Midi Wavetable Synthesis for CSEE 4840

Doga Ozesmi

Varun Varahabhotla

Lancelot Wathieu

Evan Ziebart

Table of Contents

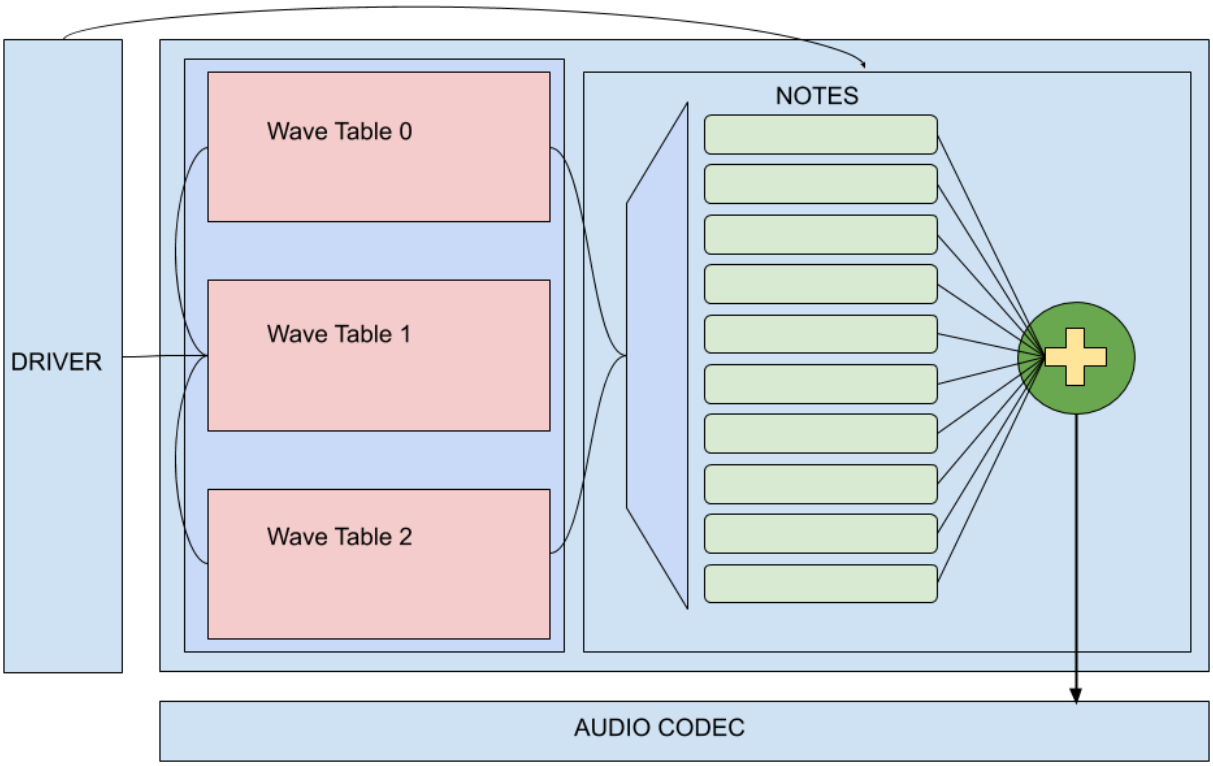
Overview

Summary

This project is an implementation of a digital wavetable synthesizer on the Terasic Cyclone DE1 board. When connected to our USB MIDI keyboard, and an audio output device such as a speaker or headphones, the board produces a sound of the corresponding pitch when a user presses a key, and the sound halts when a key is released.

In this project, there is logic to handle simultaneous key presses of up to 10 notes on the keyboard. The note sounds are implemented with enveloping and nonlinear summation to control volume. Ten wavetable signatures are available in software to represent different instrument sounds for the synth. Up to 2 of these can be active at once in the synth hardware, and they are swapped out via a software write. To further modulate the sound, a dial on the keyboard signals the software to update coefficients to determine the relative contributions of the 2 active tables. Detailed in below sections are each of the parts of the project.

Block Diagram



Wavetable Synth

Concept

In general, a wavetable synthesizer stores a signature of a sound wave in digital memory. This wave is sampled with varying gaps between sampled points in order to produce different frequencies for notes. Wavetable synthesizers are useful because there is no need to generate fundamental waves and spend time mixing these together to get desired sounds. Instead, the data is stored directly in memory and can be passed along with minimal processing.

Our design uses 16-bit sound and a sampling rate of 48kHz. Sampling is driven by the audio codec connected to the Cyclone board. When the codec requests a sample, our hardware device reads data from the wavetable, processes the signals, and then passes the sample to the audio codec to play. The number of 16-bit samples in the wavetables used by our device is the same as the sampling rate of the audio codec, which is 48k samples.

To produce different notes, the synth hardware skips over a varying number of samples among those stored in the table, producing different frequencies from the same wave signature. Our design accommodates notes in the range from C0 up to B8. Each of the notes has a different number of samples it should skip over after each sample in the wavetable. Twelve of these offsets are stored in a table for the notes in the highest supported octave. To implement lower octaves, these values are bit-shifted right.

Enveloping

Often, synthesizers use a feature called enveloping to vary the volume of a note which is being played as it advances through some duration. This is done for example to mimic the sound of a real instrument, which usually sounds loudest at the start, also known as the attack period. The note might then level off or decay from that point. Sometimes user inputs to the keyboard can control the shape and properties of the envelope as well.

Our design uses a simple version of an adsr envelope. ADSR stands for Attack, Decay, Sustain, Release. When a note is played, there is a period of volume increase from silence, labeled the attack. The sound reaches a peak, then begins a period of decreasing, called the decay. The sound levels-off at some amplitude for the sustain period. Finally, once a note is released, the sound gradually fades instead of instantaneously ending.The parameters here are the length of the attack, decay, and release periods, and the level of noise during the sustain period.

Summation

After sampling of many notes, these samples have to be added together. There are a few concerns with this however. It is likely that playing many notes will result in overflow if they are linearly added and if the range of each individual note is limited. So that this does not happen, the dynamic range of the synthesizer is immediately cut by a factor of 10. As a result there has to be a way to dynamically scale the amplitude of the sound signal. This is based both on how loud the sound wave is, and on how many notes are being played. Importantly, this scaling cannot be based on the actual amplitude of each sample because this will have a flattening effect on the note and introduce distortion similar to clipping.

Our solution was to scale each sample based on a product of (1) the volume (velocity) of each note and (2) the current enveloping coefficient. This takes into account the relative volumes of each note depending on where they are in their envelope and produces a weighted sum. If instead the summation used the total number of notes played, there could be unwanted transitions in volumes. For example, if a note is hit and then during its release a second note is hit, the total amplitude would be scaled as if 2 notes were being played but then as soon as the first note died out the overall volume would actually increase because the scale would revert to a single note coefficient.
Before 1st note dies out:
Note 1 amplitude @ t1 during its release = 10
Note 2 amplitude @ t1 during it's sustain = 100
Total amplitude = 110 * 0.75 (75% volume for 2 notes) = 82.5
After 1st note dies out:
Note 1 has died out @ t2, amplitude = 0 and note is not counted towards total
Note 2 amplitude @t2 still in sustain = 100
Total = 100*1(100% volume for one note) = 100

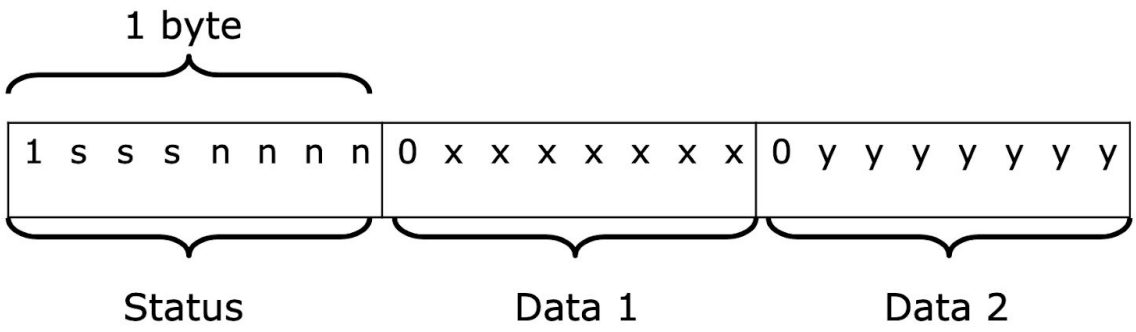
As shown in this example, not properly weighting the notes can cause situations where releasing a note, something that should make the sound quieter will actually result in an increase in volume. As a result, using the volume * envelope coefficient product allows us to properly account for the relative volumes of notes while not relying on the actual sample value which will change wildly.

MIDI Conversion

A MIDI packet reports an event that has taken place with regards to the user's interaction with the MIDI instrument, such as a key press, volume adjustment, modulation, etc. A message is a single 64 byte data packet where the first byte representing the status, the second is the first set

of data(usually an identifier of a sub-key), and the third data byte generally is magnitude, 0-127, with buttons being represented by 0 for off and 127 for on.

There are two kinds of bytes received as a part of a MIDI message: status bytes and data bytes. Status bytes always appear at the beginning and have a 1 in the most significant bit. Their purpose is to report the type of message being sent. Data bytes are arguments to the status byte and always have a 0 in the most significant bit. If the status byte remains unchanged form the previous message, a MIDI instrument will often omit the status byte to save time. In this case, the status is interpreted as the last status received from the instrument, also called the “running status”.



For our purposes, we only care about a few of the MIDI packets which are key press, release, modulation, volume control, and several keys remapped to allow for waves to be sent in.

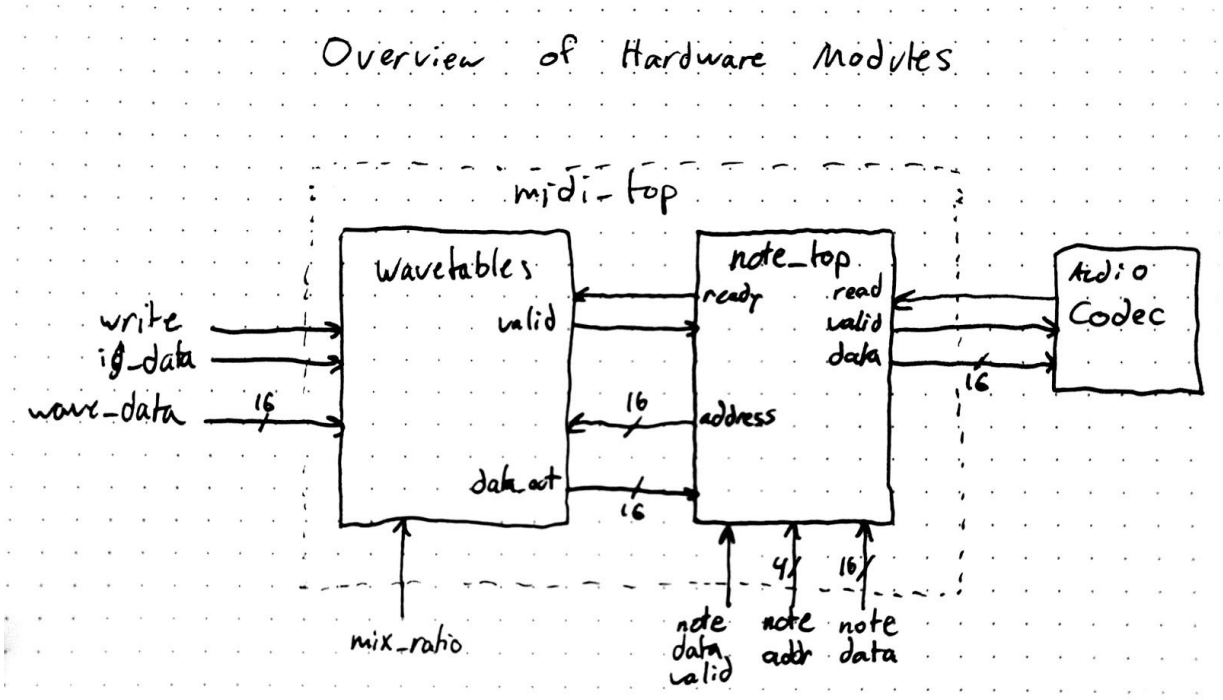
The format for key press and key release are shown below.

Event	**Status**	**Data**
Key Press	1000nnnn	0kkkkkkk 0vvvvvvv
Key Release	1001nnnn	0kkkkkkk 0vvvvvvv

Here, kkkkkkk says which key is being affected and vvvvvvv reports the velocity at which it is pressed/released. Additionally, nnnn reports the channel on which the note change should take affect.

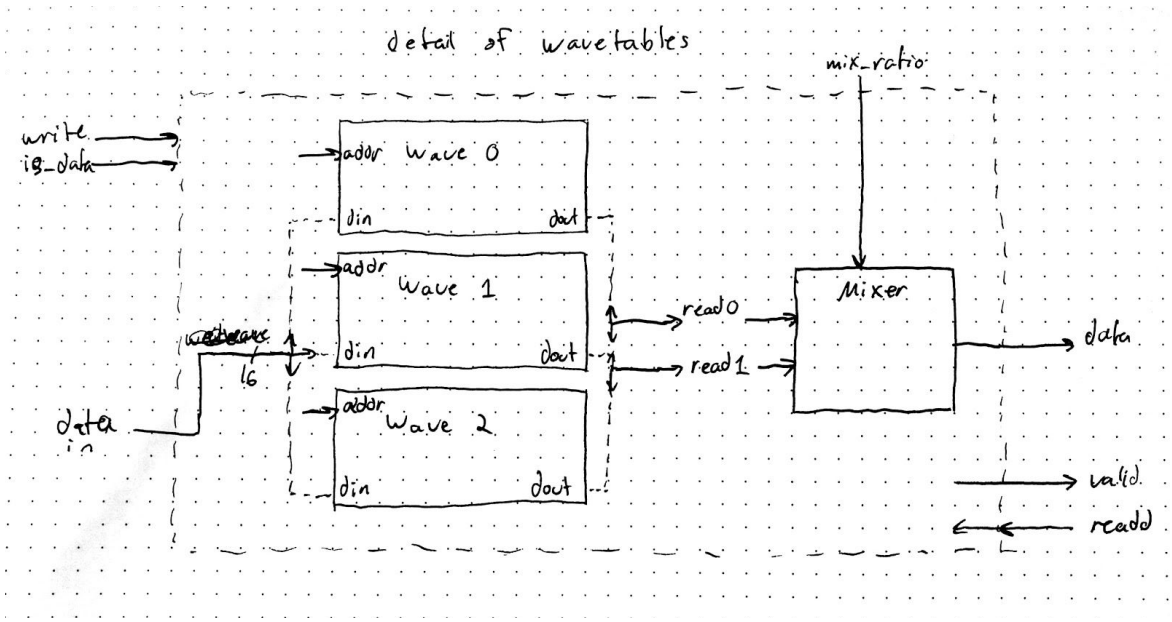
Hardware

Midi Top



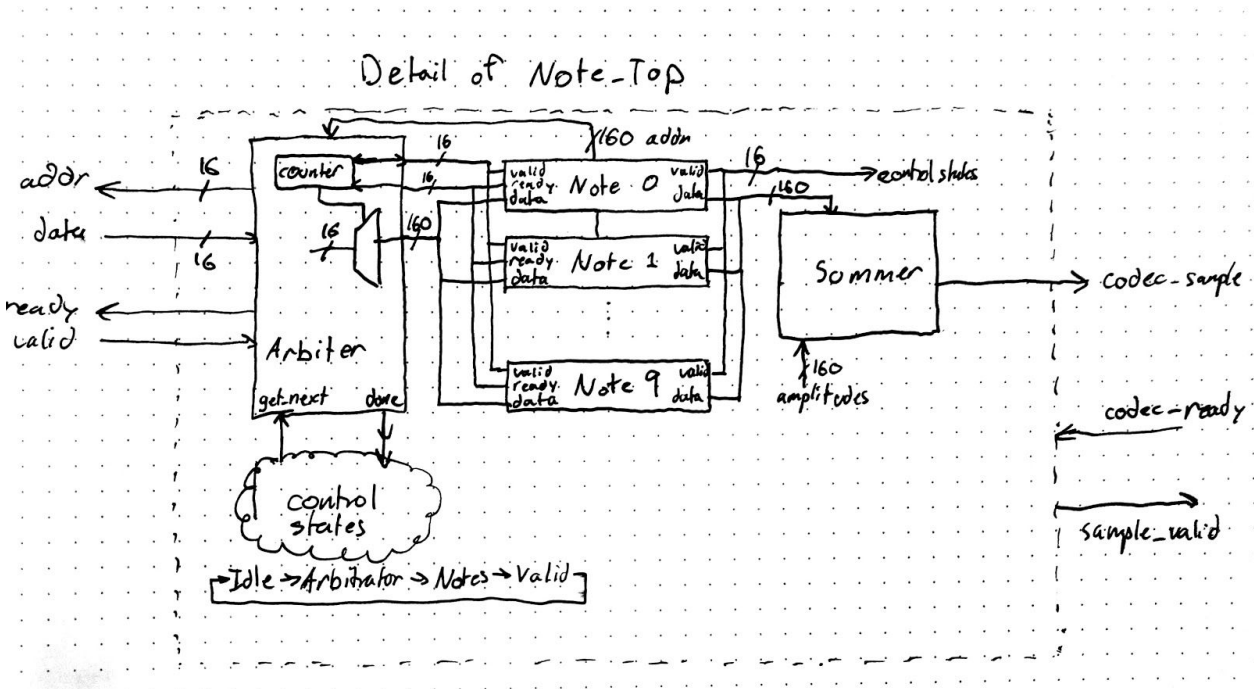
Midi_Top is a the SV module that connects to the midi_top driver. Midi_top uses the Avalon memory mapped bus to receive 4-bit-addressable 16-bit data from HPS, and also communicates with the audio codec of our FPGA. For addresses 0-9, midi_top sends a notes data to note_top, which communicates with the codec output. For addresses 10-12, midi_top sends data to the wavetables, which note_top reads from. Qsys was used to connect the HPS and audio_codecs together, and soc_system_top.sv was written based on Lab3. hps_codec_memory_test is an earlier working version of this.

Wavetables



Wavetables is a SV module that holds three full wavetables (48000 16-bit samples each). hps_codec_memory_test established that 3 buffers fits on the FPGA and is sufficiently quick to access through Avalon, so this is a tractable design. 2 wavetables are read from at a time, and feed into a mixer. One wavetable is written to. Wavetables uses a state machine to switch between these 3 buffers so that we are always reading from 2.

Note Top



The job of the note_top module is to orchestrate the sampling and processing of samples from the wavetables. In the hardware it is situated between the audio codec and the interface of the wavetable memory. It is also responsible for keeping and processing note configuration data.

Sampling is handled via the setting of three control flags. When the codec requests a sample, the get_next_sample flag is set to tell the arbiter to begin a cycle, and the notes_ready_in flag is set to instruct each of the 10 note modules to request a sample. Once the arbiter is finished, the module waits for all 10 notes to report a valid sample. At this point, the data coming from the summer is accurate, so the module sets the codec_sample_valid flag to true. Thus, there are 4 states in the protocol: IDLE for when no data has been requested, ARBITER while waiting for the arbiter to finish, NOTE while waiting for all note modules to report valid data, and VALID while waiting for the codec to accept a sample.

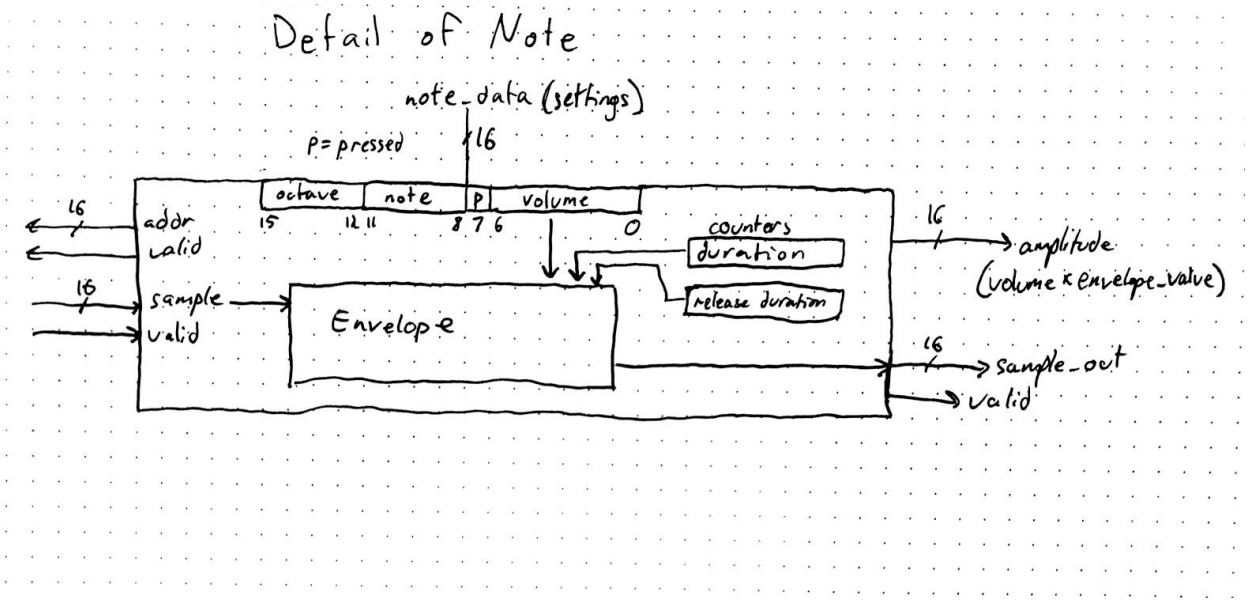
Mixer

The mixer is an all combinatorial module which takes a sample from each wavetable and adjusts a weighted sum based on the mix ratio. For example if the mix ratio is 50% each wave sample is halved and then added together. In practice the mix ratio is an 8-bit value from the modulation wheel on the keyboard. The first wave sample is multiplied by this value and then shifted back to renormalize its size. The same happens to the second wave sample except it is multiplied by (127- mix_ratio) before being shifted.

Arbiter

The arbiter coordinates accessing the wavetables between notes. It routes the address that the note is requesting and gives the data provided by the wavetables back to the note. It repeats this in a well ordered manner until all notes have been served. The arbiter is triggered by a get_next_sample signal from the note_top module which sends it in response to the Audio Codec requesting another sample. Once it receives the signal the arbiter grabs the address each note is requesting and asserts that the address it is providing to the wavetable is valid. Then once the wavetable has fetched the appropriate samples and mixed them, it asserts that the data it is providing is valid and the arbiter forwards this data to the note and asserts that the data is valid. This handshake is repeated with the arbiter as the middle man for all notes and once it has completed its round the arbiter sends a signal to note_top to say it is done at which point the get_next_sample signal is turned off.

Note



The note module is responsible for sampling the noise corresponding to a single pressed key. Its main job is to decide which address from the wavetable should be next sampled to produce a desired note. It receives configuration from the note_top module which includes octave, note, volume, and a control bit to signify whether the note is currently held down. Via a ready-valid exchange, the module requests data from the note arbiter when prompted and applies volume control and enveloping to the sample before passing the processed data to the summation module.

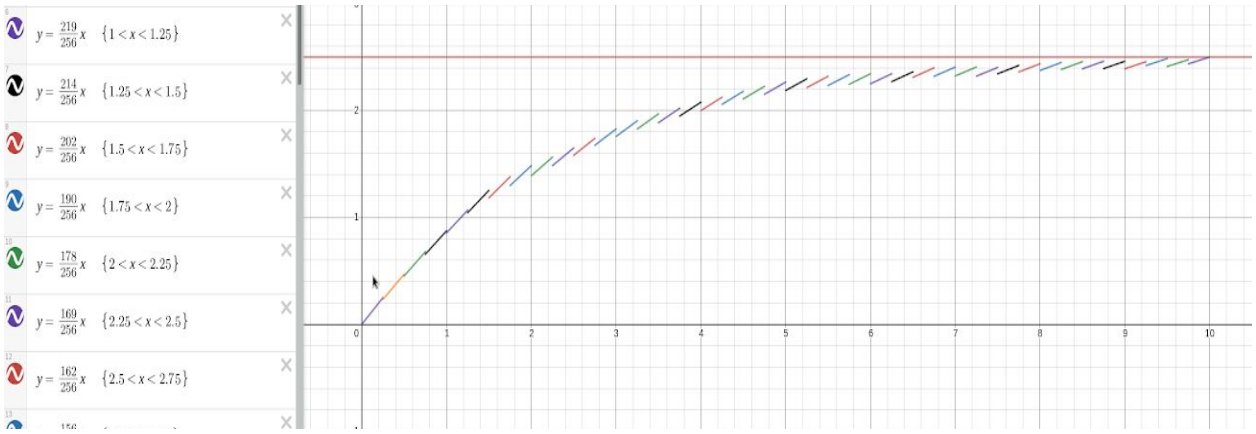
The memory offset with which to sample from the wavetable is controlled by the two parameters octave and note. The appropriate offsets for the highest possible octave are indexed from a

table. These can be shifted right by the amount given in octave to get the correct value for a lower octave of notes.

The note.sv file also includes another module called adsr_envelope for enveloping and volume control. This module processes the sample input by multiplying by the volume level and the current envelope coefficient (These are then shifted back into 16-bit). The envelope coefficient drives the level of sound as it relates to the stages of enveloping. Each stage is implemented with a different clock_div value. A counter is incremented each time a value is sampled, and once this reaches the clock div amount, it is reset and the envelope coefficient is incremented or decremented. The current stage is determined by a long duration counter which also counts up once each sample.

Summer

The summer takes the product of the current envelope coefficient and the note volume, and it sets thresholds at which progressively lower percentages of the amplitude are let through. For reference, the full graph of the level of volume as overall amplitudes increase is shown below. There are small discontinuities where the coefficient changes, but in general the trend is clear.



The maximum volume is limited to a quarter of the volume of 10 notes playing at max volume at once. This means the waves in the wavetables must still be limited in amplitude to 40% of the full theoretical range however this is significantly better than reducing by a factor of 10. The most difference comes at the lower end of the scale where the discontinuities are also the smallest.

The module shifts each sample right by 1-8 bits and then adds the result of these shift operations together. Which results are added and which are discarded is determined by a mask that the top 6 bits of the amplitude product choose. For example in the most extreme case the amplitudes should be cut to 25% of their original value so the output of the shifter which divides by 4 is the only one used will the rest are zeroed out.

Effects

A few effects modules have been written although not incorporated or rigorously test benched. The first is a bit crusher which adds a hiss to the sound. It reduces the range of the audio to merely 4 bits. This was done because reducing to anything less is not particularly noticeable except when utilizing a large dynamic range at which point very quiet sounds will be drowned out in the hiss.

The second effect is a clipping module which flattens the peaks of the audio signals so that a large amount of noise is introduced. It checks for high levels and if they are above 50% of the range chosen then it sets them to a max value.

The final effect is a tremolo which multiplies a low frequency sawtooth wave so that the volume rises and falls repeatedly. As written it has the effect of turning a sustained note into a series of 16th notes as if a violinist was rapidly moving their bow back and forth.

Software

Driver

Midi_top driver is the kernel-space connection that allows a userspace programmer to access the FPGA through the HPS. Midi_top.c shows where the kernel catches each ioctl and sends iowrites to the address of the device + an offset. The offset corresponds to HPS Avalon addresses in the hardware. We use different addresses for sending waves and the 10 notes. Hps_codec_memory_test_driver was an earlier, simpler version of this.

Lower Level Functions

note.h is the userspace file that includes functions which the user calls to send things to the kernel (ioctl). It can translate a midi number and volume into a 16-bit quantity that the accelerator expects (separates into note, octave, and volume). It also keeps track of which of 10 notes in hardware to send the next note to. To send a wavetable, a user must call start_wave with the desired table number then the 48000 data values with send_wave. To set the note it uses register offsets 0-9.

Interfacing with Libusb

We utilized libusb to interface with the connected MIDI device. We based the midi_usb code off of the libusb code used in lab 2 which utilized libusb to connect and read a keyboard. The

primary modifications to the original libusb code was to introduce a struct usb_midi_packet which is 64 bytes with the following objects within:

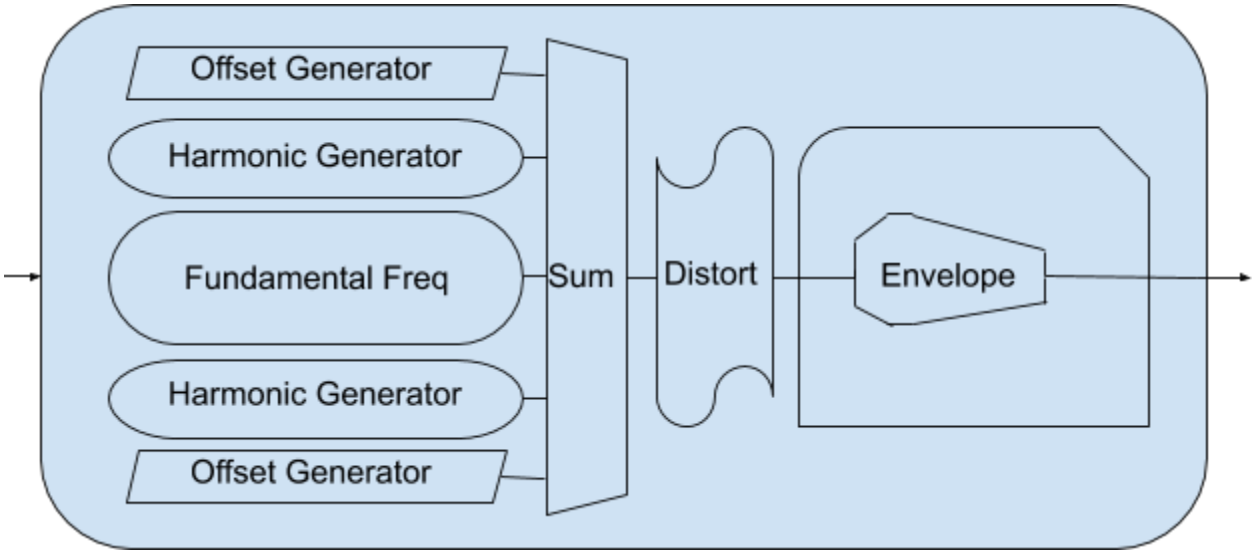
```
struct usb_midi_packet {
  uint8_t status_byte;
  uint8_t device_info;
  uint8_t note;
  uint8_t attack;
  uint8_t other_data[60];
};
```

Here the first byte is the status(described in the MIDI section). The second specifies the button, and note and attack are data1 and data2 respectively. Other data is typical just a buffer of 0's sent in due to the MIDI packets being 64 bytes in total. The process of extracting the packet is very similar to the one provided, with the primary change requiring interfaceClass and binterfaceProtocol being specified as below.

```
inter->bInterfaceClass == LIBUSB_CLASS_AUDIO &&
            inter->bInterfaceProtocol == USB_MIDI_KEYBOARD_PROTOCOL &&
            (inter->bNumEndpoints != 0)
```

Libusb supports these protocols internally.

MATLAB Wave Creation and Tests



The image above shows the process of generating the sound wave for an individual instrument. The instrument receives what type of instrument it is and what note it should play. Each instrument profile is described by some combination of 5 oscillators. Based on this information the instrument sums the fundamental frequency of the note as well as some combination of harmonic frequencies. These three central oscillators are sinusoidal generators which read from a wave table to determine what value to pump out next. In contrast the offset generators produce a triangle, sawtooth, or square wave dynamically from looking at the value of a counter. The output of these 5 oscillators are then summed together and then their combined output is fed through a filter which introduces distortion by adding a soft limit to create clipping or adding noise.

Audio amplitude levels are encoded in 16 bits and we will have the CODEC sample at 48 kHz in order to sample at well over twice the maximum frequency that humans can hear.
This means that: 16 bits * 48,000 samples * 3 waves = 288kB/s is used.
The lowest frequency that humans can hear is 20hz and as a result a single sine wave at this frequency will be sampled 2500 times for a total of 5kB needed to store the points of a sine wave. This table of values that make up a sine wave can be referenced to create a sine wave at any audible frequency.

For the sinusoidal wave table, we will use 5 kB of SRAM memory. This is because, as stated above, we will store a sine wave at 20 hz. This means (50 kHz/20 Hz)*(16 bits)*(1 Byte/8 bits) = 5 kB of the wave. With the 500 kB of SRAM in our FPGA, we have more than enough leeway for other complex waveforms we like or to allow for other systems to use the SRAM if we find out it is needed.

These waves were produced in matlab and tested in matlab. The waves that were created are sine, sawtooth, and pulse waves. Additionally, the matlab tests showed us that we could synthesize different example frequencies and waveforms added together, which is pivotal to what we are trying to achieve. Lastly, we wanted to be able to save the waveforms as binary data and then play them as audio signals.

Appendix
Hardware

Effects

bitcrush.sv

```
module bit_crusher (
    input rst,
```

```
    input en,
    input [15:0] sample_in,
    output [15:0] data_out
);

always_comb    begin
    if (rst)
       data_out = 0;
    else if(en)
       data_out = sample_in & (-1 << 12);
    else
       data_out = sample_in;
end

endmodule
```

clipper.sv

```
module clipper (
    input rst,
    input en,
    input [15:0] sample_in,
    output [15:0] data_out
);

always_comb begin
    if (rst)
       data_out = 0;
    else if (en)
       case(sample_in[15:11])
             4'h0: data_out = sample_in << 2;
             4'h1: data_out = sample_in << 2;
             4'h2: data_out = sample_in << 1;
             4'h3: data_out = sample_in << 1;
             4'h4: data_out = 16'h7FFF;
             4'h5: data_out = 16'h7FFF;
             4'h6: data_out = 16'h7FFF;
             4'h7: data_out = 16'h7FFF;
             4'h8: data_out = 16'h8000;
             4'h9: data_out = 16'h8000;
             4'hA: data_out = 16'h8000;
```

```
            4'hB: data_out = 16'h8000;
            4'hC: data_out = sample_in << 1;
            4'hD: data_out = sample_in << 1;
            4'hE: data_out = sample_in << 2;
            4'hF: data_out = sample_in << 2;
        endcase
    else
        data_out = sample_in;
end

endmodule
```

tremolo.sv

```
module tremolo (
    input clk,    // Clock
    input rst,
    input sample_in,
    output data_out
);

logic [9:0] clk_counter;
logic [9:0] aud_counter;

always_ff @(posedge clk) begin
    if(~rst) begin
        clk_counter <= 0;
        aud_counter <= 0;
    end else begin
        clk_counter <= clk_counter + 1;
        if(clk_counter == 0)
            aud_counter <= aud_counter + 1;
    end
end

always_comb    begin
    data_out = (sample_in >> 10) * aud_counter;
end

endmodule
```

Hps_codec_memory_test

Midi_top.sv

```
/*
* Avalon memory-mapped peripheral that generates VGA
*
* Stephen A. Edwards
* Columbia University
*/

module midi_top(input logic         clk,
       input logic      reset,
       input logic [15:0] writedata,
       input logic      write,
       input            chipselect,
       input logic [3:0]  address,

       input [3:0]      KEY,

       input left_chan_ready,
       input right_chan_ready,
       output logic [15:0] sample_data,
       output logic sample_valid);


   logic [6:0] volume;
   logic status_bit;
   logic [3:0] note;
   logic [3:0] octave;


   tonegen tg(.clk(clk), .reset(~KEY[3]), .volume(volume), .note(note),
.octave(octave), .left_chan_ready(left_chan_ready),
      .right_chan_ready(right_chan_ready), .sample_data(sample_data),
.sample_valid(sample_valid));


   logic [15:0] addr;
   logic [15:0] din0, din1, din2;
   logic [15:0] dout0, dout1, dout2;
   logic we;
```

```
//logic start_
//logic [15:0]

logic start;


mem table0(.clock(clk), .address(addr), .data(din0),
     .wren(we), .q(dout0));

mem table1(.clock(clk), .address(addr), .data(din1),
     .wren(we), .q(dout1));

mem table2(.clock(clk), .address(addr), .data(din2),
     .wren(we), .q(dout2));

logic done;
//assign done = ~(addr < 16'd48000);


always_ff @(posedge clk) begin
    if (reset) begin
    start <= 0;
    we   <= 0;
    octave <= 3;
          volume <= 255;
          note   <= KEY[2:0];
    we   <= 0;
    addr   <= 0;
    done   <= 0;

    //background_r <= 8'h0;
    //background_g <= 8'h0;
    //background_b <= 8'h80;
 end else if (chipselect && write) begin
    case (address)
 4'h0 : begin
    we  <= 0;
    volume     <= writedata[6:0];
    status_bit <= writedata[7];
    note       <= writedata[11:8];
    octave     <= writedata[15:12];
    end
```

```
    4'h1 : begin
       we  <= 0;
       volume     <= writedata[6:0];
       status_bit <= writedata[7];
       note       <= writedata[11:8];
       octave     <= writedata[15:12] + 2;
       end
    4'h2 : begin
       we  <= 0;
       start <= 1;
       done  <= 0;
       addr  <= 0;
       end
    4'h3 : begin
       if ( (done == 1) | (addr > 16'd47998)) begin
             done <= 1;
             we <= 0;
       end
       else begin
             start <= 0;
             if (start == 1) begin
                   we   <= 1;
                   addr <= 0;
             end else if (addr == 16'd47998) begin
                   we   <= 1;
                   addr <= addr + 1;
                   done <= 1;
             end else begin
                   we   <= 1;
                   addr <= addr + 1;
             end

             din0  <= writedata;//6; //writedata; // addr + 1;
             din1  <= writedata + 1; //7; //writedata + 1; addr + 1;
             din2  <= writedata + 2; //8; //writedata + 2; addr + 1;
             end
       end
    default: begin
       we  <= 0;
       octave <= 3;
             volume <= 255;
             note   <= KEY[2:0];
```

```
       end
       endcase
    end else begin
    we <= 0;
    if (done) begin
       addr <= 16'd4322;
       if (dout0 == 16'd0)
             note <= 0;
       else if (dout0 == 16'd4320)
             note <= 4;
       else if (dout0 == 16'd4321)
             note <= 5;
       else if (dout0 == 16'd4322 && dout1 == 16'd4323 && dout2 ==
16'd4324)
             note <= 6;
       else if (dout0 == 16'd4323)
             note <= 7;
       else if (dout0 == 16'd4324)
             note <= 8;
       else
             note <= 9;
             octave <= 4;
             volume <= 255;
       end
    end
end

endmodule

module mem( input logic       clock,
    input logic  [15:0] address,
    input logic  [15:0] data,
    input logic         wren,
    output logic [15:0] q);

logic [15:0]  mem [47999:0];

always_ff @(posedge clock) begin
    if (wren) mem[address] <= data;
    q <= mem[address];
end
```

```
endmodule
```

Soc_system_top.sv

```
// ============================================================================
// Copyright (c) 2013 by Terasic Technologies Inc.
// ============================================================================
//
// Modified 2019 by Stephen A. Edwards
//
// Permission:
//
//   Terasic grants permission to use and modify this code for use in
//   synthesis for all Terasic Development Boards and Altera
//   Development Kits made by Terasic.  Other use of this code,
//   including the selling ,duplication, or modification of any
//   portion is strictly prohibited.
//
// Disclaimer:
//
//   This VHDL/Verilog or C/C++ source code is intended as a design
//   reference which illustrates how these types of functions can be
//   implemented.  It is the user's responsibility to verify their
//   design for consistency and functionality through the use of
//   formal verification methods.  Terasic provides no warranty
//   regarding the use or functionality of this code.
//
// ============================================================
//
//  Terasic Technologies Inc

//  9F., No.176, Sec.2, Gongdao 5th Rd, East Dist, Hsinchu City, 30070.
Taiwan
//
//
//                     web: http://www.terasic.com/
//                     email: support@terasic.com
module soc_system_top(

 ///////// ADC /////////
```

```
inout           ADC_CS_N,
output     ADC_DIN,
input           ADC_DOUT,
output     ADC_SCLK,

//////// AUD ////////
input           AUD_ADCDAT,
inout           AUD_ADCLRCK,
inout           AUD_BCLK,
output     AUD_DACDAT,
inout           AUD_DACLRCK,
output     AUD_XCK,

//////// CLOCK2 ////////
input           CLOCK2_50,

//////// CLOCK3 ////////
input           CLOCK3_50,

//////// CLOCK4 ////////
input           CLOCK4_50,

//////// CLOCK ////////
input           CLOCK_50,

//////// DRAM ////////
output [12:0] DRAM_ADDR,
output [1:0]  DRAM_BA,
output     DRAM_CAS_N,
output     DRAM_CKE,
output     DRAM_CLK,
output     DRAM_CS_N,
inout [15:0]  DRAM_DQ,
output     DRAM_LDQM,
output     DRAM_RAS_N,
output     DRAM_UDQM,
output     DRAM_WE_N,

//////// FAN ////////
output     FAN_CTRL,

//////// FPGA ////////
```

```
output      FPGA_I2C_SCLK,
inout             FPGA_I2C_SDAT,

//////// GPIO ////////
inout [35:0]  GPIO_0,
inout [35:0]  GPIO_1,

//////// HEX0 ////////
output [6:0]  HEX0,

//////// HEX1 ////////
output [6:0]  HEX1,

//////// HEX2 ////////
output [6:0]  HEX2,

//////// HEX3 ////////
output [6:0]  HEX3,

//////// HEX4 ////////
output [6:0]  HEX4,

//////// HEX5 ////////
output [6:0]  HEX5,

//////// HPS ////////
inout             HPS_CONV_USB_N,
output [14:0] HPS_DDR3_ADDR,
output [2:0]  HPS_DDR3_BA,
output      HPS_DDR3_CAS_N,
output      HPS_DDR3_CKE,
output      HPS_DDR3_CK_N,
output      HPS_DDR3_CK_P,
output      HPS_DDR3_CS_N,
output [3:0]  HPS_DDR3_DM,
inout [31:0]  HPS_DDR3_DQ,
inout [3:0]   HPS_DDR3_DQS_N,
inout [3:0]   HPS_DDR3_DQS_P,
output      HPS_DDR3_ODT,
output      HPS_DDR3_RAS_N,
output      HPS_DDR3_RESET_N,
input             HPS_DDR3_RZQ,
```

```
output      HPS_DDR3_WE_N,
output      HPS_ENET_GTX_CLK,
inout             HPS_ENET_INT_N,
output      HPS_ENET_MDC,
inout             HPS_ENET_MDIO,
input             HPS_ENET_RX_CLK,
input [3:0]    HPS_ENET_RX_DATA,
input             HPS_ENET_RX_DV,
output [3:0]  HPS_ENET_TX_DATA,
output      HPS_ENET_TX_EN,
inout             HPS_GSENSOR_INT,
inout             HPS_I2C1_SCLK,
inout             HPS_I2C1_SDAT,
inout             HPS_I2C2_SCLK,
inout             HPS_I2C2_SDAT,
inout             HPS_I2C_CONTROL,
inout             HPS_KEY,
inout             HPS_LED,
inout             HPS_LTC_GPIO,
output      HPS_SD_CLK,
inout             HPS_SD_CMD,
inout [3:0]    HPS_SD_DATA,
output      HPS_SPIM_CLK,
input             HPS_SPIM_MISO,
output      HPS_SPIM_MOSI,
inout             HPS_SPIM_SS,
input             HPS_UART_RX,
output      HPS_UART_TX,
input             HPS_USB_CLKOUT,
inout [7:0]    HPS_USB_DATA,
input             HPS_USB_DIR,
input             HPS_USB_NXT,
output      HPS_USB_STP,

//////// IRDA ////////
input             IRDA_RXD,
output      IRDA_TXD,

//////// KEY ////////
input [3:0]    KEY,

//////// LEDR ////////
```

```
 output [9:0]  LEDR,

 ///////// PS2 /////////
 inout            PS2_CLK,
 inout            PS2_CLK2,
 inout            PS2_DAT,
 inout            PS2_DAT2,

 ///////// SW /////////
 input [9:0]   SW,

 ///////// TD /////////
 input            TD_CLK27,
 input [7:0]   TD_DATA,
 input            TD_HS,
 output     TD_RESET_N,
 input            TD_VS,


 ///////// VGA /////////
 output [7:0]  VGA_B,
 output     VGA_BLANK_N,
 output     VGA_CLK,
 output [7:0]  VGA_G,
 output     VGA_HS,
 output [7:0]  VGA_R,
 output     VGA_SYNC_N,
 output     VGA_VS
);

   soc_system soc_system0(
      .clk_clk                       ( CLOCK_50 ),
      .reset_reset_n                 ( 1'b1 ),

      .hps_ddr3_mem_a                ( HPS_DDR3_ADDR ),
      .hps_ddr3_mem_ba               ( HPS_DDR3_BA ),
      .hps_ddr3_mem_ck               ( HPS_DDR3_CK_P ),
      .hps_ddr3_mem_ck_n             ( HPS_DDR3_CK_N ),
      .hps_ddr3_mem_cke              ( HPS_DDR3_CKE ),
      .hps_ddr3_mem_cs_n             ( HPS_DDR3_CS_N ),
      .hps_ddr3_mem_ras_n            ( HPS_DDR3_RAS_N ),
      .hps_ddr3_mem_cas_n            ( HPS_DDR3_CAS_N ),
```

```
    .hps_ddr3_mem_we_n            ( HPS_DDR3_WE_N ),
    .hps_ddr3_mem_reset_n         ( HPS_DDR3_RESET_N ),
    .hps_ddr3_mem_dq              ( HPS_DDR3_DQ ),
    .hps_ddr3_mem_dqs             ( HPS_DDR3_DQS_P ),
    .hps_ddr3_mem_dqs_n           ( HPS_DDR3_DQS_N ),
    .hps_ddr3_mem_odt             ( HPS_DDR3_ODT ),
    .hps_ddr3_mem_dm              ( HPS_DDR3_DM ),
    .hps_ddr3_oct_rzqin           ( HPS_DDR3_RZQ ),

    .hps_hps_io_emac1_inst_TX_CLK ( HPS_ENET_GTX_CLK ),
    .hps_hps_io_emac1_inst_TXD0   ( HPS_ENET_TX_DATA[0] ),
    .hps_hps_io_emac1_inst_TXD1   ( HPS_ENET_TX_DATA[1] ),
    .hps_hps_io_emac1_inst_TXD2   ( HPS_ENET_TX_DATA[2] ),
    .hps_hps_io_emac1_inst_TXD3   ( HPS_ENET_TX_DATA[3] ),
    .hps_hps_io_emac1_inst_RXD0   ( HPS_ENET_RX_DATA[0] ),
    .hps_hps_io_emac1_inst_MDIO   ( HPS_ENET_MDIO  ),
    .hps_hps_io_emac1_inst_MDC    ( HPS_ENET_MDC   ),
    .hps_hps_io_emac1_inst_RX_CTL ( HPS_ENET_RX_DV ),
    .hps_hps_io_emac1_inst_TX_CTL ( HPS_ENET_TX_EN ),
    .hps_hps_io_emac1_inst_RX_CLK ( HPS_ENET_RX_CLK ),
    .hps_hps_io_emac1_inst_RXD1   ( HPS_ENET_RX_DATA[1]  ),
    .hps_hps_io_emac1_inst_RXD2   ( HPS_ENET_RX_DATA[2]  ),
    .hps_hps_io_emac1_inst_RXD3   ( HPS_ENET_RX_DATA[3]  ),

    .hps_hps_io_sdio_inst_CMD     ( HPS_SD_CMD           ),
    .hps_hps_io_sdio_inst_D0      ( HPS_SD_DATA[0]       ),
    .hps_hps_io_sdio_inst_D1      ( HPS_SD_DATA[1]       ),
    .hps_hps_io_sdio_inst_CLK     ( HPS_SD_CLK           ),
    .hps_hps_io_sdio_inst_D2      ( HPS_SD_DATA[2]       ),
    .hps_hps_io_sdio_inst_D3      ( HPS_SD_DATA[3]       ),

    .hps_hps_io_usb1_inst_D0      ( HPS_USB_DATA[0]      ),
    .hps_hps_io_usb1_inst_D1      ( HPS_USB_DATA[1]      ),
    .hps_hps_io_usb1_inst_D2      ( HPS_USB_DATA[2]      ),
    .hps_hps_io_usb1_inst_D3      ( HPS_USB_DATA[3]      ),
    .hps_hps_io_usb1_inst_D4      ( HPS_USB_DATA[4]      ),
    .hps_hps_io_usb1_inst_D5      ( HPS_USB_DATA[5]      ),
    .hps_hps_io_usb1_inst_D6      ( HPS_USB_DATA[6]      ),
    .hps_hps_io_usb1_inst_D7      ( HPS_USB_DATA[7]      ),
    .hps_hps_io_usb1_inst_CLK     ( HPS_USB_CLKOUT       ),
    .hps_hps_io_usb1_inst_STP     ( HPS_USB_STP          ),
    .hps_hps_io_usb1_inst_DIR     ( HPS_USB_DIR          ),
```

```
    .hps_hps_io_usb1_inst_NXT     ( HPS_USB_NXT          ),

    .hps_hps_io_spim1_inst_CLK    ( HPS_SPIM_CLK  ),
    .hps_hps_io_spim1_inst_MOSI   ( HPS_SPIM_MOSI ),
    .hps_hps_io_spim1_inst_MISO   ( HPS_SPIM_MISO ),
    .hps_hps_io_spim1_inst_SS0    ( HPS_SPIM_SS   ),

    .hps_hps_io_uart0_inst_RX     ( HPS_UART_RX     ),
    .hps_hps_io_uart0_inst_TX     ( HPS_UART_TX     ),

    .hps_hps_io_i2c0_inst_SDA     ( HPS_I2C1_SDAT   ),
    .hps_hps_io_i2c0_inst_SCL     ( HPS_I2C1_SCLK   ),

    .hps_hps_io_i2c1_inst_SDA     ( HPS_I2C2_SDAT   ),
    .hps_hps_io_i2c1_inst_SCL     ( HPS_I2C2_SCLK   ),

    .hps_hps_io_gpio_inst_GPIO09  ( HPS_CONV_USB_N ),
    .hps_hps_io_gpio_inst_GPIO35  ( HPS_ENET_INT_N ),
    .hps_hps_io_gpio_inst_GPIO40  ( HPS_LTC_GPIO ),

    .hps_hps_io_gpio_inst_GPIO48  ( HPS_I2C_CONTROL ),
    .hps_hps_io_gpio_inst_GPIO53  ( HPS_LED ),
    .hps_hps_io_gpio_inst_GPIO54  ( HPS_KEY ),
    .hps_hps_io_gpio_inst_GPIO61  ( HPS_GSENSOR_INT ),


    .midi_key(KEY[3:0]),
    .midi_l_chan_ready(rdy_left),
    .midi_r_chan_ready(rdy_right),
    .midi_s_data(sample),
    .midi_s_valid(sample_valid),


    .audio_0_avalon_left_channel_sink_data(sample),
    .audio_0_avalon_left_channel_sink_valid(sample_valid),
    .audio_0_avalon_left_channel_sink_ready(rdy_left),
    .audio_0_avalon_right_channel_sink_data(sample),
    .audio_0_avalon_right_channel_sink_valid(sample_valid),
    .audio_0_avalon_right_channel_sink_ready(rdy_right),

    .audio_0_external_interface_BCLK(AUD_BCLK),
    .audio_0_external_interface_DACDAT(AUD_DACDAT),
```

```
    .audio_0_external_interface_DACLRCK(AUD_DACLRCK),

    .audio_and_video_config_0_external_interface_SDAT(FPGA_I2C_SDAT),
    .audio_and_video_config_0_external_interface_SCLK(FPGA_I2C_SCLK),

    .audio_pll_0_audio_clk_clk(AUD_XCK)

);

 logic rdy_left;
 logic rdy_right;
 logic [15:0] sample;
 logic sample_valid;
 // The following quiet the "no driver" warnings for output
 // pins and should be removed if you use any of these peripherals

 assign ADC_CS_N = SW[1] ? SW[0] : 1'bZ;
 assign ADC_DIN = SW[0];
 assign ADC_SCLK = SW[0];

 //assign AUD_ADCLRCK = SW[1] ? SW[0] : 1'bZ;   //didnt use it
 //assign AUD_BCLK = SW[1] ? SW[0] : 1'bZ;
 //assign AUD_DACDAT = SW[0];
 //assign AUD_DACLRCK = SW[1] ? SW[0] : 1'bZ;
 //assign AUD_XCK = SW[0];

 assign DRAM_ADDR = { 13{ SW[0] } };
 assign DRAM_BA = { 2{ SW[0] } };
 assign DRAM_DQ = SW[1] ? { 16{ SW[0] } } : 16'bZ;
 assign {DRAM_CAS_N, DRAM_CKE, DRAM_CLK, DRAM_CS_N,
    DRAM_LDQM, DRAM_RAS_N, DRAM_UDQM, DRAM_WE_N} = { 8{SW[0]} };

 assign FAN_CTRL = SW[0];

 //assign FPGA_I2C_SCLK = SW[0];
 //assign FPGA_I2C_SDAT = SW[1] ? SW[0] : 1'bZ;

 assign GPIO_0 = SW[1] ? { 36{ SW[0] } } : 36'bZ;
 assign GPIO_1 = SW[1] ? { 36{ SW[0] } } : 36'bZ;

 assign HEX0 = { 7{ SW[1] } };
 assign HEX1 = { 7{ SW[2] } };
```

```
   assign HEX2 = { 7{ SW[3] } };
   assign HEX3 = { 7{ SW[4] } };
   assign HEX4 = { 7{ SW[5] } };
   assign HEX5 = { 7{ SW[6] } };

   assign IRDA_TXD = SW[0];

   assign LEDR = { 10{SW[7]} };

   assign PS2_CLK = SW[1] ? SW[0] : 1'bZ;
   assign PS2_CLK2 = SW[1] ? SW[0] : 1'bZ;
   assign PS2_DAT = SW[1] ? SW[0] : 1'bZ;
   assign PS2_DAT2 = SW[1] ? SW[0] : 1'bZ;

   assign TD_RESET_N = SW[0];

   assign {VGA_R, VGA_G, VGA_B} = { 24{ SW[0] } };
   assign {VGA_BLANK_N, VGA_CLK,
      VGA_HS, VGA_SYNC_N, VGA_VS} = { 5{ SW[0] } };


endmodule
```

Midi_top

Midi_top.sv

```
/*
* Avalon memory-mapped peripheral that generates VGA
*
* Stephen A. Edwards
* Columbia University
*/

module midi_top(input logic clk,
       input logic      reset,
       input logic [15:0]  writedata,
       input logic      write,
       input            chipselect,
       input logic [3:0]   address,
```

```
       input [3:0]      KEY,

       input left_chan_ready,
       input right_chan_ready,
       output logic [15:0] sample_data,
       output logic sample_valid);


    //note top ports
    //from wave table
    //logic nt_wave_address_ready;
    //logic nt_wave_sample_valid;
    //logic signed [15:0] nt_wave_sample;
    //logic [15:0] nt_wave_address;

    //from MIDI top
    logic nt_note_data_valid;
    logic [15:0] nt_note_data;
    logic [3:0] nt_note_addr;
    //to audio codec
    logic audio_codec_ready;
    assign audio_codec_ready = left_chan_ready & right_chan_ready;




    //wavetable ports
    logic           wt_write;       // set to true if linux sent us
something
    logic           wt_is_data;   // 0 is info, 1 is wave data
    logic [15:0] wt_wave_data; // wave data

    logic [15:0] wt_sample_address;

    logic           wt_sample_ready;
    logic           wt_sample_valid;

    logic [6:0]  wt_mix_ratio;
    logic signed [15:0] wt_data_out;


    // logic [6:0] volume;
```

```
    // logic status_bit;
    // logic [3:0] note;
    // logic [3:0] octave;



    wavetables wt(.clk(clk), .rst(reset),
    .write(wt_write), .is_data(wt_is_data) , .wave_data(wt_wave_data),

    .sample_address(wt_sample_address),
    .sample_ready(wt_sample_ready) ,.sample_valid(wt_sample_valid),

    .mix_ratio(wt_mix_ratio), //set by avalon, below

    .data_out(wt_data_out)
    );



    note_top nt(.clk(clk), .reset(reset),

       .wave_address_ready(wt_sample_ready),
.wave_sample_valid(wt_sample_valid),      // sent to/from wavetable, above
       .wave_sample(wt_data_out), .wave_address(wt_sample_address),

       .note_data_valid(nt_note_data_valid), .note_data(nt_note_data),
.note_addr(nt_note_addr), // sent by avalon, below

       .audio_codec_ready(audio_codec_ready),
.codec_sample_valid(sample_valid),                    //communicate with the
codec
       .codec_sample_data(sample_data)
    );




    always_ff @(posedge clk) begin
       nt_note_data_valid <= 0;
       wt_write <= 0;
     if (reset) begin
       nt_note_data_valid <= 0;
```

```
      wt_mix_ratio <= 0;
      wt_write <= 0;

    end else if (chipselect && write) begin
     case (address)
    4'd0 : begin
      nt_note_addr     <= 0;
      nt_note_data     <= writedata;
      nt_note_data_valid <= 1;
      end
    4'd1 : begin
      nt_note_addr     <= 1;
      nt_note_data     <= writedata;
      nt_note_data_valid <= 1;
      end
    4'd2 : begin
      nt_note_addr     <= 2;
      nt_note_data     <= writedata;
      nt_note_data_valid <= 1;
      end
    4'd3 : begin
      nt_note_addr     <= 3;
      nt_note_data     <= writedata;
      nt_note_data_valid <= 1;
      end
    4'd4 : begin
      nt_note_addr     <= 4;
      nt_note_data     <= writedata;
      nt_note_data_valid <= 1;
      end
    4'd5 : begin
      nt_note_addr     <= 5;
      nt_note_data     <= writedata;
      nt_note_data_valid <= 1;
      end
    4'd6 : begin
      nt_note_addr     <= 6;
      nt_note_data     <= writedata;
      nt_note_data_valid <= 1;
      end
    4'd7 : begin
      nt_note_addr     <= 7;
```

```
        nt_note_data     <= writedata;
        nt_note_data_valid <= 1;
        end
      4'd8 : begin
        nt_note_addr     <= 8;
        nt_note_data     <= writedata;
        nt_note_data_valid <= 1;
        end
      4'd9 : begin
        nt_note_addr     <= 9;
        nt_note_data     <= writedata;
        nt_note_data_valid <= 1;
        end
      4'd10 : begin
          //START WAVE
          wt_write   <= 1;
          wt_is_data <= 0;
          wt_wave_data  <= writedata;
        end
      4'd11 : begin
          //SEND WAVE
          wt_write   <= 1;
          wt_is_data <= 1;
          wt_wave_data  <= writedata;
        end
      4'd12 : begin
          //MODULATION
          wt_mix_ratio <= writedata;
        end
      default: begin
        wt_write   <= 0;
          wt_is_data <= 0;
          wt_wave_data  <= writedata;
        end
    endcase
      end
end

endmodule


// 48000 X 16 synchronous RAM with old data read-during-write behavior
```

```
/*
module memory( input logic          clk,
    input logic  [15:0] a,
    input logic  [15:0] din,
    input logic         we,
    output logic [15:0] dout);

logic [15:0]  mem [47999:0];

always_ff @(posedge clk) begin
    if (we) mem[a] <= din;
    dout <= mem[a];
end

endmodule
*/
```

Soc_system_top.sv

```
// ====================================================================
// Copyright (c) 2013 by Terasic Technologies Inc.
// ====================================================================
//
// Modified 2019 by Stephen A. Edwards
//
// Permission:
//
//   Terasic grants permission to use and modify this code for use in
//   synthesis for all Terasic Development Boards and Altera
//   Development Kits made by Terasic.  Other use of this code,
//   including the selling ,duplication, or modification of any
//   portion is strictly prohibited.
//
// Disclaimer:
//
//   This VHDL/Verilog or C/C++ source code is intended as a design
//   reference which illustrates how these types of functions can be
//   implemented.  It is the user's responsibility to verify their
//   design for consistency and functionality through the use of
//   formal verification methods.  Terasic provides no warranty
//   regarding the use or functionality of this code.
```

```
//
// ====================================================================
//
//  Terasic Technologies Inc

//  9F., No.176, Sec.2, Gongdao 5th Rd, East Dist, Hsinchu City, 30070.
Taiwan
//
//
//                     web: http://www.terasic.com/
//                     email: support@terasic.com
module soc_system_top(

 ///////// ADC /////////
 inout            ADC_CS_N,
 output     ADC_DIN,
 input            ADC_DOUT,
 output     ADC_SCLK,

 ///////// AUD /////////
 input            AUD_ADCDAT,
 inout            AUD_ADCLRCK,
 inout            AUD_BCLK,
 output     AUD_DACDAT,
 inout            AUD_DACLRCK,
 output     AUD_XCK,

 ///////// CLOCK2 /////////
 input            CLOCK2_50,

 ///////// CLOCK3 /////////
 input            CLOCK3_50,

 ///////// CLOCK4 /////////
 input            CLOCK4_50,

 ///////// CLOCK /////////
 input            CLOCK_50,

 ///////// DRAM /////////
 output [12:0] DRAM_ADDR,
 output [1:0]  DRAM_BA,
```

```
output      DRAM_CAS_N,
output      DRAM_CKE,
output      DRAM_CLK,
output      DRAM_CS_N,
inout [15:0]  DRAM_DQ,
output      DRAM_LDQM,
output      DRAM_RAS_N,
output      DRAM_UDQM,
output      DRAM_WE_N,

//////// FAN ////////
output      FAN_CTRL,

//////// FPGA ////////
output      FPGA_I2C_SCLK,
inout             FPGA_I2C_SDAT,

//////// GPIO ////////
inout [35:0]  GPIO_0,
inout [35:0]  GPIO_1,

//////// HEX0 ////////
output [6:0]  HEX0,

//////// HEX1 ////////
output [6:0]  HEX1,

//////// HEX2 ////////
output [6:0]  HEX2,

//////// HEX3 ////////
output [6:0]  HEX3,

//////// HEX4 ////////
output [6:0]  HEX4,

//////// HEX5 ////////
output [6:0]  HEX5,

//////// HPS ////////
inout             HPS_CONV_USB_N,
output [14:0] HPS_DDR3_ADDR,
```

```
output [2:0]  HPS_DDR3_BA,
output     HPS_DDR3_CAS_N,
output     HPS_DDR3_CKE,
output     HPS_DDR3_CK_N,
output     HPS_DDR3_CK_P,
output     HPS_DDR3_CS_N,
output [3:0]  HPS_DDR3_DM,
inout [31:0]  HPS_DDR3_DQ,
inout [3:0]   HPS_DDR3_DQS_N,
inout [3:0]   HPS_DDR3_DQS_P,
output     HPS_DDR3_ODT,
output     HPS_DDR3_RAS_N,
output     HPS_DDR3_RESET_N,
input            HPS_DDR3_RZQ,
output     HPS_DDR3_WE_N,
output     HPS_ENET_GTX_CLK,
inout            HPS_ENET_INT_N,
output     HPS_ENET_MDC,
inout            HPS_ENET_MDIO,
input            HPS_ENET_RX_CLK,
input [3:0]   HPS_ENET_RX_DATA,
input            HPS_ENET_RX_DV,
output [3:0]  HPS_ENET_TX_DATA,
output     HPS_ENET_TX_EN,
inout            HPS_GSENSOR_INT,
inout            HPS_I2C1_SCLK,
inout            HPS_I2C1_SDAT,
inout            HPS_I2C2_SCLK,
inout            HPS_I2C2_SDAT,
inout            HPS_I2C_CONTROL,
inout            HPS_KEY,
inout            HPS_LED,
inout            HPS_LTC_GPIO,
output     HPS_SD_CLK,
inout            HPS_SD_CMD,
inout [3:0]   HPS_SD_DATA,
output     HPS_SPIM_CLK,
input            HPS_SPIM_MISO,
output     HPS_SPIM_MOSI,
inout            HPS_SPIM_SS,
input            HPS_UART_RX,
output     HPS_UART_TX,
```

```
input            HPS_USB_CLKOUT,
inout [7:0]   HPS_USB_DATA,
input            HPS_USB_DIR,
input            HPS_USB_NXT,
output     HPS_USB_STP,

///////// IRDA /////////
input            IRDA_RXD,
output     IRDA_TXD,

///////// KEY /////////
input [3:0]   KEY,

///////// LEDR /////////
output [9:0]  LEDR,

///////// PS2 /////////
inout            PS2_CLK,
inout            PS2_CLK2,
inout            PS2_DAT,
inout            PS2_DAT2,

///////// SW /////////
input [9:0]   SW,

///////// TD /////////
input            TD_CLK27,
input [7:0]   TD_DATA,
input            TD_HS,
output     TD_RESET_N,
input            TD_VS,


///////// VGA /////////
output [7:0]  VGA_B,
output     VGA_BLANK_N,
output     VGA_CLK,
output [7:0]  VGA_G,
output     VGA_HS,
output [7:0]  VGA_R,
output     VGA_SYNC_N,
output     VGA_VS
```

```
);

   soc_system soc_system0(
      .clk_clk                       ( CLOCK_50 ),
      .reset_reset_n                 ( 1'b1 ),

      .hps_ddr3_mem_a                ( HPS_DDR3_ADDR ),
      .hps_ddr3_mem_ba               ( HPS_DDR3_BA ),
      .hps_ddr3_mem_ck               ( HPS_DDR3_CK_P ),
      .hps_ddr3_mem_ck_n             ( HPS_DDR3_CK_N ),
      .hps_ddr3_mem_cke              ( HPS_DDR3_CKE ),
      .hps_ddr3_mem_cs_n             ( HPS_DDR3_CS_N ),
      .hps_ddr3_mem_ras_n            ( HPS_DDR3_RAS_N ),
      .hps_ddr3_mem_cas_n            ( HPS_DDR3_CAS_N ),
      .hps_ddr3_mem_we_n             ( HPS_DDR3_WE_N ),
      .hps_ddr3_mem_reset_n          ( HPS_DDR3_RESET_N ),
      .hps_ddr3_mem_dq               ( HPS_DDR3_DQ ),
      .hps_ddr3_mem_dqs              ( HPS_DDR3_DQS_P ),
      .hps_ddr3_mem_dqs_n            ( HPS_DDR3_DQS_N ),
      .hps_ddr3_mem_odt              ( HPS_DDR3_ODT ),
      .hps_ddr3_mem_dm               ( HPS_DDR3_DM ),
      .hps_ddr3_oct_rzqin            ( HPS_DDR3_RZQ ),

      .hps_hps_io_emac1_inst_TX_CLK ( HPS_ENET_GTX_CLK ),
      .hps_hps_io_emac1_inst_TXD0    ( HPS_ENET_TX_DATA[0] ),
      .hps_hps_io_emac1_inst_TXD1    ( HPS_ENET_TX_DATA[1] ),
      .hps_hps_io_emac1_inst_TXD2    ( HPS_ENET_TX_DATA[2] ),
      .hps_hps_io_emac1_inst_TXD3    ( HPS_ENET_TX_DATA[3] ),
      .hps_hps_io_emac1_inst_RXD0    ( HPS_ENET_RX_DATA[0] ),
      .hps_hps_io_emac1_inst_MDIO    ( HPS_ENET_MDIO  ),
      .hps_hps_io_emac1_inst_MDC     ( HPS_ENET_MDC   ),
      .hps_hps_io_emac1_inst_RX_CTL ( HPS_ENET_RX_DV ),
      .hps_hps_io_emac1_inst_TX_CTL ( HPS_ENET_TX_EN ),
      .hps_hps_io_emac1_inst_RX_CLK ( HPS_ENET_RX_CLK ),
      .hps_hps_io_emac1_inst_RXD1    ( HPS_ENET_RX_DATA[1]  ),
      .hps_hps_io_emac1_inst_RXD2    ( HPS_ENET_RX_DATA[2]  ),
      .hps_hps_io_emac1_inst_RXD3    ( HPS_ENET_RX_DATA[3]  ),

      .hps_hps_io_sdio_inst_CMD      ( HPS_SD_CMD           ),
      .hps_hps_io_sdio_inst_D0       ( HPS_SD_DATA[0]       ),
      .hps_hps_io_sdio_inst_D1       ( HPS_SD_DATA[1]       ),
      .hps_hps_io_sdio_inst_CLK      ( HPS_SD_CLK           ),
```

```
    .hps_hps_io_sdio_inst_D2      ( HPS_SD_DATA[2]        ),
    .hps_hps_io_sdio_inst_D3      ( HPS_SD_DATA[3]        ),

    .hps_hps_io_usb1_inst_D0      ( HPS_USB_DATA[0]       ),
    .hps_hps_io_usb1_inst_D1      ( HPS_USB_DATA[1]       ),
    .hps_hps_io_usb1_inst_D2      ( HPS_USB_DATA[2]       ),
    .hps_hps_io_usb1_inst_D3      ( HPS_USB_DATA[3]       ),
    .hps_hps_io_usb1_inst_D4      ( HPS_USB_DATA[4]       ),
    .hps_hps_io_usb1_inst_D5      ( HPS_USB_DATA[5]       ),
    .hps_hps_io_usb1_inst_D6      ( HPS_USB_DATA[6]       ),
    .hps_hps_io_usb1_inst_D7      ( HPS_USB_DATA[7]       ),
    .hps_hps_io_usb1_inst_CLK     ( HPS_USB_CLKOUT        ),
    .hps_hps_io_usb1_inst_STP     ( HPS_USB_STP           ),
    .hps_hps_io_usb1_inst_DIR     ( HPS_USB_DIR           ),
    .hps_hps_io_usb1_inst_NXT     ( HPS_USB_NXT           ),

    .hps_hps_io_spim1_inst_CLK    ( HPS_SPIM_CLK  ),
    .hps_hps_io_spim1_inst_MOSI   ( HPS_SPIM_MOSI ),
    .hps_hps_io_spim1_inst_MISO   ( HPS_SPIM_MISO ),
    .hps_hps_io_spim1_inst_SS0    ( HPS_SPIM_SS   ),

    .hps_hps_io_uart0_inst_RX     ( HPS_UART_RX     ),
    .hps_hps_io_uart0_inst_TX     ( HPS_UART_TX     ),

    .hps_hps_io_i2c0_inst_SDA     ( HPS_I2C1_SDAT   ),
    .hps_hps_io_i2c0_inst_SCL     ( HPS_I2C1_SCLK   ),

    .hps_hps_io_i2c1_inst_SDA     ( HPS_I2C2_SDAT   ),
    .hps_hps_io_i2c1_inst_SCL     ( HPS_I2C2_SCLK   ),

    .hps_hps_io_gpio_inst_GPIO09  ( HPS_CONV_USB_N ),
    .hps_hps_io_gpio_inst_GPIO35  ( HPS_ENET_INT_N ),
    .hps_hps_io_gpio_inst_GPIO40  ( HPS_LTC_GPIO ),

    .hps_hps_io_gpio_inst_GPIO48  ( HPS_I2C_CONTROL ),
    .hps_hps_io_gpio_inst_GPIO53  ( HPS_LED ),
    .hps_hps_io_gpio_inst_GPIO54  ( HPS_KEY ),
    .hps_hps_io_gpio_inst_GPIO61  ( HPS_GSENSOR_INT ),


    .midi_key(KEY[3:0]),
    .midi_l_chan_ready(rdy_left),
```

```
    .midi_r_chan_ready(rdy_right),
    .midi_s_data(sample),
    .midi_s_valid(sample_valid),


    .audio_0_avalon_left_channel_sink_data(sample),
    .audio_0_avalon_left_channel_sink_valid(sample_valid),
    .audio_0_avalon_left_channel_sink_ready(rdy_left),
    .audio_0_avalon_right_channel_sink_data(sample),
    .audio_0_avalon_right_channel_sink_valid(sample_valid),
    .audio_0_avalon_right_channel_sink_ready(rdy_right),

    .audio_0_external_interface_BCLK(AUD_BCLK),
    .audio_0_external_interface_DACDAT(AUD_DACDAT),
    .audio_0_external_interface_DACLRCK(AUD_DACLRCK),

    .audio_and_video_config_0_external_interface_SDAT(FPGA_I2C_SDAT),
    .audio_and_video_config_0_external_interface_SCLK(FPGA_I2C_SCLK),

    .audio_pll_0_audio_clk_clk(AUD_XCK)

);

 logic rdy_left;
 logic rdy_right;
 logic [15:0] sample;
 logic sample_valid;
 // The following quiet the "no driver" warnings for output
 // pins and should be removed if you use any of these peripherals

 assign ADC_CS_N = SW[1] ? SW[0] : 1'bZ;
 assign ADC_DIN = SW[0];
 assign ADC_SCLK = SW[0];

 //assign AUD_ADCLRCK = SW[1] ? SW[0] : 1'bZ;   //didnt use it
 //assign AUD_BCLK = SW[1] ? SW[0] : 1'bZ;
 //assign AUD_DACDAT = SW[0];
 //assign AUD_DACLRCK = SW[1] ? SW[0] : 1'bZ;
 //assign AUD_XCK = SW[0];

 assign DRAM_ADDR = { 13{ SW[0] } };
 assign DRAM_BA = { 2{ SW[0] } };
```

```
   assign DRAM_DQ = SW[1] ? { 16{ SW[0] } } : 16'bZ;
   assign {DRAM_CAS_N, DRAM_CKE, DRAM_CLK, DRAM_CS_N,
      DRAM_LDQM, DRAM_RAS_N, DRAM_UDQM, DRAM_WE_N} = { 8{SW[0]} };

   assign FAN_CTRL = SW[0];

   //assign FPGA_I2C_SCLK = SW[0];
   //assign FPGA_I2C_SDAT = SW[1] ? SW[0] : 1'bZ;

   assign GPIO_0 = SW[1] ? { 36{ SW[0] } } : 36'bZ;
   assign GPIO_1 = SW[1] ? { 36{ SW[0] } } : 36'bZ;

   assign HEX0 = { 7{ SW[1] } };
   assign HEX1 = { 7{ SW[2] } };
   assign HEX2 = { 7{ SW[3] } };
   assign HEX3 = { 7{ SW[4] } };
   assign HEX4 = { 7{ SW[5] } };
   assign HEX5 = { 7{ SW[6] } };

   assign IRDA_TXD = SW[0];

   assign LEDR = { 10{SW[7]} };

   assign PS2_CLK = SW[1] ? SW[0] : 1'bZ;
   assign PS2_CLK2 = SW[1] ? SW[0] : 1'bZ;
   assign PS2_DAT = SW[1] ? SW[0] : 1'bZ;
   assign PS2_DAT2 = SW[1] ? SW[0] : 1'bZ;

   assign TD_RESET_N = SW[0];

   assign {VGA_R, VGA_G, VGA_B} = { 24{ SW[0] } };
   assign {VGA_BLANK_N, VGA_CLK,
      VGA_HS, VGA_SYNC_N, VGA_VS} = { 5{ SW[0] } };


endmodule
```

Note_top

Arbiter.sv

```
module arbiter (
    input logic clk,
    input logic reset,
    //for note_top
    input logic get_next_sample,
       output logic notes_ready,

    //for wave table
    output logic wave_address_ready,
    input logic wave_sample_valid,
    input logic signed [15:0] wave_sample,
    output logic [15:0] wave_address,
    //for note modules
    output logic signed [15:0] note_sample [9:0],
    input logic [15:0] note_address [9:0],
    output logic [9:0] note_sample_valid,
    input logic [9:0] note_address_ready
);

logic [3:0] counter;
integer i;

always_ff @(posedge clk) begin

    if (!reset) begin
       if (get_next_sample && !notes_ready) begin
             if(counter < 4'd10 ) begin
                   if (note_address_ready[counter]) begin
                         wave_address <= note_address[counter];
                         wave_address_ready <= 1;
                   end
                   if (wave_sample_valid && wave_address_ready) begin
                         note_sample[counter] <= wave_sample;
                         note_sample_valid[counter] <= 1;
                         counter <= counter + 1;
                         wave_address_ready <= 0;
                   end
```

```
            end else begin
                  notes_ready <= 1;
                  counter <= 0;
                 note_sample_valid <= 10'd0;
            end
      end else if (!get_next_sample) begin
            notes_ready <= 0;
      end
   end else begin
      for (i = 0; i < 10; i= i+1) begin
            note_sample_valid[i] <= 0;
      end

      counter <= 0;
      notes_ready <= 0;
      wave_address_ready <= 0;
   end
end

endmodule
```

Note_top.sv

```
module note_top (
    input clk,
    input reset,
    //from wave table
    output logic wave_address_ready,
    input logic wave_sample_valid,
    input logic signed [15:0] wave_sample,
    output logic [15:0] wave_address,
    //from MIDI top
    input note_data_valid,
    input logic [15:0] note_data,
    input logic [3:0] note_addr,
    //to audio codec
    input logic audio_codec_ready,
    output logic codec_sample_valid,
    output logic signed [15:0] codec_sample_data
);
```

```
logic get_next_sample;
logic arbiter_done;
//logic [9:0] push_reg;
logic [15:0] note_address [9:0];
logic signed [15:0] note_sample [9:0];
logic [9:0] note_sample_valid;
logic signed [15:0] adjusted_sample [9:0];
logic [9:0] adjusted_sample_valid;
logic [9:0] note_address_ready;
logic notes_ready_in;
logic [15:0] note_settings [9:0];
logic [15:0] amplitudes [9:0];


//logic [3:0] num_active_notes;

arbiter note_arbiter (.clk(clk), .reset(reset),
.get_next_sample(get_next_sample), .notes_ready(arbiter_done),
.wave_address_ready(wave_address_ready),
.wave_sample_valid(wave_sample_valid), .wave_sample(wave_sample),
.wave_address(wave_address), .note_sample(note_sample),
.note_address(note_address), .note_sample_valid(note_sample_valid),
.note_address_ready(note_address_ready));

summer s (.reset(reset), .notes_in(adjusted_sample),
.amplitudes(amplitudes), .wave_out(codec_sample_data));

/*genvar i;
generate
    for (i = 0; i < 10; i = i+1) begin: n
       note inst(
                  .clk,
                  .rst(reset),
                  .settings(note_settings[i]),
                  .sample_out(adjusted_sample[i]),
                  .valid_out(adjusted_sample_valid[i]),
                  .ready_in(notes_ready_in),
                  .sample_in(note_sample[i]),
                  .valid_in(note_sample_valid[i]),
                  .sample_address(note_address[i]),
                  .ready_out(note_address_ready[i]),
                  .amplitude(amplitudes[i]));
```

```
    end
endgenerate*/

note n0(.clk(clk), .rst(reset), .settings(note_settings[0]),
.sample_out(adjusted_sample[0]), .valid_out(adjusted_sample_valid[0]),
.ready_in(notes_ready_in), .sample_in(note_sample[0]),
.valid_in(note_sample_valid[0]),
.sample_address(note_address[0]), .ready_out(note_address_ready[0]),
.amplitude(amplitudes[0]));
note n1(.clk(clk), .rst(reset), .settings(note_settings[1]),
.sample_out(adjusted_sample[1]), .valid_out(adjusted_sample_valid[1]),
.ready_in(notes_ready_in), .sample_in(note_sample[1]),
.valid_in(note_sample_valid[1]),
.sample_address(note_address[1]), .ready_out(note_address_ready[1]),
.amplitude(amplitudes[1]));
note n2(.clk(clk), .rst(reset), .settings(note_settings[2]),
.sample_out(adjusted_sample[2]), .valid_out(adjusted_sample_valid[2]),
.ready_in(notes_ready_in), .sample_in(note_sample[2]),
.valid_in(note_sample_valid[2]),
.sample_address(note_address[2]), .ready_out(note_address_ready[2]),
.amplitude(amplitudes[2]));
note n3(.clk(clk), .rst(reset), .settings(note_settings[3]),
.sample_out(adjusted_sample[3]), .valid_out(adjusted_sample_valid[3]),
.ready_in(notes_ready_in), .sample_in(note_sample[3]),
.valid_in(note_sample_valid[3]),
.sample_address(note_address[3]), .ready_out(note_address_ready[3]),
.amplitude(amplitudes[3]));
note n4(.clk(clk), .rst(reset), .settings(note_settings[4]),
.sample_out(adjusted_sample[4]), .valid_out(adjusted_sample_valid[4]),
.ready_in(notes_ready_in), .sample_in(note_sample[4]),
.valid_in(note_sample_valid[4]),
.sample_address(note_address[4]), .ready_out(note_address_ready[4]),
.amplitude(amplitudes[4]));
note n5(.clk(clk), .rst(reset), .settings(note_settings[5]),
.sample_out(adjusted_sample[5]), .valid_out(adjusted_sample_valid[5]),
.ready_in(notes_ready_in), .sample_in(note_sample[5]),
.valid_in(note_sample_valid[5]),
.sample_address(note_address[5]), .ready_out(note_address_ready[5]),
.amplitude(amplitudes[5]));
note n6(.clk(clk), .rst(reset), .settings(note_settings[6]),
.sample_out(adjusted_sample[6]), .valid_out(adjusted_sample_valid[6]),
.ready_in(notes_ready_in), .sample_in(note_sample[6]),
```

```
.valid_in(note_sample_valid[6]),
.sample_address(note_address[6]), .ready_out(note_address_ready[6]),
.amplitude(amplitudes[6]));
note n7(.clk(clk), .rst(reset), .settings(note_settings[7]),
.sample_out(adjusted_sample[7]), .valid_out(adjusted_sample_valid[7]),
.ready_in(notes_ready_in), .sample_in(note_sample[7]),
.valid_in(note_sample_valid[7]),
.sample_address(note_address[7]), .ready_out(note_address_ready[7]),
.amplitude(amplitudes[7]));
note n8(.clk(clk), .rst(reset), .settings(note_settings[8]),
.sample_out(adjusted_sample[8]), .valid_out(adjusted_sample_valid[8]),
.ready_in(notes_ready_in), .sample_in(note_sample[8]),
.valid_in(note_sample_valid[8]),
.sample_address(note_address[8]), .ready_out(note_address_ready[8]),
.amplitude(amplitudes[8]));
note n9(.clk(clk), .rst(reset), .settings(note_settings[9]),
.sample_out(adjusted_sample[9]), .valid_out(adjusted_sample_valid[9]),
.ready_in(notes_ready_in), .sample_in(note_sample[9]),
.valid_in(note_sample_valid[9]),
.sample_address(note_address[9]), .ready_out(note_address_ready[9]),
.amplitude(amplitudes[9]));

enum integer {IDLE=0, ARBITER=1, NOTE=2, VALID=3} state;

always_ff @(posedge clk) begin
    if (!reset) begin
       //sends data to audio codec
       /*if (audio_codec_ready) begin
             notes_ready_in <= 1;
             if (!codec_sample_valid) begin
                   if(!arbiter_done)
                         get_next_sample <= 1;
                   else
                         get_next_sample <= 0;

                   if (adjusted_sample_valid == 10'b1111111111) begin
                         codec_sample_valid <= 1;
                   end
             end else begin
                   codec_sample_valid <= 1;
             end
             //get_next_note <= 0;
```

```
	end else begin
		notes_ready_in <= 0;
		get_next_sample <= 0;
		codec_sample_valid <= 0;
	end*/

	case(state)
		IDLE:	if(audio_codec_ready) begin
				state <= ARBITER;
				codec_sample_valid <= 0;
				get_next_sample <= 1;
				notes_ready_in <= 1;
			end else begin
				codec_sample_valid <= 0;
				get_next_sample <= 0;
				notes_ready_in <= 0;
			end
		ARBITER:if(arbiter_done) begin
				state <= NOTE;
				codec_sample_valid <= 0;
				get_next_sample <= 0;
				notes_ready_in <= 1;
			end else begin
				codec_sample_valid <= 0;
				get_next_sample <= 1;
				notes_ready_in <= 1;
			end
		NOTE:	if(adjusted_sample_valid == 10'b1111111111) begin
				state <= VALID;
				codec_sample_valid <= 1;
				get_next_sample <= 0;
				notes_ready_in <= 1;
			end else begin
				codec_sample_valid <= 0;
				get_next_sample <= 0;
				notes_ready_in <= 1;
			end
		VALID:	 if(!audio_codec_ready) begin
				state <= IDLE;
				codec_sample_valid <= 0;
				get_next_sample <= 0;
				notes_ready_in <= 0;
```

```
                        end else begin
                              codec_sample_valid <= 1;
                              get_next_sample <= 0;
                              notes_ready_in <= 1;
                        end
        endcase

        //sends data to notes
        if (note_data_valid) begin
              note_settings[note_addr] <= note_data;
        end

    end else begin
        get_next_sample <= 0;
        codec_sample_valid <= 0;
        notes_ready_in <= 0;
        for(integer i = 0; i < 10; i = i+1) begin
              note_settings[i] <= 16'b0000_0000_0_0000000;
        end
        state <= IDLE;
    end
end
```

Note.sv

```
module note (
            // control
            input clk,
            input rst,
            input logic [15:0] settings ,
            // [15:12] = octave
            // [11:8] = note
            // [7] = pressed
            // [6:0] = volume

            // to/from summer
            output logic signed [15:0] sample_out,
            output logic [15:0] amplitude,
            output logic valid_out,
            input ready_in,

            // to/from arbiter
```

```
            input logic signed [15:0] sample_in,
            input valid_in,
            output logic [15:0] sample_address,
            output logic ready_out
        );

        logic [14:0] duration; // how long since note began
        logic [14:0] released_duration; // how long note has been released
        logic smpl_clk; // high each time a new sample is requested

        // for sampling logic
        enum integer {IDLE=0, READY=1, VALID=2} state;

        // octave 8 mem_offset values
      logic [15:0] notes [12] = '{//        +-------------------------
         4186, // C                    |           _________________
         4435, // C-sharp and D-flat   |---------|_________________
         4698, // D                    |           _________________
         4978, // D-sharp and E-flat   |---------|_________________
         5274, // E                    |_____________________________
         5588, // F                    |           _________________
         5920, // F-sharp and G-flat   |---------|_________________
         6272, // G                    |           _________________
         6645, // G-sharp and A-flat   |---------|_________________
         7040, // A                    |           _________________
         7459, // A-sharp and B-flat   |---------|_________________
         7902  // B                    |
      };            //                      +-------------------------

        // settings
        logic [3:0] octave; // can only be 0-8, lower octaves are higher
values
        logic [3:0] note; // can only be 0-11
        logic pressed; // is the note pressed down
        logic last_pressed; // value of pressed in last clock cycle
        logic [6:0] volume;
        logic [15:0] mem_offset; // function of octave and note
        assign octave = settings[15:12];
        assign note = settings[11:8];
        assign pressed = settings[7];
        assign volume = settings[6:0];
        assign mem_offset = notes[note] >> octave;
```

```
logic [15:0] next_addr; // modified by mem_offset
assign next_addr = sample_address + mem_offset;

always_ff @(posedge clk) begin
      last_pressed <= pressed;

      // reset logic
      if(rst) begin
            state <= IDLE;
            sample_address <= 16'd0;
            ready_out <= 0;
            valid_out <= 0;
            duration <= 15'd0;
            released_duration <= 15'd0;
            smpl_clk <= 0;
      end else begin
            smpl_clk <= state == IDLE && ready_in;

            // reset dnote_sampleuration if the note was just pressed
            if (pressed & !last_pressed) begin
            duration <= 15'd0;
            released_duration <= 15'd0;
            end

            // sampling ready-valid handshake logic
            case (state)
            ////////////////////////////
            IDLE : if (ready_in) begin
                        state <= READY;
                        ready_out <= 1;
                        valid_out <= 0;
                  end else begin
                        state <= IDLE;
                        ready_out <= 0;
                        valid_out <= 0;
                  end

            ////////////////////////////
            READY: if (ready_in) begin
                        if (valid_in) begin
                              state <= VALID;
```

```
                              ready_out <= 0;
                              valid_out <= 1;
                        end else begin
                              state <= READY;
                              ready_out <= 1;
                              valid_out <= 0;
                        end
                  end else begin
                        state <= IDLE;
                        ready_out <= 0;
                        valid_out <= 0;
                  end

            //////////////////////////
            VALID: if (ready_in) begin
                        state <= VALID;
                        ready_out <= 0;
                        valid_out <= 1;
                  end else begin
                        state <= IDLE;
                        ready_out <= 0;
                        valid_out <= 0;
                  end
            endcase


            if (smpl_clk) begin
            // update address
            if (next_addr > 16'd47999)
                  sample_address <= next_addr - 16'd48000;
            else
                  sample_address <= next_addr;

            // update durations
            if (!pressed)
                  if (released_duration < 15'd9000)
                        released_duration <= released_duration +
15'd1;
            if (duration < 15'd19000)
                  duration <= duration + 15'd1;
      end
      end
```

```
	end

	// generating a sample
	adsr_envelope env(.*);

endmodule


// applies envelope effect on note
module adsr_envelope (
		input clk,
		input smpl_clk,
		input rst,
		input logic [14:0] duration,
		input logic [14:0] released_duration,
		input logic signed [15:0] sample_in,
		input logic [6:0] volume,
		input pressed,
		output logic signed [15:0] sample_out,
		output logic [15:0] amplitude
	);

	logic [9:0] attack_counter;
	logic [9:0] retreat_counter;
	logic [9:0] ac_div = 93;
	logic [9:0] dc_div = 300;
	logic [9:0] rc_div = 83;

	logic [7:0] envelope;
	logic signed [29:0] product;

	always_ff @(posedge clk) begin
		if (rst) begin
			attack_counter <= 0;
			retreat_counter <= 0;
			amplitude <= 0;
			sample_out <= 0;
			envelope <= 0;
		end else begin
			// reset when a new note is assigned
			if (duration == 15'd0) begin
			attack_counter <= 0;
```

```
                retreat_counter <= 0;
                end

                if (smpl_clk) begin
                // increment counters
                attack_counter <= attack_counter + 10'd1;
                if (!pressed)
                      retreat_counter <= retreat_counter + 10'd1;

                // handle attack counter clock div values
                if (duration < 15'd12000 && attack_counter == ac_div)
begin
                      attack_counter <= 10'd0;
            envelope <=  envelope + 8'd1;
                end else if (duration < 15'd18000 && attack_counter ==
dc_div) begin
                      attack_counter <= 10'd0;
                      // TODO
                            envelope <= envelope - 8'd1;
                end

                // handle retreat counter clock div values
                if (released_duration < 15'd9000 && retreat_counter ==
rc_div) begin
                      retreat_counter <= 10'd0;
                      envelope <= envelope - 8'd1;
                end
                end

                amplitude <= envelope * volume;
                product <= $signed(amplitude) * sample_in;
       sample_out <= product / (1<<14);
            end
      end
endmodule
```

summer.sv

```
module frac_mult (
       input reset,
       input logic signed [17:0] sum,
       input logic [7:0] frac,
```

```
        output logic signed [15:0] wave_out);

    //TODO test to make sure it works with negative numbers
    always_comb begin
        if(reset)
                wave_out = 0;
        else if (frac != 0)
                wave_out = frac[7] * (sum >> 1) + frac[6] * (sum >> 2) +
frac[5] * (sum >> 3) + frac[4] * (sum >> 4) + frac[3] * (sum >> 5) +
frac[2] * (sum >> 6) + frac[1] * (sum >> 7) + frac[0] * (sum >> 8);
        else
                wave_out = sum[15:0];
    end
endmodule



module summer (
        //input clk,
        input reset,
        input logic signed [15:0] notes_in [9:0],
        input logic [15:0] amplitudes [9:0],
        output logic signed [15:0] wave_out
    );

logic signed [17:0] sum;
logic [17:0] amp_sum;
logic [7:0] frac;

frac_mult fm (.*);

always_comb begin
        amp_sum = amplitudes[9] + amplitudes[8] + amplitudes[7] +
amplitudes[6] + amplitudes[5] + amplitudes[4] + amplitudes[3] +
amplitudes[2] + amplitudes[1] + amplitudes[0];
        sum = notes_in[9] + notes_in[8] + notes_in[7] + notes_in[6] +
notes_in[5] + notes_in[4] + notes_in[3] + notes_in[2] + notes_in[1] +
notes_in[0];
        case (amp_sum[17:12])
                6'd0: frac = 8'h0;
                6'd1: frac = 8'hF0;
                6'd2: frac = 8'hE8;
```

```
            6'd3: frac = 8'hE0;
            6'd4: frac = 8'hDB;
            6'd5: frac = 8'hD6;
            6'd6: frac = 8'hCA;
            6'd7: frac = 8'hBE;
            6'd8: frac = 8'hB2;
            6'd9: frac = 8'hA9;
            6'd10: frac = 8'hA2;
            6'd11: frac = 8'h9C;
            6'd12: frac = 8'h96;
            6'd13: frac = 8'h90;
            6'd14: frac = 8'h8A;
            6'd15: frac = 8'h85;
            6'd16: frac = 8'h80;
            6'd17: frac = 8'h7C;
            6'd18: frac = 8'h78;
            6'd19: frac = 8'h74;
            6'd20: frac = 8'h70;
            6'd21: frac = 8'h6C;
            6'd22: frac = 8'h68;
            6'd23: frac = 8'h64;
            6'd24: frac = 8'h60;
            6'd25: frac = 8'h5D;
            6'd26: frac = 8'h5B;
            6'd27: frac = 8'h58;
            6'd28: frac = 8'h53;
            6'd29: frac = 8'h50;
            6'd30: frac = 8'h4E;
            6'd31: frac = 8'h4C;
            6'd32: frac = 8'h4A;
            6'd33: frac = 8'h48;
            6'd34: frac = 8'h46;
            6'd35: frac = 8'h44;
            6'd36: frac = 8'h43;
            6'd37: frac = 8'h42;
            6'd38: frac = 8'h41;
            6'd39: frac = 8'h40;
            default: frac = 8'h00;
        endcase
end

endmodule
```

Tb_arbiter.sv

```
module arbiter_tb ();

logic clk;
logic reset;

logic get_next_sample;
logic notes_ready;

//for wave table
logic wave_address_ready;
logic wave_sample_valid;
logic [15:0] wave_sample;
logic [15:0] wave_address;

//for notes
logic [15:0] note_sample [9:0];
logic [15:0] note_address [9:0];
logic note_sample_valid [9:0];
logic note_address_ready [9:0];

integer i;
logic [2:0] runcount;

always_ff @(posedge clk) begin
    if (reset) begin
       get_next_sample <= 0;
       wave_address_ready <= 0;
       notes_ready <= 0;
       runcount <= 1;

       for (i = 0; i < 10; i= i+1) begin
                   note_sample_valid[i] <= 0;
       end

    end else begin
       if (notes_ready == 0) begin
             get_next_sample <= 1;
             for (i = 0; i < 10; i= i+1) begin
                   note_address_ready[i] <= 1;
```

```
                  note_address[i] <= i*runcount;
            end

            if(wave_address_ready) begin
                  wave_sample <= wave_address;
                  wave_sample_valid <= 1;
            end else begin
                  wave_sample_valid <= 0;
            end

      end else begin
            runcount = runcount + 1;
            get_next_sample <= 0;
      end
   end
end


arbiter dut(.clk, .reset,
   //for note_top
   .get_next_sample,
      .notes_ready,

   //for wave table
   .wave_address_ready,
   .wave_sample_valid,
   .wave_sample,
   .wave_address,
   //for note modules
   .note_sample,
   .note_address,
   .note_sample_valid,
   .note_address_ready
);

initial begin
      reset = 1;
      clk = 0;
      #10 reset = 0;
end

always begin
```

```
      #1 clk = !clk;
end

endmodule
```

Tb_note_top.sv

```
//`timescale 10ns / 1ns;

module tb_note_top (
      output logic left_out,
      output logic right_out
);

    logic clk;
    logic reset;
    //for wave table
    logic wave_address_ready;
    logic wave_sample_valid;
    logic signed [15:0] wave_sample;
    logic [15:0] wave_address;
    //from MIDI top
    logic note_data_valid;
    logic [15:0] note_data;
    logic [3:0] note_addr;
    //to audio codec
    logic left_ready;
    logic right_ready;
    logic codec_sample_valid;
    logic signed [15:0] codec_sample_data;

    note_top nt(.audio_codec_ready(left_ready && right_ready), .*);

    audio_fifos af( // this is the audio codec
       .sample_left(codec_sample_data),
       .sample_right(codec_sample_data),
       .sample_left_valid(codec_sample_valid),
       .sample_right_valid(codec_sample_valid),
       .*
    );
```

```
    source src(.*); // this is the wavetable

    logic [12:0] clk_counter;
    logic [1:0] note_counter;
    //simulates midi top
    always_ff @(posedge clk) begin
       if (!reset) begin
             clk_counter <= clk_counter + 1;
             if (clk_counter < 10) begin
                   note_addr <= clk_counter;
                   note_data_valid <= 1;
             end else if(clk_counter == 10) begin
                   case (note_counter)
                         2'd0: note_data <= 16'b0000_0000_1_1111111;
                         2'd1: note_data <= 16'b0000_0000_1_1111111;
                         2'd2: note_data <= 16'b0000_0000_1_1111111;
                         2'd3: note_data <= 16'b0000_0000_1_1111111;
                   endcase
                   note_data_valid <= 0;
                   note_counter <= note_counter + 1;
             end
       end else begin
             clk_counter <= 0;
             note_data <= 0;
             note_counter <= 0;
       end
    end

    initial begin
       reset = 1;
       clk = 0;
       #10 reset = 0;
    end

    always begin
       #1 clk = !clk;
    end

endmodule

module source (
      input clk,
```

```
      input reset,
      output logic signed [15:0] wave_sample,
      output logic wave_sample_valid,
      input logic [15:0] wave_address,
      input wave_address_ready
);

    logic [2:0] count;

    always_ff @(posedge clk) begin
       if (reset) begin
            count <= 0;
            wave_sample <= 0;
            wave_sample_valid <= 0;
       end else begin
            if (wave_address_ready) begin
            count <= (count == 3'd2) ? count : count + 3'd1;
                  wave_sample <= (wave_address < 16'd24000) ? 16'd13100 :
-16'd13100;
                  if(count == 3'd2) begin
                        wave_sample_valid <= 1;
                  end
            end else begin
            count <= 0;
                  wave_sample_valid <= 0;
             end
       end
    end

endmodule
```

Tb_note.sv

```
`timescale 10ns / 1ns;

module tb_note (
      output logic signed [29:0] sample_left,
      output logic signed [29:0] sample_right
);

logic clk;
```

```
logic rst;
logic [15:0] settings;

// to/from summer
logic signed [29:0] sample_out;
logic [15:0] amplitude;
logic valid_out;
logic ready_in;

// to/from arbiter
logic signed [15:0] sample_in;
logic valid_in;
logic [15:0] sample_address;
logic ready_out;

note n(.*);
source src(.*);
sink snk(.*);

assign sample_left = sample_out;
assign sample_right = sample_out;

initial begin
      rst = 1;
      clk = 0;
      settings = 16'h00ff;
      #10 rst = 0;
end

always begin
      #1 clk = ~clk;
end
endmodule

// plays role of sample buffers
module source (
      input clk,
      input rst,
      output logic signed [15:0] sample_in,
      output logic valid_in,
      input logic [15:0] sample_address,
      input ready_out
```

```
);

logic [2:0] count;

always_ff @(posedge clk) begin
      if (rst) begin
      count <= 0;
      sample_in <= 0;
      valid_in <= 0;
      end else begin
      if (ready_out)
            count <= (count == 3'd4) ? count : count + 3'd1;
      else
            count <= 0;
      valid_in = count == 3'd4;
      sample_in <= (sample_address > 16'd24000) ? 16'd255 : -16'd255;
      end
end
endmodule

// plays role of audio codec
module sink (
      input clk,
      input rst,
      output logic ready_in,
      input valid_out
);

logic [2:0] count;

always_ff @(posedge clk) begin
      if (rst) begin
      count <= 0;
      ready_in <= 0;
      end else begin
      if (valid_out)
            count <= 0;
      else
            count <= (count == 3'd3) ? count : count + 3'd1;
      ready_in <= count == 3'd3;
      end
end
```

```
endmodule
```

Tb_summer.sv

```
module summer_tb (output logic [15:0] wave);

logic clk;
logic reset;

logic [15:0] note1;
logic [15:0] note2;
logic [15:0] note3;
logic [15:0] note4;
logic [15:0] note5;
logic [15:0] note6;
logic [15:0] note7;
logic [15:0] note8;
logic [15:0] note9;
logic [15:0] note10;


logic [3:0] valid_notes;

logic [8:0] counter;

//input to note mixer
always_ff @(posedge clk) begin
    if(reset) begin
       counter <= 0;
       valid_notes <= 0;
    end else begin
       if (counter < 10) begin
             counter <= counter + 1;
             note1 <= 10;
             note2 <= 10;
             note3 <= 10;
             note4 <= 10;
             note5 <= 10;
             note6 <= 10;
             note7 <= 10;
             note8 <= 10;
```

```
            note9 <= 10;
            note10 <= 10;
        end else if (valid_notes < 10) begin
            counter <= 0;
            valid_notes = valid_notes + 1;
        end else begin
            reset <= 0;
        end
    end
end


summer dut(.reset, .note1, .note2, .note3, .note4, .note5, .note6, .note7,
.note8, .note9, .note10, .valid_notes, .wave_out(wave));

initial begin
      reset = 1;
      clk = 0;
      #10 reset = 0;
end

always begin
      #1 clk = !clk;
end

endmodule
```

Tonegen

Soc_system_top.sv

```
// ====================================================================
// Copyright (c) 2013 by Terasic Technologies Inc.
// ====================================================================
//
// Modified 2019 by Stephen A. Edwards
//
// Permission:
//
//   Terasic grants permission to use and modify this code for use in
//   synthesis for all Terasic Development Boards and Altera
```

```
//   Development Kits made by Terasic.  Other use of this code,
//   including the selling ,duplication, or modification of any
//   portion is strictly prohibited.
//
// Disclaimer:
//
//   This VHDL/Verilog or C/C++ source code is intended as a design
//   reference which illustrates how these types of functions can be
//   implemented.  It is the user's responsibility to verify their
//   design for consistency and functionality through the use of
//   formal verification methods.  Terasic provides no warranty
//   regarding the use or functionality of this code.
//
// ============================================================
//
//  Terasic Technologies Inc

//  9F., No.176, Sec.2, Gongdao 5th Rd, East Dist, Hsinchu City, 30070.
Taiwan
//
//
//                     web: http://www.terasic.com/
//                     email: support@terasic.com
module soc_system_top(

 ///////// AUD /////////
 input            AUD_ADCDAT,
 inout            AUD_ADCLRCK,
 inout            AUD_BCLK,
 output     AUD_DACDAT,
 inout            AUD_DACLRCK,
 output     AUD_XCK,

 ///////// CLOCK /////////
 input            CLOCK_50,

 ///////// KEY /////////
 input [3:0]   KEY,

 ///////// FPGA /////////
 output     FPGA_I2C_SCLK,
 inout            FPGA_I2C_SDAT
```

```
);

   logic [3:0] note;
   logic [2:0] octave = 3;
   logic [7:0] volume = 255;
   logic rdy_left;
   logic rdy_right;
   logic [15:0] sample;
   logic sample_valid;

   assign note = KEY[2:0];

   tonegen tg(.clk(CLOCK_50), .reset(~KEY[3]), .volume(volume),
.note(note), .octave(octave), .left_chan_ready(rdy_left),
      .right_chan_ready(rdy_right), .sample_data(sample),
.sample_valid(sample_valid));

   soc_system soc_system0(
      .clk_clk                       ( CLOCK_50 ),
      .reset_reset_n                 ( 1'b1 ),

      .audio_0_avalon_left_channel_sink_data          (sample),
      .audio_0_avalon_left_channel_sink_valid         (sample_valid),
      .audio_0_avalon_left_channel_sink_ready         (rdy_left),
      .audio_0_avalon_right_channel_sink_data         (sample),
      .audio_0_avalon_right_channel_sink_valid        (sample_valid),
      .audio_0_avalon_right_channel_sink_ready        (rdy_right),
      .audio_0_external_interface_BCLK                (AUD_BCLK),
      .audio_0_external_interface_DACDAT              (AUD_DACDAT),
      .audio_0_external_interface_DACLRCK             (AUD_DACLRCK),
      .audio_pll_0_audio_clk_clk                      (AUD_XCK),
      .audio_and_video_config_0_external_interface_SDAT (FPGA_I2C_SDAT),
      .audio_and_video_config_0_external_interface_SCLK (FPGA_I2C_SCLK)
  );

endmodule
```

Tonegen.sv

```
module tonegen (
       input clk,
       input reset,
```

```
    input logic [7:0] volume,
    input logic [3:0] note,
    input logic [2:0] octave,
    input left_chan_ready,
    input right_chan_ready,
    output logic [15:0] sample_data,
    output logic sample_valid
);

/* send a square wave audio signal to the codec */
logic [11:0] counter;
    logic sqrclk;
logic [11:0] clk_div;

// octave 0 clk_div values
logic [11:0] notes [12] = '{
    1468, // C
    1386, // C-sharp and D-flat
    1308, // D
    1234, // D-sharp and E-flat
    1165, // E
    1099, // F
    1038, // F-sharp and G-flat
    980, // G
    924, // G-sharp and A-flat
    873, // A
    824, // A-sharp and B-flat
    777  // B
};

always_ff @(posedge clk or posedge reset) begin
    clk_div = notes[note] >> octave;

    if (reset)
         begin
          sqrclk <= 0;
          counter <= 11'd0;
         end
    else if (left_chan_ready && right_chan_ready)
         begin
          if (!sample_valid) begin
                if (counter == clk_div) begin
```

```
                        sqrclk <= ~sqrclk;
                        counter <= 11'd0;
                    end else begin
                        counter <= counter + 11'd1;
                    end
                end
                sample_valid <= 1;
            end
        else
            begin
            sample_valid <= 0;
            end
    end

    always_comb begin
        if (sqrclk == 1'b1)
            sample_data = volume;
        else
            sample_data = -volume;
    end
endmodule
```

Wavetables

tb_mixer.sv

```
module mixer_tb (output logic [15:0] wave);

logic clk;
logic reset;

logic [15:0] wave1;
logic [15:0] wave2;

logic [6:0] mix_ratio;

logic [8:0] counter;

logic [15:0] wave_out;

//input to wave mixer
```

```
always_ff @(posedge clk) begin
    if(reset) begin
       counter <= 0;
       mix_ratio <= 7'd63;
    end else begin
       counter <= counter + 1;
       wave1 <= counter;
       wave2 <= counter;
    end
end


mixer dut(.reset, .wave1, .wave2, .mix_ratio, .wave_out(wave));

initial begin
      reset = 1;
      clk = 0;
      #10 reset = 0;
end

always begin
      #1 clk = !clk;
end

endmodule
```

Tb_wavetables.sv

```
module tb_wavetables();

logic clk;
logic rst;

logic write;      // set to true if linux sent us something
logic is_data;   // 0 is info, 1 is wave data
logic [15:0] wave_data;

logic [15:0] sample_address;
logic sample_ready;
logic sample_valid;

logic [6:0] mix_ratio;
```

```
logic [15:0] data_out;

//logic [15:0] sample_data0;
//logic [15:0] sample_data1;

wavetables w(.*);

//assign data_out = sample_data0;

initial begin
      rst = 1;
      clk = 0;
      #10 rst = 0;
end

always begin
      #1 clk = ~clk;
end


logic [15:0]sample_counter;
logic [3:0] wave_counter;
logic [15:0] read_counter;

logic [15:0] numsent;


always_ff @(posedge clk) begin
    if(!rst) begin
       if(sample_counter < 16'd48000) begin
             sample_counter = sample_counter + 1;
             if (sample_counter == 1) begin
                   write <= 1;
                   is_data <= 0;
                   if (wave_counter[0])
                         wave_data <= 1;
                   else
                         wave_data <= 0;
             end else begin
                   write <= 1;
                   is_data <= 1;
                   wave_data <= numsent;
```

```
                  numsent <= numsent + 1;
            end
      end else begin
            numsent <= 60;
            wave_counter = wave_counter + 1;
            sample_counter <= 0;
      end

      if(wave_counter > 1) begin
            if(!sample_valid) begin
                  sample_ready <= 1;
                  sample_address <= read_counter;
                  if(read_counter < 47999)
                        read_counter <= read_counter + 1;
                  else
                        read_counter <= 0;

            end else begin
                  sample_ready <= 0;
            end
      end
   end else begin
      mix_ratio <= 20;
      numsent <= 0;
      sample_address <= 47932;
      read_counter <= 0;
      wave_counter <= 0;
      sample_counter <= 0;
      write <= 0;
   end
end


endmodule
```

Wavetabbles.sv

```
module wavetables(
    input  logic         clk,
    input  logic         rst,
```

```
    input  logic         write,     // set to true if linux sent us
something
    input  logic         is_data,   // 0 is info, 1 is wave data
    input  logic [15:0] wave_data, // wave data

    input  logic [15:0] sample_address,

    input  logic         sample_ready,
    output logic         sample_valid,

    input logic [6:0] mix_ratio,
    output logic signed [15:0] data_out
);

logic [15:0] sample_data0;
logic [15:0] sample_data1;

logic  [15:0] a_wave, a_0, a_1, a_2;
//logic  [15:0] din_wave;
logic       we_wave, we_0, we_1, we_2;
logic  [15:0] dout_0, dout_1, dout_2;

//logic switch_buffers;
logic wave_num;
logic start;
logic done;

//assign switch_buffers = done;

//"S021" is State(write = 0, read0 = 2, read1 = 1) - writing to buffer 0
from software, sampledata0 = table2, sampledata1 = table1
typedef enum logic[5:0] {S012, S021, S102, S120, S201, S210} wave_state_t;
//wave_state_t wave_state, wave_next_state;
wave_state_t wave_state;
wave_state_t wave_next_state;

mixer mx (.reset(rst), .wave1(sample_data0), .wave2(sample_data1),
.mix_ratio(mix_ratio), .wave_out(data_out));

memory table0(.clk(clk),
              .a(a_0),
              .din(wave_data),
```

```
                .we(we_0),
                .dout(dout_0));

memory table1(.clk(clk),
                .a(a_1),
                .din(wave_data),
                .we(we_1),
                .dout(dout_1));

memory table2(.clk(clk),
                .a(a_2),
                .din(wave_data),
                .we(we_2),
                .dout(dout_2));

always_ff @(posedge clk or posedge rst) begin
    we_wave                     <= 0; //default
    done                <= 0;
    sample_valid <= 0;
    if (rst) begin
       start            <= 0;
       wave_state        <= S012;
       wave_num          <= 0;
       a_wave            <= 0;
    end else begin
       if (sample_ready == 1)
             sample_valid <= 1;
       wave_state <= wave_next_state;
       if (write) begin
             if ( is_data == 0) begin
                   wave_num          <= wave_data[0]; //info bit - which
sample to replace?
                   start            <= 1;
                   a_wave            <= 0;
             end else begin
                   if ( a_wave > 47998 ) begin
                         we_wave <= 0;
                   end else begin
                         start <= 0;
                         if (a_wave == 47998 ) begin
                               a_wave  <= a_wave + 1;
                               done <= 1;
```

```
                              we_wave <= 1;
                        end else begin
                              we_wave <= 1;
                              a_wave  <= a_wave + 1;
                        end
                  end
            end
      end
   end
end


// State Machine to determine
always_comb begin
    wave_next_state = wave_state; //default;
    case (wave_state)
       S012: begin
             if (done)
                   wave_next_state = (wave_num ? S210 : S102);

             a_0  = a_wave;
             we_0   = we_wave;

             a_1  = sample_address;
             a_2  = sample_address;
             we_1   = 0;
             we_2   = 0;

             sample_data0 = dout_1;
             sample_data1 = dout_2;
       end
       S021: begin
             if (done)
                   wave_next_state = (wave_num ? S120 : S201);

             a_0  = a_wave;
             we_0   = we_wave;

             a_1  = sample_address;
             a_2  = sample_address;
             we_1   = 0;
             we_2   = 0;
```

```
            sample_data0 = dout_2;
            sample_data1 = dout_1;
      end
      S102: begin
            if (done)
                  wave_next_state = (wave_num ? S201 : S012);

            a_1  = a_wave;
            we_1   = we_wave;

            a_0  = sample_address;
            a_2  = sample_address;
            we_0   = 0;
            we_2   = 0;

            sample_data0 = dout_0;
            sample_data1 = dout_2;
      end
      S120: begin
            if (done)
                  wave_next_state = (wave_num ? S021 : S210);

            a_1  = a_wave;
            we_1   = we_wave;

            a_0  = sample_address;
            a_2  = sample_address;
            we_0   = 0;
            we_2   = 0;

            sample_data0 = dout_2;
            sample_data1 = dout_0;
      end
      S201: begin
            if (done)
                  wave_next_state = (wave_num ? S102 : S021);

            a_2  = a_wave;
            we_2   = we_wave;

            a_0  = sample_address;
```

```
            a_1  = sample_address;
            we_0   = 0;
            we_1   = 0;

            sample_data0 = dout_0;
            sample_data1 = dout_1;
      end
      S210: begin
            if (done)
                  wave_next_state = (wave_num ? S012 : S120);

            a_2  = a_wave;
            we_2   = we_wave;

            a_0  = sample_address;
            a_1  = sample_address;
            we_0   = 0;
            we_1   = 0;

            sample_data0 = dout_1;
            sample_data1 = dout_0;
      end
      //default: begin
      //    wave_next_state = 5'b11111;
      //end
      endcase

end
endmodule



// 48000 X 16 synchronous RAM with old data read-during-write behavior
module memory( input logic          clk,
                  input logic  [15:0] a,
                  input logic  [15:0] din,
                  input logic         we,
                  output logic [15:0] dout);

logic [15:0]  mem [47999:0];
always_ff @(posedge clk) begin
    if (we) mem[a] <= din;
```

```
    dout <= mem[a];
end

endmodule


module mixer (
       input reset,
       input logic signed [15:0] wave1,
       input logic signed [15:0] wave2,
       input logic [6:0] mix_ratio,
       output logic signed [15:0] wave_out
       );

always_comb begin
    logic signed [31:0] out1, out2;
    logic signed [15:0] rat;
    assign rat = {9'd0,mix_ratio};
    if (!reset) begin
       out1 = wave1 * rat;
       out2 = wave2 * (127-rat);
       wave_out = ((out1 >>> 7) + (out2 >>> 7));

    end
    else
       wave_out = 0;
end

endmodule
```

Software

Future_work

Convert_wav.py

```
import librosa
import soundfile as sf
import numpy as np
```

```
# y is a numpy array of the wav file, sr = sample rate
y, sr = librosa.load('a2002011001-e02.wav', sr=48000)


# if we want to shift pitch
# y_shifted = librosa.effects.pitch_shift(y, sr, n_steps=4) # shifted by 4
half steps

# checkout for sampling rate
#
https://librosa.github.io/librosa/generated/librosa.core.samples_to_time.ht
ml#librosa.core.samples_to_time
#
https://librosa.github.io/librosa/generated/librosa.core.time_to_samples.ht
ml


y_scaled = y / np.max(np.abs(y)) * 0.4

sf.write("out_int16.wav", y_scaled, sr, 'PCM_16')
```

Hps_codec_memory_test_driver

Makefile

```
ifneq (${KERNELRELEASE},)

# KERNELRELEASE defined: we are being compiled as part of the Kernel
      obj-m := midi_top.o

else

# We are being compiled as a module: use the Kernel build system

    KERNEL_SOURCE := /usr/src/linux-headers-$(shell uname -r)
      PWD := $(shell pwd)

default: module ins test

test: note.h usbmidi.c usbmidi.h test.c
    cc -Wall note.h usbmidi.c usbmidi.h test.c -o test -lusb-1.0 -pthread

ins:
```

```
    $(shell rmmod midi_top > /dev/null 2>&1; insmod midi_top.ko > /dev/null
2>&1;)

module:
    ${MAKE} -C ${KERNEL_SOURCE} SUBDIRS=${PWD} modules

clean:
    ${MAKE} -C ${KERNEL_SOURCE} SUBDIRS=${PWD} clean
    ${RM} test usbmidi.o

TARFILES = Makefile README midi_top.h midi_top.c hello.c
TARFILE = midi_driver_sw.tar.gz
.PHONY : tar
tar : $(TARFILE)

$(TARFILE) : $(TARFILES)
    tar zcfC $(TARFILE) .. $(TARFILES:%=midi_driver_sw/%)

endif
```

Midi_top.h

```
#ifndef _MIDI_TOP_H
#define _MIDI_TOP_H

#include <linux/ioctl.h>

typedef struct {
   unsigned short note_info; //16 bit thing
} midi_top_note_t;


typedef struct {
  midi_top_note_t note;
} midi_top_arg_t;

#define MIDI_TOP_MAGIC 'q'

/* ioctls and their arguments */
#define MIDI_TOP_WRITE_NOTE _IOW(MIDI_TOP_MAGIC, 1, midi_top_arg_t *)
#define MIDI_TOP_READ_NOTE  _IOR(MIDI_TOP_MAGIC, 2, midi_top_arg_t *)
#define MIDI_TOP_START_WAVE _IOR(MIDI_TOP_MAGIC, 3, midi_top_arg_t *)
```

#define MIDI_TOP_SEND_WAVE _IOR(MIDI_TOP_MAGIC, 4, midi_top_arg_t *)

#endif

Midi_top.c

```
/* * Device driver for the VGA video generator
 *
 * A Platform device implemented using the misc subsystem
 *
 * Stephen A. Edwards
 * Columbia University
 *
 * References:
 * Linux source: Documentation/driver-model/platform.txt
 *               drivers/misc/arm-charlcd.c
 * http://www.linuxforu.com/tag/linux-device-drivers/
 * http://free-electrons.com/docs/
 *
 * "make" to build
 * insmod vga_ball.ko
 *
 * Check code style with
 * checkpatch.pl --file --no-tree midi_top.c
 */

#include "midi_top.h"
#include <linux/errno.h>
#include <linux/fs.h>
#include <linux/init.h>
#include <linux/io.h>
#include <linux/kernel.h>
#include <linux/miscdevice.h>
#include <linux/module.h>
#include <linux/of.h>
#include <linux/of_address.h>
#include <linux/platform_device.h>
#include <linux/slab.h>
#include <linux/uaccess.h>
#include <linux/version.h>

#define DRIVER_NAME "midi_top"
```

```
/* Device registers */
#define REG_NOTE(x) (x)
#define REG_NOTE_HIGH(x) ((x) + 2)
#define WAVE_START(x) ((x) + 4)
#define WAVE_DATA(x) ((x) + 6)

//#define BG_GREEN(x) ((x)+1)
//#define BG_BLUE(x) ((x)+2)

/*
 * Information about our device
 */
struct midi_top_dev {
  struct resource res;  /* Resource: our registers */
  void __iomem *virtbase; /* Where registers can be accessed in memory */
  midi_top_note_t note;
} dev;

static void
write_start_wave(midi_top_note_t *note) { // unsigned short table_number) {
  iowrite16(note->note_info, WAVE_START(dev.virtbase));
}

static void
write_send_wave(midi_top_note_t *note) { // unsigned short wave_data) {
  iowrite16(note->note_info, WAVE_DATA(dev.virtbase));
}

/*
 * Write segments of a single digit
 * Assumes digit is in range and the device information has been set up
 */
static void write_note(midi_top_note_t *note) {
  iowrite16(note->note_info, REG_NOTE(dev.virtbase));
  // iowrite8(background->green, BG_GREEN(dev.virtbase) );
  // iowrite8(background->blue, BG_BLUE(dev.virtbase) );
  dev.note = *note;
}

/*
 * Handle ioctl() calls from userspace:
 * Read or write the segments on single digits.
```

```
 * Note extensive error checking of arguments
 */
static long midi_top_ioctl(struct file *f, unsigned int cmd,
                           unsigned long arg) {
  midi_top_arg_t vla;

  switch (cmd) {

  case MIDI_TOP_START_WAVE:
      if (copy_from_user(&vla, (midi_top_note_t *)arg,
sizeof(midi_top_note_t)))
      return -EACCES;
      write_start_wave(&vla.note);
      // write_note(&vla.note);
      // write_background(&vla.background);
      break;

  case MIDI_TOP_SEND_WAVE:
      if (copy_from_user(&vla, (midi_top_note_t *)arg,
sizeof(midi_top_note_t)))
      return -EACCES;
      write_send_wave(&vla.note);
      // write_note(&vla.note);
      // write_background(&vla.background);
      break;

  case MIDI_TOP_WRITE_NOTE:
      if (copy_from_user(&vla, (midi_top_note_t *)arg,
sizeof(midi_top_note_t)))
      return -EACCES;
      write_note(&vla.note);
      // write_background(&vla.background);
      break;

  case MIDI_TOP_READ_NOTE:
      vla.note = dev.note;
      if (copy_to_user((midi_top_note_t *)arg, &vla,
sizeof(midi_top_note_t)))
      return -EACCES;
      break;

  default:
```

```
        return -EINVAL;
    }

    return 0;
}

/* The operations our device knows how to do */
static const struct file_operations midi_top_fops = {
        .owner = THIS_MODULE,
        .unlocked_ioctl = midi_top_ioctl,
};

/* Information about our device for the "misc" framework -- like a char dev
*/
static struct miscdevice midi_top_misc_device = {
        .minor = MISC_DYNAMIC_MINOR,
        .name = DRIVER_NAME,
        .fops = &midi_top_fops,
};

/*
 * Initialization code: get resources (registers) and display
 * a welcome message
 */
static int __init midi_top_probe(struct platform_device *pdev) {
    // midi_top_color_t beige = { 0xf9, 0xe4, 0xb7 };
    int ret;

    /* Register ourselves as a misc device: creates /dev/midi_top */
    ret = misc_register(&midi_top_misc_device);

    /* Get the address of our registers from the device tree */
    ret = of_address_to_resource(pdev->dev.of_node, 0, &dev.res);
    if (ret) {
        ret = -ENOENT;
        goto out_deregister;
    }

    /* Make sure we can use these registers */
    if (request_mem_region(dev.res.start, resource_size(&dev.res),
DRIVER_NAME) ==
        NULL) {
```

```
      ret = -EBUSY;
      goto out_deregister;
  }

  /* Arrange access to our registers */
  dev.virtbase = of_iomap(pdev->dev.of_node, 0);
  if (dev.virtbase == NULL) {
      ret = -ENOMEM;
      goto out_release_mem_region;
  }

  /* Set an initial color */
  // write_background(&beige);

  return 0;

out_release_mem_region:
  release_mem_region(dev.res.start, resource_size(&dev.res));
out_deregister:
  misc_deregister(&midi_top_misc_device);
  return ret;
}

/* Clean-up code: release resources */
static int midi_top_remove(struct platform_device *pdev) {
  iounmap(dev.virtbase);
  release_mem_region(dev.res.start, resource_size(&dev.res));
  misc_deregister(&midi_top_misc_device);
  return 0;
}

/* Which "compatible" string(s) to search for in the Device Tree */
#ifdef CONFIG_OF
static const struct of_device_id midi_top_of_match[] = {
      {.compatible = "csee4840,midi_top-1.0"},
      {},
};
MODULE_DEVICE_TABLE(of, midi_top_of_match);
#endif

/* Information for registering ourselves as a "platform" driver */
static struct platform_driver midi_top_driver = {
```

```
	.driver =
	{
		.name = DRIVER_NAME,
		.owner = THIS_MODULE,
		.of_match_table = of_match_ptr(midi_top_of_match),
	},
	.remove = __exit_p(midi_top_remove),
};

/* Called when the module is loaded: set things up */
static int __init midi_top_init(void) {
  pr_info(DRIVER_NAME ": init\n");
  return platform_driver_probe(&midi_top_driver, midi_top_probe);
}

/* Calball when the module is unloaded: release resources */
static void __exit midi_top_exit(void) {
  platform_driver_unregister(&midi_top_driver);
  pr_info(DRIVER_NAME ": exit\n");
}

module_init(midi_top_init);
module_exit(midi_top_exit);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Stephen A. Edwards, Columbia University");
MODULE_DESCRIPTION("MIDI top driver");
```

Test.c

```
#include <arpa/inet.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <unistd.h>

#include "note.h"
#include "usbmidi.h"
/* Update SERVER_HOST to be the IP address of
 * the chat server you are connecting to
```

```
 */
/* micro36.ee.columbia.edu */

#define BUFFER_SIZE 70000
#define MAX_MSG_LEN BUFFER_SIZE - 17

/*
 * References:
 *
 * http://beej.us/guide/bgnet/output/html/singlepage/bgnet.html
 * http://www.thegeekstuff.com/2011/12/c-socket-programming/
 *
 */

struct libusb_device_handle *keyboard;
uint8_t endpoint_address;

// char buf[BUFFER_SIZE];
// char keystate[BUFFER_SIZE];

int main() {

  int midi_top_fd;
  static const char filename[] = "/dev/midi_top";
  if ((midi_top_fd = open(filename, O_RDWR)) == -1) {
      fprintf(stderr, "could not open %s\n", filename);
      return -1;
  }

  /* Open the keyboard */
  if ((keyboard = openkeyboard(&endpoint_address)) == NULL) {

      fprintf(stderr, "Did not find a keyboard\n");
      // fbputs("Did not find a keyboard", 22, 19);
      exit(1);
  }

  // have to make our own packet...
  struct usb_midi_packet packet;
  int transferred;
  // char keystate[12];
```

```
  /* Look for and handle keypresses */
  // uint8_t data[64];

  int wave_1 = 0;
  int wave_2 = 0;

  for (;;) {
      int errcode = 0;
      memset(&packet, 0, sizeof(packet));
      errcode = libusb_bulk_transfer(keyboard, endpoint_address,
                                     (unsigned char *)&packet,
sizeof(packet),
                                     &transferred, 1000);
      if (errcode != 0 ||
      (packet.status_byte != 9 && packet.status_byte != 11 &&
      packet.status_byte != 25 && packet.status_byte != 14)) {
      continue;
      }
      // if ()

      // uint8_t data = 0;

      // printf("status_byte, not_sure: %d %d\n", packet.status_byte,
      // packet.not_sure );

      // printf("(NOTE, ATTACK): %d %d\n", packet.note, packet.attack);
      // printf("Modulation:(Status, Value):%d\n", packet.note);
      // printf("Modulation: %d\n", packet.not_sure);

      if (packet.status_byte == 25) {
      if (packet.note == 100 && packet.attack == 127) {
      printf("starting the send....\n");
      start_wave(midi_top_fd, 0);
      for (unsigned short i = 0; i < 48000; i++) {
            send_wave(midi_top_fd, i);
            // usleep(100);
      }
      printf("everything sent!..\n");
      // printf("Wave_1 Number %d and Wave_2 Number %d selected.\n",
wave_1,
      //    wave_2);
      }
```

```
      if (packet.note == 99 && packet.attack == 127) {
      wave_1++;
      printf("Wave_1 Number %d\n", wave_1);
      }
      if (packet.note == 98 && packet.attack == 127) {
      wave_1--;
      start_wave(midi_top_fd, 0);
      printf("Wave_1 Number %d\n", wave_1);
      }
      if (packet.note == 96 && packet.attack == 127) {
      wave_2++;
      printf("Wave_2 Number %d\n", wave_2);
      }
      if (packet.note == 97 && packet.attack == 127) {
      wave_2--;
      printf("Wave_2 Number %d\n", wave_2);
      }
      }
      // printf("%u %u %u %u %u %u %u %u %u %u %u....\n",
packet.status_byte,
      //    packet.not_sure, packet.note, packet.attack,
packet.other_data[0],
      //    packet.other_data[1], packet.other_data[2],
packet.other_data[3],
      //    packet.other_data[4], packet.other_data[5],
packet.other_data[6],
      //    packet.other_data[7]);

      if (packet.status_byte == 9) {
      printf("(NOTE, ATTACK): %d %d\n", packet.note, packet.attack);
      set_note(midi_top_fd, (unsigned short)packet.note,
            (unsigned short)packet.attack);
      }

      if (packet.status_byte == 11 && packet.note == 7) {
      printf("VOLUME(Value): %d\n", packet.attack);
      }
      if (packet.status_byte == 11 && packet.note == 1) {
      printf("MODULATION(Value): %d\n", packet.attack);
      }

      if (packet.status_byte == 14) {
```

```
      printf("PITCH WHEEL(VALUE): %d\n", packet.attack);
      }

      // printf("(NOTE, ATTACK, OTHER1, OTHER2): %d %d %d %d\n",
packet.note,
      // packet.attack,     packet.other_data[0], packet.other_data[1]);
      // printf("ATTACK: %d\n", packet.attack);
      // printf("transferred %d, sizeof(packet) %d: \n", transferred,
      // sizeof(packet)); printf("strlen = %d\n", strlen((char *)
&packet));
  }

  return 0;
}
```

Test_1.c

```
// version of test that iterates through wave table
#include <arpa/inet.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <unistd.h>

#include "note.h"
#include "usbmidi.h"
/* Update SERVER_HOST to be the IP address of
 * the chat server you are connecting to
 */
/* micro36.ee.columbia.edu */

#define BUFFER_SIZE 70000
#define MAX_MSG_LEN BUFFER_SIZE - 17

/*
 * References:
 *
 * http://beej.us/guide/bgnet/output/html/singlepage/bgnet.html
 * http://www.thegeekstuff.com/2011/12/c-socket-programming/
 *
 */
```

```
struct libusb_device_handle *keyboard;
uint8_t endpoint_address;

// char buf[BUFFER_SIZE];
// char keystate[BUFFER_SIZE];
void process_wave(int wave_name) {
  printf("starting the send....\n");
  start_wave(wave_name, 0);
  for (unsigned short i = 0; i < 48000; i++) {
      if (i < 24000)
      send_wave(wave_name, 0x1388);
      else
      send_wave(wave_name, 0xEC78);
      // usleep(100);
  }
  printf("everything sent!..\n");

  printf("starting the send....\n");
  start_wave(wave_name, 1);
  for (unsigned short i = 0; i < 48000; i++) {
      if (i < 24000)
      send_wave(wave_name, 0x332C);
      else
      send_wave(wave_name, 0xCCD4);
      // usleep(100);
  }
}

int file_output(char *fn) {
  int output;
  if ((output = open(fn, O_RDWR)) == -1) {
      fprintf(stderr, "could not open %s\n", fn);
      return -1;
  }
  return output;
}

int main() {
  int sin_table = file_output("../matlab/audio_files/sin_table");
  int saw_table = file_output("../matlab/audio_files/saw_table");
  int pulse_table = file_output("../matlab/audio_files/pulse_table");
```

```
  int tri_table = file_output("../matlab/audio_files/tri_table");

  // probably not a good idea to hard code these
  // int lowest_table_value = 0;
  // int total_wave_tables = 4;

  int midi_top_fd;
  static const char filename[] = "/dev/midi_top";
  if ((midi_top_fd = open(filename, O_RDWR)) == -1) {
      fprintf(stderr, "could not open %s\n", filename);
      return -1;
  }
  note_init();
  /* Open the keyboard */
  if ((keyboard = openkeyboard(&endpoint_address)) == NULL) {

      fprintf(stderr, "Did not find a keyboard\n");
      // fbputs("Did not find a keyboard", 22, 19);
      exit(1);
  }

  // have to make our own packet...
  struct usb_midi_packet packet;
  int transferred;
  // char keystate[12];

  /* Look for and handle keypresses */
  // uint8_t data[64];

  int wave_1 = -1;
  int wave_2 = -1;
  int wave1_check = 0;
  int wave2_check = 0;

  for (;;) {
      int errcode = 0;
      memset(&packet, 0, sizeof(packet));
      errcode = libusb_bulk_transfer(keyboard, endpoint_address,
                                     (unsigned char *)&packet,
sizeof(packet),
                                     &transferred, 1000);
      if (errcode != 0 ||
```

```
      (packet.status_byte != 9 && packet.status_byte != 11 &&
      packet.status_byte != 25 && packet.status_byte != 14)) {
      continue;
      }
      // if ()

      // uint8_t data = 0;

      // printf("status_byte, not_sure: %d %d\n", packet.status_byte,
      // packet.not_sure );

      // printf("(NOTE, ATTACK): %d %d\n", packet.note, packet.attack);
      // printf("Modulation:(Status, Value):%d\n", packet.note);
      // printf("Modulation: %d\n", packet.not_sure);

      if (packet.status_byte == 25) {
      if (packet.note == 100 && packet.attack == 127) {

      // printf("Wave_1 Number %d and Wave_2 Number %d selected.\n",
wave_1,
      //    wave_2);
      // process_wave(midi_top_fd);

      // Chehcks to see if Wave_1 has Changed
      if (wave1_check != wave_1) {
            if (wave_1 == 0) {
            process_wave(sin_table);
            }
            if (wave_1 == 1) {
            process_wave(saw_table);
            }
            if (wave_1 == 2) {
            process_wave(pulse_table);
            }
            if (wave_1 == 3) {
            process_wave(tri_table);
            }
      }

      // Chehcks to see if wave_2 has changed
      if (wave2_check != wave_2) {
            if (wave_2 == 0) {
```

```
            process_wave(sin_table);
            }
            if (wave_2 == 1) {
            process_wave(saw_table);
            }
            if (wave_2 == 2) {
            process_wave(pulse_table);
            }
            if (wave_2 == 3) {
            process_wave(tri_table);
            }
      }

      if (wave1_check == wave_1 && wave2_check == wave_2) {
            printf("No change in waves detected.");
      }
      }
      if (packet.note == 99 && packet.attack == 127) {
      if (wave_1 < 3) {
            wave_1++;
      }

      printf("Wave_1 Number %d\n", wave_1);
      }
      if (packet.note == 98 && packet.attack == 127) {
      if (wave_1 > 0) {
            wave_1--;
      }
      start_wave(midi_top_fd, 0);
      printf("Wave_1 Number %d\n", wave_1);
      }
      if (packet.note == 96 && packet.attack == 127) {
      if (wave_2 < 3) {
            wave_2++;
      }
      printf("Wave_2 Number %d\n", wave_2);
      }
      if (packet.note == 97 && packet.attack == 127) {
      if (wave_2 > 0) {
            wave_2--;
      }
      printf("Wave_2 Number %d\n", wave_2);
```

```
      }
      if (wave_1 != -1 || wave_2 != -1) {
      wave1_check = wave_1;
      wave2_check = wave_2;
      }
      }

      // printf("%u %u %u %u %u %u %u %u %u %u %u....\n",
packet.status_byte,
      //    packet.not_sure, packet.note, packet.attack,
packet.other_data[0],
      //    packet.other_data[1], packet.other_data[2],
packet.other_data[3],
      //    packet.other_data[4], packet.other_data[5],
packet.other_data[6],
      //    packet.other_data[7]);

      if (packet.status_byte == 9) {
      printf("(NOTE, ATTACK): %d %d\n", packet.note, packet.attack);
      set_note(midi_top_fd, (unsigned short)packet.note,
            (unsigned short)packet.attack);
      }

      if (packet.status_byte == 11 && packet.note == 7) {
      printf("VOLUME(Value): %d\n", packet.attack);
      }
      if (packet.status_byte == 11 && packet.note == 1) {
      printf("MODULATION(Value): %d\n", packet.attack);
      }

      if (packet.status_byte == 14) {
      printf("PITCH WHEEL(VALUE): %d\n", packet.attack);
      }

      // printf("(NOTE, ATTACK, OTHER1, OTHER2): %d %d %d %d\n",
packet.note,
      // packet.attack,     packet.other_data[0], packet.other_data[1]);
      // printf("ATTACK: %d\n", packet.attack);
      // printf("transferred %d, sizeof(packet) %d: \n", transferred,
      // sizeof(packet)); printf("strlen = %d\n", strlen((char *)
&packet));
  }
```

```
  return 0;
}
```

Usbmidi.c

```
#include "usbmidi.h"

#include <stdio.h>
#include <stdlib.h>

/* References on libusb 1.0 and the USB HID/keyboard protocol
 *
 * http://libusb.org
 *
http://www.dreamincode.net/forums/topic/148707-introduction-to-using-libusb
-10/
 * http://www.usb.org/developers/devclass_docs/HID1_11.pdf
 * http://www.usb.org/developers/devclass_docs/Hut1_11.pdf
 */

/*
 * Find and return a USB keyboard device or NULL if not found
 * The argument con
 *
 */
struct libusb_device_handle *openkeyboard(uint8_t *endpoint_address) {
  libusb_device **devs;
  struct libusb_device_handle *keyboard = NULL;
  struct libusb_device_descriptor desc;
  ssize_t num_devs, d;
  uint8_t i, k;

  /* Start the library */
  if (libusb_init(NULL) < 0) {
      fprintf(stderr, "Error: libusb_init failed\n");
      exit(1);
  }

  /* Enumerate all the attached USB devices */
  if ((num_devs = libusb_get_device_list(NULL, &devs)) < 0) {
```

```
    fprintf(stderr, "Error: libusb_get_device_list failed\n");
    exit(1);
}

/* Look at each device, remembering the first HID device that speaks
    the keyboard protocol */

for (d = 0; d < num_devs; d++) {
    libusb_device *dev = devs[d];
    if (libusb_get_device_descriptor(dev, &desc) < 0) {
    fprintf(stderr, "Error: libusb_get_device_descriptor failed\n");
    exit(1);
    }

    if (desc.bDeviceClass == LIBUSB_CLASS_PER_INTERFACE) {
    struct libusb_config_descriptor *config;
    libusb_get_config_descriptor(dev, 0, &config);
    for (i = 0; i < config->bNumInterfaces; i++)
    for (k = 0; k < config->interface[i].num_altsetting; k++) {
          const struct libusb_interface_descriptor *inter =
          config->interface[i].altsetting + k;

          printf("INTERFACE: %d, %d\n", inter->bInterfaceClass,
                inter->bInterfaceProtocol);

          if (inter->bInterfaceClass == LIBUSB_CLASS_AUDIO &&
          inter->bInterfaceProtocol == USB_MIDI_KEYBOARD_PROTOCOL &&
          (inter->bNumEndpoints != 0)) {
          printf("got in here...\n");
          int r;
          if ((r = libusb_open(dev, &keyboard)) != 0) {
          fprintf(stderr, "Error: libusb_open failed: %d\n", r);
          exit(1);
          }
          printf("past libusb_open...\n");
          if (libusb_kernel_driver_active(keyboard, i)) {
          libusb_detach_kernel_driver(keyboard, i);
          }
          printf("past libusb_kernel_driver_active...\n");
          libusb_set_auto_detach_kernel_driver(keyboard, i);
          printf("past libusb_set_auto_detach_kernel_driver...\n");
          if ((r = libusb_claim_interface(keyboard, i)) != 0) {
```

```
            fprintf(stderr, "Error: libusb_claim_interface failed: %d\n",
r);
            fprintf(stderr, "Error %s\n", libusb_strerror(r));
            exit(1);
            }
            printf("past libusb_claim_interface...\n");
            printf("Number of endpoints: %d\n", inter->bNumEndpoints);

            *endpoint_address = inter->endpoint[0].bEndpointAddress;
            printf("WMAXPACKETSIZE = %d\n",
inter->endpoint[0].wMaxPacketSize);
            goto found;
            }
      }
      }
  }

found:
  libusb_free_device_list(devs, 1);

  return keyboard;
}
```

```
#ifndef _USBMIDI_H
#define _USBMIDI_H

#include <libusb-1.0/libusb.h>

#define USB_HID_KEYBOARD_PROTOCOL 1
#define USB_MIDI_KEYBOARD_PROTOCOL 0

/* Modifier bits */
#define USB_LCTRL (1 << 0)
#define USB_LSHIFT (1 << 1)
#define USB_LALT (1 << 2)
#define USB_LGUI (1 << 3)
#define USB_RCTRL (1 << 4)
#define USB_RSHIFT (1 << 5)
#define USB_RALT (1 << 6)
#define USB_RGUI (1 << 7)
```

```
struct usb_midi_packet {
  uint8_t status_byte;
  uint8_t not_sure;
  uint8_t note;
  uint8_t attack;
  uint8_t other_data[60];
};

// struct usb_keyboard_packet {
//   uint8_t modifiers;
//   uint8_t reserved;
//   uint8_t keycode[6];
// };

/* Find and open a USB keyboard device.  Argument should point to
   space to store an endpoint address.  Returns NULL if no keyboard
   device was found. */
extern struct libusb_device_handle *openkeyboard(uint8_t *);

#endif
```

Matlab

Generate_pulse_table.m

```
% duty is scalar from 1 to 100, square is 50
function pulse_table = generate_pulse_table(codec_freq, mag_bits, duty)
mag = 2^(mag_bits-1)-1;
num_samples = int32( floor(codec_freq) );

disp("Size OUT OF ~500 (KiloBytes):")
disp((num_samples * mag_bits)/8192)

t_table = linspace(0, 2*pi, num_samples + 1);
t_table(end) = [];

%t_table = linspace(0, (2*pi)/4, num_samples);
pulse_table = int16( mag*square(t_table, duty) ) ;


%Write to file
```

```
fileID = fopen('pulse_table','wb');
fprintf(fileID,"%.4x ", pulse_table);
fclose(fileID);

end
```

Generate_saw_table.m

```
function saw_table = generate_saw_table(codec_freq, mag_bits)
                  %global sin_table
                  %codec_freq = 44100;
%mag_bits = 16;
rng(0,'twister');

mag = 13100;
num_samples = int32( floor(codec_freq) );

disp("Size OUT OF ~500 (KiloBytes):")
disp((num_samples * mag_bits)/8192)

t_table = linspace(0, 2*pi, num_samples + 1);
t_table(end) = [];

%t_table = linspace(0, (2*pi)/4, num_samples);
saw_table = int16( mag*sawtooth(t_table) );
%int16(linspace(-mag, mag, num_samples));
%t_table(end) = [];

% should change to int32 if mag_bits > 16
%saw_table =  int16(mag * sin(t_table));

%Write to file
fileID = fopen('saw_table','wb');
fprintf(fileID,"%.4x ", saw_table);
fclose(fileID);

end
```

Generate_sin_table.m

```
%not using lowest_freq anymore
```

```
%uses global sin_table

function sin_table = generate_sin_table(codec_freq, mag_bits)
%global sin_table
%codec_freq = 44100;
%mag_bits = 16;

mag = 2^(mag_bits-1)-1;

%num_samples = int32( floor(codec_freq/4) ) + 1 ;
num_samples = int32( floor(codec_freq) );

disp("Size OUT OF ~500 (KiloBytes):")
disp((num_samples * mag_bits)/8192)

%t_table = linspace(0, (2*pi)/4, num_samples);
t_table = linspace(0, 2*pi, num_samples + 1);
t_table(end) = [];

% should change to int32 if mag_bits > 16
sin_table =  int16(mag * sin(t_table)) ;

%Write to file
fileID = fopen('sin_table','wb');
fprintf(fileID,"%.4x ", sin_table);
fclose(fileID);

end

%^quarter_table;

%full_sin_synth = zeros(1,codec_freq);
%for i = 1:codec_freq
%     full_sin_synth(i) = sin_table_get(i, sin_table);
%end
%y = full_sin_table;

%end
```

Generate_triangle_table.m

```
%not using lowest_freq anymore
%uses global sin_table

function sin_table = generate_sin_table(codec_freq, mag_bits)
%global sin_table
%codec_freq = 44100;
%mag_bits = 16;

mag = 2^(mag_bits-1)-1;

%num_samples = int32( floor(codec_freq/4) ) + 1 ;
num_samples = int32( floor(codec_freq) );

disp("Size OUT OF ~500 (KiloBytes):")
disp((num_samples * mag_bits)/8192)

%t_table = linspace(0, (2*pi)/4, num_samples);
t_table = linspace(0, 2*pi, num_samples + 1);
t_table(end) = [];

% should change to int32 if mag_bits > 16
sin_table =  int16(mag * sin(t_table)) ;

%Write to file
fileID = fopen('sin_table','wb');
fprintf(fileID,"%.4x ", sin_table);
fclose(fileID);

end

%^quarter_table;

%full_sin_synth = zeros(1,codec_freq);
%for i = 1:codec_freq
%     full_sin_synth(i) = sin_table_get(i, sin_table);
%end
%y = full_sin_table;

%end
```

Main.m

```
%global sin_table

mag_bits    = 16;
codec_freq  = 44.1 * 1000;
seconds     = 2;
note = 440;

%Test here: https://www.szynalski.com/tone-generator/
sin_table = generate_sin_table(codec_freq, mag_bits);
saw_table = generate_saw_table(codec_freq, mag_bits);
triangle_table = generate_triangle_table(codec_freq, mag_bits);
square_table = generate_pulse_table(codec_freq, mag_bits, 50);

sin_out =         sample_wave(sin_table,        note, codec_freq, seconds);
saw_out =         sample_wave_saw(saw_table,    note, codec_freq, seconds);
triangle_out =    sample_wave(triangle_table, note, codec_freq, seconds);
pulse_out = sample_wave_saw(square_table,   note, codec_freq, seconds);



%PLOT
%t = 0:(codec_freq*seconds-1);
plotnum = round(codec_freq*seconds/200);
t=1:plotnum;
figure;
subplot(4,1,1);
%scatter(t,sin_out);
scatter(t,sin_out(1:plotnum));
subplot(4,1,2)
scatter(t,saw_out(1:plotnum));
subplot(4,1,3)
scatter(t,triangle_out(1:plotnum));
subplot(4,1,4)
scatter(t,pulse_out(1:plotnum));


%PLAY SOUND
soundsc(double(sin_out), codec_freq);
pause(seconds+.1);
soundsc(double(saw_out), codec_freq);
```

```
pause(seconds+.1);
soundsc(double(triangle_out), codec_freq);
pause(seconds+.1);
soundsc(double(pulse_out), codec_freq);
```

Sample_wave_saw.m

```
function y = sample_wave_saw(wave_table, note, codec_freq, seconds)

rng(0,'twister');

%num_samples = int32( floor(codec_freq/lowest_freq) );

sampled_wave = zeros(1 , codec_freq*seconds);

sample_every =  floor(note) ;

sampled_wave(1) = wave_table(1);
index = 1 + sample_every;

%r = randi([-64 64],1,(codec_freq*seconds));
for i = 2:(codec_freq*seconds)
      %index = min(sample,codec_freq));
      if (index+sample_every > codec_freq)
      index = codec_freq;
      end
      sampled_wave(i) = wave_table(index);
      index = mod(index - 1 + sample_every, codec_freq) + 1;
      %index = index+rando;
end

y = sampled_wave;


end
```

Sample_wave.m

```
function y = sample_wave(wave_table, note, codec_freq, seconds)
```

```
rng(0,'twister');

%num_samples = int32( floor(codec_freq/lowest_freq) );

sampled_wave = zeros(1 , codec_freq*seconds);

sample_every =  floor(note) ;
index = 1;

for i = 1:(codec_freq*seconds)

      sampled_wave(i) = wave_table(index);
      index = mod(index - 1 + sample_every, codec_freq) + 1;
      %index = index+rando;
end

y = sampled_wave;


end
```

Sawtoothwave.m

```
function y = sawtoothwave(note, codec_freq, mag_bits, seconds)

mag = 13100;
samples = int32(floor(codec_freq/note)) ;

saw_freq = zeros(1,codec_freq*seconds);
step = floor((2*mag+1)/(samples));

saw_freq(1) = 0;
for i = 2:(codec_freq*seconds)
      saw_freq(i) = mod( saw_freq(i-1) + step , 2*mag+1);
      %sample = mod( sample + sample_every, num_samples) + 1;
end

y = int16(saw_freq - mag);


%t_freq = downsample(t_table,sample_every);
%sin_freq = downsample(sin_table,sample_every);
```

```
%
%figure;
%subplot(2,1,1)
%scatter(t_table,sin_table);
%subplot(2,1,2)
%scatter(t_table,sin_freq(1:num_samples));

%soundsc(sin_freq, codec_freq);

end
```

Sinwave.m

```
function y = sinwave(note, codec_freq, mag_bits, seconds)

lowest_freq = 1;
mag = 13100;
num_samples = int32( floor(codec_freq/lowest_freq) );

%disp("Amount of space OUT OF ~500 (KiloBytes):")
%disp((num_samples*mag_bits)/8192)

t_table = linspace(0, 2*pi, num_samples+1);
t_table(end) = [];

% HAVE to change to int32 if mag_bits > 16
if mag_bits <= 16
      sin_table = int16( mag * sin(t_table) );
else
      sin_table = int32( mag * sin(t_table) );
end


sample = 1;
sample_every =  floor(note/lowest_freq) ;

sin_freq = zeros(1,codec_freq*seconds*lowest_freq);
for i = 1:(num_samples*seconds*lowest_freq)
      sin_freq(i) = sin_table(sample);
      sample = mod( sample + sample_every, num_samples) + 1;
```

```
end

y = int16(sin_freq);

%t_freq = downsample(t_table,sample_every);
%sin_freq = downsample(sin_table,sample_every);


%
%figure;
%subplot(2,1,1)
%scatter(t_table,sin_table);
%subplot(2,1,2)
%scatter(t_table,sin_freq(1:num_samples));

%soundsc(sin_freq, codec_freq);

end
```

Midi_top_driver

Midi_top.c

```
/* * Device driver for the VGA video generator
 *
 * A Platform device implemented using the misc subsystem
 *
 * Stephen A. Edwards
 * Columbia University
 *
 * References:
 * Linux source: Documentation/driver-model/platform.txt
 *               drivers/misc/arm-charlcd.c
 * http://www.linuxforu.com/tag/linux-device-drivers/
 * http://free-electrons.com/docs/
 *
 * "make" to build
 * insmod vga_ball.ko
 *
 * Check code style with
 * checkpatch.pl --file --no-tree midi_top.c
```

```
 */

#include "midi_top.h"
#include <linux/errno.h>
#include <linux/fs.h>
#include <linux/init.h>
#include <linux/io.h>
#include <linux/kernel.h>
#include <linux/miscdevice.h>
#include <linux/module.h>
#include <linux/of.h>
#include <linux/of_address.h>
#include <linux/platform_device.h>
#include <linux/slab.h>
#include <linux/uaccess.h>
#include <linux/version.h>

#define DRIVER_NAME "midi_top"

/* Device registers */
#define REG_NOTE(x) (x)
#define WAVE_START(x) ((x) + 20)
#define WAVE_DATA(x) ((x) + 22)
#define MODULATION(x) ((x) + 24)

//#define BG_GREEN(x) ((x)+1)
//#define BG_BLUE(x) ((x)+2)

/*
 * Information about our device
 */
struct midi_top_dev {
  struct resource res;  /* Resource: our registers */
  void __iomem *virtbase; /* Where registers can be accessed in memory */
  midi_top_note_t note;
} dev;

static void
write_start_wave(midi_top_note_t *note) { // unsigned short table_number) {
  iowrite16(note->note_info, WAVE_START(dev.virtbase));
}
```

```
static void
write_send_wave(midi_top_note_t *note) { // unsigned short wave_data) {
  iowrite16(note->note_info, WAVE_DATA(dev.virtbase));
}

/*
 * Write segments of a single digit
 * Assumes digit is in range and the device information has been set up
 */
static void write_note(midi_top_note_t *note) {
  iowrite16(note->note_info, dev.virtbase + (2 * note->index));
  dev.note = *note;
}

/*
 * Handle ioctl() calls from userspace:
 * Read or write the segments on single digits.
 * Note extensive error checking of arguments
 */
static long midi_top_ioctl(struct file *f, unsigned int cmd,
                           unsigned long arg) {
  midi_top_arg_t vla;

  switch (cmd) {

  case MIDI_TOP_START_WAVE:
      if (copy_from_user(&vla, (midi_top_note_t *)arg,
sizeof(midi_top_note_t)))
      return -EACCES;
      write_start_wave(&vla.note);
      // write_note(&vla.note);
      // write_background(&vla.background);
      break;

  case MIDI_TOP_SEND_WAVE:
      if (copy_from_user(&vla, (midi_top_note_t *)arg,
sizeof(midi_top_note_t)))
      return -EACCES;
      write_send_wave(&vla.note);
      // write_note(&vla.note);
      // write_background(&vla.background);
      break;
```

```
  case MIDI_TOP_WRITE_NOTE:
      if (copy_from_user(&vla, (midi_top_note_t *)arg,
sizeof(midi_top_note_t)))
      return -EACCES;
      write_note(&vla.note);
      // write_background(&vla.background);
      break;

  case MIDI_TOP_READ_NOTE:
      vla.note = dev.note;
      if (copy_to_user((midi_top_note_t *)arg, &vla,
sizeof(midi_top_note_t)))
      return -EACCES;
      break;

  default:
      return -EINVAL;
  }

  return 0;
}

/* The operations our device knows how to do */
static const struct file_operations midi_top_fops = {
      .owner = THIS_MODULE,
      .unlocked_ioctl = midi_top_ioctl,
};

/* Information about our device for the "misc" framework -- like a char dev
*/
static struct miscdevice midi_top_misc_device = {
      .minor = MISC_DYNAMIC_MINOR,
      .name = DRIVER_NAME,
      .fops = &midi_top_fops,
};

/*
 * Initialization code: get resources (registers) and display
 * a welcome message
 */
static int __init midi_top_probe(struct platform_device *pdev) {
```

```
  // midi_top_color_t beige = { 0xf9, 0xe4, 0xb7 };
  int ret;

  /* Register ourselves as a misc device: creates /dev/midi_top */
  ret = misc_register(&midi_top_misc_device);

  /* Get the address of our registers from the device tree */
  ret = of_address_to_resource(pdev->dev.of_node, 0, &dev.res);
  if (ret) {
      ret = -ENOENT;
      goto out_deregister;
  }

  /* Make sure we can use these registers */
  if (request_mem_region(dev.res.start, resource_size(&dev.res),
DRIVER_NAME) ==
      NULL) {
      ret = -EBUSY;
      goto out_deregister;
  }

  /* Arrange access to our registers */
  dev.virtbase = of_iomap(pdev->dev.of_node, 0);
  if (dev.virtbase == NULL) {
      ret = -ENOMEM;
      goto out_release_mem_region;
  }

  /* Set an initial color */
  // write_background(&beige);

  return 0;

out_release_mem_region:
  release_mem_region(dev.res.start, resource_size(&dev.res));
out_deregister:
  misc_deregister(&midi_top_misc_device);
  return ret;
}

/* Clean-up code: release resources */
static int midi_top_remove(struct platform_device *pdev) {
```

```
  iounmap(dev.virtbase);
  release_mem_region(dev.res.start, resource_size(&dev.res));
  misc_deregister(&midi_top_misc_device);
  return 0;
}

/* Which "compatible" string(s) to search for in the Device Tree */
#ifdef CONFIG_OF
static const struct of_device_id midi_top_of_match[] = {
      {.compatible = "csee4840,midi_top-1.0"},
      {},
};
MODULE_DEVICE_TABLE(of, midi_top_of_match);
#endif

/* Information for registering ourselves as a "platform" driver */
static struct platform_driver midi_top_driver = {
      .driver =
      {
            .name = DRIVER_NAME,
            .owner = THIS_MODULE,
            .of_match_table = of_match_ptr(midi_top_of_match),
      },
      .remove = __exit_p(midi_top_remove),
};

/* Called when the module is loaded: set things up */
static int __init midi_top_init(void) {
  pr_info(DRIVER_NAME ": init\n");
  return platform_driver_probe(&midi_top_driver, midi_top_probe);
}

/* Calball when the module is unloaded: release resources */
static void __exit midi_top_exit(void) {
  platform_driver_unregister(&midi_top_driver);
  pr_info(DRIVER_NAME ": exit\n");
}

module_init(midi_top_init);
module_exit(midi_top_exit);

MODULE_LICENSE("GPL");
```

```
MODULE_AUTHOR("Stephen A. Edwards, Columbia University");
MODULE_DESCRIPTION("MIDI top driver");
```

Midi_top.h

```
/* * Device driver for the VGA video generator
 *
 * A Platform device implemented using the misc subsystem
 *
 * Stephen A. Edwards
 * Columbia University
 *
 * References:
 * Linux source: Documentation/driver-model/platform.txt
 *               drivers/misc/arm-charlcd.c
 * http://www.linuxforu.com/tag/linux-device-drivers/
 * http://free-electrons.com/docs/
 *
 * "make" to build
 * insmod vga_ball.ko
 *
 * Check code style with
 * checkpatch.pl --file --no-tree midi_top.c
 */

#include "midi_top.h"
#include <linux/errno.h>
#include <linux/fs.h>
#include <linux/init.h>
#include <linux/io.h>
#include <linux/kernel.h>
#include <linux/miscdevice.h>
#include <linux/module.h>
#include <linux/of.h>
#include <linux/of_address.h>
#include <linux/platform_device.h>
#include <linux/slab.h>
#include <linux/uaccess.h>
#include <linux/version.h>

#define DRIVER_NAME "midi_top"
```

```
/* Device registers */
#define REG_NOTE(x) (x)
#define WAVE_START(x) ((x) + 20)
#define WAVE_DATA(x) ((x) + 22)
#define MODULATION(x) ((x) + 24)

//#define BG_GREEN(x) ((x)+1)
//#define BG_BLUE(x) ((x)+2)

/*
 * Information about our device
 */
struct midi_top_dev {
  struct resource res;  /* Resource: our registers */
  void __iomem *virtbase; /* Where registers can be accessed in memory */
  midi_top_note_t note;
} dev;

static void
write_start_wave(midi_top_note_t *note) { // unsigned short table_number) {
  iowrite16(note->note_info, WAVE_START(dev.virtbase));
}

static void
write_send_wave(midi_top_note_t *note) { // unsigned short wave_data) {
  iowrite16(note->note_info, WAVE_DATA(dev.virtbase));
}

/*
 * Write segments of a single digit
 * Assumes digit is in range and the device information has been set up
 */
static void write_note(midi_top_note_t *note) {
  iowrite16(note->note_info, dev.virtbase + (2 * note->index));
  dev.note = *note;
}

/*
 * Handle ioctl() calls from userspace:
 * Read or write the segments on single digits.
 * Note extensive error checking of arguments
 */
```

```
static long midi_top_ioctl(struct file *f, unsigned int cmd,
                        unsigned long arg) {
  midi_top_arg_t vla;

  switch (cmd) {

  case MIDI_TOP_START_WAVE:
      if (copy_from_user(&vla, (midi_top_note_t *)arg,
sizeof(midi_top_note_t)))
      return -EACCES;
      write_start_wave(&vla.note);
      // write_note(&vla.note);
      // write_background(&vla.background);
      break;

  case MIDI_TOP_SEND_WAVE:
      if (copy_from_user(&vla, (midi_top_note_t *)arg,
sizeof(midi_top_note_t)))
      return -EACCES;
      write_send_wave(&vla.note);
      // write_note(&vla.note);
      // write_background(&vla.background);
      break;

  case MIDI_TOP_WRITE_NOTE:
      if (copy_from_user(&vla, (midi_top_note_t *)arg,
sizeof(midi_top_note_t)))
      return -EACCES;
      write_note(&vla.note);
      // write_background(&vla.background);
      break;

  case MIDI_TOP_READ_NOTE:
      vla.note = dev.note;
      if (copy_to_user((midi_top_note_t *)arg, &vla,
sizeof(midi_top_note_t)))
      return -EACCES;
      break;

  default:
      return -EINVAL;
  }
```

```
  return 0;
}

/* The operations our device knows how to do */
static const struct file_operations midi_top_fops = {
      .owner = THIS_MODULE,
      .unlocked_ioctl = midi_top_ioctl,
};

/* Information about our device for the "misc" framework -- like a char dev
*/
static struct miscdevice midi_top_misc_device = {
      .minor = MISC_DYNAMIC_MINOR,
      .name = DRIVER_NAME,
      .fops = &midi_top_fops,
};

/*
 * Initialization code: get resources (registers) and display
 * a welcome message
 */
static int __init midi_top_probe(struct platform_device *pdev) {
  // midi_top_color_t beige = { 0xf9, 0xe4, 0xb7 };
  int ret;

  /* Register ourselves as a misc device: creates /dev/midi_top */
  ret = misc_register(&midi_top_misc_device);

  /* Get the address of our registers from the device tree */
  ret = of_address_to_resource(pdev->dev.of_node, 0, &dev.res);
  if (ret) {
      ret = -ENOENT;
      goto out_deregister;
  }

  /* Make sure we can use these registers */
  if (request_mem_region(dev.res.start, resource_size(&dev.res),
DRIVER_NAME) ==
      NULL) {
      ret = -EBUSY;
      goto out_deregister;
```

```
  }

  /* Arrange access to our registers */
  dev.virtbase = of_iomap(pdev->dev.of_node, 0);
  if (dev.virtbase == NULL) {
      ret = -ENOMEM;
      goto out_release_mem_region;
  }

  /* Set an initial color */
  // write_background(&beige);

  return 0;

out_release_mem_region:
  release_mem_region(dev.res.start, resource_size(&dev.res));
out_deregister:
  misc_deregister(&midi_top_misc_device);
  return ret;
}

/* Clean-up code: release resources */
static int midi_top_remove(struct platform_device *pdev) {
  iounmap(dev.virtbase);
  release_mem_region(dev.res.start, resource_size(&dev.res));
  misc_deregister(&midi_top_misc_device);
  return 0;
}

/* Which "compatible" string(s) to search for in the Device Tree */
#ifdef CONFIG_OF
static const struct of_device_id midi_top_of_match[] = {
      {.compatible = "csee4840,midi_top-1.0"},
      {},
};
MODULE_DEVICE_TABLE(of, midi_top_of_match);
#endif

/* Information for registering ourselves as a "platform" driver */
static struct platform_driver midi_top_driver = {
      .driver =
      {
```

```
            .name = DRIVER_NAME,
            .owner = THIS_MODULE,
            .of_match_table = of_match_ptr(midi_top_of_match),
      },
      .remove = __exit_p(midi_top_remove),
};

/* Called when the module is loaded: set things up */
static int __init midi_top_init(void) {
  pr_info(DRIVER_NAME ": init\n");
  return platform_driver_probe(&midi_top_driver, midi_top_probe);
}

/* Calball when the module is unloaded: release resources */
static void __exit midi_top_exit(void) {
  platform_driver_unregister(&midi_top_driver);
  pr_info(DRIVER_NAME ": exit\n");
}

module_init(midi_top_init);
module_exit(midi_top_exit);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Stephen A. Edwards, Columbia University");
MODULE_DESCRIPTION("MIDI top driver");
```

Note.h

```
/*
 * Userspace program that communicates with the vga_ball device driver
 * through ioctls
 *
 * Stephen A. Edwards
 * Columbia University
 */

#include <stdio.h>
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <string.h>
```

```c
#include <unistd.h>
#include <assert.h>


#include "midi_top.h"
//int midi_top_fd;

unsigned short active_notes [10];
int available [10];

void note_init() {
  memset(active_notes, 0, 10*sizeof(short));
  memset(available, -1, 10*sizeof(int));
}

void start_wave(int midi_top_fd, unsigned short data)
{
  midi_top_arg_t vla;
  vla.note.note_info = data;

  if (ioctl(midi_top_fd, MIDI_TOP_START_WAVE, &vla)) {
      perror("ioctl(MIDI_TOP_START_WAVE) failed");
      return;
  }
}

void send_wave(int midi_top_fd, short data)
{
  midi_top_arg_t vla;
  vla.note.note_info = data;

  if (ioctl(midi_top_fd, MIDI_TOP_SEND_WAVE, &vla)) {
      perror("ioctl(MIDI_TOP_SEND_WAVE) failed");
      return;
  }
}


//make sure midi_number >=12 (startin at C0)
// WRITTEN FOR TONEGEN -- we will need to make changes later
// assume volume 0-127
unsigned short to_note_info(unsigned short midi_number, unsigned short
```

```
volume)
{
  unsigned short note_info;

  assert(midi_number >= 12 && midi_number <= 119);
  assert(volume < 128);

  unsigned short note, octave;
  note   = midi_number % 12;
  octave = (midi_number / 12) - 1;

  // note - 4 bits, from 0-11 corresponding to C -> B
  // octave - 4 bits, from 0-8
  // volume - 0-127 for

  unsigned int status = (volume != 0);

  printf("note= %d, octave= %d, volume= %d\n", note, octave, volume);
  note_info = volume;
  note_info |= (status << 7);

  note_info |= (note   << 8);
  note_info |= (octave << 12);

  return note_info;
}


void display_note(unsigned short note_info)
{
  unsigned short note, octave, volume;

  volume = note_info & 127;
  note   = (note_info >> 8) & 0x000F;
  octave = note_info >> 12;

  printf("note= %d, octave= %d, volume= %d\n", note, octave, volume);
}


void set_note(int midi_top_fd, unsigned short midi_number, unsigned short
volume)
```

```
{
  // search for midi note number in active_notes
  int index = 0;
  int found = 0;
  while(index < 10 && !found) {
      if (midi_number == active_notes[index++]) found = 1;
  }

  int status = (volume != 0);
  if(found) {
      // just update the existing entry
      available[index] = !status;
  } else {
      // search for a free space
      if (status) {
      index = 0;
      while(index < 10 && !found) {
      if (available[index++]) found = 1;
      }
      }
  }

  // prepare write data
  midi_top_arg_t vla;
  vla.note.note_info = to_note_info(midi_number, volume);
  vla.note.index = index;

  // sent to the hardware
  if (found) {
      if (ioctl(midi_top_fd, MIDI_TOP_WRITE_NOTE, &vla)) {
      perror("ioctl(MIDI_TOP_WRITE_NOTE) failed");
      return;
      }
      printf("SENT: ");
      display_note(vla.note.note_info);
  }
}


/* Read and print the background color */
void print_note(int midi_top_fd) {
  midi_top_arg_t vla;
```

```
  if (ioctl(midi_top_fd, MIDI_TOP_READ_NOTE, &vla)) {
      perror("ioctl(MIDI_TOP_READ_NOTE) failed");
      return;
  }
  display_note(vla.note.note_info);
  //printf("%02x %02x %02x\n",
    // vla.background.red, vla.background.green, vla.background.blue);
}

/*
int main()
{


  //printf("MIDI top Userspace program started\n");


  int midi_top_fd;
  static const char filename[] = "/dev/midi_top";

  if ( (midi_top_fd = open(filename, O_RDWR)) == -1) {
      fprintf(stderr, "could not open %s\n", filename);
      return -1;
  }

  set_note(midi_top_fd,60, 127);

  //printf("MIDI top Userspace program terminating\n");
  return 0;
}
*/
```

Test.c

```
#include <arpa/inet.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <unistd.h>
```

```c
#include "note.h"
#include "usbmidi.h"
/* Update SERVER_HOST to be the IP address of
 * the chat server you are connecting to
 */
/* micro36.ee.columbia.edu */

#define BUFFER_SIZE 70000
#define MAX_MSG_LEN BUFFER_SIZE - 17

/*
 * References:
 *
 * http://beej.us/guide/bgnet/output/html/singlepage/bgnet.html
 * http://www.thegeekstuff.com/2011/12/c-socket-programming/
 *
 */

struct libusb_device_handle *keyboard;
uint8_t endpoint_address;

// char buf[BUFFER_SIZE];
// char keystate[BUFFER_SIZE];

int main() {

  int midi_top_fd;
  static const char filename[] = "/dev/midi_top";
  if ((midi_top_fd = open(filename, O_RDWR)) == -1) {
      fprintf(stderr, "could not open %s\n", filename);
      return -1;
  }

  /* initialize the driver program */
  note_init();

  /* Open the keyboard */
  if ((keyboard = openkeyboard(&endpoint_address)) == NULL) {

      fprintf(stderr, "Did not find a keyboard\n");
      // fbputs("Did not find a keyboard", 22, 19);
```

```
      exit(1);
  }

  // have to make our own packet...
  struct usb_midi_packet packet;
  int transferred;
  // char keystate[12];

  /* Look for and handle keypresses */
  // uint8_t data[64];

  int wave_1 = 0;
  int wave_2 = 0;

  for (;;) {
      int errcode = 0;
      memset(&packet, 0, sizeof(packet));
      errcode = libusb_bulk_transfer(keyboard, endpoint_address,
                                     (unsigned char *)&packet,
sizeof(packet),
                                     &transferred, 1000);
      if (errcode != 0 ||
      (packet.status_byte != 9 && packet.status_byte != 11 &&
      packet.status_byte != 25 && packet.status_byte != 14)) {
      continue;
      }

      if (packet.status_byte == 25) {
      if (packet.note == 100 && packet.attack == 127) {
      printf("starting the send....\n");
      start_wave(midi_top_fd, 0);
      for (unsigned short i = 0; i < 48000; i++) {
            if (i < 24000)
            send_wave(midi_top_fd, 0x1388);
            else
            send_wave(midi_top_fd, 0xEC78);
      }
      printf("everything sent!..\n");
      }
      if (packet.note == 99 && packet.attack == 127) {
      wave_1++;
      printf("Wave_1 Number %d\n", wave_1);
```

```
      }
      if (packet.note == 98 && packet.attack == 127) {
      wave_1--;
      start_wave(midi_top_fd, 0);
      printf("Wave_1 Number %d\n", wave_1);
      }
      if (packet.note == 96 && packet.attack == 127) {
      wave_2++;
      printf("Wave_2 Number %d\n", wave_2);
      }
      if (packet.note == 97 && packet.attack == 127) {
      wave_2--;
      printf("Wave_2 Number %d\n", wave_2);
      }
      }

      if (packet.status_byte == 9) {
      printf("(NOTE, ATTACK): %d %d\n", packet.note, packet.attack);
      set_note(midi_top_fd, (unsigned short)packet.note,
            (unsigned short)packet.attack);
      }

      if (packet.status_byte == 11 && packet.note == 7) {
      printf("VOLUME(Value): %d\n", packet.attack);
      }
      if (packet.status_byte == 11 && packet.note == 1) {
      printf("MODULATION(Value): %d\n", packet.attack);
      }

      if (packet.status_byte == 14) {
      printf("PITCH WHEEL(VALUE): %d\n", packet.attack);
      }
  }

  return 0;
}
```

Usbmidi.c

```
#include "usbmidi.h"

#include <stdio.h>
```

```c
#include <stdlib.h>

/* References on libusb 1.0 and the USB HID/keyboard protocol
 *
 * http://libusb.org
 *
http://www.dreamincode.net/forums/topic/148707-introduction-to-using-libusb
-10/
 * http://www.usb.org/developers/devclass_docs/HID1_11.pdf
 * http://www.usb.org/developers/devclass_docs/Hut1_11.pdf
 */

/*
 * Find and return a USB keyboard device or NULL if not found
 * The argument con
 *
 */
struct libusb_device_handle *openkeyboard(uint8_t *endpoint_address) {
  libusb_device **devs;
  struct libusb_device_handle *keyboard = NULL;
  struct libusb_device_descriptor desc;
  ssize_t num_devs, d;
  uint8_t i, k;

  /* Start the library */
  if (libusb_init(NULL) < 0) {
      fprintf(stderr, "Error: libusb_init failed\n");
      exit(1);
  }

  /* Enumerate all the attached USB devices */
  if ((num_devs = libusb_get_device_list(NULL, &devs)) < 0) {
      fprintf(stderr, "Error: libusb_get_device_list failed\n");
      exit(1);
  }

  /* Look at each device, remembering the first HID device that speaks
      the keyboard protocol */

  for (d = 0; d < num_devs; d++) {
      libusb_device *dev = devs[d];
      if (libusb_get_device_descriptor(dev, &desc) < 0) {
```

```
	fprintf(stderr, "Error: libusb_get_device_descriptor failed\n");
	exit(1);
	}

	if (desc.bDeviceClass == LIBUSB_CLASS_PER_INTERFACE) {
	struct libusb_config_descriptor *config;
	libusb_get_config_descriptor(dev, 0, &config);
	for (i = 0; i < config->bNumInterfaces; i++)
	for (k = 0; k < config->interface[i].num_altsetting; k++) {
		const struct libusb_interface_descriptor *inter =
		config->interface[i].altsetting + k;

		printf("INTERFACE: %d, %d\n", inter->bInterfaceClass,
			inter->bInterfaceProtocol);

		if (inter->bInterfaceClass == LIBUSB_CLASS_AUDIO &&
		inter->bInterfaceProtocol == USB_MIDI_KEYBOARD_PROTOCOL &&
		(inter->bNumEndpoints != 0)) {
		printf("got in here...\n");
		int r;
		if ((r = libusb_open(dev, &keyboard)) != 0) {
		fprintf(stderr, "Error: libusb_open failed: %d\n", r);
		exit(1);
		}
		printf("past libusb_open...\n");
		if (libusb_kernel_driver_active(keyboard, i)) {
		libusb_detach_kernel_driver(keyboard, i);
		}
		printf("past libusb_kernel_driver_active...\n");
		libusb_set_auto_detach_kernel_driver(keyboard, i);
		printf("past libusb_set_auto_detach_kernel_driver...\n");
		if ((r = libusb_claim_interface(keyboard, i)) != 0) {
		fprintf(stderr, "Error: libusb_claim_interface failed: %d\n",
r);
		fprintf(stderr, "Error %s\n", libusb_strerror(r));
		exit(1);
		}
		printf("past libusb_claim_interface...\n");
		printf("Number of endpoints: %d\n", inter->bNumEndpoints);

		*endpoint_address = inter->endpoint[0].bEndpointAddress;
		printf("WMAXPACKETSIZE = %d\n",
```

```
inter->endpoint[0].wMaxPacketSize);
            goto found;
            }
      }
      }
  }

found:
  libusb_free_device_list(devs, 1);

  return keyboard;
}
```

Usbmidi.h

```
#ifndef _USBMIDI_H
#define _USBMIDI_H

#include <libusb-1.0/libusb.h>

#define USB_HID_KEYBOARD_PROTOCOL 1
#define USB_MIDI_KEYBOARD_PROTOCOL 0

/* Modifier bits */
#define USB_LCTRL (1 << 0)
#define USB_LSHIFT (1 << 1)
#define USB_LALT (1 << 2)
#define USB_LGUI (1 << 3)
#define USB_RCTRL (1 << 4)
#define USB_RSHIFT (1 << 5)
#define USB_RALT (1 << 6)
#define USB_RGUI (1 << 7)

struct usb_midi_packet {
  uint8_t status_byte;
  uint8_t not_sure;
  uint8_t note;
  uint8_t attack;
  uint8_t other_data[60];
};
```

```
// struct usb_keyboard_packet {
//   uint8_t modifiers;
//   uint8_t reserved;
//   uint8_t keycode[6];
// };

/* Find and open a USB keyboard device.  Argument should point to
   space to store an endpoint address.  Returns NULL if no keyboard
   device was found. */
extern struct libusb_device_handle *openkeyboard(uint8_t *);

#endif
```