTREG PORTB
#define PIN12_PORTREG PORTB
#define PIN13_PORTREG PORTB
#define PIN14_PORTREG PORTC
#define PIN15_PORTREG PORTC
#define PIN16_PORTREG PORTC
#define PIN17_PORTREG PORTC
#define PIN18_PORTREG PORTC
#define PIN19_PORTREG PORTC

// direction control register for digital pins
#define PIN0_DDRREG DDRD
#define PIN1_DDRREG DDRD
#define PIN2_DDRREG DDRD
#define PIN3_DDRREG DDRD
#define PIN4_DDRREG DDRD
#define PIN5_DDRREG DDRD
#define PIN6_DDRREG DDRD
#define PIN7_DDRREG DDRD
#define PIN8_DDRREG DDRB
#define PIN9_DDRREG DDRB
#define PIN10_DDRREG DDRB
#define PIN11_DDRREG DDRB
#define PIN12_DDRREG DDRB
#define PIN13_DDRREG DDRB
#define PIN14_DDRREG DDRC
#define PIN15_DDRREG DDRC
#define PIN16_DDRREG DDRC
#define PIN17_DDRREG DDRC
#define PIN18_DDRREG DDRC
#define PIN19_DDRREG DDRC

// input register for digital pins
#define PIN0_PINREG PIND
#define PIN1_PINREG PIND
#define PIN2_PINREG PIND
#define PIN3_PINREG PIND
#define PIN4_PINREG PIND
#define PIN5_PINREG PIND
#define PIN6_PINREG PIND
#define PIN7_PINREG PIND
#define PIN8_PINREG PINB
#define PIN9_PINREG PINB
```

```
#define PIN10_PINREG PINB
#define PIN11_PINREG PINB
#define PIN12_PINREG PINB
#define PIN13_PINREG PINB
#define PIN14_PINREG PINC
#define PIN15_PINREG PINC
#define PIN16_PINREG PINC
#define PIN17_PINREG PINC
#define PIN18_PINREG PINC
#define PIN19_PINREG PINC
#endif // defined(__AVR_ATmega1280__) || defined(__AVR_ATmega2560__)
#endif // ArduinoPins_h
```

WaveUtil.h[24]

```
#ifndef WaveUtil_h
#define WaveUtil_h
#include <avr/pgmspace.h>

// ladayada uses this name
#define putstring(x) SerialPrint_P(PSTR(x))

// ladayada uses this name
#define putstring_nl(x) SerialPrintln_P(PSTR(x))

/** Store and print a string in flash memory.*/
#define PgmPrint(x) SerialPrint_P(PSTR(x))

/** Store and print a string in flash memory followed by a CR/LF.*/
#define PgmPrintln(x) SerialPrintln_P(PSTR(x))

int FreeRam(void);
void SerialPrint_P(PGM_P str);
void SerialPrintln_P(PGM_P str);
#endif //WaveUtil_h
```

WaveUtil.cpp[24]

```
#if ARDUINO < 100
#include <WProgram.h>
#else  // ARDUINO
#include <Arduino.h>
#endif  // ARDUINO
#include <WaveUtil.h>
//------------------------------------------------------------------------------
/** Return the number of bytes currently free in RAM. */
int FreeRam(void) {
  extern int  __bss_end;
  extern int  *__brkval;
  int free_memory;
  if((int)__brkval == 0) {
    // if no heap use from end of bss section
    free_memory = ((int)&free_memory) - ((int)&__bss_end);
  }
```

```
  else {
    // use from top of stack to heap
    free_memory = ((int)&free_memory) - ((int)__brkval);
  }
  return free_memory;
}
//------------------------------------------------------------------------------
/**
 * %Print a string in flash memory to the serial port.
 *
 * \param[in] str Pointer to string stored in flash memory.
 */
void SerialPrint_P(PGM_P str) {
  for (uint8_t c; (c = pgm_read_byte(str)); str++) Serial.write(c);
}
//------------------------------------------------------------------------------
/**
 * %Print a string in flash memory followed by a CR/LF.
 *
 * \param[in] str Pointer to string stored in flash memory.
 */
void SerialPrintln_P(PGM_P str) {
  SerialPrint_P(str);
  Serial.println();
}
```

McpDac.h[24]

```
/* Arduino WaveHC Library
 * Copyright (C) 2009 by William Greiman
 *
 * This file is part of the Arduino WaveHC Library
 *
 * This Library is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This Library is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with the Arduino WaveHC Library.  If not, see
 * <http://www.gnu.org/licenses/>.
 */
/**
 *  Macros and inline functions for MCP4921 DAC
 */
#ifndef mcpDac_h
#define mcpDac_h

#include <avr/io.h>
#include <WavePinDefs.h>
```

```
//--------------------------------------------------------------------------------
#define mcpDacCsLow() MCP_DAC_CS_PORT &= ~_BV(MCP_DAC_CS_BIT)
#define mcpDacCsHigh() MCP_DAC_CS_PORT |= _BV(MCP_DAC_CS_BIT)

#define mcpDacSckLow() MCP_DAC_SCK_PORT &= ~_BV(MCP_DAC_SCK_BIT)
#define mcpDacSckHigh() MCP_DAC_SCK_PORT |= _BV(MCP_DAC_SCK_BIT)
#define mcpDacSckPulse() {mcpDacSckHigh();mcpDacSckLow();}

#define mcpDacSdiLow() MCP_DAC_SDI_PORT &= ~_BV(MCP_DAC_SDI_BIT)
#define mcpDacSdiHigh() MCP_DAC_SDI_PORT |= _BV(MCP_DAC_SDI_BIT)
#define mcpDacSdiSet(v) if(v){mcpDacSdiHigh();}else{mcpDacSdiLow();}

// send bit b of d
#define mcpDacSendBit(d, b) {mcpDacSdiSet(d&_BV(b));mcpDacSckPulse();}

//--------------------------------------------------------------------------------
// init dac I/O ports
inline void mcpDacInit(void) {
  // set all to output mode
  MCP_DAC_CS_DDR |= _BV(MCP_DAC_CS_BIT);
  MCP_DAC_SCK_DDR |= _BV(MCP_DAC_SCK_BIT);
  MCP_DAC_SDI_DDR |= _BV(MCP_DAC_SDI_BIT);
  // chip select high
  mcpDacCsHigh();

#if USE_MCP_DAC_LDAC
  // LDAC low always - use unbuffered mode
  MCP_DAC_LDAC_DDR |= _BV(MCP_DAC_LDAC_BIT);
  MCP_DAC_LDAC_PORT &= ~_BV(MCP_DAC_LDAC_BIT);
#endif // USE_MCP_DAC_LDAC
}
//--------------------------------------------------------------------------------
// send 12 bits to dac
// trusted compiler to optimize and it does
// csLow to csHigh takes 8 - 9 usec on a 16 MHz Arduino
inline void mcpDacSend(uint16_t data) {
  mcpDacCsLow();
  // send DAC config bits
  mcpDacSdiLow();
  mcpDacSckPulse();  // DAC A
  mcpDacSckPulse();  // unbuffered
  mcpDacSdiHigh();
  mcpDacSckPulse();  // 1X gain
  mcpDacSckPulse();  // no SHDN
  // send 12 data bits
  mcpDacSendBit(data, 11);
  mcpDacSendBit(data, 10);
  mcpDacSendBit(data,  9);
  mcpDacSendBit(data,  8);
  mcpDacSendBit(data,  7);
  mcpDacSendBit(data,  6);
  mcpDacSendBit(data,  5);
  mcpDacSendBit(data,  4);
  mcpDacSendBit(data,  3);
  mcpDacSendBit(data,  2);
  mcpDacSendBit(data,  1);
```

```
  mcpDacSendBit(data,  0);
  mcpDacCsHigh();
}

#endif //mcpDac_h
```

FatStructs.h[24]

```
#ifndef FatStructs_h
#define FatStructs_h
/**
 * \file
 * FAT file structures
 */
/*
 * mostly from Microsoft document fatgen103.doc
 * http://www.microsoft.com/whdc/system/platform/firmware/fatgen.mspx
 */
//------------------------------------------------------------------------------
        /** Value for byte 510 of boot block or MBR */
#define BOOTSIG0        0X55
       /** Value for byte 511 of boot block or MBR */
#define BOOTSIG1        0XAA
//------------------------------------------------------------------------------
/**
 * \struct partitionTable
 * \brief MBR partition table entry
 *
 * A partition table entry for a MBR formatted storage device.
 * The MBR partition table has four entries.
 */
struct partitionTable {
          /**
           * Boot Indicator . Indicates whether the volume is the active
           * partition.  Legal values include: 0X00. Do not use for booting.
           * 0X80 Active partition.
           */
  uint8_t  boot;
          /**
            * Head part of Cylinder-head-sector address of the first block in
            * the partition. Legal values are 0-255. Only used in old PC BIOS.
            */
  uint8_t  beginHead;
          /**
           * Sector part of Cylinder-head-sector address of the first block in
           * the partition. Legal values are 1-63. Only used in old PC BIOS.
           */
  unsigned beginSector : 6;
           /** High bits cylinder for first block in partition. */
  unsigned beginCylinderHigh : 2;
          /**
           * Combine beginCylinderLow with beginCylinderHigh. Legal values
           * are 0-1023.  Only used in old PC BIOS.
           */
```

```
  uint8_t  beginCylinderLow;
          /**
           * Partition type. See defines that begin with PART_TYPE_ for
           * some Microsoft partition types.
           */
  uint8_t  type;
          /**
           * head part of cylinder-head-sector address of the last sector in the
           * partition.  Legal values are 0-255. Only used in old PC BIOS.
           */
  uint8_t  endHead;
          /**
           * Sector part of cylinder-head-sector address of the last sector in
           * the partition.  Legal values are 1-63. Only used in old PC BIOS.
           */
  unsigned endSector : 6;
           /** High bits of end cylinder */
  unsigned endCylinderHigh : 2;
          /**
           * Combine endCylinderLow with endCylinderHigh. Legal values
           * are 0-1023.  Only used in old PC BIOS.
           */
  uint8_t  endCylinderLow;
           /** Logical block address of the first block in the partition. */
  uint32_t firstSector;
           /** Length of the partition, in blocks. */
  uint32_t totalSectors;
};
/** Type name for partitionTable */
typedef struct partitionTable part_t;
//------------------------------------------------------------------------------
/**
 * \struct masterBootRecord
 *
 * \brief Master Boot Record
 *
 * The first block of a storage device that is formatted with a MBR.
 */
struct masterBootRecord {
           /** Code Area for master boot program. */
  uint8_t  codeArea[440];
           /** Optional WindowsNT disk signature. May contain more boot code. */
  uint32_t diskSignature;
           /** Usually zero but may be more boot code. */
  uint16_t usuallyZero;
           /** Partition tables. */
  part_t   part[4];
           /** First MBR signature byte. Must be 0X55 */
  uint8_t  mbrSig0;
           /** Second MBR signature byte. Must be 0XAA */
  uint8_t  mbrSig1;
};
/** Type name for masterBootRecord */
typedef struct masterBootRecord mbr_t;
//------------------------------------------------------------------------------
/**
 * \struct biosParmBlock
 *
```

```
 * \brief BIOS parameter block
 *
 *  The BIOS parameter block describes the physical layout of a FAT volume.
 */
struct biosParmBlock{
          /**
           * Count of bytes per sector. This value may take on only the
           * following values: 512, 1024, 2048 or 4096
           */
  uint16_t bytesPerSector;
          /**
           * Number of sectors per allocation unit. This value must be a
           * power of 2 that is greater than 0. The legal values are
           * 1, 2, 4, 8, 16, 32, 64, and 128.
           */
  uint8_t  sectorsPerCluster;
          /**
           * Number of sectors before the first FAT.
           * This value must not be zero.
           */
  uint16_t reservedSectorCount;
          /** The count of FAT data structures on the volume. This field should
           *  always contain the value 2 for any FAT volume of any type.
           */
  uint8_t  fatCount;
          /**
          * For FAT12 and FAT16 volumes, this field contains the count of
          * 32-byte directory entries in the root directory. For FAT32 volumes,
          * this field must be set to 0. For FAT12 and FAT16 volumes, this
          * value should always specify a count that when multiplied by 32
          * results in a multiple of bytesPerSector.  FAT16 volumes should
          * use the value 512.
          */
  uint16_t rootDirEntryCount;
          /**
           * This field is the old 16-bit total count of sectors on the volume.
           * This count includes the count of all sectors in all four regions
           * of the volume. This field can be 0; if it is 0, then totalSectors32
           * must be nonzero.  For FAT32 volumes, this field must be 0. For
           * FAT12 and FAT16 volumes, this field contains the sector count, and
           * totalSectors32 is 0 if the total sector count fits
           * (is less than 0x10000).
           */
  uint16_t totalSectors16;
          /**
           * This dates back to the old MS-DOS 1.x media determination and is
           * no longer usually used for anything.  0xF8 is the standard value
           * for fixed (nonremovable) media. For removable media, 0xF0 is
           * frequently used. Legal values are 0xF0 or 0xF8-0xFF.
           */
  uint8_t  mediaType;
          /**
           * Count of sectors occupied by one FAT on FAT12/FAT16 volumes.
           * On FAT32 volumes this field must be 0, and sectorsPerFat32
           * contains the FAT size count.
           */
  uint16_t sectorsPerFat16;
           /** Sectors per track for interrupt 0x13. Not used otherwise. */
```

```
  uint16_t sectorsPerTrtack;
           /** Number of heads for interrupt 0x13.  Not used otherwise. */
  uint16_t headCount;
          /**
           * Count of hidden sectors preceding the partition that contains this
           * FAT volume. This field is generally only relevant for media
           *  visible on interrupt 0x13.
           */
  uint32_t hidddenSectors;
          /**
           * This field is the new 32-bit total count of sectors on the volume.
           * This count includes the count of all sectors in all four regions
           * of the volume.  This field can be 0; if it is 0, then
           * totalSectors16 must be nonzero.
           */
  uint32_t totalSectors32;
          /**
           * Count of sectors occupied by one FAT on FAT32 volumes.
           */
  uint32_t sectorsPerFat32;
          /**
           * This field is only defined for FAT32 media and does not exist on
           * FAT12 and FAT16 media.
           * Bits 0-3 -- Zero-based number of active FAT.
           *             Only valid if mirroring is disabled.
           * Bits 4-6 -- Reserved.
           * Bit 7   -- 0 means the FAT is mirrored at runtime into all FATs.
                *        -- 1 means only one FAT is active; it is the one
referenced in bits 0-3.
           * Bits 8-15      -- Reserved.
           */
  uint16_t fat32Flags;
          /**
           * FAT32 version. High byte is major revision number.
           * Low byte is minor revision number. Only 0.0 define.
           */
  uint16_t fat32Version;
          /**
           * Cluster number of the first cluster of the root directory for FAT32.
           * This usually 2 but not required to be 2.
           */
  uint32_t fat32RootCluster;
          /**
           * Sector number of FSINFO structure in the reserved area of the
           * FAT32 volume. Usually 1.
           */
  uint16_t fat32FSInfo;
          /**
           * If nonzero, indicates the sector number in the reserved area
           * of the volume of a copy of the boot record. Usually 6.
           * No value other than 6 is recommended.
           */
  uint16_t fat32BackBootBlock;
          /**
           * Reserved for future expansion. Code that formats FAT32 volumes
           * should always set all of the bytes of this field to 0.
           */
  uint8_t  fat32Reserved[12];
```

```
};
/** Type name for biosParmBlock */
typedef struct biosParmBlock bpb_t;
//------------------------------------------------------------------------------
/**
 * \struct fat32BootSector
 *
 * \brief Boot sector for a FAT16 or FAT32 volume.
 *
 */
struct fat32BootSector {
          /** X86 jmp to boot program */
  uint8_t  jmpToBootCode[3];
          /** informational only - don't depend on it */
  char     oemName[8];
          /** BIOS Parameter Block */
  bpb_t    bpb;
          /** for int0x13 use value 0X80 for hard drive */
  uint8_t  driveNumber;
          /**used by Windows NT - should be zero for FAT */
  uint8_t  reserved1;
          /** 0X29 if next three fields are valid */
  uint8_t  bootSignature;
          /** usually generated by combining date and time */
  uint32_t volumeSerialNumber;
          /** should match volume label in root dir */
  char     volumeLabel[11];
          /** informational only - don't depend on it */
  char     fileSystemType[8];
          /** X86 boot code */
  uint8_t  bootCode[420];
          /** must be 0X55 */
  uint8_t  bootSectorSig0;
          /** must be 0XAA */
  uint8_t  bootSectorSig1;
};
//------------------------------------------------------------------------------
// End Of Chain values for FAT entries
        /** Minimum value for FAT16 EOC.  Use to test for EOC. */
#define FAT16EOC_MIN 0XFFF8
        /** Minimum value for FAT32 EOC.  Use to test for EOC. */
#define FAT32EOC_MIN 0X0FFFFFF8
        /** FAT16 end of chain value used by Microsoft. */
#define FAT16EOC 0XFFFF
        /** FAT32 end of chain value used by Microsoft. */
#define FAT32EOC 0X0FFFFFFF
        /** Mask a for FAT32 entry. Entries are 28 bits. */
#define FAT32MASK 0X0FFFFFFF

/** Type name for fat32BootSector */
typedef struct fat32BootSector fbs_t;
//------------------------------------------------------------------------------
/**
 * \struct directoryEntry
 * \brief FAT short directory entry
 *
 * Short means short 8.3 name, not the entry size.
 *
```

```
 * Date Format. A FAT directory entry date stamp is a 16-bit field that is
 * basically a date relative to the MS-DOS epoch of 01/01/1980. Here is the
 * format (bit 0 is the LSB of the 16-bit word, bit 15 is the MSB of the
 * 16-bit word):
 *
 * Bits 9-15: Count of years from 1980, valid value range 0-127
 * inclusive (1980-2107).
 *
 * Bits 5-8: Month of year, 1 = January, valid value range 1-12 inclusive.
 *
 * Bits 0-4: Day of month, valid value range 1-31 inclusive.
 *
 * Time Format. A FAT directory entry time stamp is a 16-bit field that has
 * a granularity of 2 seconds. Here is the format (bit 0 is the LSB of the
 * 16-bit word, bit 15 is the MSB of the 16-bit word).
 *
 * Bits 11-15: Hours, valid value range 0-23 inclusive.
 *
 * Bits 5-10: Minutes, valid value range 0-59 inclusive.
 *
 * Bits 0-4: 2-second count, valid value range 0-29 inclusive (0 - 58 seconds).
 *
 * The valid time range is from Midnight 00:00:00 to 23:59:58.
 */
struct directoryEntry {
          /**
           * Short 8.3 name.
           * The first eight bytes contain the file name with blank fill.
           * The last three bytes contain the file extension with blank fill.
           */
  uint8_t  name[11];
         /** Entry attributes.
          *
          * The upper two bits of the attribute byte are reserved and should
          * always be set to 0 when a file is created and never modified or
          * looked at after that.  See defines that begin with DIR_ATT_.
          */
  uint8_t  attributes;
         /**
          * Reserved for use by Windows NT. Set value to 0 when a file is
          * created and never modify or look at it after that.
          */
  uint8_t  reservedNT;
         /**
          * The granularity of the seconds part of creationTime is 2 seconds
          * so this field is a count of tenths of a second and its valid
          * value range is 0-199 inclusive. (WHG note - seems to be hundredths)
          */
  uint8_t  creationTimeTenths;
          /** Time file was created. */
  uint16_t creationTime;
          /** Date file was created. */
  uint16_t creationDate;
         /**
          * Last access date. Note that there is no last access time, only
          * a date.  This is the date of last read or write. In the case of
          * a write, this should be set to the same date as lastWriteDate.
          */
```

```
  uint16_t lastAccessDate;
          /**
           * High word of this entry's first cluster number (always 0 for a
           * FAT12 or FAT16 volume).
           */
  uint16_t firstClusterHigh;
           /** Time of last write. File creation is considered a write. */
  uint16_t lastWriteTime;
           /** Date of last write. File creation is considered a write. */
  uint16_t lastWriteDate;
           /** Low word of this entry's first cluster number. */
  uint16_t firstClusterLow;
           /** 32-bit unsigned holding this file's size in bytes. */
  uint32_t fileSize;
};
//------------------------------------------------------------------------------
// Macros for directory entries
//
        /** Type name for directoryEntry */
typedef struct directoryEntry dir_t;
        /** escape for name[0] = 0XE5 */
#define DIR_NAME_0XE5          0X05
        /** name[0] value for entry that is free after being "deleted" */
#define DIR_NAME_DELETED       0XE5
        /** name[0] value for entry that is free and no allocated entries follow
*/
#define DIR_NAME_FREE          0X00
        /** file is read-only */
#define DIR_ATT_READ_ONLY      0X01
        /** File should hidden in directory listings */
#define DIR_ATT_HIDDEN         0X02
        /** Entry is for a system file */
#define DIR_ATT_SYSTEM         0X04
        /** Directory entry contains the volume label */
#define DIR_ATT_VOLUME_ID      0X08
        /** Entry is for a directory */
#define DIR_ATT_DIRECTORY      0X10
       /** Old DOS archive bit for backup support */
#define DIR_ATT_ARCHIVE        0X20
       /** Test value for long name entry.  Test is
           d->attributes & DIR_ATT_LONG_NAME_MASK) == DIR_ATT_LONG_NAME. */
#define DIR_ATT_LONG_NAME      0X0F
        /** Test mask for long name entry */
#define DIR_ATT_LONG_NAME_MASK 0X3F
        /** defined attribute bits */
#define DIR_ATT_DEFINED_BITS   0X3F
        /** Directory entry is part of a long name */
#define DIR_IS_LONG_NAME(dir)\
           (((dir).attributes & DIR_ATT_LONG_NAME_MASK) == DIR_ATT_LONG_NAME)
        /** Mask for file/subdirectory tests */
#define DIR_ATT_FILE_TYPE_MASK (DIR_ATT_VOLUME_ID | DIR_ATT_DIRECTORY)
        /** Directory entry is for a file */
#define DIR_IS_FILE(dir) (((dir).attributes & DIR_ATT_FILE_TYPE_MASK) == 0)
        /** Directory entry is for a subdirectory */
#define DIR_IS_SUBDIR(dir)\
            (((dir).attributes & DIR_ATT_FILE_TYPE_MASK) == DIR_ATT_DIRECTORY)
        /** Directory entry is for a file or subdirectory */
#define DIR_IS_FILE_OR_SUBDIR(dir) (((dir).attributes & DIR_ATT_VOLUME_ID) == 0)
```

```
#endif //FatStructs_h
```

FatReader.h[24]

```
/* Arduino FatReader Library
 * Copyright (C) 2009 by William Greiman
 *
 * This file is part of the Arduino FatReader Library
 *
 * This Library is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This Library is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.

 * You should have received a copy of the GNU General Public License
 * along with the Arduino FatReader Library.  If not, see
 * <http://www.gnu.org/licenses/>.
 */
#ifndef FatReader_h
#define FatReader_h
#include <SdReader.h>
#include <FatStructs.h>

// flags for ls()
/** ls() flag to print modify date */
#define LS_FLAG_FRAGMENTED 1
/** ls() flag to print file size */
#define LS_SIZE 2
/** ls() flag for recursive list of subdirectories */
#define LS_R 4

// offsets for structures used in volume init
/** Offset to BIOS Parameter Block in FAT Boot Sector */
#define BPB_OFFSET 11
/** Byte count for part of BIOS Parameter Block to be read by init() */
#define BPB_COUNT 37
/** offset to partition table in mbr */
#define PART_OFFSET (512-64-2)

// format dir.name into name[13] as standard 8.3 string
void dirName(dir_t &dir, char name[]);
// Print name field of dir_t struct in 8.3 format
void printEntryName(dir_t &dir);
//------------------------------------------------------------------------------
/** \class FatVolume
 * \brief FatVolume provides access to FAT volumes.
 */
class FatVolume {
  /** Allow FatReader access to FatVolume private data. */
  friend class FatReader;
```

```
  uint8_t blocksPerCluster_;
  uint32_t blocksPerFat_;
  uint32_t clusterCount_;
  uint32_t dataStartBlock_;
  uint8_t fatCount_;
  uint32_t fatStartBlock_;
  uint8_t fatType_;
  SdReader *rawDevice_;
  uint16_t rootDirEntryCount_;
  uint32_t rootDirStart_;
  uint32_t totalBlocks_;
  uint8_t chainIsContiguous(uint32_t cluster);
  uint32_t chainSize(uint32_t cluster);
  uint8_t isEOC(uint32_t cluster)
    {return  cluster >= (fatType_ == 16 ? FAT16EOC_MIN : FAT32EOC_MIN);}
  uint32_t nextCluster(uint32_t cluster);
  uint8_t rawRead(uint32_t block, uint16_t offset, uint8_t *dst, uint16_t count)
    {return rawDevice_->readData(block, offset, dst, count);}
  uint8_t validCluster(uint32_t cluster) {
   return (1 < cluster && cluster < (clusterCount_ + 2));}
public:
/** Create an instance of FatVolume */
  FatVolume(void) : fatType_(0){}
  /**
   * Initialize a FAT volume.  Try partition one first then try super
   * floppy format.
   *
   * \param[in] dev The SdReader where the volume is located.
   *
   * \return The value one, true, is returned for success and
   * the value zero, false, is returned for failure.  Reasons for
   * failure include not finding a valid partition, not finding a valid
   * FAT file system or an I/O error.
   */
  uint8_t init(SdReader &dev) { return init(dev, 1) ? 1 : init(dev, 0);}
  uint8_t init(SdReader &dev, uint8_t part);

  // inline functions that return volume info
  /** \return The volume's cluster size in blocks. */
  uint8_t blocksPerCluster(void) {return blocksPerCluster_;}
  /** \return The number of blocks in one FAT. */
  uint32_t blocksPerFat(void)  {return blocksPerFat_;}
  /** \return The total number of clusters in the volume. */
  uint32_t clusterCount(void) {return clusterCount_;}
  /** \return The logical block number for the start of file data. */
  uint32_t dataStartBlock(void) {return dataStartBlock_;}
  /** \return The number of FAT structures on the volume. */
  uint8_t fatCount(void) {return fatCount_;}
  /** \return The logical block number for the start of the first FAT. */
  uint32_t fatStartBlock(void) {return fatStartBlock_;}
  /** \return The FAT type of the volume. Values are 12, 16 or 32. */
  uint8_t fatType(void) {return fatType_;}
  /** Raw device for this volume */
  SdReader *rawDevice(void) {return rawDevice_;}
  /** \return The number of entries in the root directory for FAT16 volumes. */
  uint32_t rootDirEntryCount(void) {return rootDirEntryCount_;}
  /** \return The logical block number for the start of the root directory
       on FAT16 volumes or the first cluster number on FAT32 volumes. */
```

```
  uint32_t rootDirStart(void) {return rootDirStart_;}
  /** \return The total number of blocks in the volume. */
  uint32_t totalBlocks(void) {return totalBlocks_;}
};
//------------------------------------------------------------------------------
/** \class FatReader
 * \brief FatReader implements a minimal FAT16/FAT32 file reader class.
 */
class FatReader {
// values for type_
/** File is contiguous file */
#define FILE_IS_CONTIGUOUS 0X08
/** File type mask */
#define FILE_TYPE_MASK 0X07
/** This FatReader has not been opened. */
#define FILE_TYPE_CLOSED 0X00
/** FatReader for a file */
#define FILE_TYPE_NORMAL 0X01
/** FatReader for a FAT16 root directory */
#define FILE_TYPE_ROOT16 0X02
/** FatReader for a FAT32 root directory */
#define FILE_TYPE_ROOT32 0X03
/** FatReader for a subdirectory */
#define FILE_TYPE_SUBDIR 0X04
/** Test value for directory type */
#define FILE_TYPE_MIN_DIR FILE_TYPE_ROOT16
  uint8_t type_;
  uint32_t fileSize_;
  uint32_t readCluster_;
  uint32_t readPosition_;
  uint32_t firstCluster_;
  FatVolume *vol_;
  int16_t readBlockData(uint8_t *dst, uint16_t count);
  void lsR(dir_t &d, uint8_t flags, uint8_t indent);
public:
/** Create an instance of FatReader. */
  FatReader(void) : type_(FILE_TYPE_CLOSED) {}
  void ls(uint8_t flags = 0);
  uint8_t openRoot(FatVolume &vol);
  uint8_t open(FatVolume &vol, dir_t &dir);
  uint8_t open(FatReader &dir, char *name);
  uint8_t open(FatReader &dir, uint16_t index);
  void optimizeContiguous(void);
  int16_t read(void *buf, uint16_t count);
  int8_t readDir(dir_t &dir);
  void rewind(void);
  uint8_t seekCur(uint32_t pos);
  //inline functions
  /** Close this instance of FatReader. */
  void close(void) {type_ = FILE_TYPE_CLOSED;}
  /** \return The total number of bytes in a file or directory. */
  uint32_t fileSize(void) {return fileSize_;}
  /**
   * Type of this FatReader.  You should use isFile() or isDir()
   * instead of type() if possible.
   *
   * \return The file or directory type.
   */
```

```
  uint8_t fileType(void) {return type_ & FILE_TYPE_MASK;}
  /** \return The first cluster number for a file or directory. */
  uint32_t firstCluster(void) {return firstCluster_;}
  /**
   * \return True if the bit for optimized reads is set.
   * See optimizeContiguous(). */
  uint8_t isContiguous(void) {return type_ & FILE_IS_CONTIGUOUS;}
  /** \return True if this is a FatReader for a directory else false */
  uint8_t isDir(void) {return fileType() >= FILE_TYPE_MIN_DIR;}
  /** \return True if this is a FatReader for a file else false */
  uint8_t isFile(void) {return fileType() == FILE_TYPE_NORMAL;}
  /** \return True if FatReader is for an open file/directory else false */
  uint8_t isOpen(void) {return fileType() != FILE_TYPE_CLOSED;}
  /** \return The current cluster number for a file or directory. */
  uint32_t readCluster(void) {return readCluster_;}
  /** \return The read position for a file or directory. */
  uint32_t readPosition(void) {return readPosition_;}
  /**
   * Set the read position for a file or directory to \a pos.
   *
   * \param[in] pos The new read position in bytes from the beginning
   * of the file.
   *
   * \return The value one, true, is returned for success and
   * the value zero, false, is returned for failure.
   */
  uint8_t seekSet(uint32_t pos) {
    if (pos >= readPosition_) return seekCur(pos - readPosition_);
    rewind(); return seekCur(pos);}
  /** Parent volume */
  FatVolume *volume(void) {return vol_;}
};
#endif//FatReader_h
```

FatReader.cpp[24]

```
/* Arduino FatReader Library
 * Copyright (C) 2009 by William Greiman
 *
 * This file is part of the Arduino FatReader Library
 *
 * This Library is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This Library is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.

 * You should have received a copy of the GNU General Public License
 * along with the Arduino FatReader Library.  If not, see
 * <http://www.gnu.org/licenses/>.
 */
#include <string.h>
```

```
#if ARDUINO < 100
#include <WProgram.h>
#else  // ARDUINO
#include <Arduino.h>
#endif  // ARDUINO
#include <FatReader.h>
//------------------------------------------------------------------------------
/**
 *  Format the name field of the dir_t struct \a dir into the 13 byte array
 *  \a name in the standard 8.3 short name format.
 */
void dirName(dir_t &dir, char name[]) {
  uint8_t j = 0;
  for (uint8_t i = 0; i < 11; i++) {
    if (dir.name[i] == ' ')continue;
    if (i == 8) name[j++] = '.';
    name[j++] = dir.name[i];
  }
  name[j] = 0;
}
//------------------------------------------------------------------------------
/**
 * Print the name field of a dir_t structure in 8.3 format.
 * Append a '/' if it is a subdirectory.
 *
 */
void printEntryName(dir_t &dir) {
  for (uint8_t i = 0; i < 11; i++) {
    if (dir.name[i] == ' ')continue;
    if (i == 8) Serial.write('.');
    Serial.write(dir.name[i]);
  }
  if (DIR_IS_SUBDIR(dir)) {
    // indicate subdirectory
    Serial.write('/');
  }
}
//------------------------------------------------------------------------------
/**
 *  List file in a directory
 *
 */
void FatReader::ls(uint8_t flags) {
  dir_t d;
  if (isDir()) lsR(d, flags, 0);
}
//------------------------------------------------------------------------------
// recursive part of ls()
void FatReader::lsR(dir_t &d, uint8_t flags, uint8_t indent) {
  while (readDir(d) > 0) {

    // print any indent spaces
    for (int8_t i = 0; i < indent; i++) {
      Serial.write(' ');
    }
    printEntryName(d);

    if (DIR_IS_SUBDIR(d)) {
```

```
      Serial.println();
      // recursive call if LS_R
      if (flags & LS_R) {
        FatReader s;
        if (s.open(*vol_, d)) {
          s.lsR(d, flags, indent + 2);
        }
      }
    }
    else {
      if (flags & LS_FLAG_FRAGMENTED) {
        uint32_t c = (uint32_t)d.firstClusterHigh << 16;
        c |= d.firstClusterLow;

        // fragmented if has clusters and not contiguous
        char f = c && !vol_->chainIsContiguous(c) ? '*' : ' ';
        Serial.write(' ');
        Serial.write(f);
      }
      if (flags & LS_SIZE) {
        Serial.write(' ');
        Serial.print(d.fileSize);
      }
      Serial.println();
    }
  }
}
//------------------------------------------------------------------------------
/** return the next cluster in a chain */
uint32_t FatVolume::nextCluster(uint32_t cluster) {
  if (!validCluster(cluster)) {
    return 0;
  }
  if (fatType_ == 32) {
    uint32_t next;
    uint32_t block = fatStartBlock_ + (cluster >> 7);
    uint16_t offset = 0X1FF & (cluster << 2);
    if (!rawRead(block, offset, (uint8_t *)&next, 4)) {
      return 0;
    }
    return next;
  }
  if (fatType_ == 16) {
    uint16_t next;
    uint32_t block = fatStartBlock_ + (cluster >> 8);
    uint16_t offset = 0X1FF & (cluster << 1);
    if (!rawRead(block, offset, (uint8_t *)&next, 2)) {
      return 0;
    }
    return next;
  }
  return 0;
}
//------------------------------------------------------------------------------
/**
 * Open a file or subdirectory by index.
 *
 * \param[in] dir An open FatReader instance for the directory.
```

```
 *
 * \param[in] index The \a index for a file or subdirectory in the
 * directory \a dir. \a index is the byte offset divided by 32 of
 * the directory entry for the file or subdirectory.
 *
 *  To determine the index for a file open it by name. The index for the
 *  file is then is: (dir.readPosition()/32 -1)
 *
 *
 * \return The value one, true, is returned for success and
 * the value zero, false, is returned for failure.
 * Reasons for failure include the FAT volume has not been initialized, \a dir
 * is not a directory, \a name is invalid, the file or subdirectory does not
 *  exist, or an I/O error occurred.
 */
uint8_t FatReader::open(FatReader &dir, uint16_t index) {
  dir_t d;

  // position directory file to entry
  if (!dir.seekSet(32UL*index)) return false;

  // read entry
  if (dir.read(&d, 32) != 32) return false;

  // must be a real file or directory
  if (!DIR_IS_FILE_OR_SUBDIR(d)
    || d.name[0] == DIR_NAME_FREE
    || d.name[0] == DIR_NAME_DELETED) {
    return false;
  }
  return open(*dir.volume(), d);
}
//------------------------------------------------------------------------------
/**
 * Open a file or subdirectory by name.
 *
 * \note The file or subdirectory, \a name, must be in the specified
 * directory, \a dir, and must have a DOS 8.3 name.
 *
 * \param[in] dir An open FatReader instance for the directory.
 *
 * \param[in] name A valid 8.3 DOS name for a file or subdirectory in the
 * directory \a dir.
 *
 * \return The value one, true, is returned for success and
 * the value zero, false, is returned for failure.
 * Reasons for failure include the FAT volume has not been initialized, \a dir
 * is not a directory, \a name is invalid, the file or subdirectory does not
 *  exist, or an I/O error occurred.
 */
uint8_t FatReader::open(FatReader &dir, char *name) {
  dir_t entry;
  char dname[13];

  dir.rewind();
  while(dir.readDir(entry) > 0) {
    dirName(entry, dname);
    if (strcasecmp(dname, name)) continue;
```

```
    return open(*(dir.vol_), entry);
  }
  return false;
}
//------------------------------------------------------------------------------
/**
 * Open a file or subdirectory by directory structure.
 *
 * \param[in] vol The FAT volume that contains the file or subdirectory.
 *
 * \param[in] dir The directory structure describing the file or subdirectory.
 *
 * \return The value one, true, is returned for success and
 * the value zero, false, is returned for failure.
 * Reasons for failure include the FAT volume, \a vol, has not been initialized,
 * \a vol is a FAT12 volume or \a dir is not a valid directory entry.
 */
uint8_t FatReader::open(FatVolume &vol, dir_t &dir) {
  if (vol.fatType() < 16) return false;
  if (dir.name[0] == 0 || dir.name[0] == DIR_NAME_DELETED) {
    return false;
  }
  firstCluster_ = (uint32_t)dir.firstClusterHigh << 16;
  firstCluster_ |= dir.firstClusterLow;
  if (DIR_IS_FILE(dir)) {
    type_ = FILE_TYPE_NORMAL;
    fileSize_ = dir.fileSize;
  }
  else if (DIR_IS_SUBDIR(dir)) {
    type_ = FILE_TYPE_SUBDIR;
    fileSize_ = vol.chainSize(firstCluster_);
  }
  else {
    return false;
  }
  vol_ = &vol;
  rewind();
  return true;
}
//------------------------------------------------------------------------------
/**
 * Open a volume's root directory.
 *
 * \param[in] vol The FAT volume containing the root directory to be opened.
 *
 * \return The value one, true, is returned for success and
 * the value zero, false, is returned for failure.
 * Reasons for failure include the FAT volume has not been initialized
 * or it a FAT12 volume.
 */
uint8_t FatReader::openRoot(FatVolume &vol) {
  if(vol.fatType() == 16) {
    type_ = FILE_TYPE_ROOT16;
    firstCluster_ = 0;
    fileSize_ = 32*vol.rootDirEntryCount();
  }
  else if (vol.fatType() == 32) {
    type_ = FILE_TYPE_ROOT32;
```

```
    firstCluster_ = vol.rootDirStart();
    fileSize_ = vol.chainSize(firstCluster_);
  }
  else {
    return false;
  }
  vol_ = &vol;
  rewind();
  return true;
}
//------------------------------------------------------------------------------
/**
 * Check for a contiguous file and enable optimized reads if the
 * file is contiguous.
 */
void FatReader::optimizeContiguous(void) {
  if (isOpen() && firstCluster_) {
    if (vol_->chainIsContiguous(firstCluster_)) {
      type_ |= FILE_IS_CONTIGUOUS;
    }
  }
}
//------------------------------------------------------------------------------
/**
 * Read data from a file at starting at the current read position.
 *
 * \param[out] buf Pointer to the location that will receive the data.
 *
 * \param[in] count Maximum number of bytes to read.
 *
 * \return For success read() returns the number of bytes read.
 * A value less than \a count, including zero, will be returned
 * if end of file is reached.
 * If an error occurs, read() returns -1.  Possible errors include
 * read() called before a file has been opened, corrupt file system
 * or an I/O error occurred.
 */
int16_t FatReader::read(void *buf, uint16_t count) {
  uint8_t *dst = (uint8_t *)buf;
  uint16_t nr = 0;
  int16_t n = 0;
  while (nr < count && (n = readBlockData(dst, count - nr)) > 0) {
    if (!seekCur(n)) return -1;
    dst += n;
    nr += n;
  }
  return n < 0 ? -1 : nr;
}
//------------------------------------------------------------------------------
// read maximum amount possible from current physical block
int16_t FatReader::readBlockData(uint8_t *dst, uint16_t count) {
  uint32_t block;
  uint16_t offset = readPosition_ & 0X1FF;
  if (count > (512 - offset)) count = 512 - offset;
  if (count > (fileSize_ - readPosition_)) count = fileSize_ - readPosition_;
  if (fileType() == FILE_TYPE_ROOT16) {
    block = vol_->rootDirStart() + (readPosition_ >> 9);
  }
```

```
  else {
    uint8_t bpc = vol_->blocksPerCluster();
    block = vol_->dataStartBlock() + (readCluster_ - 2)*bpc
                        + ((readPosition_ >> 9) & (bpc -1));
  }
  return vol_->rawRead(block, offset, dst, count) ? count : -1;
}
//------------------------------------------------------------------------------
/**
 * Read the next directory entry from a directory file.
 *
 * \param[out] dir The dir_t struct that will receive the data.
 *
 * \return For success readDir() returns the number of bytes read.
 * A value of zero will be returned if end of file is reached.
 * If an error occurs, readDir() returns -1.  Possible errors include
 * readDir() called before a directory has been opened, this is not
 * a directory file or an I/O error occurred.
 */
int8_t FatReader::readDir(dir_t &dir) {
  int8_t n;
  //if not a directory file return an error
  if (!isDir()) return -1;
  while ((n = read((uint8_t *)&dir, sizeof(dir_t))) == sizeof(dir_t)
       && dir.name[0] != DIR_NAME_FREE) {
    if (dir.name[0] == DIR_NAME_DELETED || dir.name[0] == '.') continue;
    if (DIR_IS_FILE(dir) || DIR_IS_SUBDIR(dir)) return n;
  }
  return n < 0 ? n : 0;
}
//------------------------------------------------------------------------------
/** Set read position to start of file */
void FatReader::rewind(void) {
  readCluster_ = firstCluster_;
  readPosition_ = 0;
}
/**
 * Set the read position for a file or directory to the current position plus
 * \a offset.
 *
 * \param[in] offset The amount to advance the read position.
 *
 * \return The value one, true, is returned for success and
 * the value zero, false, is returned for failure.
 */
uint8_t FatReader::seekCur(uint32_t offset) {

  uint32_t newPos = readPosition_ + offset;

  // can't position beyond end of file
  if (newPos > fileSize_) return false;

  // number of clusters forward
  uint32_t nc = (newPos >> 9)/vol_->blocksPerCluster()
                - (readPosition_ >> 9)/vol_->blocksPerCluster();

  // set new position - only corrupt file system can cause error now
  readPosition_ = newPos;
```

```
  // no clusters if FAT16 root
  if (fileType() == FILE_TYPE_ROOT16) return true;

  // don't need to read FAT if contiguous
  if (isContiguous()) {
    readCluster_ += nc;
    return true;
  }

  // read FAT chain while nc != 0
  while (nc-- != 0) {
    if (!(readCluster_ = vol_->nextCluster(readCluster_))) {
      return false;
    }
  }
  return true;
}
//------------------------------------------------------------------------------
/** check for contiguous chain */
uint8_t FatVolume::chainIsContiguous(uint32_t cluster) {
  uint32_t next;
  while((next = nextCluster(cluster))) {
    if (next != (cluster + 1)) {
      return isEOC(next);
    }
    cluster = next;
  }
  return false;
}
//------------------------------------------------------------------------------
/** return the number of bytes in a cluster chain */
uint32_t FatVolume::chainSize(uint32_t cluster) {
  uint32_t size = 0;
  while ((cluster = nextCluster(cluster))) {
    size += 512*blocksPerCluster_;
  }
  return size;
}
//------------------------------------------------------------------------------
/**
 *  Initialize a FAT volume.
 *
 * \param[in] dev The SD card where the volume is located.
 *
 * \param[in] part The partition to be used.  Legal values for \a part are
 * 1-4 to use the corresponding partition on a device formatted with
 * a MBR, Master Boot Record, or zero if the device is formatted as
 * a super floppy with the FAT boot sector in block zero.
 *
 * \return The value one, true, is returned for success and
 * the value zero, false, is returned for failure.  Reasons for
 * failure include not finding a valid partition, not finding a valid
 *  FAT file system in the specified partition or an I/O error.
 */
uint8_t FatVolume::init(SdReader &dev, uint8_t part) {
  uint8_t buf[BPB_COUNT];
  uint32_t volumeStartBlock = 0;
```

```
  rawDevice_ = &dev;
  // if part == 0 assume super floppy with FAT boot sector in block zero
  // if part > 0 assume mbr volume with partition table
  if (part) {
    if (part > 4) return false;

    if (!rawRead(volumeStartBlock, PART_OFFSET + 16*(part-1), buf, 16)) {
      return false;
    }
    part_t *part = (part_t *)buf;
    if ((part->boot & 0X7F) !=0  ||
      part->totalSectors < 100 ||
      part->firstSector == 0) {
      //not a valid partition
      return false;
    }
    volumeStartBlock = part->firstSector;
  }
  if (!rawRead(volumeStartBlock, BPB_OFFSET, buf, BPB_COUNT)) {
    return false;
  }
  bpb_t *bpb = (bpb_t *)buf;
  if (bpb->bytesPerSector != 512 ||
    bpb->fatCount == 0 ||
    bpb->reservedSectorCount == 0 ||
    bpb->sectorsPerCluster == 0 ||
    (bpb->sectorsPerCluster & (bpb->sectorsPerCluster - 1)) != 0) {
       // not valid FAT volume
      return false;
  }
  fatCount_ = bpb->fatCount;
  blocksPerCluster_ = bpb->sectorsPerCluster;
  blocksPerFat_ = bpb->sectorsPerFat16 ? bpb->sectorsPerFat16 :
bpb->sectorsPerFat32;
  rootDirEntryCount_ = bpb->rootDirEntryCount;
  fatStartBlock_ = volumeStartBlock + bpb->reservedSectorCount;
  rootDirStart_ = fatStartBlock_ + bpb->fatCount*blocksPerFat_;
  dataStartBlock_ = rootDirStart_ + ((32*bpb->rootDirEntryCount + 511)/512);
  totalBlocks_ = bpb->totalSectors16 ? bpb->totalSectors16 : bpb->totalSectors32;
  clusterCount_ = (totalBlocks_ - (dataStartBlock_ - volumeStartBlock))
                  /bpb->sectorsPerCluster;
  if (clusterCount_ < 4085) {
    fatType_ = 12;
  }
  else if (clusterCount_ < 65525) {
    fatType_ = 16;
  }
  else {
    rootDirStart_ = bpb->fat32RootCluster;
    fatType_ = 32;
  }
  return true;
}
```

XV. Appendix B: Detailed Sources of Purchased Materials

• Screws & nuts set. Available from:

http://item.taobao.com/item.htm?spm=a230r.1.14.8.n09Dho&id=37273502548&ns=1#detail

• Wood board. Available from:

http://detail.tmall.com/item.htm?spm=a230r.1.14.1.QRpK2K&id=19534326667&ad_id=&am_id=&cm_id=140105335569ed55e27b&pm_id=

• ABS plastic board. Available from:

http://detail.tmall.com/item.htm?id=35129477604&spm=a1z09.2.9.14.5VVZm7&_u=pah0pge019e

• ABS plastic board. Available from:

http://detail.tmall.com/item.htm?id=35129477604&spm=a1z09.2.9.14.5VVZm7&_u=pah0pge019e

• DC adjustable regulator for Arduino (reducing voltage). Available from:

http://detail.tmall.com/item.htm?id=19556881143&spm=a1z09.2.9.68.uQNcye&_u=pah0pge8e09

• DC adjustable regulator for Arduino (boosting voltage). Available from:

http://detail.tmall.com/item.htm?id=36530439809&spm=a1z09.2.9.87.uQNcye&_u=pah0pgebe11

• Switchers. Available from:

http://item.taobao.com/item.htm?spm=a1z09.2.9.105.uQNcye&id=16948110975&_u=pah0pge3ebc

• Mini amplifier. Available from:

http://item.taobao.com/item.htm?spm=a1z09.2.9.121.uQNcye&id=22241040699&_u=pah0pge62fa

• Fiberglass panel. Available from:

http://detail.tmall.com/item.htm?id=13573247268&spm=a1z09.2.9.91.s6Hpau&_u=pah0pge0ae0

• Fiberglass panel. Available from:

http://detail.tmall.com/item.htm?id=13573247268&spm=a1z09.2.9.91.s6Hpau&_u=pah0pge0ae0

• Resistor. Available from:

http://item.taobao.com/item.htm?spm=a1z09.2.9.67.s6Hpau&id=12723702428&_u=pah0pgeb75e

• Resistor. Available from:

http://item.taobao.com/item.htm?spm=a1z09.2.9.67.s6Hpau&id=12723702428&_u=pah0pgeb75e

• Photo resistor. Available from:

http://item.taobao.com/item.htm?spm=a1z09.2.9.78.s6Hpau&id=4441680237&_u=pah0pge2c4d

• Thimble. Available from:

http://item.taobao.com/item.htm?spm=a1z09.2.9.28.STqAaH&id=14422596825&_u=qah0pgeaeea

• Laser generator. Available from:

http://item.taobao.com/item.htm?spm=a1z09.2.9.126.k3HJWz&id=8500073719&_u=pah0pgeb61c

• Touch switcher. Available from:

http://item.taobao.com/item.htm?spm=a1z09.2.9.109.Lat47s&id=13065786552&_u=qah0pge339a

• Arduino. Available from:

http://detail.tmall.com/item.htm?spm=a230r.1.14.78.94aztO&id=20094329408

• Arduino expansion board. Available from:

http://item.taobao.com/item.htm?spm=a1z09.2.9.165.Lat47s&id=4096258356&_u=qah0pge03ae

• WAV module for Arduino board. Available from:

http://item.taobao.com/item.htm?spm=a1z09.2.9.143.Lat47s&id=8476641542&_u=qah0pgeaa1645

• SD card module for Arduino board. Available from:

http://item.taobao.com/item.htm?spm=a1z09.2.9.153.Lat47s&id=4096247566&_u=qah0pge446e

• SD card. Available from:

http://trade.taobao.com/trade/detail/tradeSnap.htm?spm=a1z09.2.9.179.k3HJWz&tradeID=70679
7540283862&snapShot=true

• Audio connector. Available from:

http://detail.tmall.com/item.htm?id=38627430726&spm=a1z09.2.9.198.F8nNZ1&_u=pah0pge56e3

• Adapter. Available from:

http://item.taobao.com/item.htm?spm=a1z09.2.9.159.Lat47s&id=4096297188&_u=qah0pgec15f

• Hot glue. Available from:

http://detail.tmall.com/item.htm?spm=a230r.1.14.47.8qxA3R&id=39164806069&_u=qah0pgedb77

• Photo sensitive transistor. Available from:

http://item.taobao.com/item.htm?spm=a1z09.2.9.254.F8nNZ1&id=10427884482&_u=pah0pge2fcf

• Hexagon copper socket. Available from:

http://detail.tmall.com/item.htm?id=18761858666&spm=a1z09.2.9.194.STqAaH&_u=qah0pge6d96

• Hexagon plastic socket. Available from:

http://item.taobao.com/item.htm?spm=a1z09.2.9.256.STqAaH&id=17647682021&_u=qah0pge88bf

• Hexagon plastic socket. Available from:

http://item.taobao.com/item.htm?spm=a1z09.2.9.267.STqAaH&id=17647682021&_u=qah0pge88bf

• Hexagon plastic socket. Available from:

http://item.taobao.com/item.htm?spm=a1z09.2.9.273.STqAaH&id=17647682021&_u=qah0pge88bf

• Electrical tape. Available from:

http://detail.tmall.com/item.htm?id=15532646345&spm=a1z09.2.9.188.ljrs9l&_u=qah0pge4d94

• Battery box. Available from:

http://item.taobao.com/item.htm?spm=a1z09.2.9.222.Lat47s&id=12668831445&_u=qah0pgef039

• Battery (GP2200mAh). Available from:

http://detail.tmall.com/item.htm?id=15099887509&spm=a1z09.2.9.48.AhOxtg&_u=qah0pge38df

• Battery. Available from:

http://detail.tmall.com/item.htm?spm=a230r.1.14.1.TJD8Lp&id=13157726931&ad_id=&am_id=&cm_id=140105335569ed55e27b&pm_id=

• DuPont line. Available from:

http://detail.tmall.com/item.htm?id=21642135234&spm=a1z09.2.9.252.ljrs9l&_u=qah0pge44a2

• DuPont line. Available from:

http://detail.tmall.com/item.htm?id=26648068250&spm=a1z09.2.9.264.ljrs9l&_u=qah0pge1898

• DuPont line. Available from:

http://detail.tmall.com/item.htm?id=19338838061&spm=a1z09.2.9.273.ljrs9l&_u=qah0pge3a52

• 502 glue. Available from:

http://item.taobao.com/item.htm?spm=a1z09.2.9.49.ljrs9l&id=15415565237&_u=qah0pge28e6

• Digital ranging sensor. Available from:

http://item.taobao.com/item.htm?spm=a1z09.2.9.209.Lat47s&id=14898953905&_u=qah0pge439e46

• Bread board. Available from:

http://detail.tmall.com/item.htm?spm=a230r.1.14.33.vgKkwv&id=16513870165&_u=qah0pge120b

• Laser generator. Available from:

http://item.taobao.com/item.htm?spm=a1z09.2.9.237.F8nNZ1&id=2634853413&_u=pah0pge643a

• Conductive rubber. Available from:

http://item.taobao.com/item.htm?spm=a230r.1.14.40.7vFgPd&id=36137728090&ns=1&_u=qah0pge4189#detail

• Ultrasonic ranging sensor. Available from:

http://item.taobao.com/item.htm?spm=a1z09.2.9.197.Lat47s&id=35140847142&_u=qah0pgef5c7

• Resistor. Available from:

http://item.taobao.com/item.htm?spm=a1z09.2.9.67.s6Hpau&id=12723702428&_u=pah0pgeb75e

• Resistor. Available from:

http://item.taobao.com/item.htm?spm=a1z09.2.9.67.s6Hpau&id=12723702428&_u=pah0pgeb75e

• Resistor. Available from:

http://item.taobao.com/item.htm?spm=a1z09.2.9.67.s6Hpau&id=12723702428&_u=pah0pgeb75e

• Resistor. Available from:

http://item.taobao.com/item.htm?spm=a1z09.2.9.67.s6Hpau&id=12723702428&_u=pah0pgeb75e

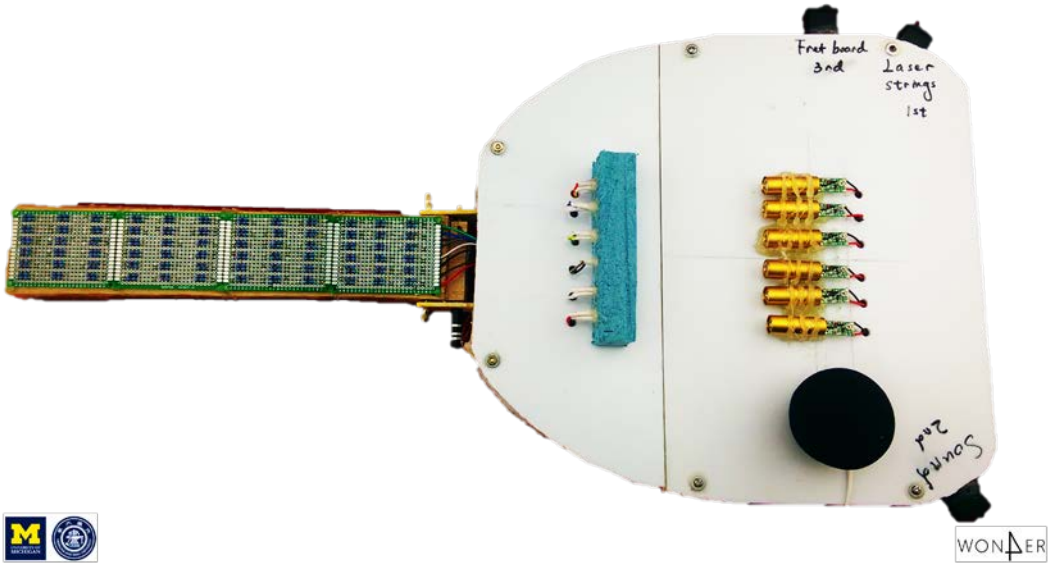
• Spray paint. Available from:

http://item.taobao.com/item.htm?spm=a1z09.2.9.239.STqAaH&id=39043844225&_u=qah0pgee00e

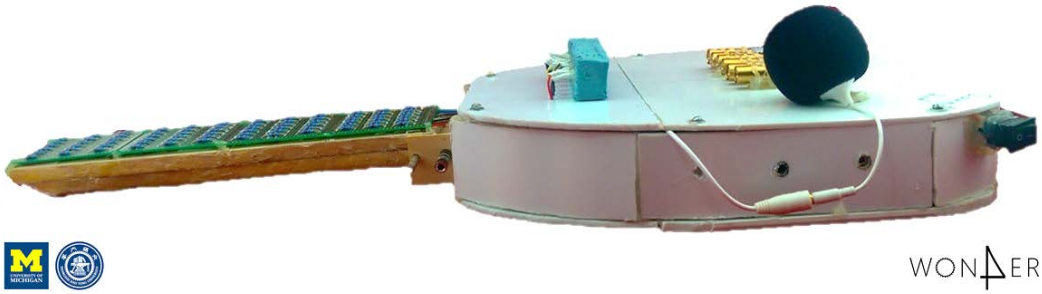
XVI. Appendix C: Graphs and Videos

Graphs

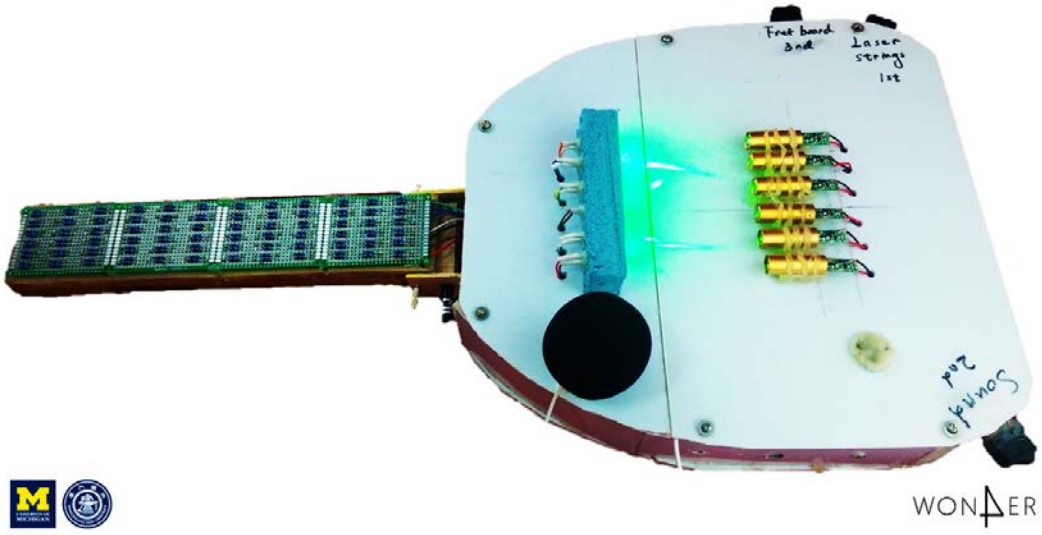
View from the top:



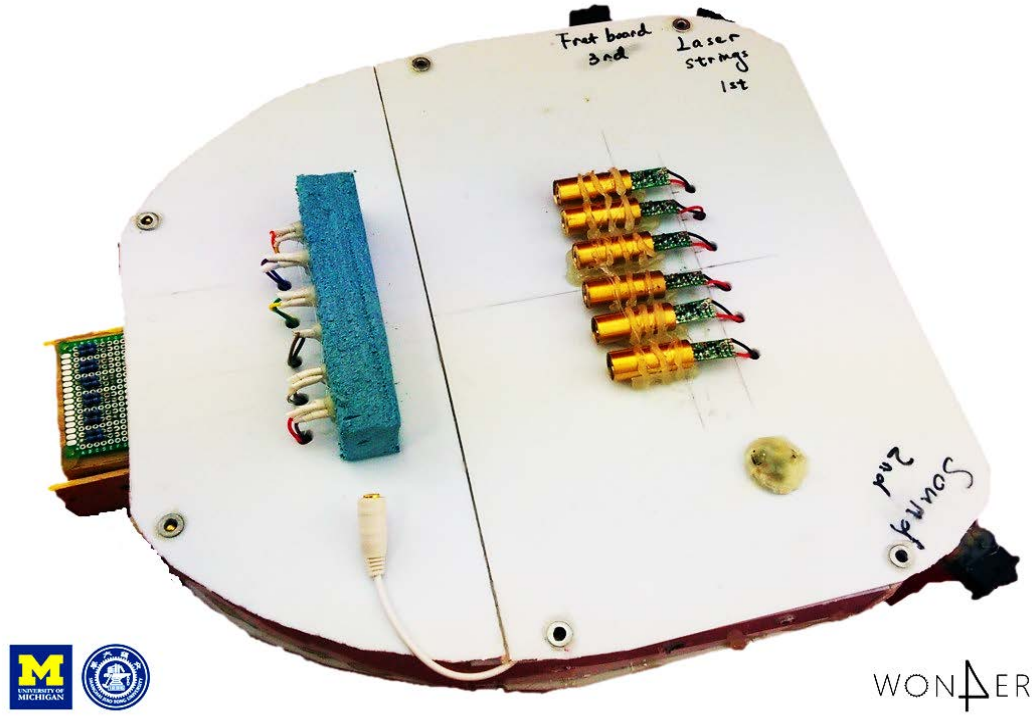
View from the left:



View when the lasers are on:



View when the fret board is retracted:



Demo Videos

The 3D Preview for our "Portable Laser Guitar" are shown in the following webpages:

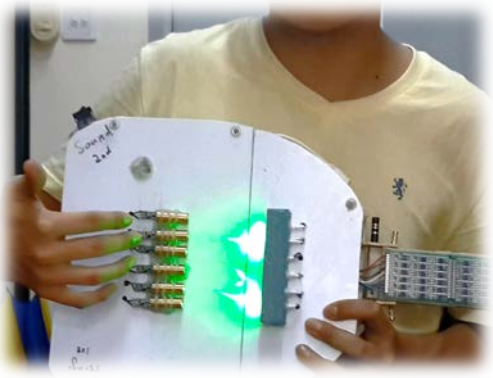
YouKu: http://v.youku.com/v_show/id_XNzQ2NTc4MjY0.html

YouTube: https://www.youtube.com/watch?v=LZErYLt636o

The DEMO for our "Portable Laser Guitar" are shown in the following webpages:

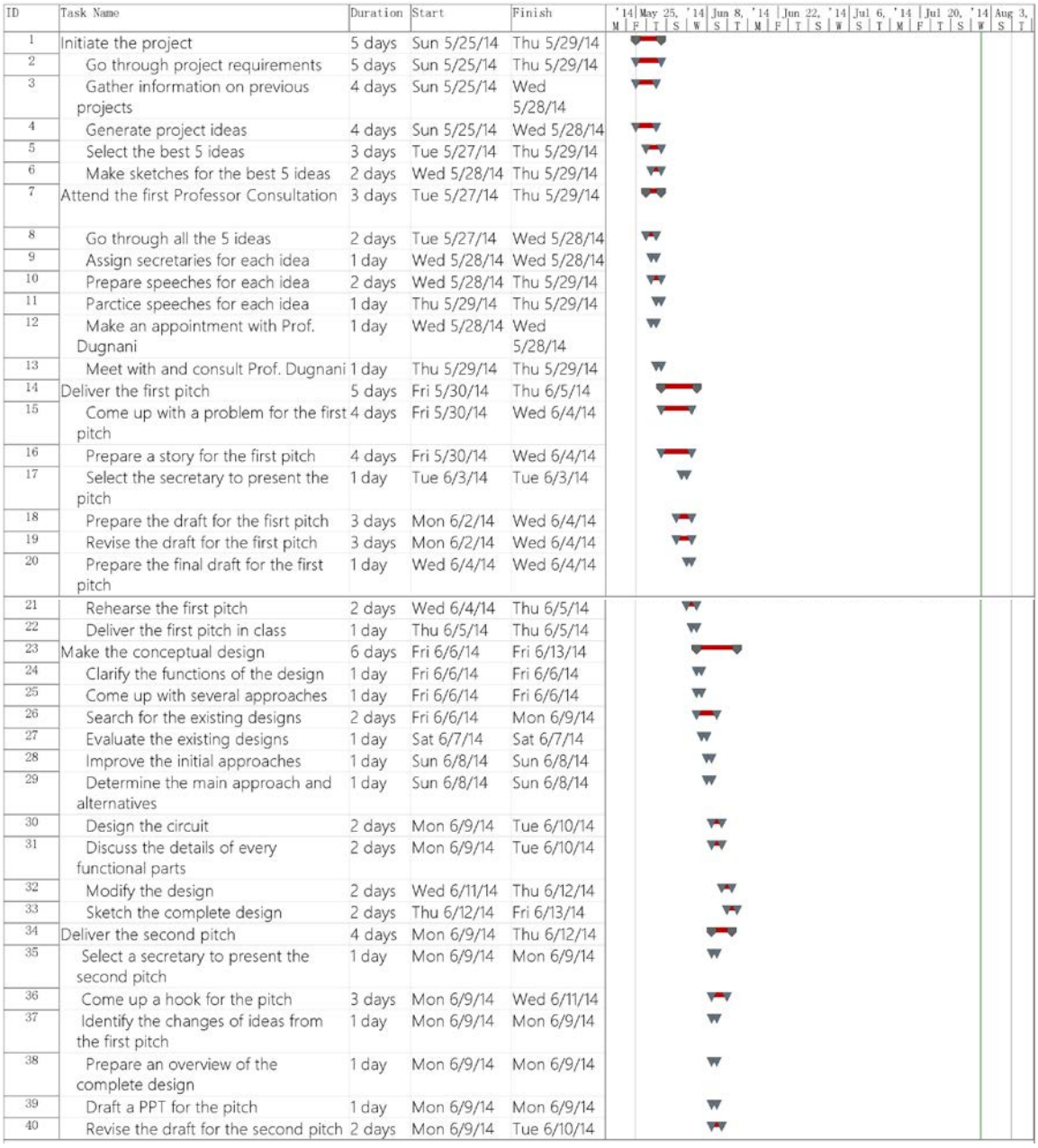
YouKu: http://v.youku.com/v_show/id_XNzQ2NTc5MDIw.html

YouTube: https://www.youtube.com/watch?v=EAGU8Xj4gFk&feature=youtu.be

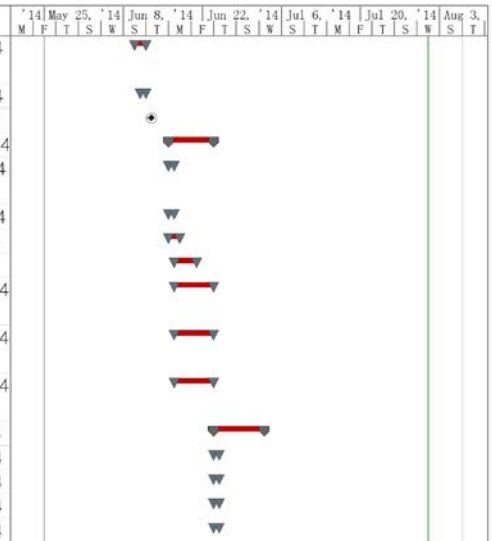



XVII. Appendix D: Complete Schedule

ID	Task Name	Duration	Start	Finish
1	Initiate the project	5 days	Sun 5/25/14	Thu 5/29/14
2	Go through project requirements	5 days	Sun 5/25/14	Thu 5/29/14
3	Gather information on previous projects	4 days	Sun 5/25/14	Wed 5/28/14
4	Generate project ideas	4 days	Sun 5/25/14	Wed 5/28/14
5	Select the best 5 ideas	3 days	Tue 5/27/14	Thu 5/29/14
6	Make sketches for the best 5 ideas	2 days	Wed 5/28/14	Thu 5/29/14
7	Attend the first Professor Consultation	3 days	Tue 5/27/14	Thu 5/29/14
8	Go through all the 5 ideas	2 days	Tue 5/27/14	Wed 5/28/14
9	Assign secretaries for each idea	1 day	Wed 5/28/14	Wed 5/28/14
10	Prepare speeches for each idea	2 days	Wed 5/28/14	Thu 5/29/14
11	Parctice speeches for each idea	1 day	Thu 5/29/14	Thu 5/29/14
12	Make an appointment with Prof. Dugnani	1 day	Wed 5/28/14	Wed 5/28/14
13	Meet with and consult Prof. Dugnani	1 day	Thu 5/29/14	Thu 5/29/14
14	Deliver the first pitch	5 days	Fri 5/30/14	Thu 6/5/14
15	Come up with a problem for the first pitch	4 days	Fri 5/30/14	Wed 6/4/14
16	Prepare a story for the first pitch	4 days	Fri 5/30/14	Wed 6/4/14
17	Select the secretary to present the pitch	1 day	Tue 6/3/14	Tue 6/3/14
18	Prepare the draft for the fisrt pitch	3 days	Mon 6/2/14	Wed 6/4/14
19	Revise the draft for the first pitch	3 days	Mon 6/2/14	Wed 6/4/14
20	Prepare the final draft for the first pitch	1 day	Wed 6/4/14	Wed 6/4/14
21	Rehearse the first pitch	2 days	Wed 6/4/14	Thu 6/5/14
22	Deliver the first pitch in class	1 day	Thu 6/5/14	Thu 6/5/14
23	Make the conceptual design	6 days	Fri 6/6/14	Fri 6/13/14
24	Clarify the functions of the design	1 day	Fri 6/6/14	Fri 6/6/14
25	Come up with several approaches	1 day	Fri 6/6/14	Fri 6/6/14
26	Search for the existing designs	2 days	Fri 6/6/14	Mon 6/9/14
27	Evaluate the existing designs	1 day	Sat 6/7/14	Sat 6/7/14
28	Improve the initial approaches	1 day	Sun 6/8/14	Sun 6/8/14
29	Determine the main approach and alternatives	1 day	Sun 6/8/14	Sun 6/8/14
30	Design the circuit	2 days	Mon 6/9/14	Tue 6/10/14
31	Discuss the details of every functional parts	2 days	Mon 6/9/14	Tue 6/10/14
32	Modify the design	2 days	Wed 6/11/14	Thu 6/12/14
33	Sketch the complete design	2 days	Thu 6/12/14	Fri 6/13/14
34	Deliver the second pitch	4 days	Mon 6/9/14	Thu 6/12/14
35	Select a secretary to present the second pitch	1 day	Mon 6/9/14	Mon 6/9/14
36	Come up a hook for the pitch	3 days	Mon 6/9/14	Wed 6/11/14
37	Identify the changes of ideas from the first pitch	1 day	Mon 6/9/14	Mon 6/9/14
38	Prepare an overview of the complete design	1 day	Mon 6/9/14	Mon 6/9/14
39	Draft a PPT for the pitch	1 day	Mon 6/9/14	Mon 6/9/14
40	Revise the draft for the second pitch	2 days	Mon 6/9/14	Tue 6/10/14

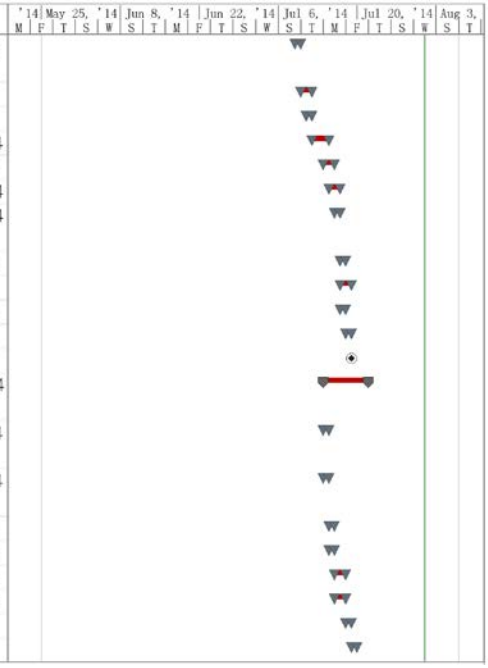


ID	Task Name	Duration	Start	Finish
41	Prepare the final draft for the second pitch	2 days	Tue 6/10/14	Wed 6/11/14
42	Rehearse the second pitch	1 day	Wed 6/11/14	Wed 6/11/14
43	Deliver the second pitch in class	1 day	Thu 6/12/14	Thu 6/12/14
44	Purchase the materials	6 days	Mon 6/16/14	Mon 6/23/14
45	Identify the necessary components in the design	1 day	Mon 6/16/14	Mon 6/16/14
46	Make a shopping list	1 day	Mon 6/16/14	Mon 6/16/14
47	Learn about the laser generators	2 days	Mon 6/16/14	Tue 6/17/14
48	Buy the appropriate laser generators	4 days	Tue 6/17/14	Fri 6/20/14
49	Purchase the suitable sound playback module	5 days	Tue 6/17/14	Mon 6/23/14
50	Buy the electric resistances and the diastimeters	5 days	Tue 6/17/14	Mon 6/23/14
51	Purchase any other materials needed to complete the project	5 days	Tue 6/17/14	Mon 6/23/14
52	Build the first prototype	7 days	Tue 6/24/14	Wed 7/2/14
53	Book a discussion room	1 day	Tue 6/24/14	Tue 6/24/14
54	Hold a meeting	1 day	Tue 6/24/14	Tue 6/24/14
55	Design the appearance	1 day	Tue 6/24/14	Tue 6/24/14
56	Analyze the structure of the guitar	1 day	Tue 6/24/14	Tue 6/24/14
57	Analyze the structure of the laser sensor	1 day	Tue 6/24/14	Tue 6/24/14
58	Analyze the structure of the audio components	1 day	Tue 6/24/14	Tue 6/24/14
59	Design and construct the circuit of the prototype	3 days	Thu 6/26/14	Mon 6/30/14
60	Design the program of arduino boarc	2 days	Sun 6/29/14	Mon 6/30/14
61	Assemble the prototype	1 day	Tue 7/1/14	Tue 7/1/14
62	Get advice from the teacher	1 day	Wed 7/2/14	Wed 7/2/14
63	Write the progress report	2 days	Fri 7/4/14	Mon 7/7/14
64	Hold a meeting	1 day	Fri 7/4/14	Fri 7/4/14
65	Make a scheldule of the whole process	1 day	Fri 7/4/14	Fri 7/4/14
66	Discuss the current progress	2 days	Fri 7/4/14	Mon 7/7/14
67	Orgainize the remain progress and completed progress	1 day	Sat 7/5/14	Sat 7/5/14
68	Draw a gnatt chart	2 days	Sat 7/5/14	Mon 7/7/14
69	Write the progress report	2 days	Sat 7/5/14	Mon 7/7/14
70	Test the first prototype	3 days	Wed 7/9/14	Fri 7/11/14
71	Book a discussion room	1 day	Wed 7/9/14	Wed 7/9/14
72	Prepare tools	1 day	Thu 7/10/14	Thu 7/10/14
73	Hold a meeting	1 day	Thu 7/10/14	Thu 7/10/14
74	Discuss which test to use	1 day	Thu 7/10/14	Thu 7/10/14
75	Design a fuctional test	1 day	Thu 7/10/14	Thu 7/10/14
76	Test the prototype	1 day	Fri 7/11/14	Fri 7/11/14
77	Build the second prototpye	8 days	Wed 7/9/14	Fri 7/18/14
78	Book a discussion room	1 day	Wed 7/9/14	Wed 7/9/14
79	Hold a meeting	1 day	Wed 7/9/14	Wed 7/9/14
80	Discuss the shortage of first prototye	1 day	Wed 7/9/14	Wed 7/9/14
81	Get the several replaced solutions	1 day	Wed 7/9/14	Wed 7/9/14

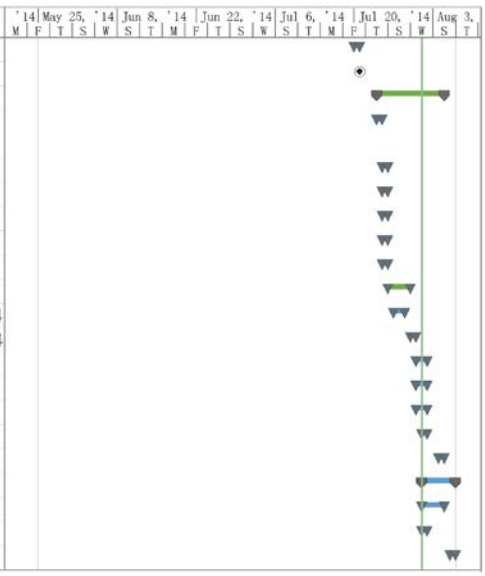


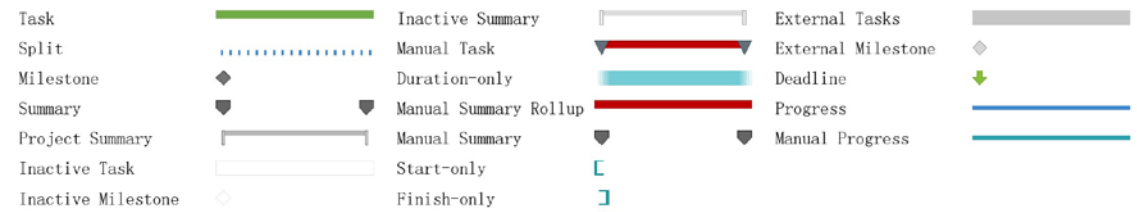
Task
Split
Milestone
Summary
Project Summary
Inactive Task
Inactive Milestone
Inactive Summary
Manual Task
Duration-only
Manual Summary Rollup
Manual Summary
Start-only
Finish-only
External Tasks
External Milestone
Deadline
Progress
Manual Progress

ID	Task Name	Duration	Start	Finish
82	Decide the practicability of the replacement	1 day	Wed 7/9/14	Wed 7/9/14
83	Determine the modified solutions	2 days	Thu 7/10/14	Fri 7/11/14
84	Consult with the TAs and teachers	1 day	Fri 7/11/14	Fri 7/11/14
85	Search for the materials	2 days	Sat 7/12/14	Mon 7/14/14
86	Build the solutions' part	2 days	Mon 7/14/14	Tue 7/15/14
87	Test the all parts	2 days	Tue 7/15/14	Wed 7/16/14
88	Carry out robustness test and repeat the test	1 day	Wed 7/16/14	Wed 7/16/14
89	Record the result of test	1 day	Thu 7/17/14	Thu 7/17/14
90	Anaylse the result	2 days	Thu 7/17/14	Fri 7/18/14
91	Discuss the result	1 day	Thu 7/17/14	Thu 7/17/14
92	Consult the teacher about the result	1 day	Fri 7/18/14	Fri 7/18/14
93	Get advice from the teacher	1 day	Fri 7/18/14	Fri 7/18/14
94	Attend the symposium and deliver the oral presentation	6 days	Mon 7/14/14	Mon 7/21/14
95	Have the meeting and discuss about the content	1 day	Mon 7/14/14	Mon 7/14/14
96	Choose the secretary for the oral presentation	1 day	Mon 7/14/14	Mon 7/14/14
97	Get familiar with the requirement	1 day	Tue 7/15/14	Tue 7/15/14
98	Deliver the tasks for everyone	1 day	Tue 7/15/14	Tue 7/15/14
99	Design ppt and video	2 days	Wed 7/16/14	Thu 7/17/14
100	Prepare the oral content	2 days	Wed 7/16/14	Thu 7/17/14
101	Rehearse	1 day	Fri 7/18/14	Fri 7/18/14
102	Revise the draft	1 day	Sat 7/19/14	Sat 7/19/14

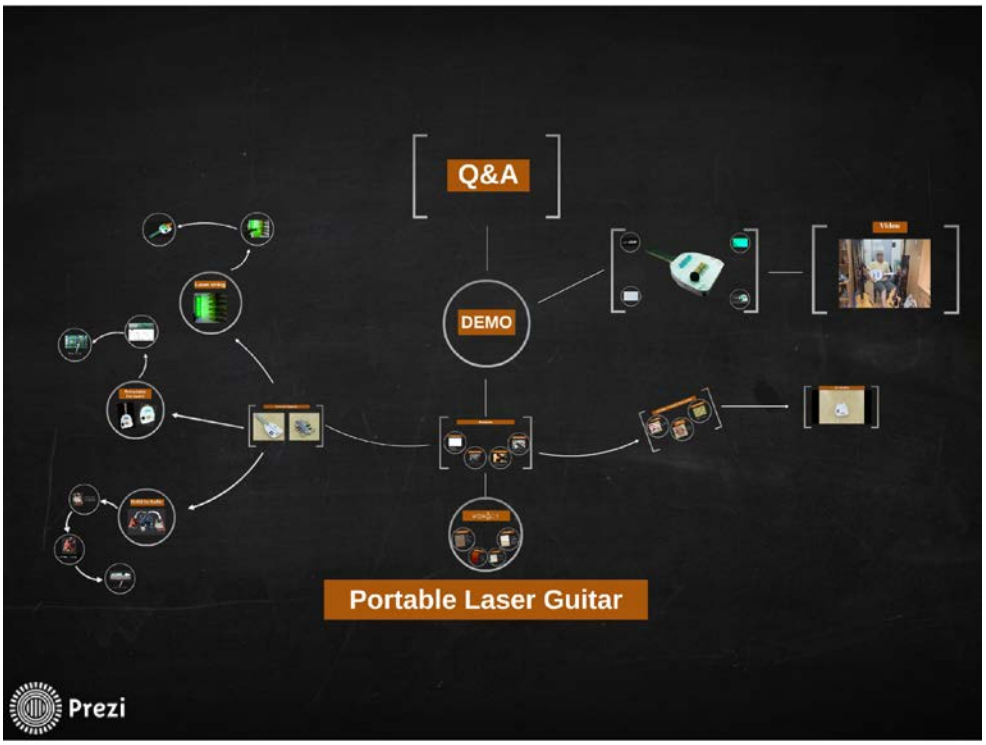


ID	Task Name	Duration	Start	Finish
103	Attend the symposium	1 day	Sun 7/20/14	Sun 7/20/14
104	Deliver the oral presentation	1 day	Sun 7/20/14	Sun 7/20/14
105	Write the final report	8 days	Thu 7/24/14	Mon 8/4/14
106	Learn the requirement for the final report	1 day	Thu 7/24/14	Thu 7/24/14
107	Hold the meeting	1 day	Fri 7/25/14	Fri 7/25/14
108	Discuss the structure of the report	1 day	Fri 7/25/14	Fri 7/25/14
109	Discuss the details for every part	1 day	Fri 7/25/14	Fri 7/25/14
110	Search the material for every part	1 day	Fri 7/25/14	Fri 7/25/14
111	Deliver the task for everyone	1 day	Fri 7/25/14	Fri 7/25/14
112	Write the initial draft	3 days	Sat 7/26/14	Tue 7/29/14
113	Revise the first draft	2 days	Sun 7/27/14	Mon 7/28/14
114	Write the second draft	1 day	Wed 7/30/14	Wed 7/30/14
115	Check the reference	2 days	Thu 7/31/14	Fri 8/1/14
116	Check the appendix	2 days	Thu 7/31/14	Fri 8/1/14
117	Check the format of all the parts	2 days	Thu 7/31/14	Fri 8/1/14
118	Revise the second report	1 day	Fri 8/1/14	Fri 8/1/14
119	Hand in the report	1 day	Mon 8/4/14	Mon 8/4/14
120	Attend the final Expo	4 days	Fri 8/1/14	Wed 8/6/14
121	Prepare for the Expo	2 days	Fri 8/1/14	Mon 8/4/14
122	Invite the visitors	1 day	Fri 8/1/14	Fri 8/1/14
123	Attend the Expo	1 day	Wed 8/6/14	Wed 8/6/14

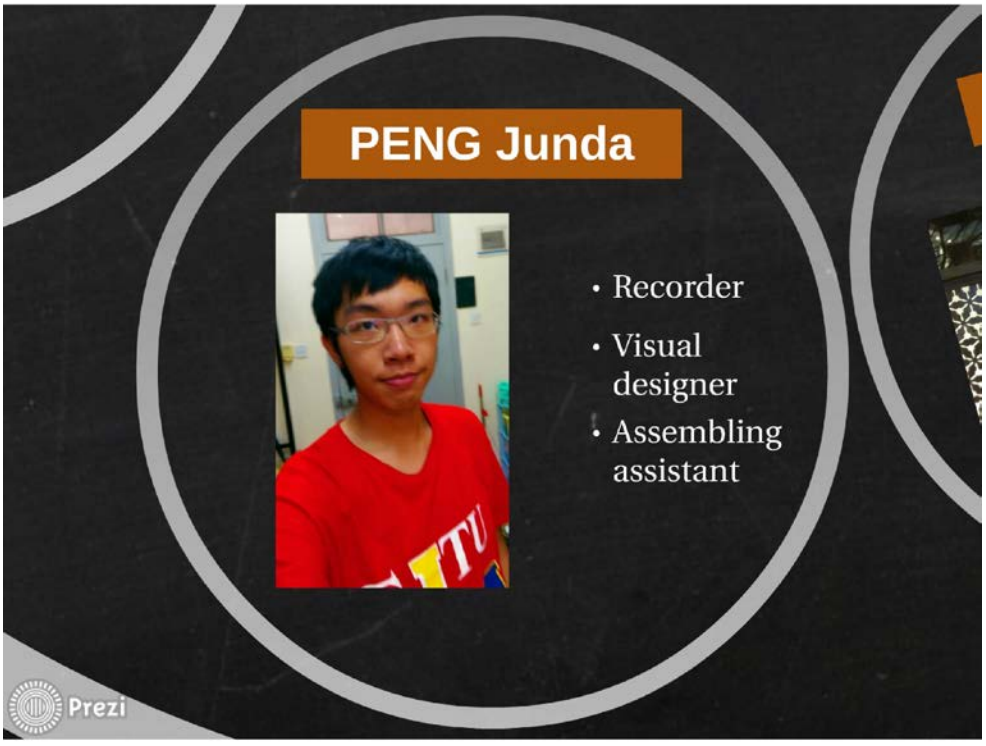




XVIII. Appendix E: Symposium Slides




LIU Xieyang
• Leader
• Programmer
• Circuit engineer
• Assembling engineer
Prezi


PENG Junda
• Recorder
• Visual designer
• Assembling assistant
Prezi

YU Tong

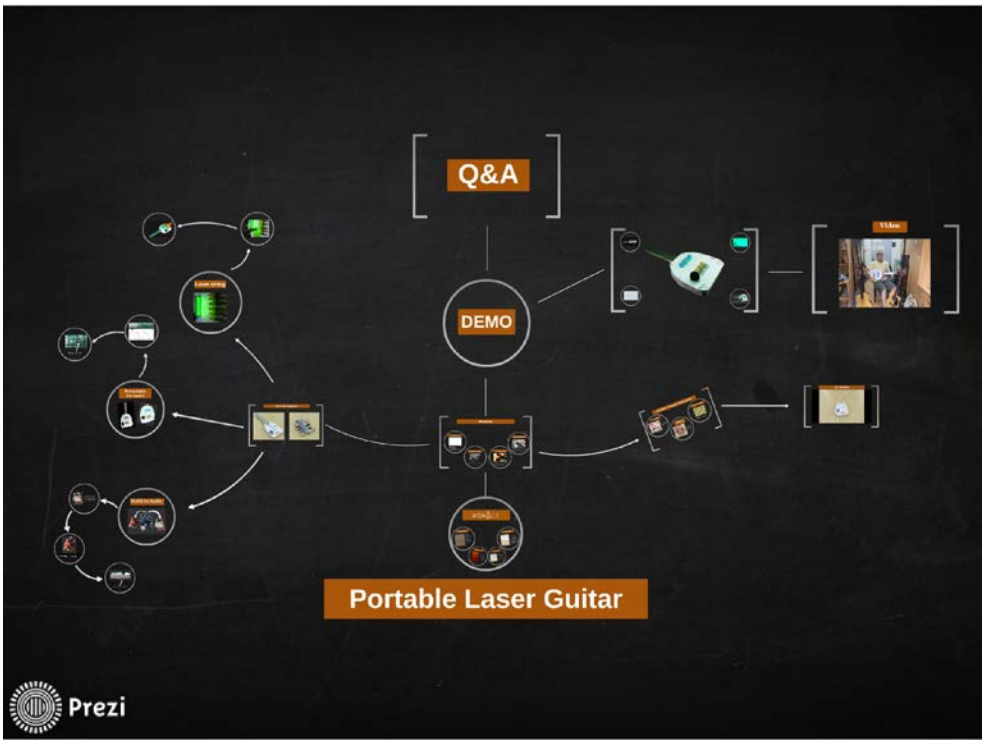
- Visual designer
- Chief finance officer
- Assembling assistant

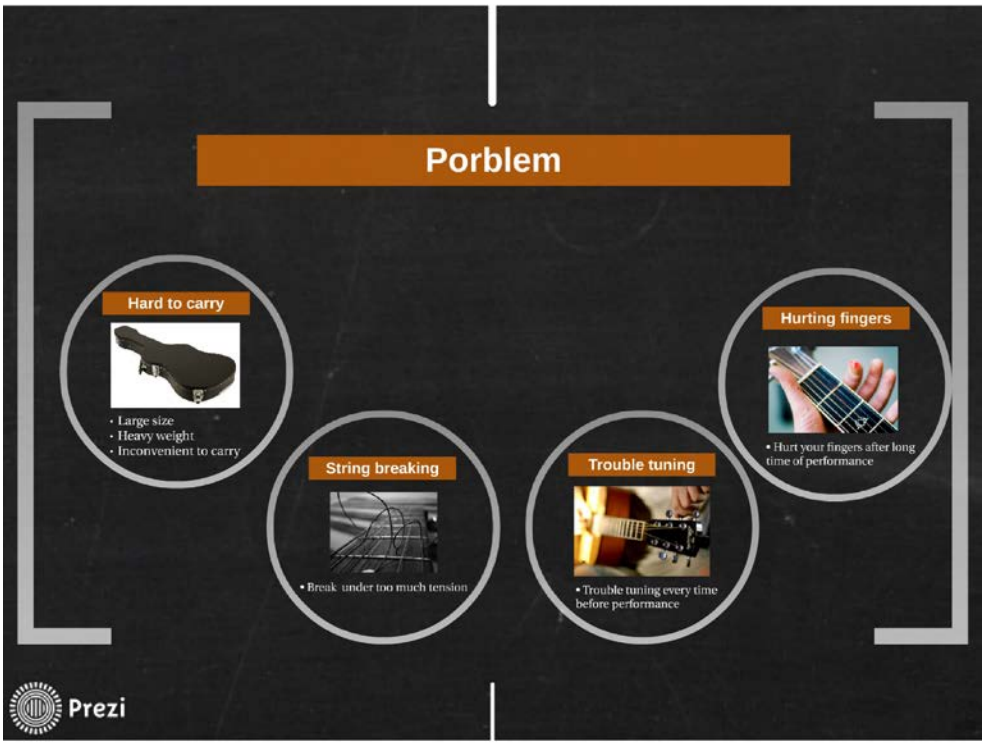
Prezi

YU Zhiwei

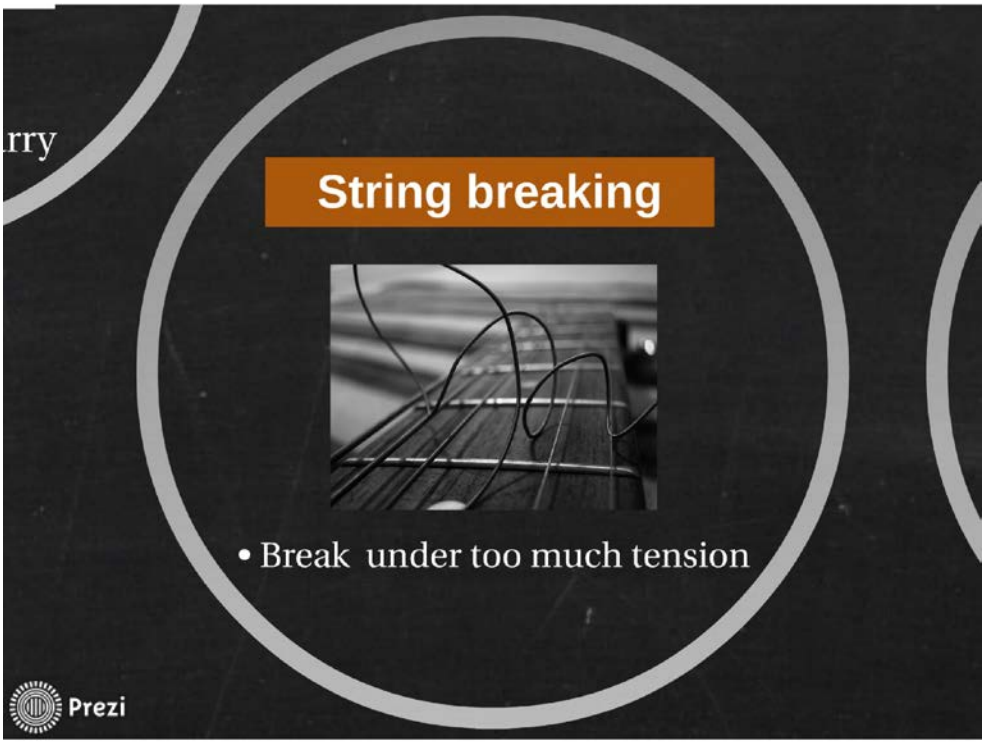
- Recorder
- Assembling assistant
- Programming advisor

Prezi


Q&A
DEMO
Video
Portable Laser Guitar
Prezi

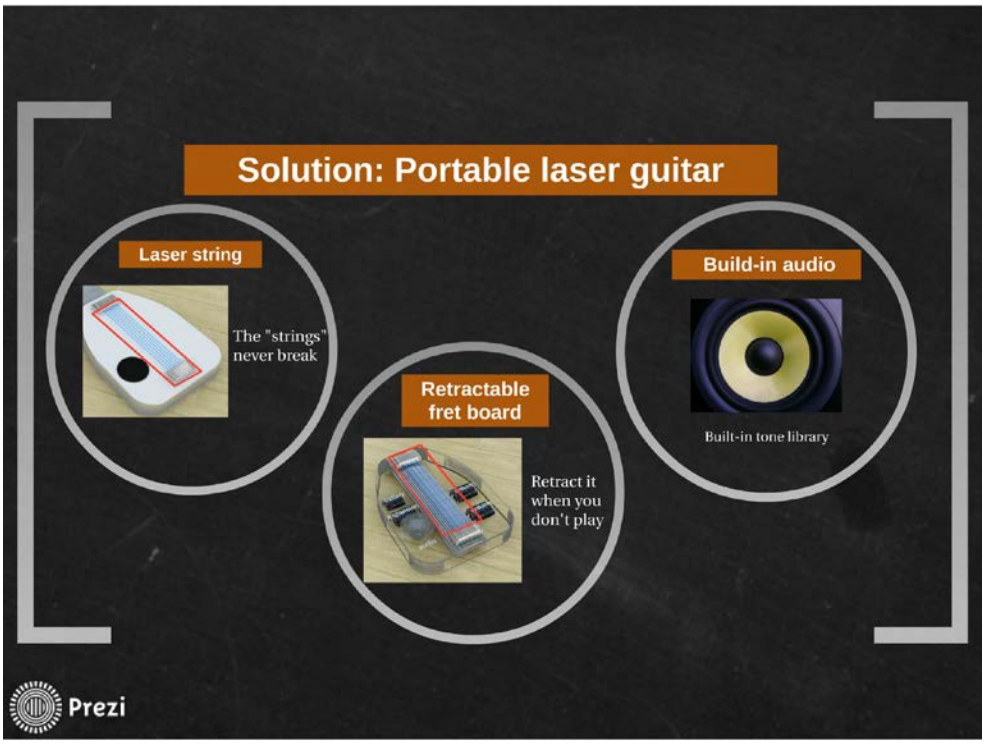

Porblem
Hard to carry
• Large size
• Heavy weight
• Inconvenient to carry
String breaking
• Break under too much tension
Trouble tuning
• Trouble tuning every time before performance
Hurting fingers
• Hurt your fingers after long time of performance
Prezi

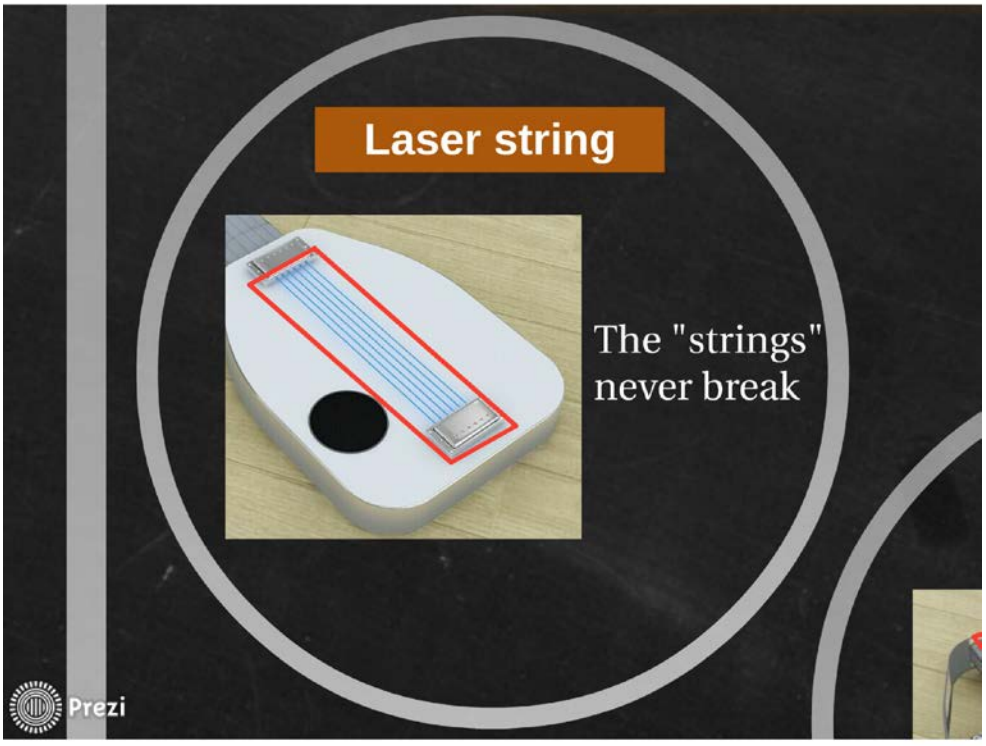

Hard to carry
• Large size
• Heavy weight
• Inconvenient to carry
Prezi

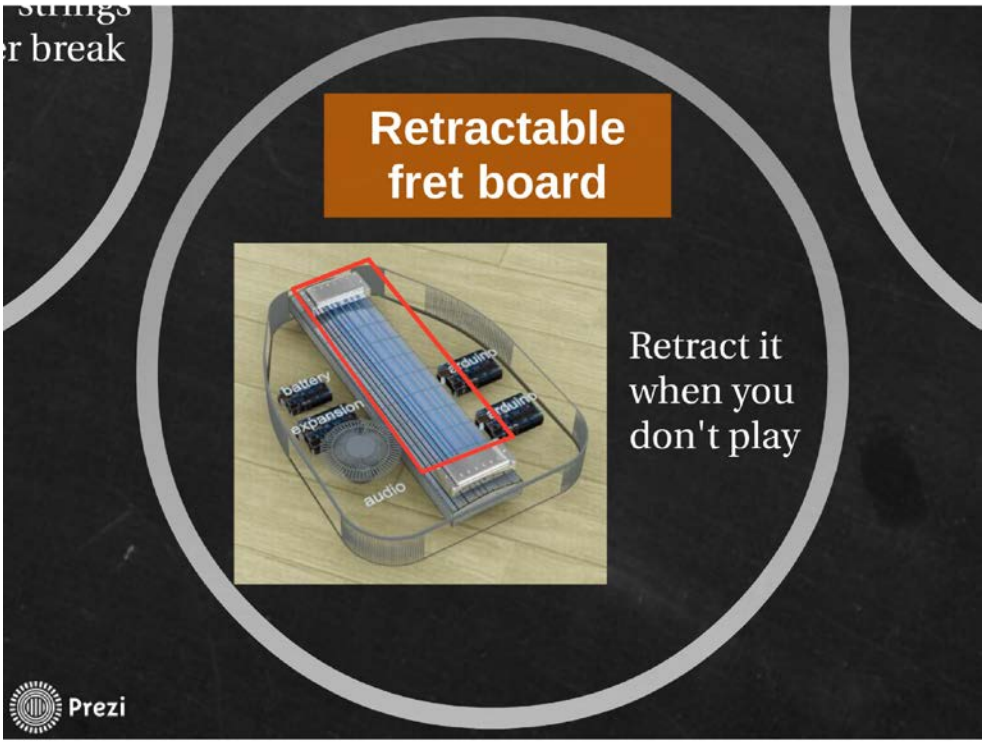

String breaking
• Break under too much tension
Prezi


• Hurt
time of
Trouble tuning
• Trouble tuning every time before performance
Prezi


Hurting fingers
• Hurt your fingers after long time of performance
ing
Prezi

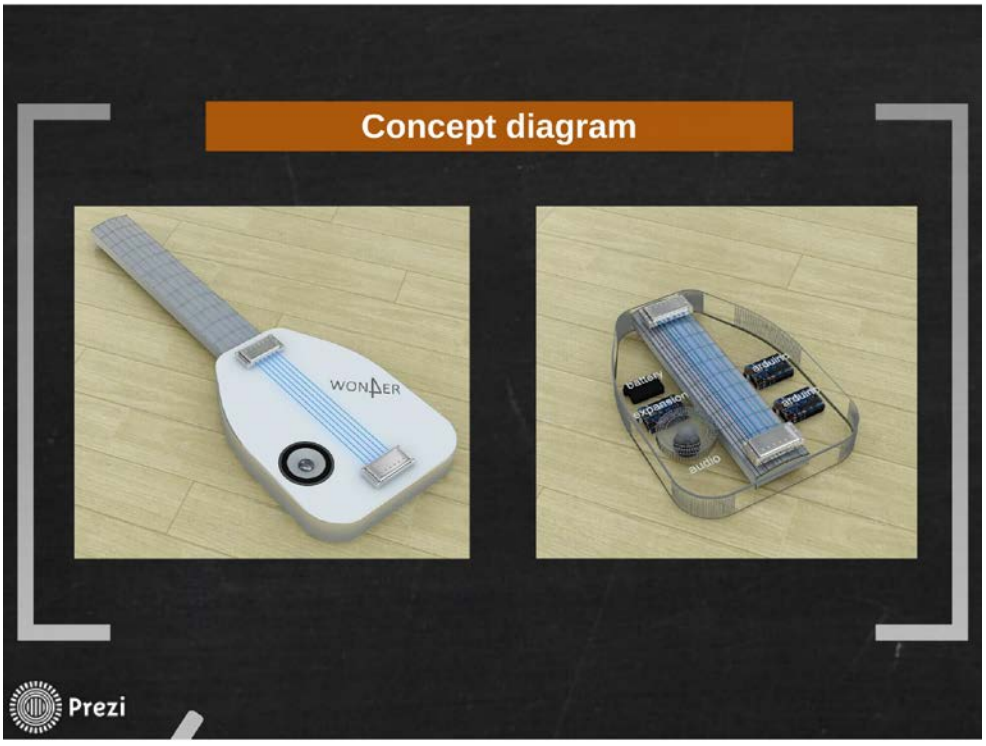

Solution: Portable laser guitar
Laser string
The "strings" never break
Retractable fret board
Retract it when you don't play
Build-in audio
Built-in tone library
Prezi


Laser string
The "strings" never break
Prezi

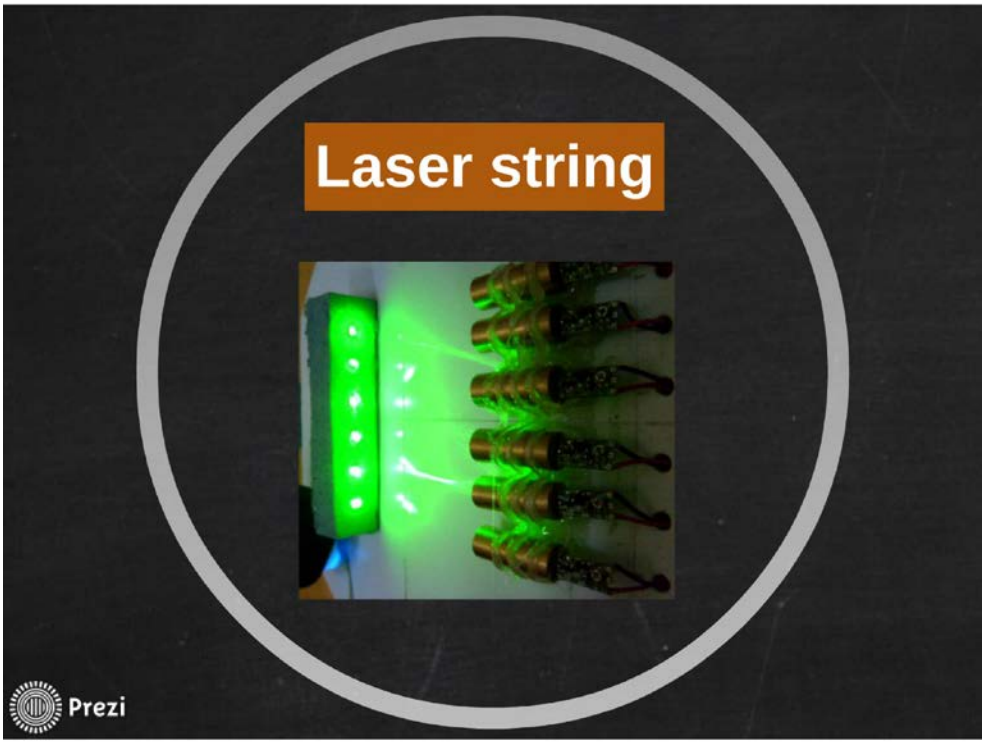

r break
Retractable fret board
battery
expansion
audio
Retract it when you don't play
Prezi


Build-in audio
Built-in tone library
Prezi

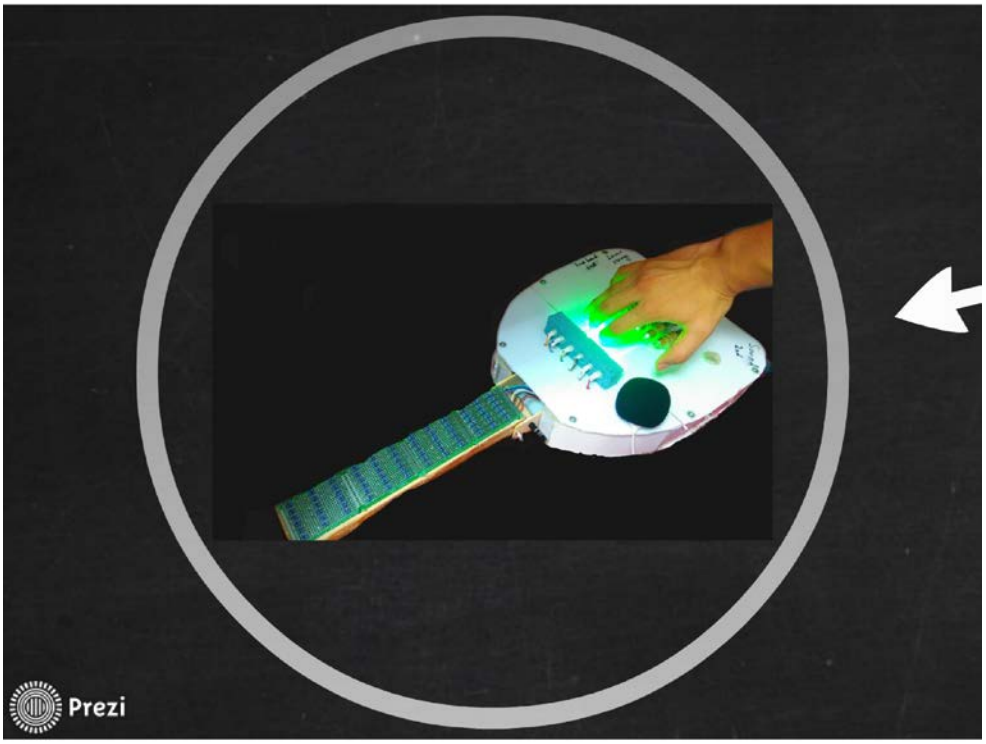

3D Preview
Prezi

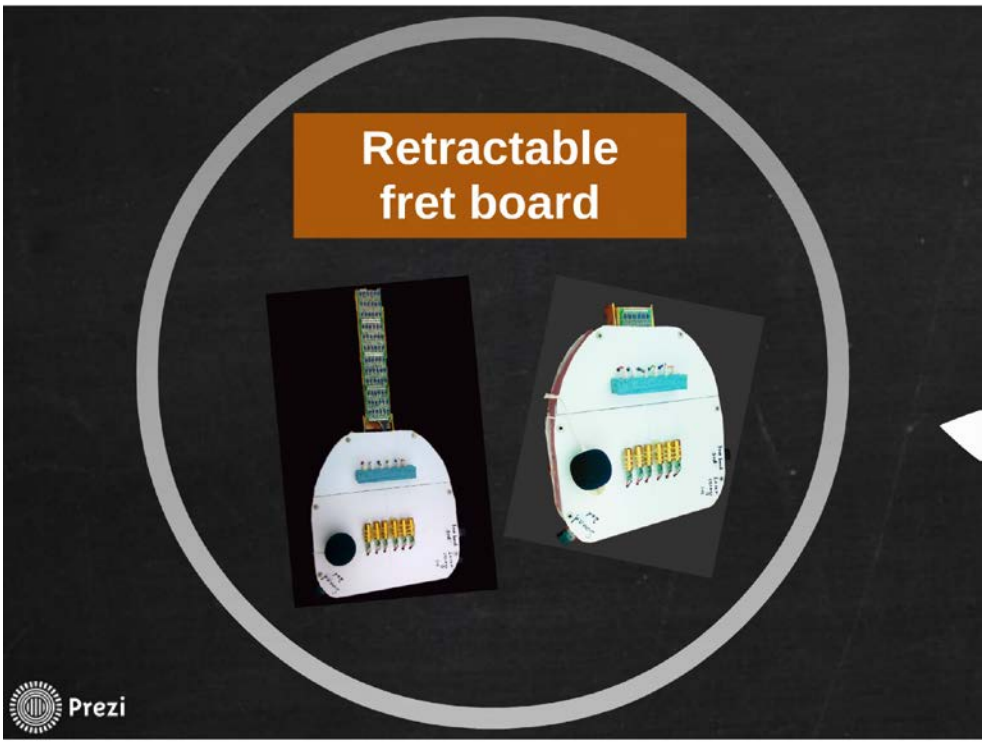

Concept diagram
Prezi

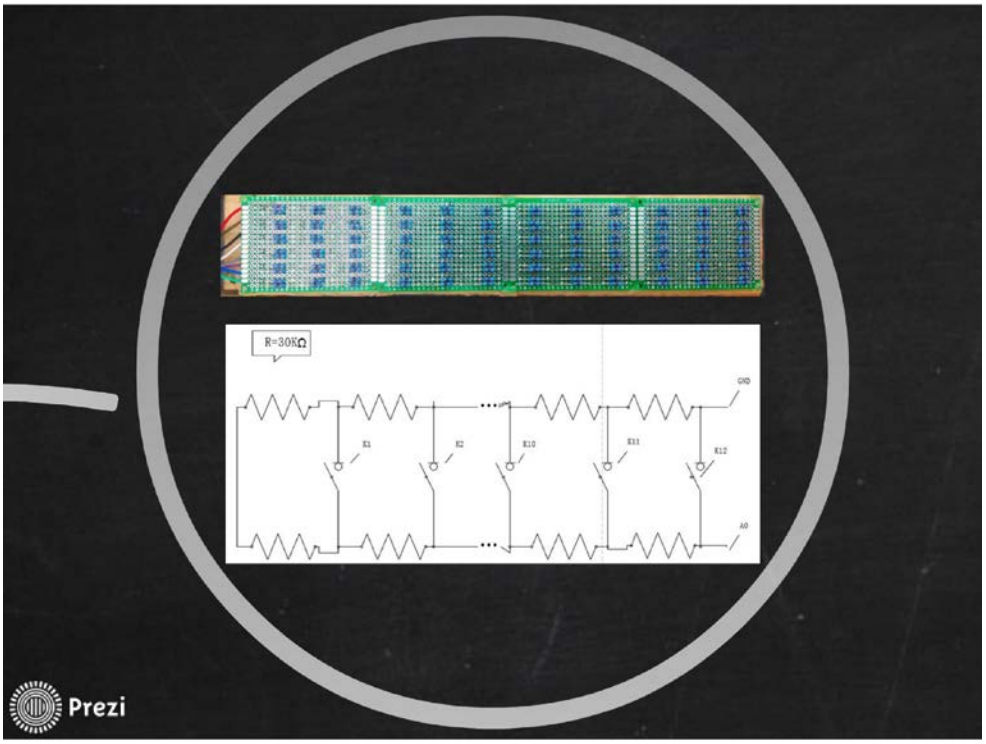

battery
expansion
arduino
arduino
audio
Prezi

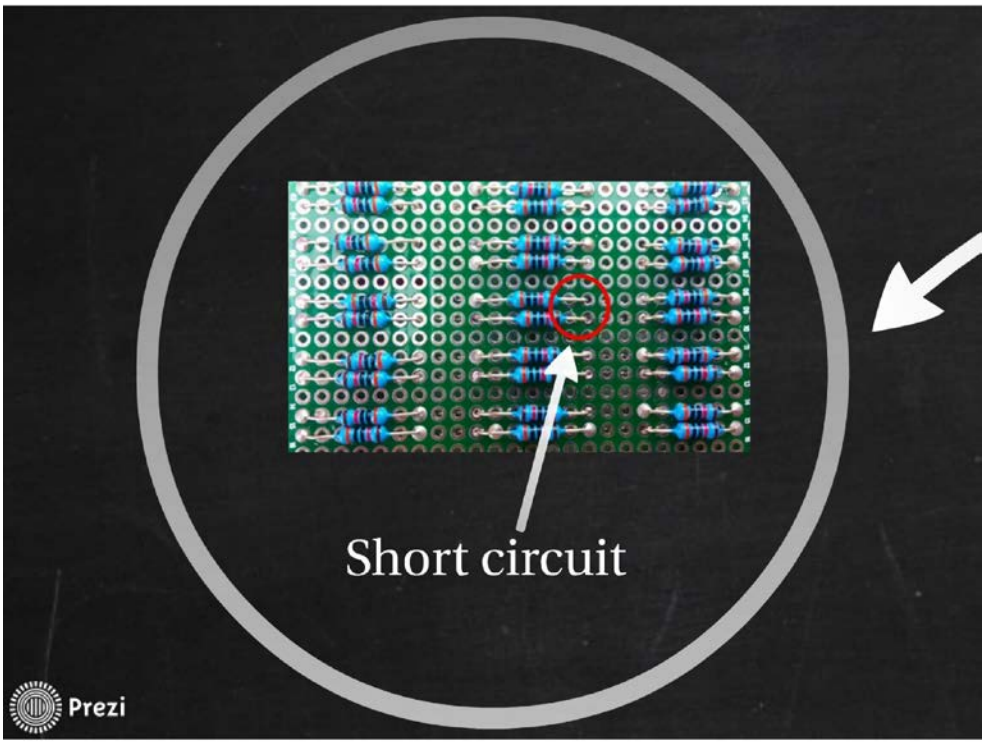

Laser string
Prezi

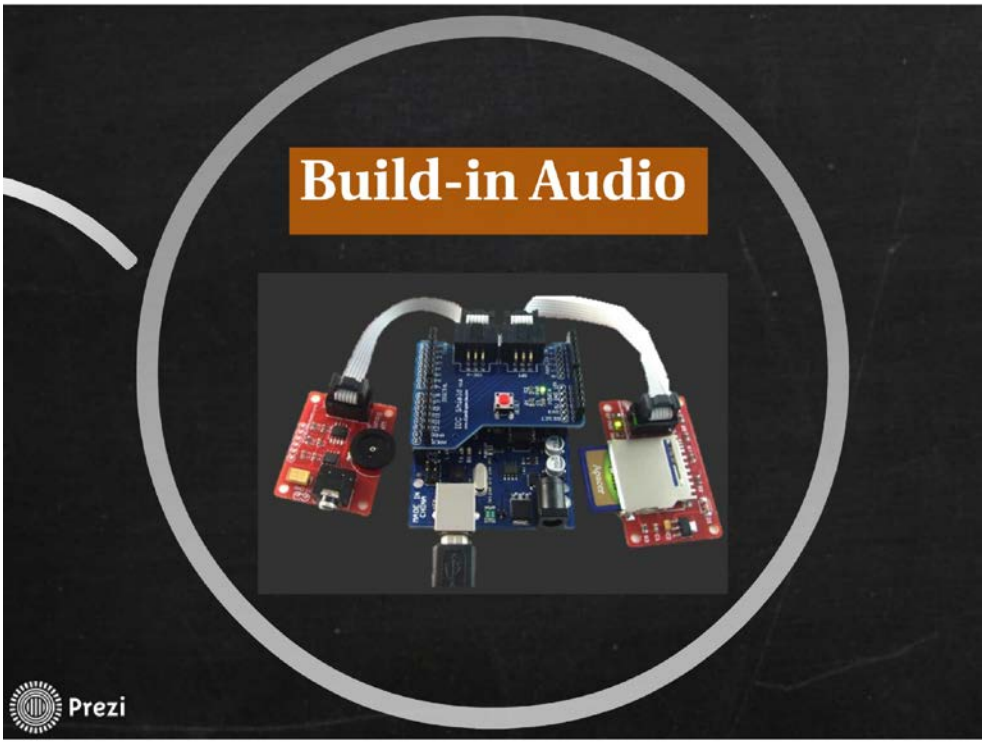

Prezi

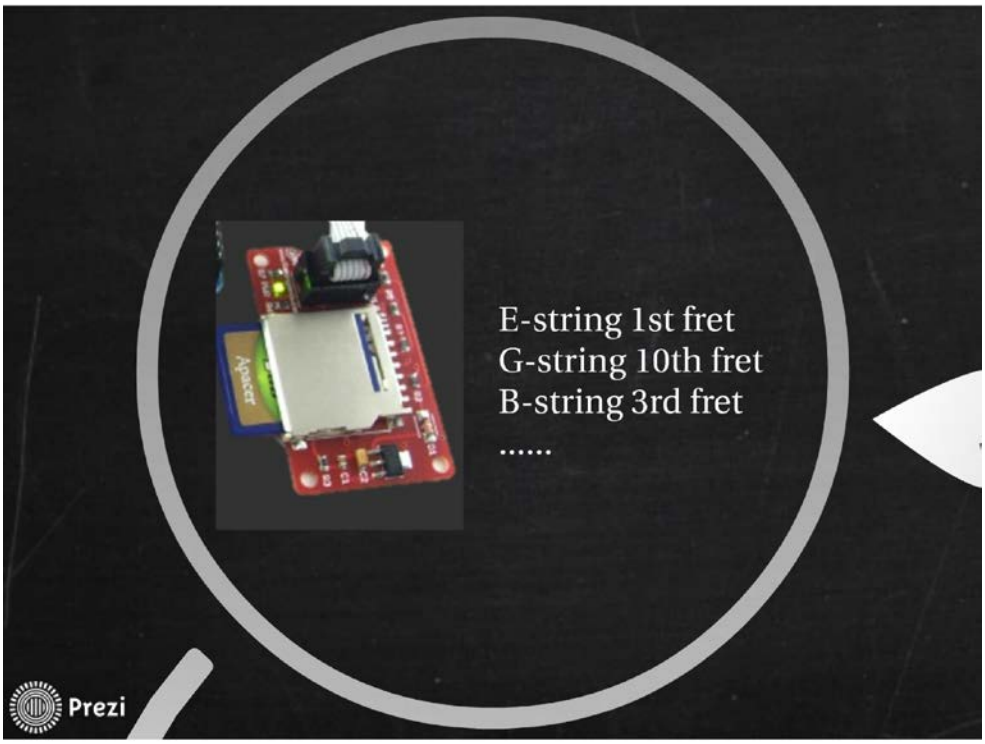

Prezi

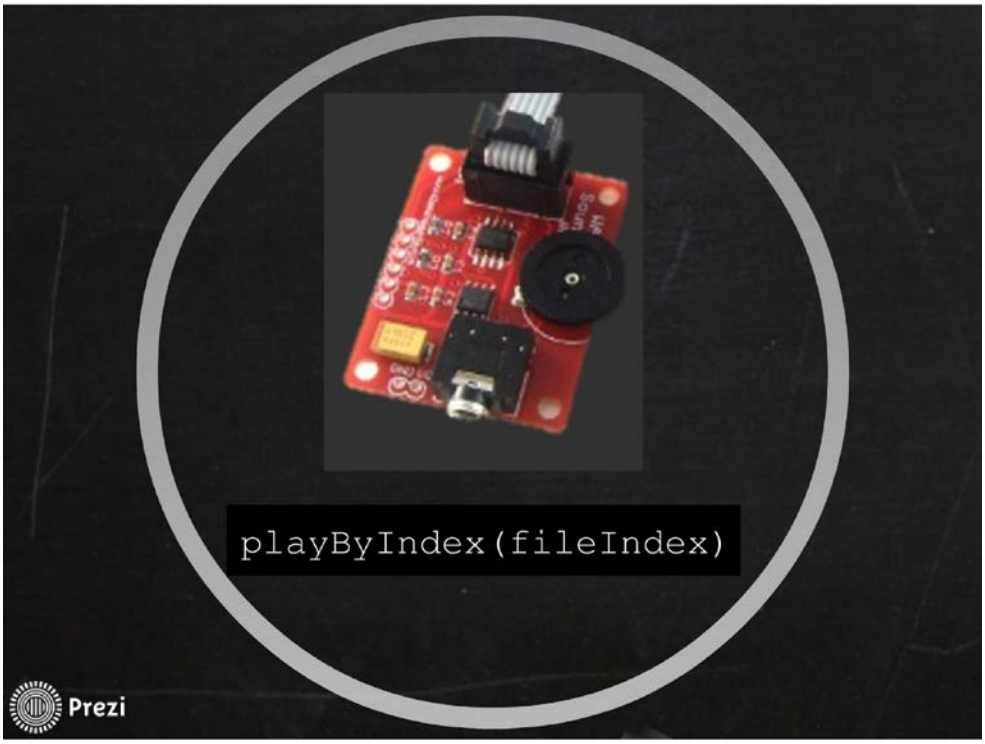

Retractable
fret board
Prezi

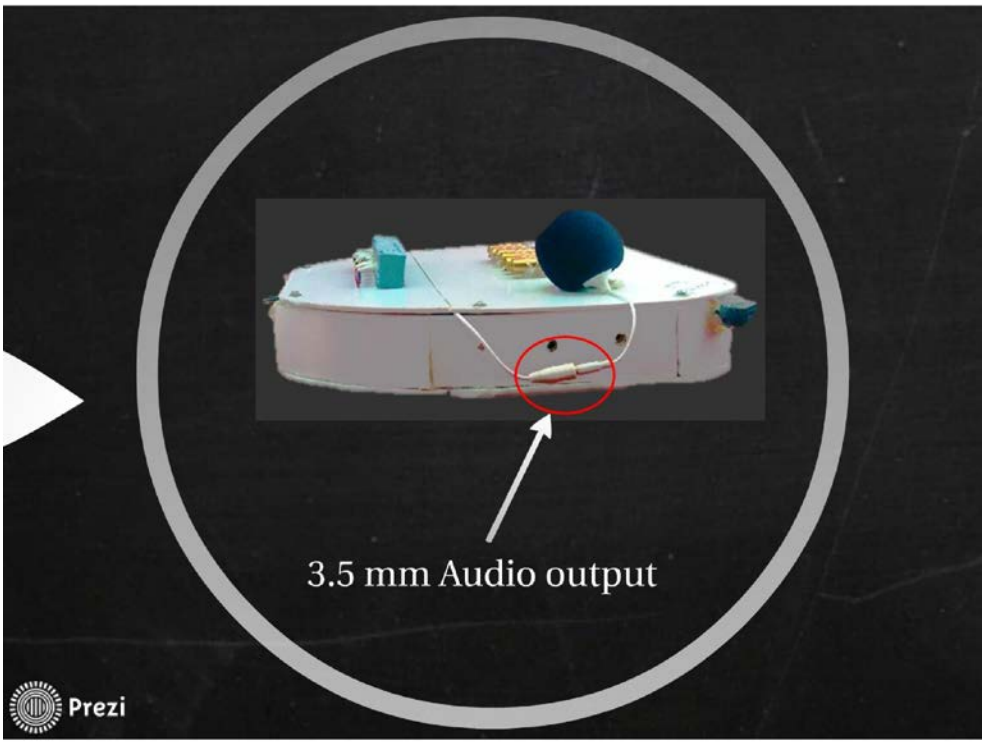

R=30KΩ
Prezi


Short circuit
Prezi

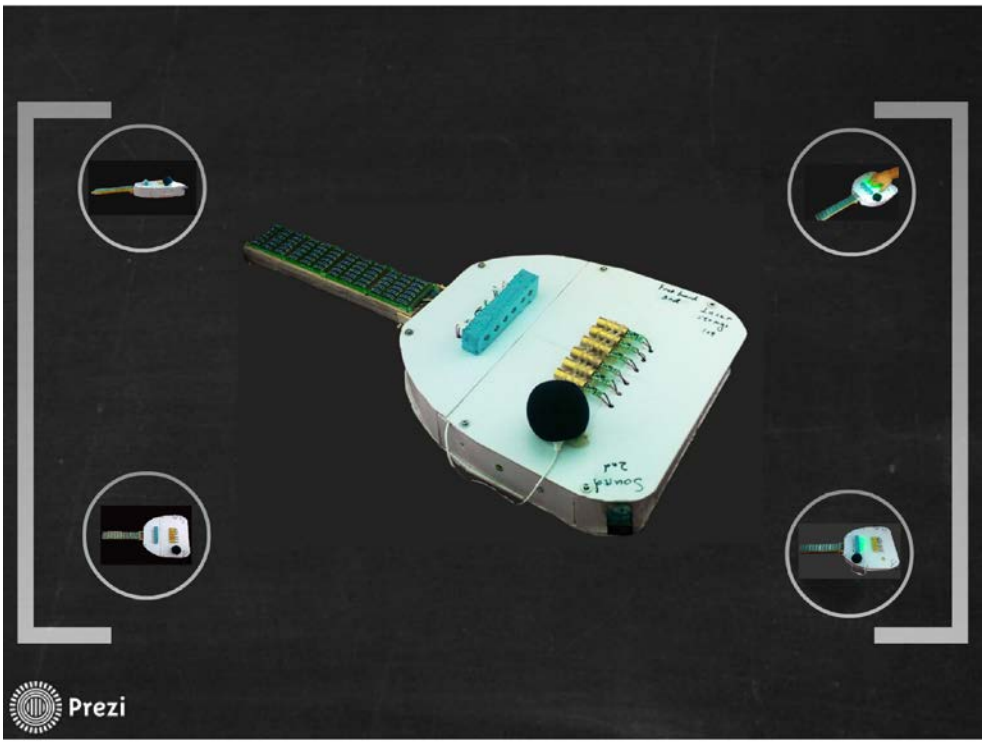

Build-in Audio
Prezi

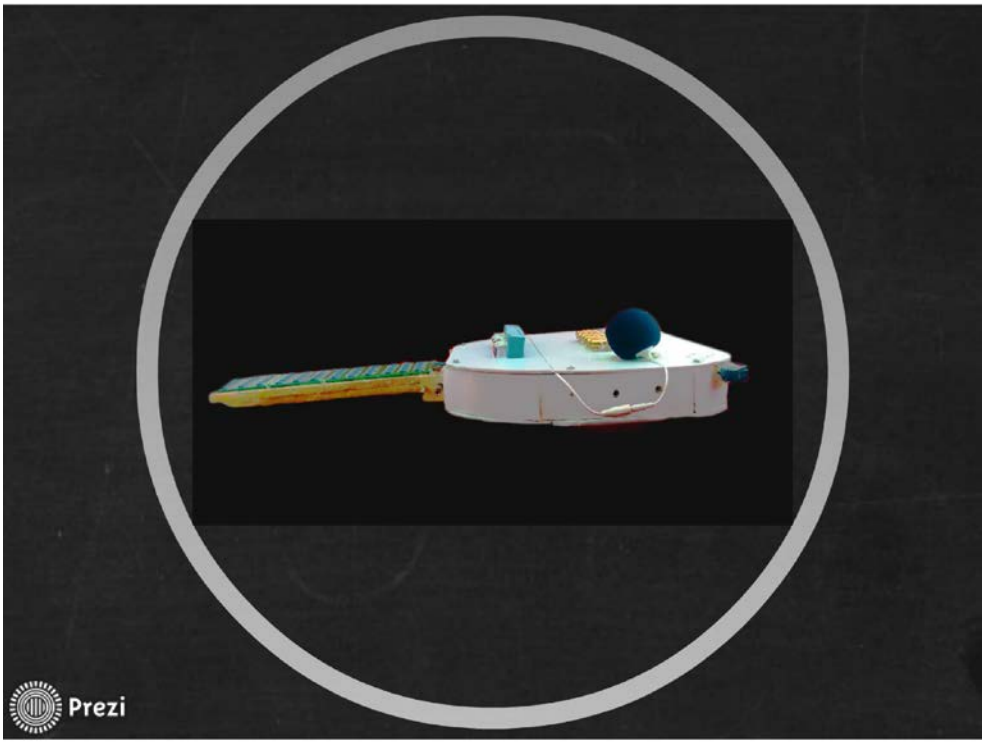

E-string 1st fret
G-string 10th fret
B-string 3rd fret
.....
Prezi

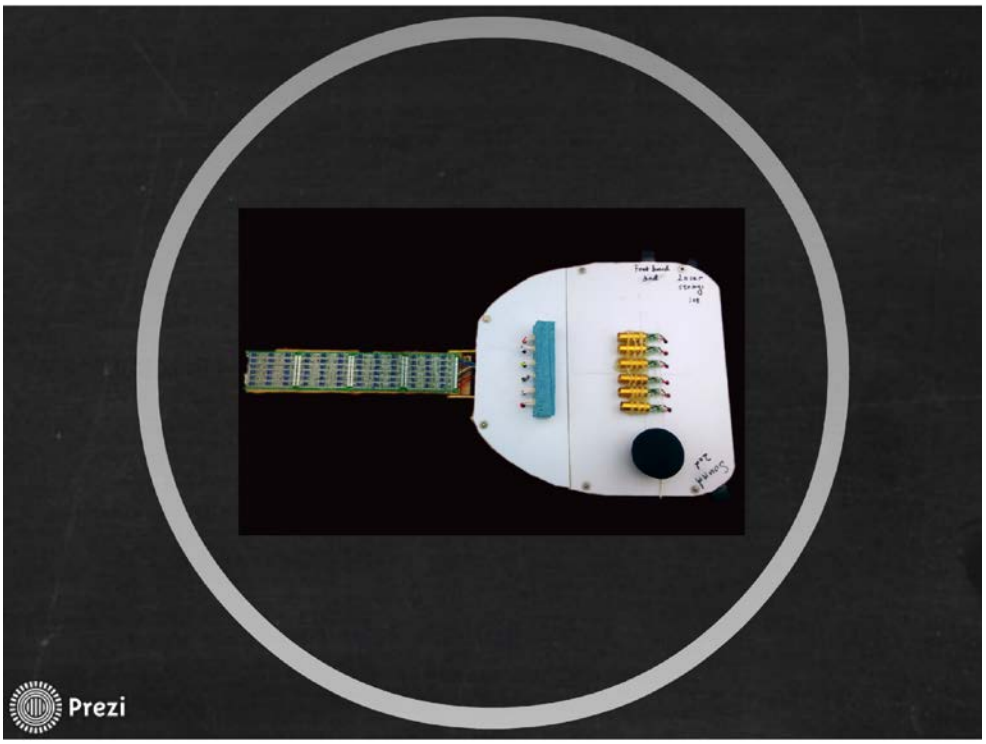

playByIndex(fileIndex)
Prezi

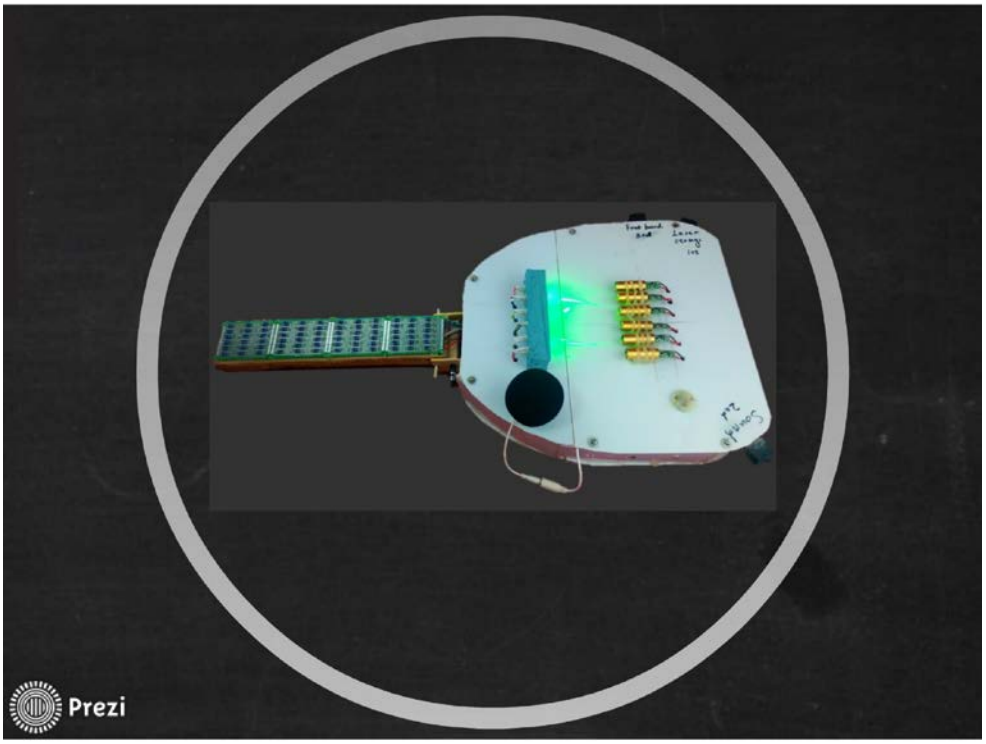

3.5 mm Audio output
Prezi

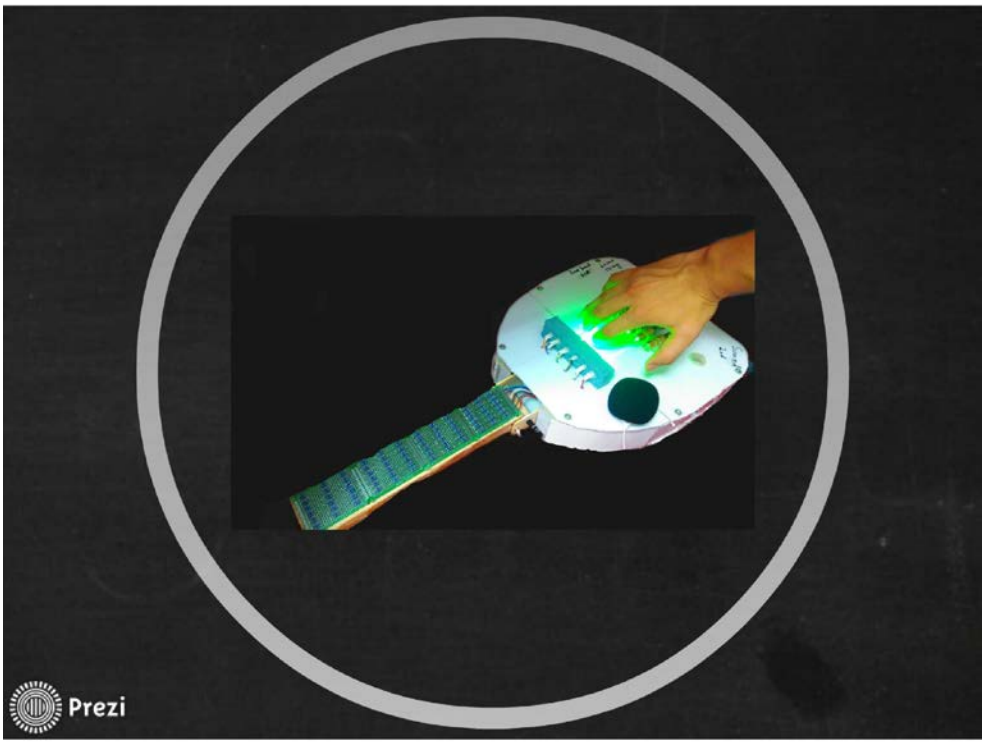

DEMO
Prezi

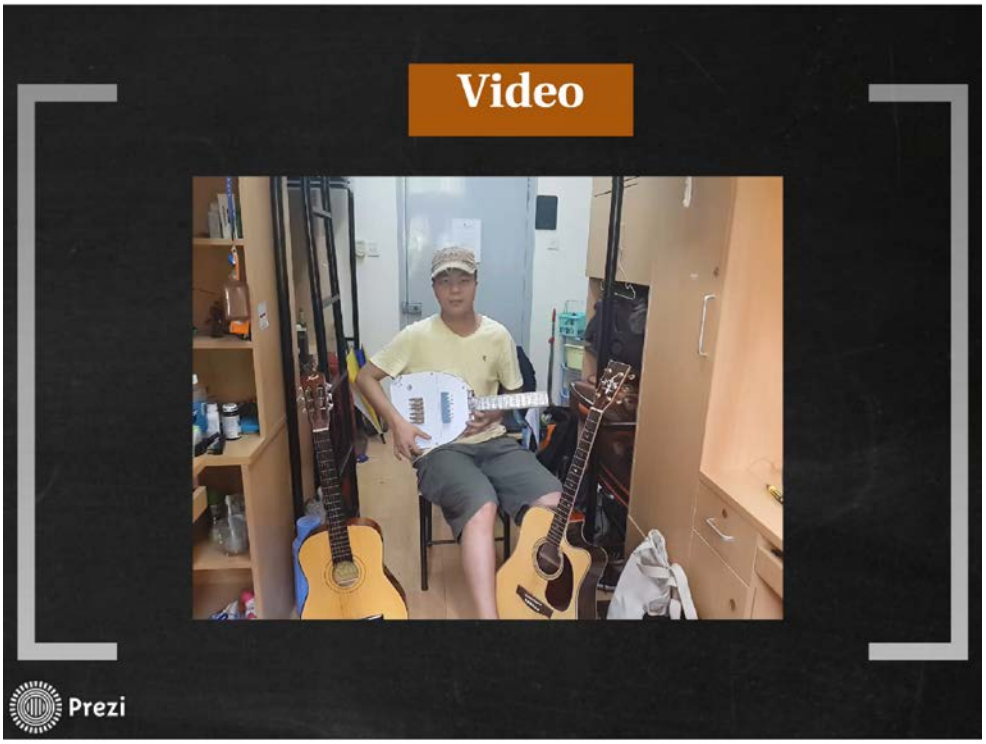

Prezi


Prezi

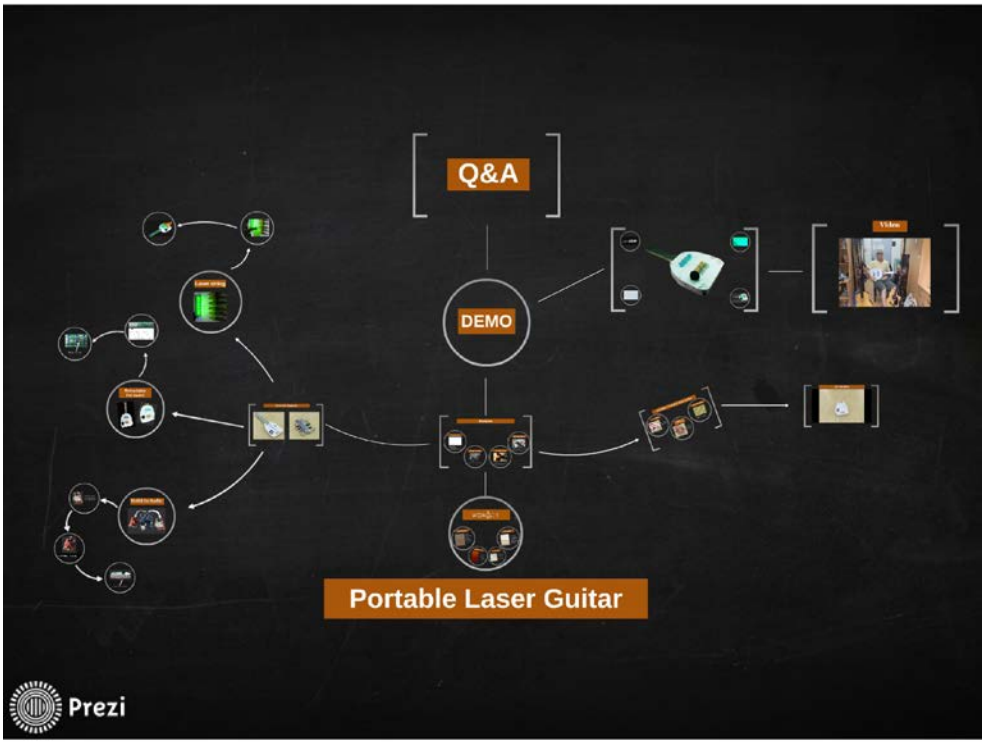

Prezi


Prezi


Prezi


Video
Prezi


Q&A
Prezi


Q&A
DEMO
Portable Laser Guitar
Prezi