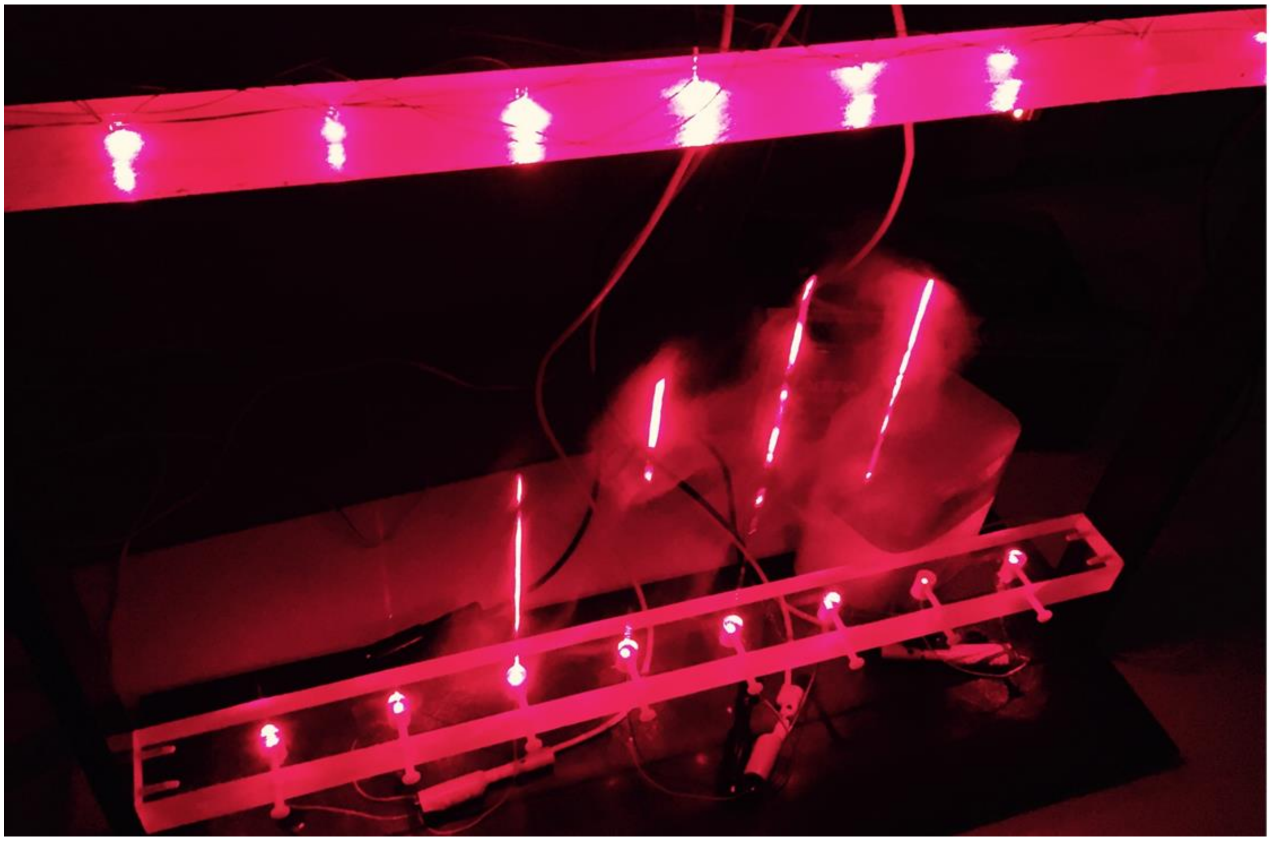


LASER HARP

An instrument played by blocking lasers. Sound is made either by using the harp’s built-in synthesizer, or by plugging in an external synthesizer.

Edwin Rodriguez
Peter Crinklaw
Qiushi Jiang

Abstract

The Laser Harp is an instrument with laser strings. When a laser on the harp is blocked, this action is detected by the software on a Nios II processor inside an Altera DE2 board. The software determines which laser is blocked and generates a MIDI signal according to MIDI 1.0 standard. The note is then sent to the synthesizer component which calculates the note frequency used to create a digital audio signal. This digital audio signal is sent to the audio core on the DE2 board, causing sound to be transmitted through the 1/8’’ line out audio jack on the board. Beside the internal synthesizer, the Laser Harp also has a MIDI output jack which can be connected to other external synthesizers.

Table of contents

Table of Contents

Functional requirements of project

The functional requirements of this project that were met are as follows:

1. The instrument:
   a. has 8 laser strings to allow a broad spectrum of playable notes.
   b. is adjustable so that the laser strings could be aligned to the photodiodes.
   c. is clean and robust so that the instrument can be used for faculty events.
   d. has a very low latency.
   e. can switch between Major, Minor, Blues and Doran scales
   f. can change base key

2. The internal synthesizer:
   a. can produce sound through the line-out jack
   b. can produce notes of any pitch.
   c. produces a clean sine wave.

3. The external interface:
   a. produces MIDI signals which follow MIDI 1.0 standard.
   b. sends the signals to a MIDI jack.
   c. produces the correct signals for the MIDI jack (Serial information, power, …)

The presented implementation of the Laser Harp met most of the requirements that were laid out for it. However, the synthesizer was not able to play multiple notes at once, and did not have a sound envelope. Due to problems with integration at a late stage of the project, these requirements were not implemented as efforts were diverted to create a working build of the instrument.

Design and description of operation

The system’s overall goal is to read the laser inputs and generate sound accordingly. It is divided into two main parts. The **MIDI signal generator** produces MIDI messages according to the laser inputs. The **Synthesizer** reads these MIDI messages and sends an audio signal to the DE2’s line-out jack.

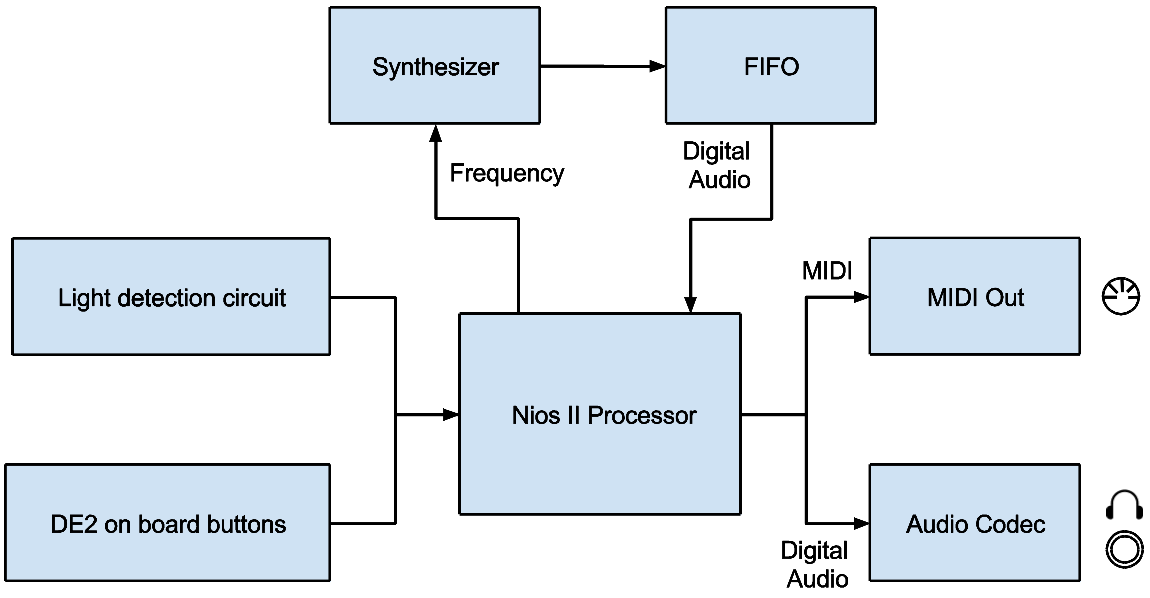


Figure1. System Block Diagram

The laser input is detected by photodiodes. The voltage across the photodiodes increases when a laser is pointed directly at them. The voltage reading is 0.2V for ambient light and 0.5V for laser light. This voltage is amplified by with a comparator circuit implemented using op-amps. The comparator circuit sets the voltage threshold to 0.35V. When the input is higher than 0.35V, the op-amp drives the output to VDD, otherwise to VEE. In our case VDD is set to 3.3V, VEE is set to ground. The output from the op-amp is read by the DE2 GPIO pins through the **Laser PIO** component. The MIDI generator component reads this data and construct MIDI message according to the user selected music scale.

The MIDI generator is notified of laser blockages by interrupts. When an interrupt is received, the MIDI signal generator task begins to construct a MIDI message. It uses a memory mapped interface to write this message to its **MIDI Serializer.** This produces a signal that can be interpreted by a third-party MIDI sink device. It also invokes the synthesizer's software component, passing it the MIDI message (in particular, the note pitch).

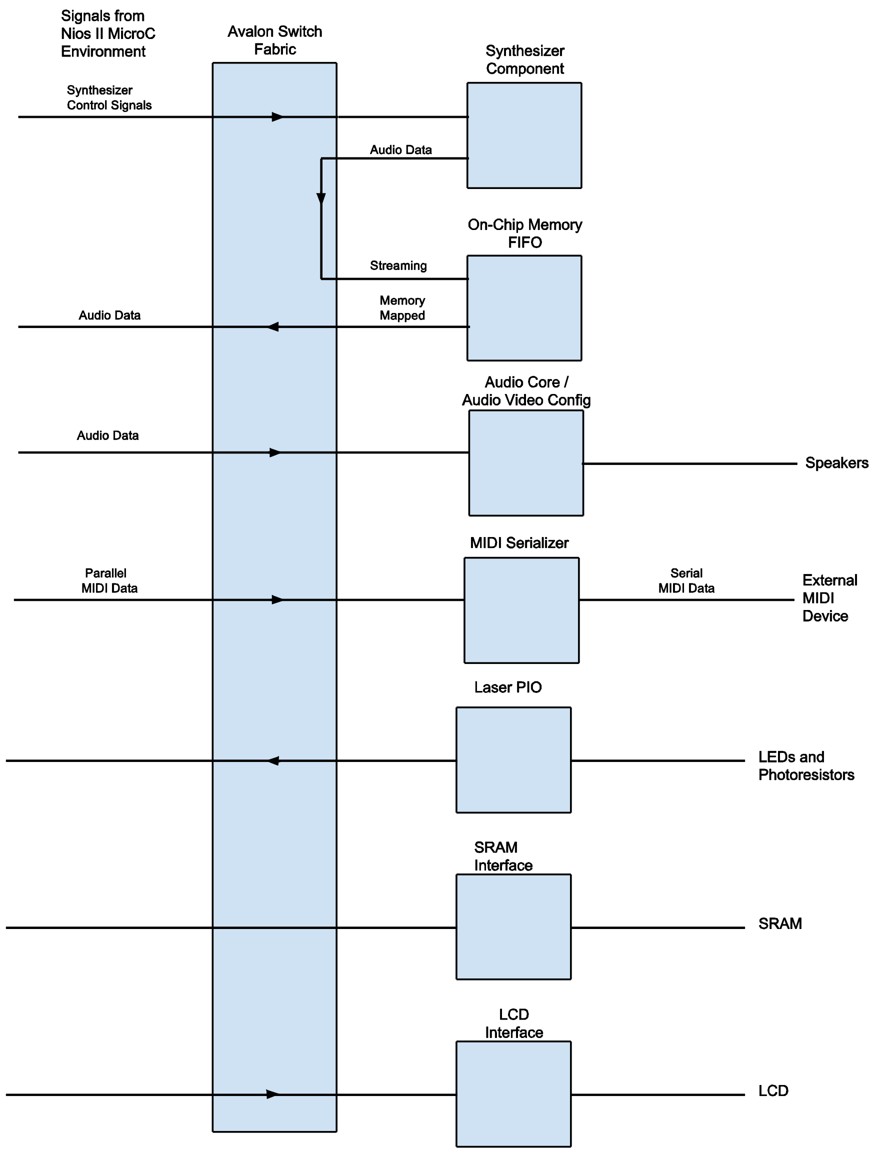


Figure2. Hardware Overview

The synthesizer software component reads the MIDI signals and sends commands to the synthesizer hardware. It receives instructions from the MIDI generator through ordinary C-style function calls. While not dealing with MIDI messages, the audio task reads the audio signal produced by the hardware component through an Altera FIFO and pushes it to the Audio Core. The Audio Core is a university program provided by Altera, which interfaces with the digital-analog converter on Altera DE2 board. By using a FIFO, the software is allowed to abandon the audio task for a short period of time without risking corruption in the audio data.

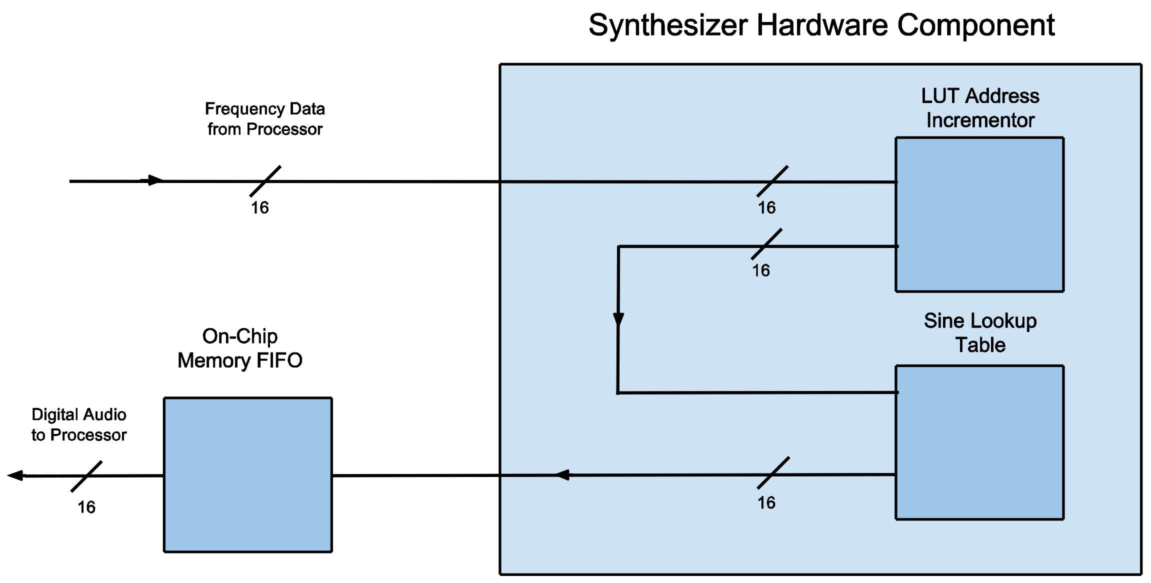

Synthesizer Hardware Component
Frequency Data from Processor
16
16
LUT Address Incrementor
16
Sine Lookup Table
On-Chip Memory FIFO
Digital Audio to Processor
16
16

Figure3. Synthesizer Hardware

The synthesizer hardware component takes inputs specifying the frequency of the wave that will be generated. It receives this input through from the NIOS II processor through a memory mapped interface. The frequency is represented as a phase increment, which tells the **LUT Address Incrementor** how much to increment its internal counter by every clock cycle. Frequency is related to phase increment by the following formula:

$$\frac{frequency}{max\ frequency} = \frac{phase\ increment}{max\ phase\ increment}$$

The maximum frequency is the audio output rate (32 kHz) and the maximum phase increment is 2^16 (as it is 16 bits wide). The value of the LUT Address Incrementor’s counter is used to address the **Sine Lookup Table**. The larger the phase increment, the faster the counter increments, and the faster the Sine LUT is stepped through. A large phase increment is basically the difference between outputting *sin(*5*t)* as opposed to *sin(t)*, thus resulting in a higher frequency output wave. The Sine LUT produces a 16 bit audio signal, which is streamed to the On-Chip FIFO memory core, which can be read by the processor through a memory mapped interface.

An expanded version of the synthesizer was created that allowed for multiple notes to be played at once. It contained 3 address incrementors per laser, the outputs of each being fed into either the sine lookup table, a square wave generator, or a sawtooth wave generator. The 24 resulting waveforms were summed together, producing a more complex audio signal. Due to time constraints, this version could not be integrated with the rest of the system.

Bill of materials

5mW Red Dot Laser Diode Module

Part Number: SKU: 154145

Quantity: 20

Unit Cost: USD 1.34

Total Cost: USD 26.80

Specs: Output laser power: < 4mW

Wavelength: 650nm

Working Voltage: 3V

Certification: IIIC

Supplier Link:http://www.dx.com/p/150mw-red-dot-laser-diode-module-154145#.VSSOTpN0fXo

Datasheet: N/A

PIN Silicon Photodiode Type OP906

Part Number:	***Manufacturer*:**	OP906
	Supplier:	365-1084-ND

Quantity: 20

Unit Cost: USD 0.61

Total Cost: USD 12.20

Specs: Spectral Range: 500nm ~ 1100nm

Wavelength: 935nm

Response: 5 ns

Current: 1 nA

Wavelength: 650nm

Working Voltage: 3V

Certification: IIIC

Supplier Link:http://www.digikey.com/product-detail/en/OP906/365-1084-ND/498710

Datasheet: http://optekinc.com/datasheets/OP906.PDF

Infrared Proximity Sensor - Sharp GP2Y0A21YK

Part Number: Sharp GP2Y0A21YK

Quantity:	1
Unit Cost:	USD 13.95
Total Cost:	USD 13.95
Specs:	Detecting distance: 10cm to 80 cm
Supplier Link:	https://www.sparkfun.com/products/242
Datasheet:	https://www.sparkfun.com/datasheets/Components/GP2Y0A21YK.pdf

MIDI Connector - Female Right Angle

Part Number:	PRT-09536
Quantity:	2
Unit Cost:	USD 1.95
Total Cost:	USD 3.90
Supplier Link:	https://www.sparkfun.com/products/9536
Datasheet:	https://www.sparkfun.com/datasheets/Prototyping/Connectors/MIDI-RA.pdf

OP-AMP

Part Number:	MC33174P
Quantity:	3
Unit Cost:	USD 0.48
Total Cost:	USD 1.44
Supplier Link:	ECE Department
Datasheet:	http://www.onsemi.com/pub_link/Collateral/MC33171-D.PDF

10 KΩ Resistor

Part Number:	N/A
Quantity:	1
Unit Cost:	USD 0.11
Total Cost:	USD 0.11
Supplier Link:	ECE Department
Datasheet:	N/A

1 KΩ Resistor

Part Number:	N/A
Quantity:	3
Unit Cost:	USD 0.11
Total Cost:	USD 0.33
Supplier Link:	ECE Department
Datasheet:	N/A

220 Ω Resistor

Part Number:	N/A
Quantity:	1
Unit Cost:	USD 0.11
Total Cost:	USD 0.11
Supplier Link:	ECE Department
Datasheet:	N/A

8 Pin IC Socket

Part Number:	N/A
Quantity:	2
Unit Cost:	USD 0.18
Total Cost:	USD 0.36
Supplier Link:	ECE Department
Datasheet:	N/A

16 Pin IC Socket

Part Number:	N/A
Quantity:	3
Unit Cost:	USD 0.25
Total Cost:	USD 0.75
Supplier Link:	ECE Department

Datasheet: N/A

Altera/Terasic DE2 Development Board

Part Number: P0301

Quantity: 1

Unit Cost: USD 495.00

Total Cost: USD 495.00

Supplier Link: http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=53&No=30

Datasheet: http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=53&No=30&PartNo=4

Perforated Board

Part Number: N/A

Quantity: 1

Unit Cost: USD 7.50

Total Cost: USD 7.50

Supplier Link: N/A

Datasheet: N/A

40-pin Ribbon Cable

Part Number: N/A

Quantity: 1

Unit Cost: USD 10.00

Total Cost: USD 10.00

Supplier Link: N/A

Datasheet: N/A

16-pin Ribbon Cable

Part Number:	N/A
Quantity:	1
Unit Cost:	USD 10.00
Total Cost:	USD 10.00
Supplier Link:	N/A
Datasheet:	N/A

Machine Shop Materials and Frame Build Costs

Part Number:	N/A
Quantity:	2 Hours
Unit Cost:	CAD 50.00 / Hour
Total Cost:	CAD 100.00
Supplier Link:	N/A
Datasheet:	N/A

Screws and Plastic Feet for Perforated Board

Part Number:	N/A
Quantity:	4
Unit Cost:	USD 0.02
Total Cost:	USD 0.08
Supplier Link:	N/A
Datasheet:	N/A

Total Project cost:	USD 682.53
Project Supplier:	ECE Department

Reusable Design Units

2012 Application Notes: audio_altera_university_ip_cores

This application was used to set up the audio core as well as guide the team on the basic functionality of the audio core. We had some problems in transferring the application note from SOPC to QSys due the differences in the programs. To resolve the issues we needed to add a 27.0 KHz clock to audio core.

2012 Voice Manipulator Project

The synthesizer from this project was deconstructed and some of its VHDL components and structure were adapted into our synthesizer.

Datasheet

Performance

MIDI messages take 2ms to process, allowing 500 laser blockages to be processed per second.

The software pipes 32K 16-bit audio data per second to the D/A converter.

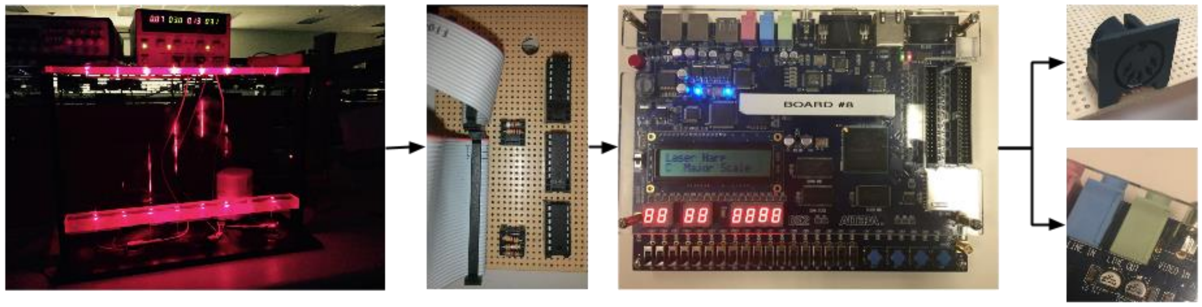


Figure4. User Perspective Block Diagram

Operating Conditions

The Laser Harp can be operated in any environment but its functionality is dependent on the amount of ambient light which is seen by the photodiodes. The Laser Harp will not function if the photodiodes detect that the ambient light is sufficiently bright enough to produce a voltage equal or higher to the pre-set threshold of 3.2 V. For this reason it is recommended that the laser harp is played in an indoor environment or a dark environment so that the ambient light does not interfere with the pre-set threshold voltage. This also maximizes the look and appeal of the Laser Harp.

Power Usage

Idle Power: 3.15 mA * 9 V = 28.35 mW

Standby Power: 3.15 mA * 9 V = 28.35 mW

Peak Power: 5.05 mA * 9V = 45.45 mW

I/O Description

Type of Signal	Description	Pin
Input Button	**Change Scale**	**Key[2]**
Input Button	**Change Key**	**Key[1]**
Input GPIO	**Laser signals**	**GPIO_1[0-7]**
Output GPIO	**MIDI Out**	**GPIO_1[9]**
Output GPIO	**5V power supply for lasers**	**GPIO_1[10]**
Output GPIO	**3.3 V power supply for op-amp**	**GPIO_1[28]**
Output GPIO	**Ground**	**GPIO_1[11]**
Output	**Line out for synthesizer**	**J3_Line_Out**

Background Reading

Understanding MIDI - The key to creating and conducting the music to your own MTV video

- IEEE Potentials Volume 13 Issue 1
- Quick overview of the MIDI standard, useful for the MIDI component of our system.

Sound on Sound Magazine: Synth Secrets series

- A series explaining the mechanics of what makes music sound the way it does, explaining concepts such as reverb and sound envelopes. This will useful for getting our synthesizer component to sound as nice as possible.

Interfacing a processor core in FPGA to an audio system

- Master thesis which explains the use of the audio codec for the NIOS II system.
- http://www-ug.eecg.toronto.edu/msl/nios_devices/dev_audio.html

Software design

The Laser Harp software is composed of the main software package and two supporting drivers for the MIDI generator and the synthesizer. These software files work with the with the hardware harmoniously and allow the harp to have a simple design with low latency response.
The main software package has three main tasks: laser input task, UI task (LCD), and audio task. Laser input task listens to changes in the lasers through interrupts. Once a laser interrupt is detected, the task calls the MIDI driver and starts processing the note. The UI task updates the LCD screen with the base key and scale that are currently selected. The audio task reads in the contents of a FIFO (which is populated by the synthesizer) and sends this data to the to the Audio Core. The synthesizer produces a *signed* 2's complement audio signal while the audio core expects an *unsigned* signal. A conversion is required in order to correctly reproduce the sound. This conversion is down by adding 0x8000 to all the elements inside the FIFO. While the audio *could* be streamed directly from the synthesizer to the audio core with no software involvement, we chose to do it this way in order to gain the reliability of using Altera's sanctioned software API.

The MIDI driver's role is to map the lasers to MIDI note numbers and to serialize the MIDI note numbers into MIDI messages. Notes are serialized by writing a 30 bit MIDI message to a MIDI out VHDL component using a memory mapped interface.

The synthesizer driver has two main functionalities. First it calculates the note frequency which is used to calculate the phase increment of the note. This phase increment is then sent to the synthesizer using a memory mapped interface. Secondly, the driver helps manage the incoming notes from the harp. It does this by implementing a note dispatcher that controls which notes are played. If the dispatcher is full, it drops the oldest note and adds the current note to the dispatcher.

Test plan

Software

Each VHDL component was tested using a testbench with ModelSim. Additionally, ModelSim simulations were used to test the behaviour of the system under several key test cases including:

- Multiple lasers played at the same time which was used to determine how the system operates under parallel inputs
- Rapid and consecutive input sequences from the lasers which was used to test the system for backlog data.
- Rapid input from many lasers which was used to determine if the system held under extreme conditions.

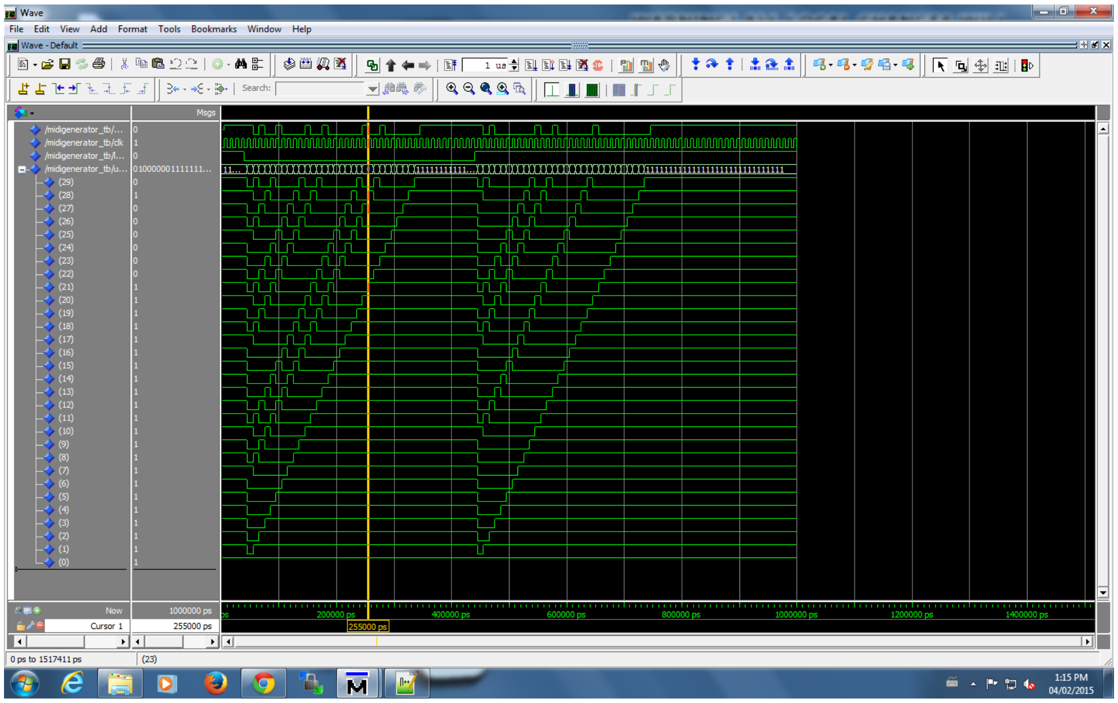
Software components were tested in two stages. First the software fragments were tested with known input data and the resulting output was compared to the expected output data. Successful fragments were integrated into the main build and the components were tested again under similar environments (known data inputs). Secondly, once the hardware was integrated, the software was run and tested under real conditions.

Hardware

The brightness of each laser when powered with 3.3V was measured to confirm that all were below 5mW, where eye damage would start to become a concern.
The photodiodes were tested before being mounted onto the frame. This was done by measuring the voltage through the photodiode under ambient light, aligned laser, and misaligned laser conditions. Once this information was known, the lasers and photodiodes were mounted onto the harp. One final voltage reading from every photodiode was taken to ensure that all aligned and slightly misaligned lasers were producing an acceptable voltage.

Results of experiments and characterization

For the MIDI generator, we have created a VHDL testbench and run simulations with modelSIM. Figure 5 demonstrates the typical scenario that occurs when a singular laser is blocked and then unblocked. In response, the shift register is loaded with a 30 bit MIDI message, which is shifted out.



	Msgs
/midigenerator_tb/...	0
/midigenerator_tb/clk	1
/midigenerator_tb/l...	0
/midigenerator_tb/u...	010000001111111...
(29)	0
(28)	1
(27)	0
(26)	0
(25)	0
(24)	0
(23)	0
(22)	0
(21)	1
(20)	1
(19)	1
(18)	1
(17)	1
(16)	1
(15)	1
(14)	1
(13)	1
(12)	1
(11)	1
(10)	1
(9)	1
(8)	1
(7)	1
(6)	1
(5)	1
(4)	1
(3)	1
(2)	1
(1)	1
(0)	1
Now	1000000 ps
Cursor 1	255000 ps

Figure5. MIDI Signal Generator Test Bench Output

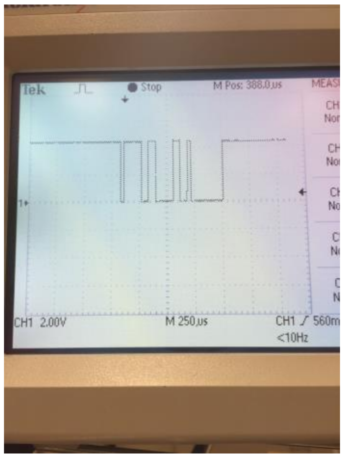
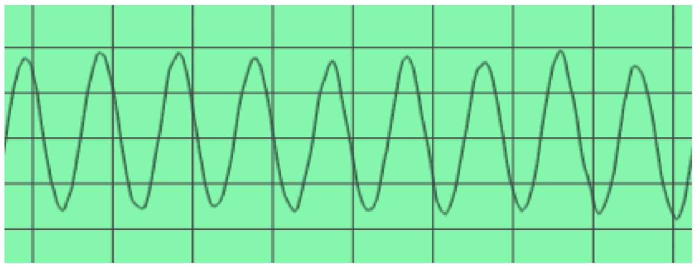


Figure6. MIDI Signal Generator Oscillator Output

The DE2's audio component, the Wolfson WM8731 Audio Codec, was first tested using a Quartis project built by Sean Hunter, Michael Wong, and Thomas Zylstra. Our system was able to send audio output to the the line-out, using the mic-in as inputs. When our synthesizer was used as input, an audio signal was able to be produced. Initially, the wave was not a proper sine wave. This was found to be due to failure to properly handle the conversion from signed audio data to unsigned data. After adjustments were made, a clean sine wave was produced. This was verified using an Android app, "Spectrum Analyzer", where the following image was taken from.



Synthesizer's audio output

In testing the software components, several problems arose. Memory corruption problems were occurring quite frequently. These were tested by comparing the memory map from a working build and compared to the broken build. Due to the unknown nature of these incidents, the memory problem was not solved. As a work around, print statements had to be inserted as padding for our code preventing our programs' memory from being overwritten at runtime.

Each laser's brightness at 3.3V was found to be well below 5mW, averaging about 2mW.

Safety

Lasers with a power output less than 5mW lasers are used in this project. This will ensure that we maintain our project with a class IIIC classification. This means that an exposure time of ~0.25 seconds will cause some damage to a human eye. It can be assumed however that a person will blink before an before an exposure time of ~0.25 seconds occurs and eye damage will not occur. Using this assumption will allow the project be considered “eye safe”.

Although the laser is “eye safe”, the following precautions will be taken:

- The output of each laser module was re-measured and compared to the datasheet.
- All metal and reflective components (where possible) were removed from the hardware.
- Laser mounts are insured to be adjustable and rigid when locked in a specified position.
- Team members took the required laser safety courses to operate the specified lasers.
- Lasers are positioned so that they are not pointing towards the audience.
- The lasers are secured such that no two lasers will ever be pointed at the same location (This will result in a laser beam > 5mW)

Failsafe

Shut down all lasers, and speakers when any failures happens.

Max Voltage: 5V

One of the op-amps requires 5V, which is the highest voltage of any component in our system.

Max Power: 45.5mW

This occurs when lasers are triggered at a very high rate.

Stored Energy: None

No battery is used in this project.

Operating Temperature: 15 to +32 °C

Both max and min are limited by the DE2 board.

Environmental impact

RoHS Status :

The electronic devices/components used in the projects are:

- Altera DE2 Development and Education board
- Type OP906 Photodiode
- S-5 DIY Laser Diodes
- MC33171 Op-amp

All of the components listed above are in compliance with RoHS standard, and they do not contain Cadmium, Chromium, Lead, Mercury, Polybrominated Biphenyls, Polybrominated Diphenyl Ethers.

Other environmental Impact:

- Light pollution: The artificial light from the laser diodes could cause light pollution when misdirected.
- Energy Consumption : The device consumes electricity, more detail see sustainability section.

Sustainability

Power Consumption - Active

DE2 Power Consumption = 1 * 9V * 1.3A = 11.7W

Laser Power Consumption = 8 * 3.3V * 3.0 mA = 79.2mW = 0.0792W

Op-amp Power Consumption = 8 * 3.3V * 250uA = 6.6 mW = 0.0066W

Speaker Power Consumption = 2 * 12V * 1.8A = 43.2W

(Assuming a pair of medium/small size computer speaker)

Total Power Consumption = 11.7W + 0.0792W + 0.0066W + 43.2W = 54.996W ≈ 55W

Power Consumption - Idle

Even when no one is playing the harp, the overall power consumption stays at 55W same as active mode.

Power Consumption - Sleep

N/A, the device does not support any type of sleep mode.

Energy Cost per Year

- Theoretical Maximum Consumption
  55W * 3600 seconds/hour * 24 hours / day * 365 days / year = 1734480000 J = 481.8kWh

- Assuming Two Hours Daily Usage
  55W * 3600 seconds/hour * 2 hours / day * 365 days / year = 144540000J = 40.15 kWh

CO2 Production per Year

Emission Factor: 6.8927 × 10^-4 metric tons CO2 / kWh (Reference 3)

- Theoretical Maximum Consumption
  481.8kWh * 6.8927 × 10-4^ metric tons CO2 / kWh = 0.3321 metric tons CO2

- Assuming Two Hours Daily Usage

40.15kWh * 6.8927 × 10-4^ metric tons CO2 / kWh = 0.02767 metric tons CO2

Area of Solar Cells Required in Edmonton

Annual power output of 1 m^2 solar panel in Edmonton: 1100 kWh/m^2 (Reference 4)

- Theoretical Maximum Consumption

  481.8kWh /1100 kWh/m^2 = 0.438 m^2

- Assuming Two Hours Daily Usage

  40.15kWh / 1100 kWh/m^2 = 0.0365 m^2

References

1 → The Laser Harp Pages, [Online]. Available:http://www.laserharp.org/print.php?type=A&item_id=5. Accessed Sunday March 1, 2015.

2 → The Association of Musical Electronics Industry, *MIDI Media Adaptation Layer for IEEE-1394.* http://www.midi.org/techspecs/rp27v10spec%281394%29.pdf . Accessed: Monday Feb 2, 2015.

3 → University of Alberta, Laser Safety Manual, http://www.ehs.ualberta.ca/en/LabPPE/~/media/Environmental%20Health%20and%20Safety/Documents/Radiation/Laser_Safety_Manual.pdf. Accessed: Monday Feb 23, 2015.

4 → United State Environmental Protection Agency, Green Power Equivalency Calculator Methodologies http://www.epa.gov/greenpower/pubs/calcmeth.htm Accessed: Monday March 2, 2015.

5 → Solargis, World Map of Global Horizontal Irradiation http://solargis.info/doc/_pics/freemaps/1000px/ghi/SolarGIS-Solar-map-World-map-en.png Accessed: Monday March 2, 2015.

Appendices

Quick start manual

A1.1 Re-assembling the Laser Harp

1. Plug in the 40 pins ribbon cable into DE2 board **GPIO_1** expansion header.
2. Power on DE2 board.
3. Play!

A1.2 Connecting IO to the Lab Computers

In order to connect the Laser Harp to the lab computers please do the following three steps:

1. Connect the 40-pin ribbon cable to Expansion Header 2 on the Altera DE2 board. Be sure that Pin 1 of the ribbon cable matches with pin 1 on Header 2.
2. (Optional) Connect a set of speakers to the line out jack on the Altera DE2 board in order to use the internal synthesizer.
3. (Optional) Connect the Laser Harp to an external MIDI synthesizer (such as the voice manipulator) if an external synthesizer is required.

A1.3 Configuring the Laser Harp for Demonstrations

Hardware:

In order to configure the Laser Harp simply download the project files from the course website. Once downloaded, open the niosII_microc_lab1.qpf file to open the Quartus II project. Program the board the the niosII_microc_lab1.sof file found in the output_files folder. If you need to flash the board, program the board with the niosII_microc_lab1.pof file. If problems occur and the board is unable to be program or flashed, rebuild the project by generating the niosII_system.qsys file with QSys. Once done, simply compile the project with Quartus II and re-program the board.

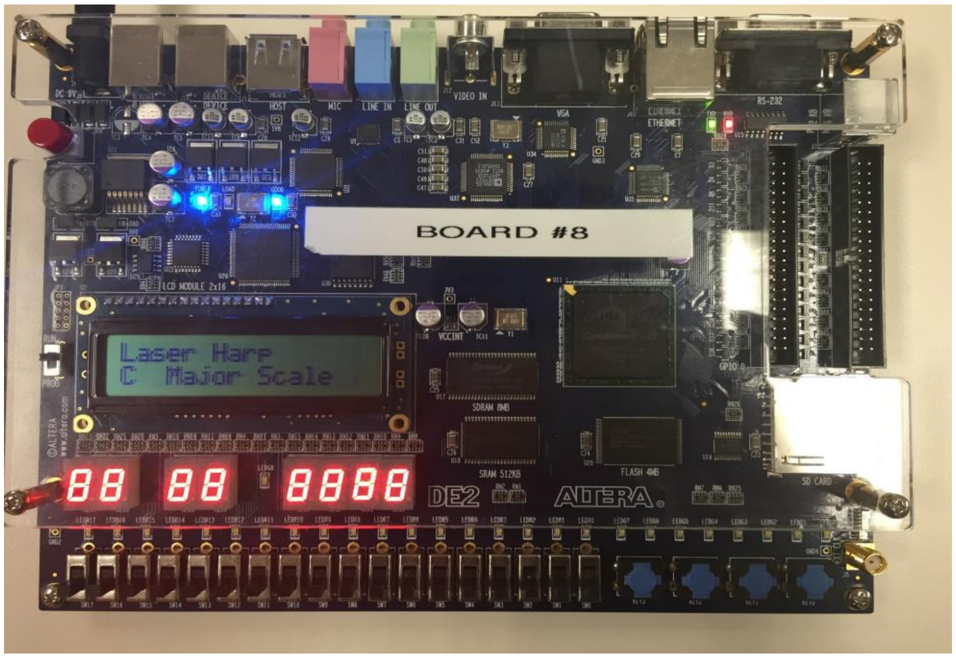
Software:

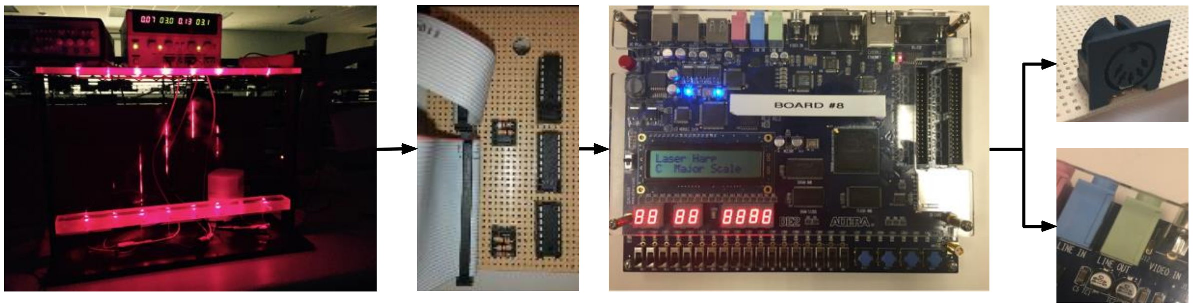
Open Nios II Software Build Tools for Eclipse and load the projects in the software folder. If Eclipse prompts you with errors, recreate the project and copy the MIDI_Driver.h, MIDI_Driver.c, Synthesizer_Driver.c, and Synthesizer_Driver.h files from the old project to your new project.

Lastly copy the contents of the “file_name” onto the new hello_worldII.c file and built/run the project.

A1.4 Laser Harp User Guide

Once the program is running you will see the Altera DE2 board in the following state:





The LCD screen will display the name of the current scale and the base note of the first laser. These values can be changed by using the built-in push buttons.

Key [1]: This key will change the music scale. The Laser Harp is currently programming to have the following four music scales:

Major [C D E F G A B C]
Minor [C D Eb F G Ab Bb C]
Blue [C Eb F Gb G Bb C]
Dorian [C D Eb F G A Bb C]

Key [2]: This key will change the base note of the first laser. Once a base note is changed, the note mapping of all the other lasers are also shifted so that the selected music scale is maintained. In order to keep the range of the notes within a user-friendly bound, the shift will wrap around F4 to A3.

A1.5 Additional Information for Open House Demonstrations

The harp has a simple monophonic synthesizer. In order for the Laser Harp to captivate audiences it is recommended that both the internal synthesizer and an external synthesizer (like the Voice Manipulator synthesizer) are played at the same time.

The ribbon cables on the laser interface circuit cannot be removed as we were not able to find a socket for the ribbon cables. Additionally the circuit is wired-wrapped so the interface board requires some care when transporting.

Appendices

Future work

- Make the synthesizer polyphonic, allowing more than one note to be played at a time.
- Add an envelope to the synthesizer, allowing notes to ring out.
- Add Sensors (IR/Pedal) to control octave shift of single/all lasers
- Add Sensors (IR/Pedal) to implement note velocity.
- Add more instrument support to the synthesizer.

Appendices

main.c

//Contains 3 main task, they are LaserInputTask, UITask and AudioTask.

```
#include <stdio.h>
#include "includes.h"
#include "altera_avalon_fifo_util.h"
#include "altera_avalon_fifo_regs.h"
#include "altera_avalon_fifo.h"
#include "altera_up_avalon_audio.h"
#include "altera_up_avalon_audio_and_video_config.h"
#include "altera_avalon_pio_regs.h"
#include "sys/alt_irq.h"
#include "altera_up_avalon_character_lcd.h"

#define   BUFFER_SIZE      128

/* Definition of Task Stacks */
#define   AUDIO_TASK_STACKSIZE   100000
#define   LASER_INPUT_TASK_STACKSIZE  5000
#define   UI_TASK_STACKSIZE 5000

/* Definition of Task Priorities */
#define LASER_INPUT_TASK_PRIORITY     1
#define UI_TASK_PRIORITY              2
#define AUDIO_TASK_PRIORITY           3

OS_STK    audioTaskStack[AUDIO_TASK_STACKSIZE];
OS_STK    laserInputTaskStack[LASER_INPUT_TASK_STACKSIZE];
OS_STK    uiTaskStack[UI_TASK_STACKSIZE];

OS_EVENT * laserStatusChanged;
OS_EVENT * settingChanged;

int scaleIndex = 0;
int baseTranspose = 0;
/* Prints "Hello World" and sleeps for three seconds */
void audioTask(void* pdata){
     INT8U err;
    alt_up_audio_dev * audio_dev;
    alt_up_av_config_dev * audio_config_dev;
    volatile unsigned int level;
    unsigned int buffer[BUFFER_SIZE];
    int i = 0;

    audio_config_dev =
alt_up_av_config_open_dev("/dev/audio_and_video_config_0");
    if ( audio_config_dev == NULL)
        printf("Error: could not open audio config device \n");
    else
        printf("Opened audio config device \n");

    /* Open Devices */
    audio_dev = alt_up_audio_open_dev ("/dev/audio_0");
    if ( audio_dev == NULL)
        printf("Error: could not open audio device \n");
    else
        printf("Opened audio device \n");

    /* Configure WM8731 */
    alt_up_av_config_reset(audio_config_dev);
    alt_up_audio_reset_audio_core(audio_dev);

    /* Write to configuration registers in the audio codec; see
datasheet for what these values mean */
    alt_up_av_config_write_audio_cfg_register(audio_config_dev, 0x0,
0x17);
    alt_up_av_config_write_audio_cfg_register(audio_config_dev, 0x1,
0x17);
    alt_up_av_config_write_audio_cfg_register(audio_config_dev, 0x2,
0x68);
    alt_up_av_config_write_audio_cfg_register(audio_config_dev, 0x3,
0x68);
    alt_up_av_config_write_audio_cfg_register(audio_config_dev, 0x4,
0x15);
    alt_up_av_config_write_audio_cfg_register(audio_config_dev, 0x5,
0x06);
    alt_up_av_config_write_audio_cfg_register(audio_config_dev, 0x6,
0x00);
     altera_avalon_fifo_init(FIFO_0_OUT_CSR_BASE, 0x0, 10,
FIFO_0_OUT_CSR_FIFO_DEPTH-10);
     altera_avalon_fifo_read_fifo(FIFO_0_OUT_BASE,
FIFO_0_OUT_CSR_BASE);

     while (1){
          level =
altera_avalon_fifo_read_level(FIFO_0_OUT_CSR_BASE);
```

```
            if (level>0){
                  for (i=0;i<level;i++){
                        buffer[i]=
((altera_avalon_fifo_read_fifo(FIFO_0_OUT_BASE,
FIFO_0_OUT_CSR_BASE)>>16) + 0x7fff)/2;
                  }
            }
         alt_up_audio_write_fifo (audio_dev, buffer, level,
ALT_UP_AUDIO_RIGHT);
         alt_up_audio_write_fifo (audio_dev, buffer, level,
ALT_UP_AUDIO_LEFT);
            OSTimeDlyHMSM(0, 0, 0, 1);
      }

}

/* Prints "Hello World" and sleeps for three seconds */
void laserInputTask(void* pdata){
      INT8U err;
      alt_u32* laserStatusPointer =(alt_u32* )SWITCH_BASE;
      alt_u32* midiOutPointer = (alt_u32*) MIDIOUT_0_BASE;
      alt_u32* voiceAddr =SYNTHESIZER_0_BASE;

      alt_u32 laserStatus = 0;
      alt_u32 previousLaserStatus = 0;
      alt_u32 differentBits = 0;
      int noteType;
      int laserIndex=0;

      while (1){
            OSSemPend(laserStatusChanged, 0, &err);
            previousLaserStatus = laserStatus;
      // save previous laserstatus
            laserStatus = 255 - *laserStatusPointer;
      // inverse 8bits input
            differentBits = previousLaserStatus^laserStatus;       //
XOR: 100 ^ 110 = 010

            for (laserIndex=0;laserIndex<8;laserIndex++){
                  //if laser i is changed
                  if( ((1 << laserIndex) & (differentBits)) == (1 <<
laserIndex) ){
                        //get notetype 1 on   0 off
                        noteType = ((laserStatus &  (1 << laserIndex)) ==
(1 << laserIndex)) ?  1: 0;
```

```
                        //write to midi out
                        *midiOutPointer =
generateMidiData(getPitch(scaleIndex,laserIndex)+baseTranspose,noteTyp
e);
                        //write to on board synth

      sendNoteSynthesizer(getPitch(scaleIndex,laserIndex)+baseTranspose
,noteType);
                  }
            }
      }
}

void uiTask(void* pdata){
      INT8U err;
      alt_up_character_lcd_dev * char_lcd_dev;
      // open the Character LCD port
      char_lcd_dev = alt_up_character_lcd_open_dev
("/dev/character_lcd_0");
      if ( char_lcd_dev == NULL){
            alt_printf ("Error: could not open character LCD
device\n");
      }
      else{
            alt_printf ("Opened character LCD device\n");
      }
      alt_up_character_lcd_init (char_lcd_dev);
      alt_up_character_lcd_cursor_off(char_lcd_dev);

      alt_up_character_lcd_set_cursor_pos(char_lcd_dev, 0, 0);
      alt_up_character_lcd_string(char_lcd_dev, "Laser Harp");

      while(1){
            OSSemPend(settingChanged, 0, &err);
            alt_up_character_lcd_set_cursor_pos(char_lcd_dev, 0, 1);
         switch(baseTranspose){
            case -5:
                  alt_up_character_lcd_string(char_lcd_dev, "G  ");
                  break;
            case -4:
                  alt_up_character_lcd_string(char_lcd_dev, "Ab ");
                  break;
            case -3:
                  alt_up_character_lcd_string(char_lcd_dev, "A  ");
                  break;
```

```
        case -2:
             alt_up_character_lcd_string(char_lcd_dev, "Bb ");
             break;
        case -1:
             alt_up_character_lcd_string(char_lcd_dev, "B  ");
             break;
        case 0:
             alt_up_character_lcd_string(char_lcd_dev, "C  ");
             break;
        case 1:
             alt_up_character_lcd_string(char_lcd_dev, "C# ");
             break;
        case 2:
             alt_up_character_lcd_string(char_lcd_dev, "D  ");
             break;
        case 3:
             alt_up_character_lcd_string(char_lcd_dev, "Eb ");
             break;
        case 4:
             alt_up_character_lcd_string(char_lcd_dev, "E  ");
             break;
        case 5:
             alt_up_character_lcd_string(char_lcd_dev, "F  ");
             break;
        case 6:
             alt_up_character_lcd_string(char_lcd_dev, "F# ");
             break;
      }


      switch(scaleIndex){
        case 0:
             alt_up_character_lcd_string(char_lcd_dev, "Major Scale
");
             break;
        case 1:
             alt_up_character_lcd_string(char_lcd_dev, "Minor Scale
");
             break;
        case 2:
             alt_up_character_lcd_string(char_lcd_dev, "Blue Scale
");
             break;
        case 3:
                alt_up_character_lcd_string(char_lcd_dev, "Doran Scale
");
                break;
        }

     }
}

static void laserChangeHandler( void * context){
     OSSemPost(laserStatusChanged);
     IOWR_ALTERA_AVALON_PIO_EDGE_CAP(SWITCH_BASE, 0xff);
}

static void button1Handler(void * context){
     scaleIndex ++;
     if (scaleIndex>3){
          scaleIndex = 0;
     }
     OSSemPost(settingChanged);
     IOWR_ALTERA_AVALON_PIO_EDGE_CAP(BUTTON_1_BASE, 0x1);
}
static void button2Handler(void * context){
     baseTranspose ++;
     if (baseTranspose>6){
          baseTranspose = -5;
     }
     OSSemPost(settingChanged);
     IOWR_ALTERA_AVALON_PIO_EDGE_CAP(BUTTON_2_BASE, 0x1);
}

/* The main function creates two task and starts multi-tasking */
int main(void){
     laserStatusChanged = OSSemCreate(0);
     settingChanged = OSSemCreate(1);
     alt_ic_isr_register(SWITCH_IRQ_INTERRUPT_CONTROLLER_ID, //alt_u32
ic_id
                              SWITCH_IRQ, //alt_u32 irq
                              laserChangeHandler, //alt_isr_func isr
                              NULL,
                              NULL);
     IOWR_ALTERA_AVALON_PIO_IRQ_MASK(SWITCH_BASE,0xff);
     IOWR_ALTERA_AVALON_PIO_EDGE_CAP(SWITCH_BASE, 0xff);

     alt_ic_isr_register(BUTTON_1_IRQ_INTERRUPT_CONTROLLER_ID,
//alt_u32 ic_id
```

```
                                BUTTON_1_IRQ, //alt_u32 irq
                                button1Handler, //alt_isr_func isr
                                NULL,
                                NULL);
     IOWR_ALTERA_AVALON_PIO_IRQ_MASK(BUTTON_1_BASE,0x1);
     IOWR_ALTERA_AVALON_PIO_EDGE_CAP(BUTTON_1_BASE, 0x1);

     alt_ic_isr_register(BUTTON_2_IRQ_INTERRUPT_CONTROLLER_ID,
//alt_u32 ic_id
                                BUTTON_2_IRQ, //alt_u32 irq
                                button2Handler, //alt_isr_func isr
                                NULL,
                                NULL);
     IOWR_ALTERA_AVALON_PIO_IRQ_MASK(BUTTON_2_BASE,0x1);
     IOWR_ALTERA_AVALON_PIO_EDGE_CAP(BUTTON_2_BASE, 0x1);

     OSTaskCreateExt(audioTask,
                        NULL,
                        (void *)&audioTaskStack[AUDIO_TASK_STACKSIZE-
1],
                        AUDIO_TASK_PRIORITY,
                        AUDIO_TASK_PRIORITY,
                        audioTaskStack,
                        AUDIO_TASK_STACKSIZE,
                        NULL,
                        0);

     OSTaskCreateExt(laserInputTask,
                        NULL,
                        (void
*)&laserInputTaskStack[LASER_INPUT_TASK_STACKSIZE-1],
                        LASER_INPUT_TASK_PRIORITY,
                        LASER_INPUT_TASK_PRIORITY,
                        laserInputTaskStack,
                        LASER_INPUT_TASK_STACKSIZE,
                        NULL,
                        0);

     OSTaskCreateExt(uiTask,
                        NULL,
                        (void *)&uiTaskStack[UI_TASK_STACKSIZE-1],
                        UI_TASK_PRIORITY,
                        UI_TASK_PRIORITY,
                        uiTaskStack,
                        UI_TASK_STACKSIZE,
                              NULL,
                              0);

       OSStart();
       printf(".");printf(".");printf(".");printf(".");printf(".");print
f(".");printf(".");printf(".");

       return 0;
}
```

Synthesizer VHDL

//Main control component for the synthesizer hardware component

```
library IEEE;
    use IEEE.std_logic_1164.all;
    use IEEE.std_logic_textio.all;
    use IEEE.std_logic_arith.all;
    use IEEE.numeric_bit.all;
    use IEEE.numeric_std.all;
    use IEEE.std_logic_signed.all;
    use IEEE.std_logic_unsigned.all;

use work.SynthesizerPackage.all;

entity Synthesizer is
      port (
      -- system signals
      clk         : in  std_logic;
      reset_n       : in  std_logic;

      -- Frequency control
      --Bottom 16 bits are for first oscillator, next 16 are for
second, next 16 are for third
      phase_increments          : in  PHASE_INCS;
      audio_output              : out std_logic_vector(15 downto 0);
      audio_output_valid        : out std_logic
      );
end Synthesizer;

architecture synthesizer of Synthesizer is

      signal target_lut_addresses      : LUT_ADDRESSES;
      signal audioData      : WAVE_ARRAY;

      component AddressIncrementor is
      port (
        -- system signals
        clk            : in  std_logic;
        reset_n        : in  std_logic;

        -- NCO frequency control
        phase_inc      : in  std_logic_vector(15 downto 0);

        -- Output waveforms
        lut_address    : out std_logic_vector(11 downto 0)
        );
```

```
	end component;

	component SinLut is
		port (
			clk      : in  std_logic;

			--Address input
			address  : in LUT_ADDRESSES;

			--Sine output
			audioData : out WAVE_ARRAY
		);
	end component;

	begin

	anAddressIncrementor: AddressIncrementor
		port map(
			clk => clk,
			reset_n => reset_n,
			phase_inc => phase_increments(0)(15 downto 0),
			lut_address => target_lut_addresses(0)
		);

	aSinLut: SinLut
		port map(
			clk => clk,
			address => target_lut_addresses,
			audioData => audioData
		);

	send_output: process(clk, reset_n)
	begin
		if rising_edge(clk) then
			audio_output <= audioData(0);
		end if;
	end process send_output;

	audio_output_valid <= '1';

end synthesizer;
```