COLUMBIA UNIVERSITY

Leap-Motion Piano

Motion-sensed Virtual Piano

Patrice Liang (pl2279)
Matthew Patey (mep2167)
Vanshil Shah (vs2409)
Kevin Walters (kmw2168)

5/14/2014

Table of Contents

Overview

This project implements a virtual piano using human input processed by a Leap Motion device. The device uses two monochromatic IR cameras and three infrared LEDs to detect hand and finger movements in an interaction space of eight cubic feet and sends the data to the Leap Motion controller software, which uses math to infer three-dimensional position data. Since the Leap Motion needs to run on x86 architecture, we are unable to use the Hard Processor System (HPS) on the SoCKIT board to read input from the device. Instead, we collect finger position data on a separate computer and send it through Ethernet communication to the HPS.

For the purpose of this project, we process data from one finger to display a sprite cursor on the VGA screen. The project's video component displays a front view of keys from one octave of a piano and a sprite cursor that tracks the user's finger movement by converting raw position data from the Leap Motion device into VGA coordinate data. When the user simulates a key press, the screen displays this action and the corresponding audio note is generated through the SoCKIT board audio output. To ensure realistic output, we stream raw audio files of piano notes from the HPS to the audio hardware.

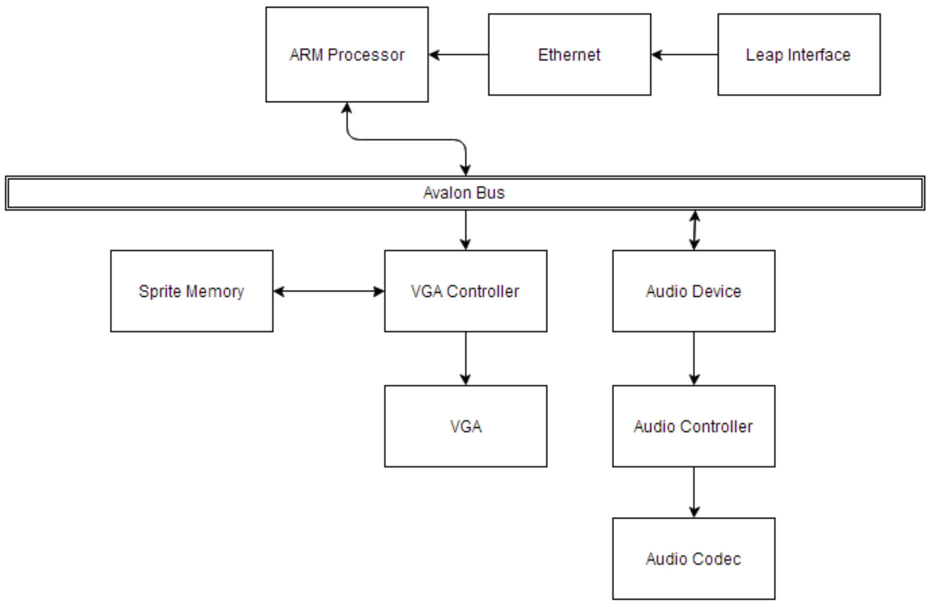
Design and Implementation

Architecture Overview

As shown in the diagram below, the ARM Processor interacts with Ethernet directly, and the VGA Controller and Audio Device through the Avalon Bus. Its primary functions are as follows:

- receiving coordinate data from the Leap Motion device through Ethernet communication
- displaying piano keys and simulating key presses on the VGA display through the VGA Controller
- producing corresponding audio notes through the Audio Controller

The sprite memory module interacts with the VGA Controller to display the cursor image on screen.



Software - Overview

The software side of the project includes both userspace and driver programs. The software in general is in charge of 3 major components:

- communication through Ethernet (on HPS and a separate workstation)
- detecting key presses on the virtual piano using finger positions (HPS)
- generating audio and video data (HPS)

The Ethernet communication is done by sending finger position information over a UDP connection from a separate workstation running Java. This data packet is received on the HPS through a C program. This position data is converted from physical measurements (millimeters) to pixels. This data is then used to determine when a key is pressed based on its position and velocity, and the software writes the appropriate video and audio information to the hardware through the drivers. We will now go into a detailed analysis of the software side of our project.

Software - Details

Our project consists of both userspace programs and kernel drivers. Userspace programs communicate with their respective drivers, which in turn communicate with the connected hardware peripherals. The software is split into a number of different files, each focusing on a specific goal. In general, the software takes care of receiving finger location information through UDP packets, sending appropriate audio files to the hardware to be played, and maintaining the state of keys that are pressed. Each functionality is described in its appropriate file below.

Userspace Programs

LeapSender.java

This program parses the Leap information to send the data to the SoCKIT board through UDP packets. It takes the first finger of the first hand received from the Leap device and retrieves the coordinates and the velocity in the y direction. It then puts these into a byte stream and sends it over the UDP socket to the SoCKIT board. The userspace program coordrcv.c gets this data.

coordrcv.c

This is the file that is run from the command line to allow our project to run. It begins by opening the VGA and audio driver file descriptors. It then immediately writes the data of our cursor image to the driver, so it can be stored in hardware. The next task it completes is opening a UDP socket, which receives packets from the Leap Motion device. The program then continuously reads packets from the socket. This data is parsed, providing us with x and y coordinates (in millimeters) and a y velocity of the finger being tracked. These coordinates are then translated into pixel coordinates so they can be displayed on the screen. Based on these coordinates and the y velocity, the program the determines if a key has been pressed. If so, it indicates the specific key to the VGA driver (vga_piano.c) and the corresponding audio file to the audio thread (see below for audio details).

audio.c

This file deals with the audio component of the userspace programs. An audio thread is created when coordrcv.c is first run. This threading allows the audio to write to the audio driver (audio_driver.c) while the VGA component can simultaneously continue to update its coordinates to the VGA driver. The beginning of the audio thread opens every audio file that can be played and stores each in a buffer. When a key press is determined, the appropriate key is sent to the audio thread. Based on this information, the correct buffer is accessed and the audio file's data is then sent to the audio driver 2 KB at a time.

When a key is pressed, we compare this key value to the previous key value and send another play signal only when they are not equal, thereby ensuring that the same audio file is not played again if the key press persists. We also include a thread-safe queue in queue.c to enable cancellation of the currently-playing audio file if another key is pressed. After every 2 KB chunk of audio data is sent to the driver, the queue is checked. If an item is in the queue, the file pointer being played is changed to the file pointer located in the queue. The audio then continues sending data to the driver, but with the data of the new file, offering a seamless transition between keys.

Driver Programs

vga_piano.c

This is the device driver that sends information from the userspace to the VGA hardware. There are three functions that the driver utilizes: write_coord, write_data, and write_keypress. write_coord reads an x and y coordinate, each 16 bits, from the userspace program coordrcv.c. It then writes this data to the hardware using an iowrite32 command. write_data is only used during the initializing of coordrcv.c, and sends 32 bits of data per pixel of the image we write. write_keypress writes a 32-bit integer, representing the key that has been determined to be pressed by the software.

audio_driver.c

This is the device driver that sends information from the userspace to the audio hardware. There are two writing functions, write_data and send_play_signal, and one read function, poll. poll is used to read a 16-bit number from the hardware to see if one of the audio buffers is available to be written to. If not, it will wait before writing. send_play_signal writes a 1 (16 bits) to the hardware if it has been sent new data to be played. These two functions work together to ensure that no audio data is overwritten in the hardware due to the slow clock of the audio codec. Finally, write_data actually writes the audio data to the hardware. As long as one buffer is free, a series of 1024 16-bit values will be written one chunk at a time to the hardware. When all of the 2 KB data has been sent, a play signal will be sent to the hardware.

Memory Requirements

The memory requirements of our project mainly comes from the buffers in the audio device and the memory for the cursor sprite. The cursor sprite consists of a 32x32 pixel image. Each pixel is comprised of four bytes, R, G, B, and alpha (A), which uses 4 KB of memory. The audio device uses two buffers, each with a capacity of 1024 16-bit audio data points, for a total of 4 KB. This brings the total memory usage to 8 KB.

Hardware - Overview

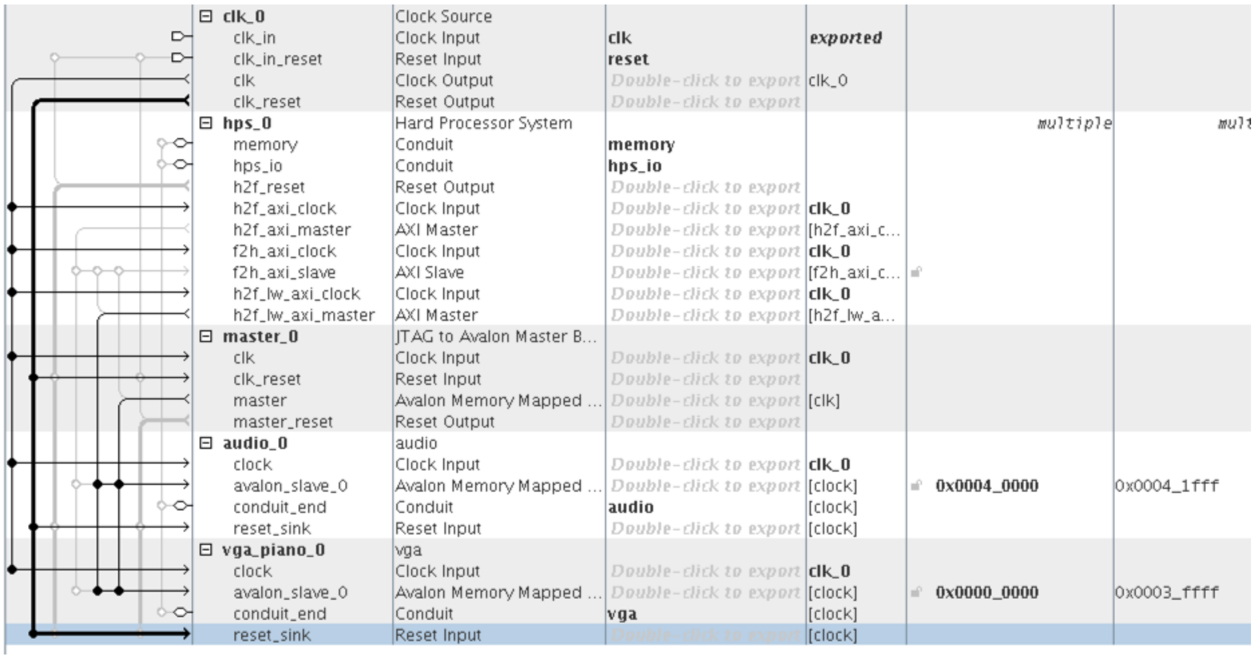
The hardware components of this project are primarily responsible for the following:

- displaying the user interface on the VGA screen, which includes piano keys and a tracking cursor
- reading audio data from the HPS and playing the note that corresponds to a key press

Hardware - Details

Peripherals

Peripherals used in this project include the Leap Motion device, the VGA screen, and the audio codec. The following screenshot from Qsys shows the peripheral connections:



⊟ clk_0	Clock Source				
clk_in	Clock Input	**clk**	*exported*		
clk_in_reset	Reset Input	**reset**			
clk	Clock Output	*Double-click to export*	clk_0		
clk_reset	Reset Output	*Double-click to export*			
⊟ hps_0	Hard Processor System			*multiple*	*mult*
memory	Conduit	**memory**			
hps_io	Conduit	**hps_io**			
h2f_reset	Reset Output	*Double-click to export*			
h2f_axi_clock	Clock Input	*Double-click to export*	**clk_0**		
h2f_axi_master	AXI Master	*Double-click to export*	[h2f_axi_c…		
f2h_axi_clock	Clock Input	*Double-click to export*	**clk_0**		
f2h_axi_slave	AXI Slave	*Double-click to export*	[f2h_axi_c…		
h2f_lw_axi_clock	Clock Input	*Double-click to export*	**clk_0**		
h2f_lw_axi_master	AXI Master	*Double-click to export*	[h2f_lw_a…		
⊟ master_0	JTAG to Avalon Master B…				
clk	Clock Input	*Double-click to export*	**clk_0**		
clk_reset	Reset Input	*Double-click to export*			
master	Avalon Memory Mapped …	*Double-click to export*	[clk]		
master_reset	Reset Output	*Double-click to export*			
⊟ audio_0	audio				
clock	Clock Input	*Double-click to export*	**clk_0**		
avalon_slave_0	Avalon Memory Mapped …	*Double-click to export*	[clock]	**0x0004_0000**	0x0004_1fff
conduit_end	Conduit	**audio**	[clock]		
reset_sink	Reset Input	*Double-click to export*	[clock]		
⊟ vga_piano_0	vga				
clock	Clock Input	*Double-click to export*	**clk_0**		
avalon_slave_0	Avalon Memory Mapped …	*Double-click to export*	[clock]	**0x0000_0000**	0x0003_ffff
conduit_end	Conduit	**vga**	[clock]		
reset_sink	Reset Input	*Double-click to export*	[clock]		

VGA

The video display of the user interface is primarily handled in hardware by VGA_Piano.sv and VGA_Piano_Emulator.sv. The emulator displays the piano keys and the background on the screen through raster scanning logic. In addition to taking as input a 50MHz clock and cursor coordinates from the VGA_Piano module, it also takes as input an 8-bit keypress indicator that indicates which key is pressed. Keys are numbered 1 to 12, left-to-right from the bottom row up. A value of 0 indicates that no key is pressed. Using this information, the emulator determines where and when to display a pressed key.

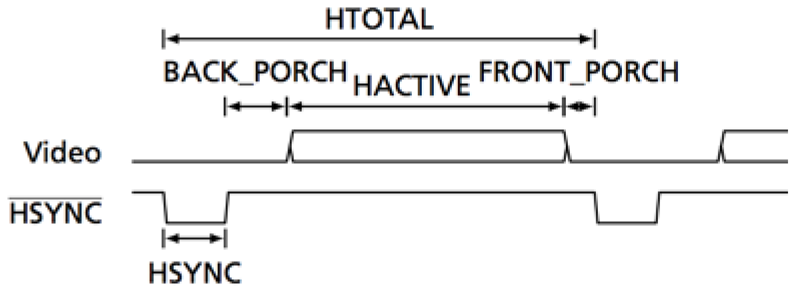
To display the cursor on the screen, the emulator first determines where to draw it based on input from VGA_Piano. It then requests pixel information by outputting an address selector to VGA_Piano. VGA_Piano sends the address selector to the sprite_memory module, which is responsible for initially storing the sprite cursor data in memory and subsequently outputting appropriate pixels based on what is requested by the address selector. VGA_Piano then outputs the correct RGBA values to the emulator to display the cursor.

Software-Hardware Interface:

The VGA_Piano module interfaces with software. It takes as inputs a 32-bit writedata and a 16-bit address. The value of address indicates whether writedata contains the sprite cursor data, the cursor coordinates, or the keypress information. Based on the value of address, the hardware will write writedata to sprite memory, split it into cursorx and cursory positions, or read the first 8 bits into keypress.

VGA Timing:

The timing diagram below shows how the VGA signals are output.



Audio:

The SoCKIT board includes a SSM2603 low power audio codec. This codec runs on the I2C bus with write address 0x34. The audio hardware we wrote has a I2C bus controller (i2c_controller.sv) and a I2C configuration sender (i2c_av_config.sv). The clock of the I2C bus is derived by dividing the main FPGA clock by 128 to get a clock of approximately 390 KHz. i2c_av_config sends the configuration values to the audio codec using the I2C controller so that the audio codec can be written to by our audio codec controller (audio_codec.sv). Some of the configurations that we send to the I2C bus include a sample rate of 44.1 KHz, a sample size of 16 bits, 0 dB volume, etc. At the end of the configuration, the audio codec is given a power on signal, after which we can directly send data to the audio codec.[1] Our audio interface (audio_interface.sv) is based on a buffer system with two buffers, each of size 2048 bytes. The HPS writes to one buffer while the other plays, and the buffers switch as the codec finishes playing and the HPS finishes writing data.

Hardware-Hardware:

The audio codec runs at a clock speed not directly derivable from the main FPGA clock, so we used the Megawizard function to create a precision clock of 11.2896 MHz. Our audio interface, which communicates with the HPS, sends data to the audio codec using the audio clock, which is then output from the DAC on the 3mm audio jack. The audio codec is consecutively given data from one buffer at a time, allowing for seamless playing.

Software-Hardware:

The audio interface is connected through Qsys to the HPS. The HPS constantly polls the audio peripheral for its status. Once the peripheral indicates that it is ready to accept data, the HPS writes to one of the buffers using 16-bit data. When the buffer is filled, the audio codec is signalled to start playing from the first buffer. The HPS then writes to the second buffer, and so on.

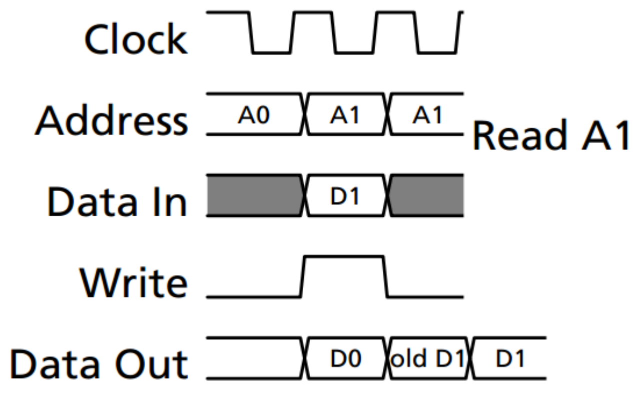
Memory:

The cursor sprite data is stored in memory and accessed by the sprite hardware when drawing to the screen. The memory controller takes read and write requests and passes them on to the memory.

[1] http://zhehaomao.com/blog/fpga/2014/01/15/sockit-8.html

Hardware-Hardware:

Below is timing for the interface between the memory controller and memory hardware:



Address is a 16-bit signal that informs the memory which address to access. Data In and Data Out are 32 bits wide and transfer data between the devices. When the value of Write is 1, the memory reads data from Data In into the specified address. When the value of Write is 0, the memory writes data from the specified address onto Data Out.

Challenges

Cursor Sprite

During the raster scan for the VGA, the hardware determines if it is within the bounds of the cursor. If it is, it fetches the current pixel from the sprite memory and paints that instead of the piano or background. The problem with this strategy is that the data from the sprite memory comes one cycle after the desired address is sent to the memory, meaning that values are a cycle old, causing the image to be painted one pixel to the right. While this was not a big problem, it also caused the column one to the left of the sprite to all be painted with the sprites 0, 0 pixel. This occurred because when the raster scan was not within the bounds of the cursor the memory always sent out the first value, so for the first column being one behind meant painting the pixel sent from the memory the previous cycle when the raster asked for 0, 0 because it was not within the cursor bounds. This was solved by starting the cursor painting one cycle after starting the data request. Thus, the first time that the raster scan paints a cursor pixel in a given row, the pixel was requested the cycle before.

Audio

Playing audio also presented our group some challenges. We decided to stream audio data from the HPS to the audio hardware by using two buffers, one of which was filled by the HPS while the other was sent to be played by sending it to the audio codec. To facilitate this, a ready flag was allocated for each buffer. A 1 meant that the buffer was full and ready to be played, and a 0 meant that if had been played, and is either not started or in the process of being filled by the HPS. The HPS could tell whether it should send a frame of data to the hardware based on these flags. As such, the flag must be set to 1 when the HPS fills it, and set to 0 when playback reaches the end of the buffer. This was complicated by the fact that playback and writes from the HPS operate on two different clocks, so the two different sections of the hardware could not both write to the same register. This was solved by giving each section of the hardware its own register operating on the corresponding clock that had a 1 written to it to represent an event (e.g. finished playing the buffer). Then, logic running on one clock could read both of the intermediate buffers and update the flag (on the same clock) if necessary.

Design Changes

The original design for the video used a frame buffer stored in RAM. One hardware unit would combine the piano image and the cursor in the frame buffer, and a separate unit would read from the hardware during the VGA raster scan. We realized, however, that a frame buffer was an overkill for our needs, considering that we had one small sprite and a piano that was drawn in hardware logic. Considering the significant memory needs of frame buffers and the complexity of adding a memory controller for the DRAM to our design, we decided that using a frame buffer would do little more than complicate our design. We decided to merge the two hardware units by combining the piano and the cursor on the fly during the raster scan. This removed the approximately 1 MB memory requirements of a frame buffer, leaving the video hardware with just 4 KB memory requirements from the sprite memory, which easily fits in the FPGA fabric.

Timeline

Milestone 1

For the first milestone, we displayed a temporary cursor on the VGA screen that tracks movement from one finger through the Leap Motion device. This involved creating the Leap Interface by writing software that sends raw data from the Leap API to the SoCKIT board via Ethernet and implementing coordinate conversion for VGA display.

Milestone 2

We displayed one octave of piano keys on the VGA screen and set up the sprite memory hardware for cursor display. We also implemented key press logic in software to recognize a key press and determine the specific key pressed.

Milestone 3

On the video front, we tied together key presses and key animation on the VGA screen. For this milestone, we set up audio components to output simple sine wave audio independent of key presses.

Final Phase

After the last milestone, we implemented the audio component to output raw audio files of piano notes that correspond to key presses. We also fixed small bugs and errors, and cleaned and commented the code.

Reflections

Kevin

This was a very challenging but very fulfilling project to undertake. There were many challenges along the way, but I am very proud of how our project turned out. Throughout the process we learned many lessons, including the importance of starting early and making deadlines. What stood out the most for me was how much this process stressed the differences in developing software and hardware. Developing in software is very straightforward - if you have a series of bugs, you you can fix one, recompile, and continue with the others. This is not possible in hardware because the compilation times are so long. So a major theme that I took away from this project (and course) is that hardware must be approached with a completely different mindset than software. Overall I really enjoyed working on this project, and am very proud of our efforts and our final result.

Matt

This project gave me an understanding of how hardware and software interact in a way that no theoretical class could. Too often unexplained abstractions leave holes in mental models, but by designing this project from the userspace software down to the hardware registers, I feel like I finally see the whole picture. Additionally, I have a greater appreciation for the complexity and challenges of designing hardware such as video that is often taken for granted. I never thought that I would feel so accomplished by getting a few seconds of an audio file to play.

Patrice

Though I had worked on group projects of similar scale in the past, this is by far the one that involved the widest range of technical applications. At every stage of the project, we faced an unseen challenge and learned something new. Aside from the technical aspects, this also taught me not to take the easy way out if it compromises the quality of the project in the long run. In the case of our project, we briefly toyed with the idea of generating sine waves to output as audio instead of playing more realistic piano notes from audio files, which requires a much more complex implementation. We decided pretty quickly that doing so would greatly undermine the quality of our project and I learned that an initially daunting problem can be worked out most of the time, step by step. In terms of logistics, I found milestones to be extremely helpful and following them to the best of our abilities benefited us tremendously in the end.

Vanshil

Going into this project I had never programmed anything in hardware before and in hindsight, I knew nothing about device drivers or device trees. I had taken COMS 4118 (OS) and thought that I understood but how wrong I was. This project was fantastic because now I feel like I have finally cracked the surface of knowledge about the mysterious device that is the computer. Before doing this project, I had only read about buses and memory, but now I have first hand experience in programming and using those. The computer is an extremely intricate device and this project has only made me appreciate it even more. The project has given me invaluable experience in hardware and software alike and it has motivated me to explore as much as I can in this subject. After this project and this class, I think I would like to pursue a career in embedded systems and learn as much as I can.

Distribution of Work

Kevin

Leap interface, VGA hardware, VGA userspace, audio driver, audio userspace

Matt

Leap interface, VGA driver, memory, audio driver, audio userspace, audio hardware

Patrice

Leap interface, VGA hardware, VGA userspace, audio userspace

Vanshil

Leap interface, VGA driver, memory, audio userspace, audio hardware

Future Work

We started this project with a very ambitious mindset. Unfortunately, but inevitably, we were not able to complete everything we had planned. Below we will discuss some implementations that can be done in the future.

Additional Fingers

Our project currently only supports the usage of one finger. Fortunately, adding more fingers to the design would be relatively simple. The first step would be to include additional coordinates and velocities for each finger added and to send these through the UDP socket to the SoCKIT board userspace program.

Drawing the fingers

Data from each finger would then be parsed from the packet and sent to the VGA hardware through the driver sequentially. An additional 4 bits would be needed to differentiate which finger's coordinates are being updated. In the hardware, additional coordinate information would have to be kept for each finger to be tracked. The logic for drawing each cursor would remain exactly the same, but duplicated for each additional finger being tracked.

Drawing the key presses

To determine key presses, the logic would need to be applied for all fingers. Instead of using one variable that stores which key is pressed, as is done currently, we can use a bitmap of 12 bits, where each bit represents a key. We can then call write_keypress with this new bitmap, and the hardware will use this mapping to determine the drawing logic for the keys.

Multiple audio

After implementing multiple cursors, the next extension would be to play multiple keys at once.

Luckily, this is all managed in software. We essentially need to handle playing a chord and fixing our current key-cancellation logic so that multiple key presses are possible from an audio perspective. Theoretically, the data of each note being played is added together to play a chord. Therefore, when sending the data to the driver, every 2 bytes from each audio file would be added after first be scaled down by some value (i.e. by the number of notes being played). After doing this for 2 KB, it is then sent to the driver.

To handle the key-cancellation logic, each finger would need to have its own queue. In sending the audio data, each finger's own queue is checked to see if a new key has been pressed by it. Using this, pressing a new key with the same finger cancels the current audio being played due to that finger, but the rest of the keys' audio continues as expected.

Interrupts

One of the problems we ran into in finishing our project is how to ensure we do not overwrite any unplayed audio data when trying to write more data. To fix this, we implemented a dual-buffer system. With this, we can poll the hardware to see if at least one buffer is ready to be filled. If not, we continue to loop and poll until a buffer is ready, at which point we continue writing data to the audio component. Instead of this, we would like to use interrupts. Interrupts are useful because we don't need to continuously ask the hardware if it is ready - we can simply wait until the hardware tells us that it is ready, thereby saving a lot of traffic on the bus.

Condition Variables

Condition variables can be useful to us in waiting for a new note to be played, if none is already being played. Currently, this is done in an infinite while loop, while checking the queue. In each loop iteration, we know there is something to play if popping from the queue returns a non-null value. This functions correctly, but is unfortunately a waste of the processor's resources. A better implementation would be to use condition variables. Condition variables are used in multi-threaded programs and allow one thread to sleep until a certain condition is met. In our situation, if no note is currently being played, we can simply have the thread wait (sleep) on the condition that nothing is in the queue. Once something is pushed into the queue, we can signal (wake up) the audio thread, and it can pop from the queue to play the audio.

Continuous Note Playing

An additional feature that could be provided in our project is the implementation of continuous note playing. Currently, holding down the same key will play its corresponding note only once. While this is similar to how real pianos work, this isn't a real piano! Therefore, the functionality may be desired to be more like an electric keyboard, which continues to play the note as long as the key is pressed. One possible solution to this would be to take the last segments of the audio file being played, temporarily store it, and simply continue to play it after the file has finished. This can continue until that key is no longer pressed.

ALSA-Compatible Driver

An ALSA driver is an Advanced Linux Sound Driver. ALSA drivers provide audio and MIDI functionality to Linux devices. In creating an ALSA driver, we would then be able to play any audio directly from Linux on the board using our audio peripheral.

Source Code

Software

audio.c

```
#include <stdio.h>
#include <fcntl.h>
#include "audio_driver.h"
#include "queue.h"
#include <pthread.h>

FILE *audio_files[12];
int audio_fd;
audio_arg_t aa;
extern pthread_mutex_t key_mutex;
queue *audio_queue;

void send_to_audio_thread(unsigned int key) {
       key -= 1; // let interface present keys from 1 - 12
       if (key < 12) {
              push(audio_queue, audio_files[key]);
       }
}

void poll_buffer() {
       // get device status from driver
       if (ioctl(audio_fd, AUDIO_READ_DATA, &aa))
              perror("ioctl(AUDIO_READ_DATA) failed");
       //aa.data contains the signal
}

void write_audio_data(short *data, int length) {
       // send audio data frame to device
       aa.data = data;
       aa.length = length;
       if (ioctl(audio_fd, AUDIO_WRITE_DATA, &aa))
              perror("ioctl(AUDIO_WRITE_DATA) failed");
}

int init_audio() {
       // open audio driver
       static const char file[] = "/dev/audio";
       if ( (audio_fd = open(file, O_RDWR)) == -1) {
              fprintf(stderr, "could not open %s\n", file);
              return -1;
       }

       char filename[100];
       int i;

       // open 12 key audio files
       for (i = 1; i < 13; i++) {
              sprintf(filename, "raw_audio/key%d.raw", i);
              FILE *audio = fopen(filename, "r");
              fseek(audio, 0L, SEEK_END);
              int sz = ftell(audio);
              rewind(audio);
              printf("opening file of length %d\n", sz);
              if (!audio) {
                     perror("could not open file\n");
                     return -1;
              }
              audio_files[i-1] = audio;
       }

       // thread safe queue used to send audio files to audio thread
       audio_queue = initQueue();
       return 0;
}

void play_audio(FILE *note_file) {
       short data[1024];
       size_t read;

       // read from file, loop until the file is done or the buffer is full
       while ((read = fread(data, 2, 1024, note_file)) > 0) {
              do { // poll until buffer is open in device
                     poll_buffer();
              } while (aa.data == 3);

              write_audio_data(data, (unsigned int)read);

              // check if we should stop current file and start playing new one
              void *queue_val = pop(audio_queue);
              if (queue_val != NULL) {
                     rewind(note_file);
                     note_file = (FILE *)queue_val;
              }
       }

       rewind(note_file);
}


// starting point of audio thread
void audio_thread_fn(void *arg) {
       while(1) { // loop, popping off of queue, play if something there
              void *queue_val = pop(audio_queue);
              if (queue_val != NULL) {
                     FILE *note_file = (FILE *)queue_val;
                     play_audio(note_file);
              }
       }
}
```

audio_driver.c

```c
/*
 * Device driver for the Audio Player
 *
 * 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 audio_driver.ko
 *
 * Check code style with
 * checkpatch.pl --file --no-tree audio_driver.c
 */

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

#define DRIVER_NAME "audio"
#define MAX_MEM_ADDRESS 2049 // last address of writable memory

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

static void read_data(audio_arg_t *aa) {
	// read from buffer status address
	aa->data = (short)ioread16(dev.virtbase+MAX_MEM_ADDRESS+1);
}

static void send_play_signal(void) {
	// write to play address
	iowrite16(0x0001, dev.virtbase+MAX_MEM_ADDRESS-1);
}

static int write_data(audio_arg_t *aa) {
	short data[1024];
	unsigned int num_bytes = aa->length * 2;
```

```
        int poll_data;

        // copy audio data frame from user space
        if (copy_from_user(data, aa->data, num_bytes))
                return -EACCES;

        int address;
        int i = 0;

        // loop over address space sending audio data words
        for (address = 0; address < num_bytes; address += 2) {
                iowrite16(data[i], dev.virtbase+address);
                i++;
        }

        send_play_signal();

        return 0;
}

/*
 * Handle ioctl() calls from userspace:
 * Note extensive error checking of arguments
 */
static long audio_ioctl(struct file *f, unsigned int cmd, unsigned long arg)
{
        audio_arg_t aa;

        if (copy_from_user(&aa, (audio_arg_t *) arg, sizeof(audio_arg_t))) //gets aa
struct from userspace
                return -EACCES;
        switch (cmd) {
        case AUDIO_READ_DATA:
                read_data(&aa);//read buffer status data into aa.data
                if (copy_to_user((void *)arg, &aa,sizeof(audio_arg_t))) //write struct
back to userspace
                        return -EACCES;
                break;
        case AUDIO_WRITE_DATA:
                if (aa.length > 1024) {
                        return -EINVAL;
                }
                return write_data(&aa);
        default:
                return -EINVAL;
        }
        return 0;
}

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

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

```c
};

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

	/* Register ourselves as a misc device: creates /dev/audio */
	ret = misc_register(&audio_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;
	}

	return 0;

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

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

/* Which "compatible" string(s) to search for in the Device Tree */
#ifdef CONFIG_OF
static const struct of_device_id audio_of_match[] = {
	{ .compatible = "altr,audio" },
	{},
};
MODULE_DEVICE_TABLE(of, audio_of_match);
#endif
```

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

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

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

module_init(audio_init);
module_exit(audio_exit);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Leap-Motion, Columbia University");
MODULE_DESCRIPTION("Audio playback device");
```

audio_driver.h

```
#ifndef _AUDIO_DRIVER_H
#define _AUDIO_DRIVER_H

#include <linux/ioctl.h>

typedef struct {
  short *data;
  unsigned int length;
} audio_arg_t;

#define AUDIO_MAGIC 'q'

/* ioctls and their arguments */
#define AUDIO_WRITE_DATA _IOW(AUDIO_MAGIC, 1, audio_arg_t *)
#define AUDIO_READ_DATA _IOWR(AUDIO_MAGIC, 2, audio_arg_t *)

#endif
```

coordrcv.c

```
/**
 *
 * Our userspace program that communicates with the Leap and with the VGA and audio
peripherals
 *     Receives and converts coordinates from the Leap device, uses keypress logic to
tell the hardware peripherals what to display and play
 */


#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <arpa/inet.h>
#include <fcntl.h>
#include <unistd.h>
#include <pthread.h>
#include "vga_piano.h"

#define PORT htons(4000)
#define BUFFER_SIZE 4096
#define KEYPRESS_VEL -10
#define WHITE_KEY_WIDTH 91
#define BLACK_KEY_WIDTH 30
#define WHITE_KEY_HEIGHT 400
#define WHITE_KEY_HEIGHT_PRESSED 430
#define BLACK_KEY_HEIGHT 380
#define BLACK_KEY_HEIGHT_PRESSED 395
#define SCREEN_WIDTH_MM 340
#define SCREEN_WIDTH_PIXELS 640
#define SCREEN_HEIGHT_MM 270
#define SCREEN_HEIGHT_PIXELS 480
#define SCREEN_OFFSET 100 //distance in millimeters from bottom of screen to Leap
#define CURSOR_SIZE 32

void write_data(unsigned short address, unsigned int data);
void write_keypress(unsigned short keypress);
void write_coords(const unsigned short xcoord, const unsigned short ycoord);

extern void audio_thread_fn(void *);
extern send_to_audio_thread(unsigned int key);

int vga_piano_fd;

/* Variables to deal with audio */
int key;
int pressed_key;
pthread_t audio_thread;
int play = 0;

int main()
{
       init();
       //variables to deal with receiving data packets
       int sockfd;
       struct sockaddr_in saddr;
       char recvBuf[BUFFER_SIZE+1];
```

```
int n;

//open a new UDP socket
if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0 )
{
        fprintf(stderr, "Error: Could not create socket\n");
        exit (1);
}

saddr.sin_family = AF_INET;
saddr.sin_port = PORT;
saddr.sin_addr.s_addr = htonl(INADDR_ANY);

if (bind (sockfd, (struct sockaddr *)&saddr, sizeof(saddr)) < 0)
{
        fprintf(stderr, "Error: Could not bind to a port");
        exit (1);
}

printf("%s","Beginning to read packets\n");

//write initial values to the hardware
write_coords(100, 100);
write_keypress(0);
pressed_key = 0;
key = 0;
//audio thread for concurrent playback and VGA updates
pthread_create(&audio_thread, NULL, &audio_thread_fn, NULL);

while (1)
{
        memset(recvBuf, 0, BUFFER_SIZE+1);
        //receive packet from UDP socket
        n = recvfrom(sockfd, recvBuf, BUFFER_SIZE - 1, 0, NULL, NULL);

        //get data from the packet
        int *coord = (int *)recvBuf;
        int x = ntohl(*coord++);
        int y = ntohl(*coord++);
        int y_vel = ntohl(*coord);

        //translate from millimeters to pixels
        get_coords(&x,&y,&y_vel);

        //keep the cursor within the screen borders
        if (x < 0)
                x = 0;
        else if (x > SCREEN_WIDTH_PIXELS-CURSOR_SIZE)
                x = SCREEN_WIDTH_PIXELS-CURSOR_SIZE;

        if (y < 0)
                y = 0;
        else if (y > SCREEN_HEIGHT_PIXELS-CURSOR_SIZE)
                y = SCREEN_HEIGHT_PIXELS-CURSOR_SIZE;
        write_coords((unsigned short)x, (unsigned short)y);

        //if a different key has been pressed, tell the audio and VGA drivers
        int old_key = key;
        key = isKeypress(x,y, y_vel);
        if (key != old_key) {
```

```
                    write_keypress((unsigned short) key);
                    if (key > 0) {
                          send_to_audio_thread(key);
                    }
              }
       }

       return 0;
}

/*Use the coordinates and velocity to determine if a key has been pressed
  Returns the number of the key pressed*/
int isKeypress(int x, int y, int y_vel) {
       int x_mid = x + CURSOR_SIZE/2;//for a 32x32 sprite
       int y_bot = y + CURSOR_SIZE;
       int keypress = 0;

       //Determine if a black key has been pressed
       if (x_mid > (WHITE_KEY_WIDTH - BLACK_KEY_WIDTH/2) && x_mid < (WHITE_KEY_WIDTH +
BLACK_KEY_WIDTH/2) && y_bot >= BLACK_KEY_HEIGHT && (y_vel < KEYPRESS_VEL || key == 8))
              keypress = 8;
       else if (x_mid > (2*WHITE_KEY_WIDTH - BLACK_KEY_WIDTH/2) && x_mid <
(2*WHITE_KEY_WIDTH + BLACK_KEY_WIDTH/2) && y_bot >= BLACK_KEY_HEIGHT && (y_vel <
KEYPRESS_VEL || key == 9))
              keypress = 9;
       else if (x_mid > (4*WHITE_KEY_WIDTH - BLACK_KEY_WIDTH/2) && x_mid <
(4*WHITE_KEY_WIDTH + BLACK_KEY_WIDTH/2) && y_bot >= BLACK_KEY_HEIGHT && (y_vel <
KEYPRESS_VEL || key == 10))
              keypress = 10;
       else if (x_mid > (5*WHITE_KEY_WIDTH - BLACK_KEY_WIDTH/2) && x_mid <
(5*WHITE_KEY_WIDTH + BLACK_KEY_WIDTH/2) && y_bot >= BLACK_KEY_HEIGHT && (y_vel <
KEYPRESS_VEL || key == 11))
              keypress = 11;
       else if (x_mid > (6*WHITE_KEY_WIDTH - BLACK_KEY_WIDTH/2) && x_mid <
(6*WHITE_KEY_WIDTH + BLACK_KEY_WIDTH/2) && y_bot >= BLACK_KEY_HEIGHT && (y_vel <
KEYPRESS_VEL || key == 12))
              keypress = 12;
       //Determine if a white key has been pressed
       else if (x_mid < WHITE_KEY_WIDTH && y_bot >= WHITE_KEY_HEIGHT && (y_vel <
KEYPRESS_VEL || key == 1))
              keypress = 1;
       else if (x_mid > WHITE_KEY_WIDTH && x_mid < 2*WHITE_KEY_WIDTH && y_bot >=
WHITE_KEY_HEIGHT && (y_vel < KEYPRESS_VEL || key == 2))
              keypress = 2;
       else if (x_mid > 2*WHITE_KEY_WIDTH && x_mid < 3*WHITE_KEY_WIDTH && y_bot >=
WHITE_KEY_HEIGHT && (y_vel < KEYPRESS_VEL || key == 3))
              keypress = 3;
       else if (x_mid > 3*WHITE_KEY_WIDTH && x_mid < 4*WHITE_KEY_WIDTH && y_bot >=
WHITE_KEY_HEIGHT && (y_vel < KEYPRESS_VEL || key == 4))
              keypress = 4;
       else if (x_mid > 4*WHITE_KEY_WIDTH && x_mid < 5*WHITE_KEY_WIDTH && y_bot >=
WHITE_KEY_HEIGHT && (y_vel < KEYPRESS_VEL || key == 5))
              keypress = 5;
       else if (x_mid > 5*WHITE_KEY_WIDTH && x_mid < 6*WHITE_KEY_WIDTH && y_bot >=
WHITE_KEY_HEIGHT && (y_vel < KEYPRESS_VEL || key == 6))
              keypress = 6;
       else if (x_mid > 6*WHITE_KEY_WIDTH && x_mid < 7*WHITE_KEY_WIDTH && y_bot >=
WHITE_KEY_HEIGHT && (y_vel < KEYPRESS_VEL || key == 7))
              keypress = 7;
       else
```

```
              keypress = 0;
       return keypress;
}

/* Converts from Leap coordinates (millimeters) to pixels*/
int get_coords(int *x, int *y, int *y_vel)
{
       //middle of screen is 170 mm = 320 pixels
       int tempx, tempy;
       tempx = *x;
       tempy = *y;
       //handle Leap coordinate system offset
       tempx = tempx + SCREEN_WIDTH_MM/2;
       //convert from millimeters to pixels
       *x = tempx  * SCREEN_WIDTH_PIXELS/SCREEN_WIDTH_MM;

       //subtract distance from the Leap to the bottom of the monitor used
       tempy -= SCREEN_OFFSET;
       tempy = tempy * SCREEN_HEIGHT_PIXELS/SCREEN_HEIGHT_MM;
       //invert to a top-down coordinate system
       *y = SCREEN_HEIGHT_PIXELS - tempy;

       return 0;
}

/* Initializes the file descriptor used to communicate with the vga_piano peripheral *
int init() {
  vga_piano_arg_t vla;
  int i;
  static const char filename[] = "/dev/vga_piano";
  static const char imageFileName[] = "greenCircle.dat";
  FILE *fp;

  printf("Userspace program initialized\n");

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

  fp = fopen(imageFileName, "rb");
  if(!fp) {
    fprintf(stdout, "could not open %s\n", imageFileName);
    return -1;
  }

  //writes cursor data to the driver
       for(i = 0; i < CURSOR_SIZE*CURSOR_SIZE; i++) {
       unsigned int temp;
              fread((void *)&temp, sizeof(int), 1, fp);
              write_data(i*4, temp);
       }

  init_audio();

  return 0;
}


/* Write the coordinates of the cursor to the VGA hardware through the driver */
```

```
void write_coords(const unsigned short xcoord, const unsigned short ycoord) {
  vga_piano_arg_t vla;
  vla.xcoord = xcoord;
  vla.ycoord = ycoord;
  if (ioctl(vga_piano_fd, VGA_PIANO_WRITE_COORD, &vla)) {
    perror("ioctl(VGA_PIANO_WRITE_COORD) failed");
    return;
  }
}

/* Write the number of a pressed key to the VGA hardware through the driver*/
void write_keypress(unsigned short keypress) {
  vga_piano_arg_t vla;
  vla.keypress = keypress;
  if (ioctl(vga_piano_fd, VGA_PIANO_WRITE_KEYPRESS, &vla)) {
    perror("ioctl(VGA_PIANO_WRITE_KEYPRESS) failed");
    return;
  }
}

/* Write data to the VGA hardware through the driver*/
void write_data(unsigned short address, unsigned int data) {
	vga_piano_arg_t vla;
	vla.address = address;
	vla.data = data;

	if (ioctl(vga_piano_fd, VGA_PIANO_WRITE_DATA, &vla)) {
		perror("ioctl(VGA_PIANO_WRITE_DATA) failed");
		return;
	}
}
```

LeapSender.java

```
package laptop;
/******************************************************************************\
* Copyright (C) 2012-2013 Leap Motion, Inc. All rights reserved.               *
* Leap Motion proprietary and confidential. Not for distribution.              *
* Use subject to the terms of the Leap Motion SDK Agreement available at       *
* https://developer.leapmotion.com/sdk_agreement, or another agreement         *
* between Leap Motion and you, your company or other organization.             *
\******************************************************************************/

import java.io.IOException;
import java.lang.Math;
import java.util.concurrent.TimeUnit;

import com.leapmotion.leap.*;
import com.leapmotion.leap.Gesture.State;
import java.nio.ByteBuffer; // for converting int array to byte array
import java.nio.IntBuffer;

// Networking imports
import java.net.DatagramSocket;
import java.net.DatagramPacket;
import java.net.InetAddress;
```

```
class LeapSender {

       final static int SIZE = 3;

    public static void main(String[] args) throws IOException {


        Controller controller = new Controller();
        int[] arrayToSend = new int[SIZE];

        while(true) {
              try {
                            TimeUnit.MILLISECONDS.sleep(30);
                     } catch (InterruptedException e) {
                            // TODO Auto-generated catch bloc
                            e.printStackTrace();
                            break;
                     }
              Frame frame = controller.frame();
              System.out.println("Frame id: " + frame.id()
                     + ", timestamp: " + frame.timestamp()
                     + ", hands: " + frame.hands().count()
                     + ", fingers: " + frame.fingers().count());

              if (!frame.hands().isEmpty()) {
                     // Get the first hand
                     Hand hand = frame.hands().get(0);

                     FingerList fingers = hand.fingers();
                     if(!fingers.isEmpty()) {
                            Finger finger = fingers.get(0);
                            Vector fingerPos = finger.stabilizedTipPosition();

                            int xPosition;
                            int yPosition;
                            int yVelocity;

                            xPosition = (int)fingerPos.getX();
                            yPosition = (int)fingerPos.getY();
                            yVelocity = (int)finger.tipVelocity().getY();

                            arrayToSend[0] = xPosition;
                            arrayToSend[1] = yPosition;
                            arrayToSend[2] = yVelocity;

                            byte[] toSend = intArray2ByteArray(arrayToSend);
                            sendData(toSend);
                     }

                     /*
                      * Code to include later perhaps if we modify our design to
include multiple fingers
                      *

                     int handCount = 1;
                     for(Hand hand: frame.hands()) {
                            // Check if the hand has any fingers
                            System.out.println("Hand " + handCount);
                            System.out.println("Hand Time Visible: " +
hand.timeVisible());
                              FingerList fingers = hand.fingers();
                              if (!fingers.isEmpty()) {
                                     int i = 1;
                                     // Print out the finger positions
                                     for (Finger finger : fingers) {
                                            System.out.println("Finger " + i);
                                            Vector tipPos =
finger.stabilizedTipPosition();
                                            System.out.println("X: " + tipPos.getX());
                                            Vector tipVel = finger.tipVelocity();
                                            System.out.println("Y velocity: " +
tipVel.getY());
                                            i++;
                                     }
                              }
                              handCount++;
                       }
                       */
               }
         }
    }

    // takes byte array of x pos, y pos, y velocity
    static void sendData(byte[] data) throws IOException {
              DatagramSocket socket = new DatagramSocket(9876);
              InetAddress IPAddress = InetAddress.getByName("192.168.1.102");
              DatagramPacket packet = new DatagramPacket(data, data.length, IPAddress,
4000);
              socket.send(packet);
              socket.close();
    }

    static byte [] intArray2ByteArray(int[] data){
              ByteBuffer byteBuffer = ByteBuffer.allocate(data.length*4);
              IntBuffer intBuffer = byteBuffer.asIntBuffer();
              intBuffer.put(data);
              byte[] output = byteBuffer.array();
              return output;
    }
}
```

Makefile

```
ifneq (${KERNELRELEASE},)

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

else
```

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

       KERNEL_SOURCE := /usr/src/linux
        PWD := $(shell pwd)

default: module coordrcv

coordrcv: coordrcv.o audio.o queue.o
       gcc -o coordrcv coordrcv.o audio.o queue.o -lpthread

coordrcv.o: coordrcv.c

audio.o: audio.c

queue.o: queue.c

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

clean:
       ${MAKE} -C ${KERNEL_SOURCE} SUBDIRS=${PWD} clean
       ${RM} coordrcv

socfpga.dtb : socfpga.dtb
       dtc -O dtb -o socfpga.dtb socfpga.dts

endif
```

queue.c

```
#include <stdlib.h>
#include <pthread.h>
#include "queue.h"

struct Node {
       void *data;
       struct Node *next;
};

typedef struct Node Node;

struct Queue {
       Node *head;
       Node *tail;
```

```
	pthread_mutex_t queue_mutex;
};

typedef struct Queue Queue;

Queue *initQueue() {
	Queue *queue = malloc(sizeof(Queue));
    queue->head = NULL;
    queue->tail = NULL;
    pthread_mutex_init(&queue->queue_mutex, NULL);
    return queue;
}

void destroyQueue(Queue *queue) {
	Node *node = queue->head;

	while (node != NULL) {
		Node *next = node->next;
		free(node);
		node = next;
	}

	pthread_mutex_destroy(&queue->queue_mutex);
	free(queue);
}

void push(Queue *queue, void *data) {
	pthread_mutex_lock(&queue->queue_mutex);
	Node *node = malloc(sizeof(Node));
	node->data = data;
	node->next = NULL;

	if (queue->head == NULL) {
		queue->head = node;
	}
	else {
		queue->tail->next = node;
	}

	queue->tail = node;
	pthread_mutex_unlock(&queue->queue_mutex);
}

void *pop(Queue *queue) {
	if (queue->head == NULL) {
		return NULL;
	}

	Node *node = queue->head;
	void *data = node->data;

	if (queue->tail == node) { // node is only node in queue, acquire tail lock
		pthread_mutex_lock(&queue->queue_mutex);

		if (node->next == NULL) { // now that we have lock, check if it is still
only node
			queue->tail = NULL;
			queue->head = NULL;
		}
		else {
```

```
                    queue->head = node->next;
                }

                pthread_mutex_unlock(&queue->queue_mutex);
        }
        else {
                queue->head = node->next;
        }

        free(node);
        return data;
}
```

queue.h

```
typedef struct Queue queue;


queue *initQueue();
void destroyQueue(queue *q);
void push(queue *q, void *data);
void *pop(queue *q);
```

socfpga.dts

```
/*
 *  Copyright (C) 2012 Altera Corporation <www.altera.com>
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program.  If not, see <http://www.gnu.org/licenses/>.
 *
 * dtc -O dtb -o socfpga.dtb socfpga.dts
 */
```

```
/dts-v1/;
/include/ "socfpga.dtsi"

/ {
	model = "Altera SOCFPGA Cyclone V";
	compatible = "altr,socfpga-cyclone5", "altr,socfpga";

	chosen {
		bootargs = "console=ttyS0,57600";
	};

	memory {
		name = "memory";
		device_type = "memory";
		reg = <0x0 0x40000000>; /* 1 GB */
	};

	aliases {
		/* this allow the ethaddr uboot environmnet variable contents
		 * to be added to the gmac1 device tree blob.
		 */
		ethernet0 = &gmac1;
	};

	soc {
		clkmgr@ffd04000 {
			clocks {
				osc1 {
					clock-frequency = <25000000>;
				};
			};
		};

		dcan0: d_can@ffc00000 {
			status = "disabled";
		};

		dcan1: d_can@ffc10000 {
			status = "disabled";
		};
		dwmmc0@ff704000 {
			num-slots = <1>;
			supports-highspeed;
			broken-cd;
			altr,dw-mshc-ciu-div = <4>;
			altr,dw-mshc-sdr-timing = <0 3>;

			slot@0 {
				reg = <0>;
				bus-width = <4>;
			};
		};

		ethernet@ff700000 {
			status = "disabled";
		};

		ethernet@ff702000 {
			phy-mode = "rgmii";
			phy-addr = <0xffffffff>; /* probe for phy addr */
		};

		i2c1: i2c@ffc05000 {
			status = "disabled";
		};

		i2c2: i2c@ffc06000 {
			status = "disabled";
		};

		i2c3: i2c@ffc07000 {
			status = "disabled";
		};

		qspi: spi@ff705000 {
				compatible = "cadence,qspi";
				#address-cells = <1>;
				#size-cells = <0>;
				reg = <0xff705000 0x1000>,
					<0xffa00000 0x1000>;
				interrupts = <0 151 4>;
				master-ref-clk = <400000000>;
```

```
                        ext-decoder = <0>;  /* external decoder */
                        num-chipselect = <4>;
                        fifo-depth = <128>;
                        bus-num = <2>;

                        flash0: n25q00@0 {
                                #address-cells = <1>;
                                #size-cells = <1>;
                                compatible = "n25q00";
                                reg = <0>;    /* chip select */
                                spi-max-frequency = <100000000>;
                                page-size = <256>;
                                block-size = <16>; /* 2^16, 64KB */
                                quad = <1>;      /* 1-support quad */
                                tshsl-ns = <200>;
                                tsd2d-ns = <255>;
                                tchsh-ns = <20>;
                                tslch-ns = <20>;

                                partition@0 {
                                        /* 8MB for raw data. */
                                        label = "Flash 0 Raw Data";
                                        reg = <0x0 0x800000>;
                                };
                                partition@800000 {
                                        /* 8MB for jffs2 data. */
                                        label = "Flash 0 jffs2 Filesystem";
                                        reg = <0x800000 0x800000>;
                                };
                        };

                };

        sysmgr@ffd08000 {
                cpu1-start-addr = <0xffd080c4>;
        };

        timer0@ffc08000 {
                clock-frequency = <100000000>;
        };

        timer1@ffc09000 {
                clock-frequency = <100000000>;
        };

        timer2@ffd00000 {
                clock-frequency = <25000000>;
        };

        timer3@ffd01000 {
                clock-frequency = <25000000>;
        };

        serial0@ffc02000 {
                clock-frequency = <100000000>;
        };

        serial1@ffc03000 {
                clock-frequency = <100000000>;
        };

        usb0: usb@ffb00000 {
                status = "disabled";
        };

        usb1: usb@ffb40000 {
                ulpi-ddr = <0>;
        };

        i2c0: i2c@ffc04000 {
                speed-mode = <0>;
        };

        leds {
                compatible = "gpio-leds";
                hps0 {
                        label = "hps_led0";
                        gpios = <&gpio1 15 1>;
                };

                hps1 {
                        label = "hps_led1";
                        gpios = <&gpio1 14 1>;
                };
```

```
                        hps2 {
                                label = "hps_led2";
                                gpios = <&gpio1 13 1>;
                        };

                        hps3 {
                                label = "hps_led3";
                                gpios = <&gpio1 12 1>;
                        };
                };

                lightweight_bridge: bridge@0xff200000 {
                        #address-cells = <1>;
                        #size-cells = <1>;
                        ranges = < 0x0 0xff200000 0x200000 >;

                        compatible = "simple-bus";

                        vga_piano: vga_piano@0 {
                                 compatible = "altr,vga_piano";
                                 reg = <0x0 0x40000>;
                        };

                        audio: audio@40000 {
                                compatible = "altr,audio";
                                reg = <0x40000 0x4000>;
                        };
                };
        };
};

&i2c0 {
        lcd: lcd@28 {
                compatible = "newhaven,nhd-0216k3z-nsw-bbw";
                reg = <0x28>;
                height = <2>;
                width = <16>;
                brightness = <8>;
        };

        eeprom@51 {
              compatible = "atmel,24c32";
              reg = <0x51>;
              pagesize = <32>;
       };

       rtc@68 {
              compatible = "dallas,ds1339";
              reg = <0x68>;
       };
};
```

vga_piano.c

```
/*
 * Device driver for the VGA Piano Emulator
 *
 * A Platform device implemented using the misc subsystem
 *
 * Base code courtesy of:
 * 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_piano.ko
 *
 * Check code style with
 * checkpatch.pl --file --no-tree vga_piano.c
 */

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

#define DRIVER_NAME "vga_piano"
#define MAX_MEM_ADDRESS 4095

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

//writes coordinates to the VGA hardware
static void write_coord(u16 xcoord, u16 ycoord)
{
	unsigned short coords[2];
	coords[0] = xcoord;
	coords[1] = ycoord;
	iowrite32(*(int *)coords, dev.virtbase+MAX_MEM_ADDRESS + 1);
}

//writes data to the VGA hardware
static void write_data(u16 address, u32 data) {
	iowrite32(data, dev.virtbase+address);
}

//writes the key that was pressed to the VGA hardware
static void write_keypress(u32 keypress) {
	iowrite32( keypress, dev.virtbase + MAX_MEM_ADDRESS + 5);
}

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

	if (copy_from_user(&vla, (vga_piano_arg_t *) arg,
			   sizeof(vga_piano_arg_t)))
		return -EACCES;
	switch (cmd) {
	case VGA_PIANO_WRITE_DATA:
		if (vla.address > MAX_MEM_ADDRESS - 3) {
			return -EINVAL;
		}
		write_data(vla.address, vla.data);
		break;
	case VGA_PIANO_WRITE_COORD:
		if (vla.xcoord >= 640 || vla.ycoord >= 480)
			return -EINVAL;
		write_coord(vla.xcoord, vla.ycoord);
		break;
	case VGA_PIANO_WRITE_KEYPRESS:
		if (vla.keypress > 15) {
			return -EINVAL;
		}
		write_keypress(vla.keypress);
		break;
	default:
		return -EINVAL;
	}
	return 0;
}
```

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

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

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

	/* Register ourselves as a misc device: creates /dev/vga_piano */
	ret = misc_register(&vga_piano_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;
	}

	return 0;

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

/* Clean-up code: release resources */
static int vga_piano_remove(struct platform_device *pdev)
{
	iounmap(dev.virtbase);
	release_mem_region(dev.res.start, resource_size(&dev.res));
```

```
        misc_deregister(&vga_piano_misc_device);
        return 0;
}

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

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

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

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

module_init(vga_piano_init);
module_exit(vga_piano_exit);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Team Leap-Motion: Patrice Liang, Matthew Patey, Vanshil Shah, Kevin
Walters; Columbia University");
MODULE_DESCRIPTION("VGA Piano Emulator");
```

vga_piano.h

```
#ifndef _VGA_PIANO_H
#define _VGA_PIANO_H


#include <linux/ioctl.h>
```

```
//holds VGA piano arguments and data
typedef struct {
  unsigned int data;
  unsigned short xcoord, ycoord, keypress, address;
} vga_piano_arg_t;

#define VGA_PIANO_MAGIC 'q'

/* ioctls and their arguments */
#define VGA_PIANO_WRITE_DATA _IOW(VGA_PIANO_MAGIC, 1, vga_piano_arg_t *)
#define VGA_PIANO_WRITE_COORD _IOW(VGA_PIANO_MAGIC, 2, vga_piano_arg_t *)
#define VGA_PIANO_WRITE_KEYPRESS _IOW(VGA_PIANO_MAGIC, 3, vga_piano_arg_t *)

#endif
```

Hardware

audio_codec.sv

```
/*
 * Based on code written by Howard Mao
 * http://zhehaomao.com/blog/fpga/2014/01/15/sockit-8.html
 */

module audio_codec (
    input  clk,
    input  reset,
    output [1:0]  sample_end,
    output [1:0]  sample_req,
    input  [15:0] audio_output,
    output [15:0] audio_input,
    // 1 - left, 0 - right
    input  [1:0] channel_sel,

    output AUD_ADCLRCK,
    input AUD_ADCDAT,
    output AUD_DACLRCK,
    output AUD_DACDAT,
    output AUD_BCLK
);

reg [7:0] lrck_divider;
reg [1:0] bclk_divider;

reg [15:0] shift_out;
reg [15:0] shift_temp;
reg [15:0] shift_in;
```

```
wire lrck = !lrck_divider[7];

assign AUD_ADCLRCK = lrck;
assign AUD_DACLRCK = lrck;
assign AUD_BCLK = bclk_divider[1];
assign AUD_DACDAT = shift_out[15];

always @(posedge clk) begin
    if (reset) begin
        lrck_divider <= 8'hff;
        bclk_divider <= 2'b11;
    end else begin
        lrck_divider <= lrck_divider + 1'b1;
        bclk_divider <= bclk_divider + 1'b1;
    end
end

assign sample_end[1] = (lrck_divider == 8'h40);
assign sample_end[0] = (lrck_divider == 8'hc0);
assign audio_input = shift_in;
assign sample_req[1] = (lrck_divider == 8'hfe);
assign sample_req[0] = (lrck_divider == 8'h7e);

wire clr_lrck = (lrck_divider == 8'h7f);
wire set_lrck = (lrck_divider == 8'hff);
// high right after bclk is set
wire set_bclk = (bclk_divider == 2'b10 && !lrck_divider[6]);
// high right before bclk is cleared
wire clr_bclk = (bclk_divider == 2'b11 && !lrck_divider[6]);

always @(posedge clk) begin
    if (reset) begin
        shift_out <= 16'h0;
        shift_in <= 16'h0;
        shift_in <= 16'h0;
    end else if (set_lrck || clr_lrck) begin
        // check if current channel is selected
        if (channel_sel[set_lrck]) begin
            shift_out <= audio_output;
            shift_temp <= audio_output;
            shift_in <= 16'h0;
        // repeat the sample from the other channel if not
        end else shift_out <= shift_temp;
    end else if (set_bclk == 1) begin
        // only read in if channel is selected
        if (channel_sel[lrck])
            shift_in <= {shift_in[14:0], AUD_ADCDAT};
    end else if (clr_bclk == 1) begin
        shift_out <= {shift_out[14:0], 1'b0};
    end
end

endmodule
```

audio_interface.sv

```
/*
 * playback portion based on code by Howard Mao
 * http://zhehaomao.com/blog/fpga/2014/01/15/sockit-8.html
 */

module audio_interface (
        input audio_clk,
    input  sample_req,
    output [15:0] audio_output,

        //Inputs from the HPS
        input hps_clk,
        input reset,
        input chipselect,
        input [11:0] address,
        input [15:0] data,
        input write,
        input read,

        output [15:0] readdata,
        output [3:0] LEDS
);

//2 buffer audio memory
reg [15:0] wavData0 [1023:0];
reg [15:0] wavData1 [1023:0];

logic writeBuffer; // the buffer currently being written to
logic playBuffer; // the buffer currently being played from

logic ready0; // flag high when buffer 0 ready to be played
logic ready1; // flag high when buffer 1 ready to be played

// represent events of buffer becoming ready or no longer being ready, used to update
ready flags
logic enterReady0;
logic enterReady1;
logic exitReady0;
logic exitReady1;

reg [10:0] index = 10'd0; // current position in playback
reg [15:0] dat; // assigned to audio data out

assign audio_output = dat;
assign LEDS[0] = ready0;
assign LEDS[1] = ready1;
assign LEDS[2] = index[10];
assign LEDS[3] = '1;

always_ff @(posedge hps_clk or posedge reset) begin
       if (reset) begin
              ready0 <= 0;
              ready1 <= 0;
       end
       else begin
              // ready flag logic: on event flag, update value accordingly
              // otherwise, value is unchanged
              if (enterReady0)
                     ready0 <= 1;
```

```
            else if (exitReady0)
                  ready0 <= 0;

            if (enterReady1)
                  ready1 <= 1;
            else if (exitReady1)
                  ready1 <= 0;
      end
end

// hps reads and writes
always_ff @(posedge hps_clk or posedge reset) begin
      if (reset) begin
            writeBuffer <= 0;
            enterReady0 <= 0;
            enterReady1 <= 0;
      end
      else if (chipselect && read) begin
            if (address == 11'd1025) begin // only one valid read address, used for
buffer status
                  // return bitwise or of ready flags
                  if(!ready1 && !ready0)
                        readdata <= 16'd0;
                  else if (ready0 && !ready1)
                        readdata <= 16'd1;
                  else if(!ready0 && ready1)
                        readdata <= 16'd2;
                  else if(ready1 && ready0)
                        readdata <= 16'd3;
                  //readdata <= !ready0 || !ready1;
            end
            enterReady0 <= 0;
            enterReady1 <= 0;
      end
      else begin
            if(writeBuffer && write && chipselect) begin // currently writing to
buffer 1
                  if(address == 11'd1024) begin // play signal, sets ready flag for
currently writing buffer
                        enterReady1 <= 1; // set event flag to enter ready 1 state
                        writeBuffer <= !writeBuffer; // switch write buffer to be
buffer 0
                  end else begin
                        wavData1[address] <= data; // write data into buffer 1
                        enterReady1 <= 0;
                  end
                  enterReady0 <= 0;
            end else if (!writeBuffer && write && chipselect) begin // writing to
buffer 0
                  if(address == 11'd1024) begin // play signal, sets ready flag for
currently writing buffer
                        enterReady0 <= 1; // set event flag to enter ready 0 state
                        writeBuffer <= !writeBuffer; // switch write buffer to be
buffer 1
                  end else begin
                        wavData0[address] <= data;
                        enterReady0 <= 0;
                  end
                  enterReady1 <= 0;
            end else begin
```

```
                    enterReady0 <= 0;
                    enterReady1 <= 0;
            end
      end
end

// audio playback
always_ff @(posedge audio_clk or posedge reset) begin
      if (reset) begin
            playBuffer <= 0;
            exitReady0 <= 0;
            exitReady1 <= 0;
      end else begin
            if (sample_req && playBuffer && ready1) begin // playing from buffer 1
and it is ready
                    dat <= wavData1[index];
                    if (index == 10'd1023) begin // reached end of buffer
                          exitReady1 <= 1; // set flag to exit ready 1 state
                          playBuffer <= 0; // set playbuffer to buffer 0
                          index <= 10'h0;
                    end else begin
                          exitReady1 <= 0;
                          index <= index + 1'b1;
                    end
                    exitReady0 <= 0;
            end else if (sample_req && !playBuffer && ready0) begin // playing from
buffer 0 and it is ready
                    dat <= wavData0[index];
                    if (index == 10'd1023) begin // reached end of buffer
                          exitReady0 <= 1; // set flag to exit ready 0 state
                          playBuffer <= 1; // set playbuffer to buffer 1
                          index <= 10'h0;
                    end else begin
                          exitReady0 <= 0;
                          index <= index + 1'b1;
                    end
                    exitReady1 <= 0;
            end else if(sample_req) begin // if the next buffer to play is not ready
send 0's
                    dat <= 16'h0;
                    exitReady0 <= 0;
                    exitReady1 <= 0;
            end else begin // don't send anything if the next sample has not been
requested
                    exitReady0 <= 0;
                    exitReady1 <= 0;
            end
      end
end


endmodule
```

clock_pll.v

```
// megafunction wizard: %Altera PLL v13.1%
// GENERATION: XML
// clock_pll.v

// Generated using ACDS version 13.1.1 166 at 2014.04.24.17:53:21

`timescale 1 ps / 1 ps
module clock_pll_1 (
             input  wire  refclk,   //  refclk.clk
             input  wire  rst,      //   reset.reset
             output wire  outclk_0, // outclk0.clk
             output wire  outclk_1  // outclk1.clk
      );

      clock_pll_0002 clock_pll_inst (
             .refclk   (refclk),   //  refclk.clk
             .rst      (rst),      //   reset.reset
             .outclk_0 (outclk_0), // outclk0.clk
             .outclk_1 (outclk_1), // outclk1.clk
             .locked   ()          // (terminated)
      );

endmodule
// Retrieval info: <?xml version="1.0"?>
//<!--
//      Generated by Altera MegaWizard Launcher Utility version 1.0
//      ************************************************************
//      THIS IS A WIZARD-GENERATED FILE. DO NOT EDIT THIS FILE!
//      ************************************************************
//      Copyright (C) 1991-2014 Altera Corporation
//      Any megafunction design, and related net list (encrypted or decrypted),
//      support information, device programming or simulation file, and any other
//      associated documentation or information provided by Altera or a partner
//      under Altera's Megafunction Partnership Program may be used only to
//      program PLD devices (but not masked PLD devices) from Altera.  Any other
//      use of such megafunction design, net list, support information, device
//      programming or simulation file, or any other related documentation or
//      information is prohibited for any other purpose, including, but not
//      limited to modification, reverse engineering, de-compiling, or use with
//      any other silicon devices, unless such use is explicitly licensed under
//      a separate agreement with Altera or a megafunction partner.  Title to
//      the intellectual property, including patents, copyrights, trademarks,
//      trade secrets, or maskworks, embodied in any such megafunction design,
//      net list, support information, device programming or simulation file, or
//      any other related documentation or information provided by Altera or a
//      megafunction partner, remains with Altera, the megafunction partner, or
//      their respective licensors.  No other licenses, including any licenses
//      needed under any third party's intellectual property, are provided herein.
//-->
// Retrieval info: <instance entity-name="altera_pll" version="13.1" >
// Retrieval info:  <generic name="debug_print_output" value="false" />
// Retrieval info:  <generic name="debug_use_rbc_taf_method" value="false" />
// Retrieval info:  <generic name="device_family" value="Cyclone V" />
// Retrieval info:  <generic name="device" value="Unknown" />
// Retrieval info:  <generic name="gui_device_speed_grade" value="2" />
// Retrieval info:  <generic name="gui_pll_mode" value="Integer-N PLL" />
// Retrieval info:  <generic name="gui_reference_clock_frequency" value="50.0" />
// Retrieval info:  <generic name="gui_channel_spacing" value="0.0" />
// Retrieval info:  <generic name="gui_operation_mode" value="direct" />
// Retrieval info:  <generic name="gui_feedback_clock" value="Global Clock" />
// Retrieval info:  <generic name="gui_fractional_cout" value="32" />
// Retrieval info:  <generic name="gui_dsm_out_sel" value="1st_order" />
// Retrieval info:  <generic name="gui_use_locked" value="false" />
// Retrieval info:  <generic name="gui_en_adv_params" value="false" />
// Retrieval info:  <generic name="gui_number_of_clocks" value="2" />
// Retrieval info:  <generic name="gui_multiply_factor" value="1" />
// Retrieval info:  <generic name="gui_frac_multiply_factor" value="1" />
// Retrieval info:  <generic name="gui_divide_factor_n" value="1" />
// Retrieval info:  <generic name="gui_cascade_counter0" value="false" />
// Retrieval info:  <generic name="gui_output_clock_frequency0" value="11.2896" />
// Retrieval info:  <generic name="gui_divide_factor_c0" value="1" />
// Retrieval info:  <generic name="gui_actual_output_clock_frequency0" value="0 MHz"
/>
// Retrieval info:  <generic name="gui_ps_units0" value="ps" />
// Retrieval info:  <generic name="gui_phase_shift0" value="0" />
// Retrieval info:  <generic name="gui_phase_shift_deg0" value="0.0" />
// Retrieval info:  <generic name="gui_actual_phase_shift0" value="0" />
// Retrieval info:  <generic name="gui_duty_cycle0" value="50" />
// Retrieval info:  <generic name="gui_cascade_counter1" value="false" />
// Retrieval info:  <generic name="gui_output_clock_frequency1" value="50.0" />
// Retrieval info:  <generic name="gui_divide_factor_c1" value="1" />
// Retrieval info:  <generic name="gui_actual_output_clock_frequency1"
value="49.772727 MHz" />
// Retrieval info:  <generic name="gui_ps_units1" value="ps" />
// Retrieval info:  <generic name="gui_phase_shift1" value="0" />
// Retrieval info:  <generic name="gui_phase_shift_deg1" value="0.0" />
// Retrieval info:  <generic name="gui_actual_phase_shift1" value="0" />
// Retrieval info:  <generic name="gui_duty_cycle1" value="50" />
// Retrieval info:  <generic name="gui_cascade_counter2" value="false" />
// Retrieval info:  <generic name="gui_output_clock_frequency2" value="100.0" />
// Retrieval info:  <generic name="gui_divide_factor_c2" value="1" />
// Retrieval info:  <generic name="gui_actual_output_clock_frequency2" value="0 MHz"
/>
// Retrieval info:  <generic name="gui_ps_units2" value="ps" />
// Retrieval info:  <generic name="gui_phase_shift2" value="0" />
// Retrieval info:  <generic name="gui_phase_shift_deg2" value="0.0" />
// Retrieval info:  <generic name="gui_actual_phase_shift2" value="0" />
// Retrieval info:  <generic name="gui_duty_cycle2" value="50" />
// Retrieval info:  <generic name="gui_cascade_counter3" value="false" />
// Retrieval info:  <generic name="gui_output_clock_frequency3" value="100.0" />
// Retrieval info:  <generic name="gui_divide_factor_c3" value="1" />
// Retrieval info:  <generic name="gui_actual_output_clock_frequency3" value="0 MHz"
/>
// Retrieval info:  <generic name="gui_ps_units3" value="ps" />
// Retrieval info:  <generic name="gui_phase_shift3" value="0" />
// Retrieval info:  <generic name="gui_phase_shift_deg3" value="0.0" />
// Retrieval info:  <generic name="gui_actual_phase_shift3" value="0" />
// Retrieval info:  <generic name="gui_duty_cycle3" value="50" />
// Retrieval info:  <generic name="gui_cascade_counter4" value="false" />
// Retrieval info:  <generic name="gui_output_clock_frequency4" value="100.0" />
// Retrieval info:  <generic name="gui_divide_factor_c4" value="1" />
// Retrieval info:  <generic name="gui_actual_output_clock_frequency4" value="0 MHz"
/>
// Retrieval info:  <generic name="gui_ps_units4" value="ps" />
// Retrieval info:  <generic name="gui_phase_shift4" value="0" />
// Retrieval info:  <generic name="gui_phase_shift_deg4" value="0.0" />
// Retrieval info:  <generic name="gui_actual_phase_shift4" value="0" />
// Retrieval info:  <generic name="gui_duty_cycle4" value="50" />
// Retrieval info:  <generic name="gui_cascade_counter5" value="false" />
// Retrieval info:  <generic name="gui_output_clock_frequency5" value="100.0" />
// Retrieval info:  <generic name="gui_divide_factor_c5" value="1" />
// Retrieval info:  <generic name="gui_actual_output_clock_frequency5" value="0 MHz"
/>
// Retrieval info:  <generic name="gui_ps_units5" value="ps" />
// Retrieval info:  <generic name="gui_phase_shift5" value="0" />
// Retrieval info:  <generic name="gui_phase_shift_deg5" value="0.0" />
// Retrieval info:  <generic name="gui_actual_phase_shift5" value="0" />
// Retrieval info:  <generic name="gui_duty_cycle5" value="50" />
// Retrieval info:  <generic name="gui_cascade_counter6" value="false" />
// Retrieval info:  <generic name="gui_output_clock_frequency6" value="100.0" />
// Retrieval info:  <generic name="gui_divide_factor_c6" value="1" />
// Retrieval info:  <generic name="gui_actual_output_clock_frequency6" value="0 MHz"
/>
// Retrieval info:  <generic name="gui_ps_units6" value="ps" />
// Retrieval info:  <generic name="gui_phase_shift6" value="0" />
// Retrieval info:  <generic name="gui_phase_shift_deg6" value="0.0" />
// Retrieval info:  <generic name="gui_actual_phase_shift6" value="0" />
// Retrieval info:  <generic name="gui_duty_cycle6" value="50" />
// Retrieval info:  <generic name="gui_cascade_counter7" value="false" />
// Retrieval info:  <generic name="gui_output_clock_frequency7" value="100.0" />
// Retrieval info:  <generic name="gui_divide_factor_c7" value="1" />
// Retrieval info:  <generic name="gui_actual_output_clock_frequency7" value="0 MHz"
/>
// Retrieval info:  <generic name="gui_ps_units7" value="ps" />
// Retrieval info:  <generic name="gui_phase_shift7" value="0" />
// Retrieval info:  <generic name="gui_phase_shift_deg7" value="0.0" />
// Retrieval info:  <generic name="gui_actual_phase_shift7" value="0" />
// Retrieval info:  <generic name="gui_duty_cycle7" value="50" />
// Retrieval info:  <generic name="gui_cascade_counter8" value="false" />
// Retrieval info:  <generic name="gui_output_clock_frequency8" value="100.0" />
// Retrieval info:  <generic name="gui_divide_factor_c8" value="1" />
// Retrieval info:  <generic name="gui_actual_output_clock_frequency8" value="0 MHz"
/>
// Retrieval info:  <generic name="gui_ps_units8" value="ps" />
// Retrieval info:  <generic name="gui_phase_shift8" value="0" />
// Retrieval info:  <generic name="gui_phase_shift_deg8" value="0.0" />
// Retrieval info:  <generic name="gui_actual_phase_shift8" value="0" />
// Retrieval info:  <generic name="gui_duty_cycle8" value="50" />
// Retrieval info:  <generic name="gui_cascade_counter9" value="false" />
// Retrieval info:  <generic name="gui_output_clock_frequency9" value="100.0" />
// Retrieval info:  <generic name="gui_divide_factor_c9" value="1" />
// Retrieval info:  <generic name="gui_actual_output_clock_frequency9" value="0 MHz"
/>
// Retrieval info:  <generic name="gui_ps_units9" value="ps" />
// Retrieval info:  <generic name="gui_phase_shift9" value="0" />
// Retrieval info:  <generic name="gui_phase_shift_deg9" value="0.0" />
// Retrieval info:  <generic name="gui_actual_phase_shift9" value="0" />
// Retrieval info:  <generic name="gui_duty_cycle9" value="50" />
// Retrieval info:  <generic name="gui_cascade_counter10" value="false" />
// Retrieval info:  <generic name="gui_output_clock_frequency10" value="100.0" />
// Retrieval info:  <generic name="gui_divide_factor_c10" value="1" />
// Retrieval info:  <generic name="gui_actual_output_clock_frequency10" value="0 MHz"
/>
// Retrieval info:  <generic name="gui_ps_units10" value="ps" />
// Retrieval info:  <generic name="gui_phase_shift10" value="0" />
// Retrieval info:  <generic name="gui_phase_shift_deg10" value="0.0" />
// Retrieval info:  <generic name="gui_actual_phase_shift10" value="0" />
// Retrieval info:  <generic name="gui_duty_cycle10" value="50" />
// Retrieval info:  <generic name="gui_cascade_counter11" value="false" />
// Retrieval info:  <generic name="gui_output_clock_frequency11" value="100.0" />
// Retrieval info:  <generic name="gui_divide_factor_c11" value="1" />
// Retrieval info:  <generic name="gui_actual_output_clock_frequency11" value="0 MHz"
/>
// Retrieval info:  <generic name="gui_ps_units11" value="ps" />
// Retrieval info:  <generic name="gui_phase_shift11" value="0" />
// Retrieval info:  <generic name="gui_phase_shift_deg11" value="0.0" />
// Retrieval info:  <generic name="gui_actual_phase_shift11" value="0" />
// Retrieval info:  <generic name="gui_duty_cycle11" value="50" />
// Retrieval info:  <generic name="gui_cascade_counter12" value="false" />
// Retrieval info:  <generic name="gui_output_clock_frequency12" value="100.0" />
// Retrieval info:  <generic name="gui_divide_factor_c12" value="1" />
// Retrieval info:  <generic name="gui_actual_output_clock_frequency12" value="0 MHz"
/>
// Retrieval info:  <generic name="gui_ps_units12" value="ps" />
// Retrieval info:  <generic name="gui_phase_shift12" value="0" />
// Retrieval info:  <generic name="gui_phase_shift_deg12" value="0.0" />
// Retrieval info:  <generic name="gui_actual_phase_shift12" value="0" />
// Retrieval info:  <generic name="gui_duty_cycle12" value="50" />
// Retrieval info:  <generic name="gui_cascade_counter13" value="false" />
// Retrieval info:  <generic name="gui_output_clock_frequency13" value="100.0" />
// Retrieval info:  <generic name="gui_divide_factor_c13" value="1" />
// Retrieval info:  <generic name="gui_actual_output_clock_frequency13" value="0 MHz"
/>
// Retrieval info:  <generic name="gui_ps_units13" value="ps" />
// Retrieval info:  <generic name="gui_phase_shift13" value="0" />
// Retrieval info:  <generic name="gui_phase_shift_deg13" value="0.0" />
// Retrieval info:  <generic name="gui_actual_phase_shift13" value="0" />
// Retrieval info:  <generic name="gui_duty_cycle13" value="50" />
// Retrieval info:  <generic name="gui_cascade_counter14" value="false" />
// Retrieval info:  <generic name="gui_output_clock_frequency14" value="100.0" />
// Retrieval info:  <generic name="gui_divide_factor_c14" value="1" />
// Retrieval info:  <generic name="gui_actual_output_clock_frequency14" value="0 MHz"
/>
// Retrieval info:  <generic name="gui_ps_units14" value="ps" />
// Retrieval info:  <generic name="gui_phase_shift14" value="0" />
// Retrieval info:  <generic name="gui_phase_shift_deg14" value="0.0" />
// Retrieval info:  <generic name="gui_actual_phase_shift14" value="0" />
// Retrieval info:  <generic name="gui_duty_cycle14" value="50" />
// Retrieval info:  <generic name="gui_cascade_counter15" value="false" />
// Retrieval info:  <generic name="gui_output_clock_frequency15" value="100.0" />
// Retrieval info:  <generic name="gui_divide_factor_c15" value="1" />
// Retrieval info:  <generic name="gui_actual_output_clock_frequency15" value="0 MHz"
/>
// Retrieval info:  <generic name="gui_ps_units15" value="ps" />
// Retrieval info:  <generic name="gui_phase_shift15" value="0" />
// Retrieval info:  <generic name="gui_phase_shift_deg15" value="0.0" />
// Retrieval info:  <generic name="gui_actual_phase_shift15" value="0" />
// Retrieval info:  <generic name="gui_duty_cycle15" value="50" />
// Retrieval info:  <generic name="gui_cascade_counter16" value="false" />
// Retrieval info:  <generic name="gui_output_clock_frequency16" value="100.0" />
// Retrieval info:  <generic name="gui_divide_factor_c16" value="1" />
// Retrieval info:  <generic name="gui_actual_output_clock_frequency16" value="0 MHz"
/>
// Retrieval info:  <generic name="gui_ps_units16" value="ps" />
// Retrieval info:  <generic name="gui_phase_shift16" value="0" />
// Retrieval info:  <generic name="gui_phase_shift_deg16" value="0.0" />
// Retrieval info:  <generic name="gui_actual_phase_shift16" value="0" />
// Retrieval info:  <generic name="gui_duty_cycle16" value="50" />
// Retrieval info:  <generic name="gui_cascade_counter17" value="false" />
// Retrieval info:  <generic name="gui_output_clock_frequency17" value="100.0" />
// Retrieval info:  <generic name="gui_divide_factor_c17" value="1" />
// Retrieval info:  <generic name="gui_actual_output_clock_frequency17" value="0 MHz"
/>
// Retrieval info:  <generic name="gui_ps_units17" value="ps" />
// Retrieval info:  <generic name="gui_phase_shift17" value="0" />
// Retrieval info:  <generic name="gui_phase_shift_deg17" value="0.0" />
// Retrieval info:  <generic name="gui_actual_phase_shift17" value="0" />
// Retrieval info:  <generic name="gui_duty_cycle17" value="50" />
// Retrieval info:  <generic name="gui_pll_auto_reset" value="Off" />
// Retrieval info:  <generic name="gui_pll_bandwidth_preset" value="Auto" />
// Retrieval info:  <generic name="gui_en_reconf" value="false" />
// Retrieval info:  <generic name="gui_en_dps_ports" value="false" />
// Retrieval info:  <generic name="gui_en_phout_ports" value="false" />
// Retrieval info:  <generic name="gui_phout_division" value="1" />
// Retrieval info:  <generic name="gui_en_lvds_ports" value="false" />
// Retrieval info:  <generic name="gui_mif_generate" value="false" />
// Retrieval info:  <generic name="gui_enable_mif_dps" value="false" />
// Retrieval info:  <generic name="gui_dps_cntr" value="C0" />
// Retrieval info:  <generic name="gui_dps_num" value="1" />
// Retrieval info:  <generic name="gui_dps_dir" value="Positive" />
// Retrieval info:  <generic name="gui_refclk_switch" value="false" />
// Retrieval info:  <generic name="gui_refclk1_frequency" value="100.0" />
// Retrieval info:  <generic name="gui_switchover_mode" value="Automatic Switchover"
/>
// Retrieval info:  <generic name="gui_switchover_delay" value="0" />
// Retrieval info:  <generic name="gui_active_clk" value="false" />
// Retrieval info:  <generic name="gui_clk_bad" value="false" />
// Retrieval info:  <generic name="gui_enable_cascade_out" value="false" />
// Retrieval info:  <generic name="gui_cascade_outclk_index" value="0" />
// Retrieval info:  <generic name="gui_enable_cascade_in" value="false" />
// Retrieval info:  <generic name="gui_pll_cascading_mode" value="Create an adjpllin
signal to connect with an upstream PLL" />
// Retrieval info:  <generic name="AUTO_REFCLK_CLOCK_RATE" value="-1" />
// Retrieval info: </instance>
// IPFS_FILES : clock_pll.vo
// RELATED_FILES: clock_pll.v, clock_pll_0002.v
```

i2c_av_config.sv

```
/*
 * Based on code written by Howard Mao
 * http://zhehaomao.com/blog/fpga/2014/01/15/sockit-8.html
 */

module i2c_av_config (
    input clk,
    input reset,

    output i2c_sclk,
    inout  i2c_sdat

 //   output [3:0] status
);

reg [23:0] i2c_data;
reg [15:0] lut_data;
reg [3:0]  lut_index = 4'd0;
```

```
parameter LAST_INDEX = 4'ha;

reg  i2c_start = 1'b0;
wire i2c_done;
wire i2c_ack;

i2c_controller control (
    .clk (clk),
    .i2c_sclk (i2c_sclk),
    .i2c_sdat (i2c_sdat),
    .i2c_data (i2c_data),
    .start (i2c_start),
    .done (i2c_done),
    .ack (i2c_ack)
);

always @(*) begin
    case (lut_index)
        4'h0: lut_data <= 16'h0c10; // power on everything except out
        4'h1: lut_data <= 16'h0017; // left input
        4'h2: lut_data <= 16'h0217; // right input
        4'h3: lut_data <= 16'h0479; // left output
        4'h4: lut_data <= 16'h0679; // right output
        4'h5: lut_data <= 16'h08d4; // analog path
        4'h6: lut_data <= 16'h0a04; // digital path
        4'h7: lut_data <= 16'h0e01; // digital IF
        4'h8: lut_data <= 16'h1020; // sampling rate
        4'h9: lut_data <= 16'h0c00; // power on everything
        4'ha: lut_data <= 16'h1201; // activate
        default: lut_data <= 16'h0000;
    endcase
end

reg [1:0] control_state = 2'b00;

//assign status = lut_index;

always @(posedge clk) begin
    if (reset) begin
        lut_index <= 4'd0;
        i2c_start <= 1'b0;
        control_state <= 2'b00;
    end else begin
        case (control_state)
            2'b00: begin
                i2c_start <= 1'b1;
                i2c_data <= {8'h34, lut_data};
                control_state <= 2'b01;
            end
            2'b01: begin
                i2c_start <= 1'b0;
                control_state <= 2'b10;
            end
            2'b10: if (i2c_done) begin
                if (i2c_ack) begin
                    if (lut_index == LAST_INDEX)
                        control_state <= 2'b11;
                    else begin
                        lut_index <= lut_index + 1'b1;
                        control_state <= 2'b00;
                    end
                end else
                    control_state <= 2'b00;
            end
        endcase
    end
end

endmodule
```

i2c_controller.sv

```
/*
 * Based on code written by Howard Mao
 * http://zhehaomao.com/blog/fpga/2014/01/15/sockit-8.html
 */

module i2c_controller (
    input  clk,

    output i2c_sclk,
    inout  i2c_sdat,

    input  start,
    output done,
    output ack,

    input [23:0] i2c_data
);

reg [23:0] data;

reg [4:0] stage;
reg [6:0] sclk_divider;
reg clock_en = 1'b0;

// don't toggle the clock unless we're sending data
// clock will also be kept high when sending START and STOP symbols
assign i2c_sclk = (!clock_en) || sclk_divider[6];
wire midlow = (sclk_divider == 7'h1f);

reg sdat = 1'b1;
// rely on pull-up resistor to set SDAT high
assign i2c_sdat = (sdat) ? 1'bz : 1'b0;

reg [2:0] acks;

parameter LAST_STAGE = 5'd29;

assign ack = (acks == 3'b000);
assign done = (stage == LAST_STAGE);

always @(posedge clk) begin
    if (start) begin
        sclk_divider <= 7'd0;
        stage <= 5'd0;
        clock_en = 1'b0;
        sdat <= 1'b1;
        acks <= 3'b111;
        data <= i2c_data;
    end else begin
        if (sclk_divider == 7'd127) begin
            sclk_divider <= 7'd0;

            if (stage != LAST_STAGE)
                stage <= stage + 1'b1;

            case (stage)
                // after start
                5'd0:  clock_en <= 1'b1;
                // receive acks
                5'd9:  acks[0] <= i2c_sdat;
                5'd18: acks[1] <= i2c_sdat;
                5'd27: acks[2] <= i2c_sdat;
                // before stop
                5'd28: clock_en <= 1'b0;
            endcase
        end else
            sclk_divider <= sclk_divider + 1'b1;

        if (midlow) begin
            case (stage)
                // start
                5'd0:  sdat <= 1'b0;
                // byte 1
                5'd1:  sdat <= data[23];
                5'd2:  sdat <= data[22];
                5'd3:  sdat <= data[21];
                5'd4:  sdat <= data[20];
                5'd5:  sdat <= data[19];
                5'd6:  sdat <= data[18];
                5'd7:  sdat <= data[17];
                5'd8:  sdat <= data[16];
                // ack 1
                5'd9:  sdat <= 1'b1;
                // byte 2
                5'd10: sdat <= data[15];
                5'd11: sdat <= data[14];
                5'd12: sdat <= data[13];
                5'd13: sdat <= data[12];
                5'd14: sdat <= data[11];
                5'd15: sdat <= data[10];
                5'd16: sdat <= data[9];
                5'd17: sdat <= data[8];
                // ack 2
                5'd18: sdat <= 1'b1;
                // byte 3
                5'd19: sdat <= data[7];
                5'd20: sdat <= data[6];
                5'd21: sdat <= data[5];
                5'd22: sdat <= data[4];
                5'd23: sdat <= data[3];
                5'd24: sdat <= data[2];
                5'd25: sdat <= data[1];
                5'd26: sdat <= data[0];
                // ack 3
                5'd27: sdat <= 1'b1;
                // stop
                5'd28: sdat <= 1'b0;
                5'd29: sdat <= 1'b1;
            endcase
        end
    end
end

endmodule
```

SoCKit_golden_top.v

```
// ============================================================================
// Copyright (c) 2013 by Terasic Technologies Inc.
// ============================================================================
//
// 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
//
// ============================================================================
//Date:  Mon Jul  1 14:21:10 2013
// ============================================================================


//`define ENABLE_DDR3
`define ENABLE_HPS
//`define ENABLE_HSMC

module SoCKit_golden_top(

      ///////// AUD /////////
      input                AUD_ADCDAT,
      inout                AUD_ADCLRCK,
      inout                 AUD_BCLK,
      output                AUD_DACDAT,
      inout                 AUD_DACLRCK,
      output                AUD_I2C_SCLK,
      inout                 AUD_I2C_SDAT,
      output                AUD_MUTE,
      output                AUD_XCK,

`ifdef ENABLE_DDR3
      ///////// DDR3 /////////
      output      [14:0] DDR3_A,
      output      [2:0]  DDR3_BA,
      output             DDR3_CAS_n,
      output             DDR3_CKE,
      output             DDR3_CK_n,
      output             DDR3_CK_p,
      output             DDR3_CS_n,
      output      [3:0]  DDR3_DM,
      inout       [31:0] DDR3_DQ,
      inout       [3:0]  DDR3_DQS_n,
      inout       [3:0]  DDR3_DQS_p,
      output             DDR3_ODT,
      output             DDR3_RAS_n,
      output             DDR3_RESET_n,
      input              DDR3_RZQ,
      output             DDR3_WE_n,
`endif /*ENABLE_DDR3*/

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

`ifdef ENABLE_HPS
      ///////// HPS /////////
      input              HPS_CONV_USB_n,
      output      [14:0] HPS_DDR3_A,
      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       [3:0]  HPS_FLASH_DATA,
      output            HPS_FLASH_DCLK,
      output            HPS_FLASH_NCSO,
      inout             HPS_GSENSOR_INT,
      inout             HPS_I2C_CLK,
      inout             HPS_I2C_SDA,
      inout      [3:0]  HPS_KEY,
      inout             HPS_LCM_BK,
      output            HPS_LCM_D_C,
      output            HPS_LCM_RST_N,
      input             HPS_LCM_SPIM_CLK,
      output            HPS_LCM_SPIM_MOSI,
      output            HPS_LCM_SPIM_SS,
      output     [3:0]  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,
      output            HPS_SPIM_SS,
      input      [3:0]  HPS_SW,
      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,
`endif /*ENABLE_HPS*/

`ifdef ENABLE_HSMC
      ///////// HSMC /////////
      input      [2:1]  HSMC_CLKIN_n,
      input      [2:1]  HSMC_CLKIN_p,
      output     [2:1]  HSMC_CLKOUT_n,
      output     [2:1]  HSMC_CLKOUT_p,
      output            HSMC_CLK_IN0,
      output            HSMC_CLK_OUT0,
      inout      [3:0]  HSMC_D,
      input      [7:0]  HSMC_GXB_RX_p,
      output     [7:0]  HSMC_GXB_TX_p,
      input             HSMC_REF_CLK_p,
      inout      [16:0] HSMC_RX_n,
      inout      [16:0] HSMC_RX_p,
      output            HSMC_SCL,
      inout             HSMC_SDA,
      inout      [16:0] HSMC_TX_n,
      inout      [16:0] HSMC_TX_p,
`endif /*ENABLE_HSMC*/

      ///////// IRDA /////////
      input             IRDA_RXD,

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

      ///////// LED /////////
      output     [3:0]  LED,
      ///////// OSC /////////
      input                OSC_50_B3B,
      input                OSC_50_B4A,
      input                OSC_50_B5B,
      input                OSC_50_B8A,

      ///////// PCIE /////////
      input                PCIE_PERST_n,
      output               PCIE_WAKE_n,

      ///////// RESET /////////
      input                RESET_n,

      ///////// SI5338 /////////
      inout                SI5338_SCL,
      inout                SI5338_SDA,

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

      ///////// TEMP /////////
      output               TEMP_CS_n,
      output               TEMP_DIN,
      input                TEMP_DOUT,
      output               TEMP_SCLK,

      ///////// USB /////////
      input                USB_B2_CLK,
      inout         [7:0]  USB_B2_DATA,
      output               USB_EMPTY,
      output               USB_FULL,
      input                USB_OE_n,
      input                USB_RD_n,
      input                USB_RESET_n,
      inout                USB_SCL,
      inout                USB_SDA,
      input                USB_WR_n,

      ///////// 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


);



//=======================================================
//  REG/WIRE declarations
//=======================================================
wire reset = !KEY[0];
wire main_clk;
wire audio_clk;
wire [1:0] sample_end;
wire [1:0] sample_req;
wire [15:0] audio_output;
wire [15:0] audio_input;

clock_pll_1 pll (
    .refclk (OSC_50_B8A),
    .rst (reset),
    .outclk_0 (audio_clk),
    .outclk_1 (main_clk)
);

i2c_av_config av_config (
    .clk (main_clk),
    .reset (reset),
    .i2c_sclk (AUD_I2C_SCLK),
    .i2c_sdat (AUD_I2C_SDAT),
//    .status (LED)
);

assign AUD_XCK = audio_clk;
assign AUD_MUTE = 1;

audio_codec ac (
    .clk (audio_clk),
    .reset (reset),
    .sample_end (sample_end),
    .sample_req (sample_req),
    .audio_output (audio_output),
    .audio_input (audio_input),
    .channel_sel (2'b10),

    .AUD_ADCLRCK (AUD_ADCLRCK),
    .AUD_ADCDAT (AUD_ADCDAT),
    .AUD_DACLRCK (AUD_DACLRCK),
    .AUD_DACDAT (AUD_DACDAT),
    .AUD_BCLK (AUD_BCLK)
);

/*
audio_effects ae (
    .clk (audio_clk),
    .sample_end (sample_end[1]),
    .sample_req (sample_req[1]),
    .audio_output (audio_output),
    .audio_input  (audio_input),

        .hps_clk (main_clk),
        .address (address),
        .data(writedata),
        .write (write),
        .LEDS(LED)
);
*/

    project u0 (
        .clk_clk                        (OSC_50_B4A),                        //
clk.clk
        .reset_reset_n                  (RESET_n),                   //  reset.reset_
        .memory_mem_a                   (HPS_DDR3_A),                    //
```

```
memory.mem_a
        .memory_mem_ba                  (HPS_DDR3_BA),                  //
.mem_ba
        .memory_mem_ck                  (HPS_DDR3_CK_p),                  //
.mem_ck
        .memory_mem_ck_n                (HPS_DDR3_CK_n),                //
.mem_ck_n
        .memory_mem_cke                 (HPS_DDR3_CKE),                 //
.mem_cke
        .memory_mem_cs_n                (HPS_DDR3_CS_n),                //
.mem_cs_n
        .memory_mem_ras_n               (HPS_DDR3_RAS_n),               //
.mem_ras_n
        .memory_mem_cas_n               (HPS_DDR3_CAS_n),               //
.mem_cas_n
        .memory_mem_we_n                (HPS_DDR3_WE_n),                //
.mem_we_n
        .memory_mem_reset_n             (HPS_DDR3_RESET_n),             //
.mem_reset_n
        .memory_mem_dq                  (HPS_DDR3_DQ),                  //
.mem_dq
        .memory_mem_dqs                 (HPS_DDR3_DQS_p),                 //
.mem_dqs
        .memory_mem_dqs_n               (HPS_DDR3_DQS_n),               //
.mem_dqs_n
        .memory_mem_odt                 (HPS_DDR3_ODT),                 //
.mem_odt
        .memory_mem_dm                  (HPS_DDR3_DM),                  //
.mem_dm
        .memory_oct_rzqin               (HPS_DDR3_RZQ),               //
.oct_rzqin
        .hps_io_hps_io_emac1_inst_TX_CLK (HPS_ENET_GTX_CLK), //
hps_io.hps_io_emac1_inst_TX_CLK
        .hps_io_hps_io_emac1_inst_TXD0   (HPS_ENET_TX_DATA[0]),   //
.hps_io_emac1_inst_TXD0
        .hps_io_hps_io_emac1_inst_TXD1   (HPS_ENET_TX_DATA[1]),   //
.hps_io_emac1_inst_TXD1
        .hps_io_hps_io_emac1_inst_TXD2   (HPS_ENET_TX_DATA[2]),   //
.hps_io_emac1_inst_TXD2
        .hps_io_hps_io_emac1_inst_TXD3   (HPS_ENET_TX_DATA[3]),   //
.hps_io_emac1_inst_TXD3
        .hps_io_hps_io_emac1_inst_RXD0   (HPS_ENET_RX_DATA[0]),   //
.hps_io_emac1_inst_RXD0
        .hps_io_hps_io_emac1_inst_MDIO   (HPS_ENET_MDIO),   //
.hps_io_emac1_inst_MDIO
        .hps_io_hps_io_emac1_inst_MDC    (HPS_ENET_MDC),    //
.hps_io_emac1_inst_MDC
        .hps_io_hps_io_emac1_inst_RX_CTL (HPS_ENET_RX_DV), //
.hps_io_emac1_inst_RX_CTL
        .hps_io_hps_io_emac1_inst_TX_CTL (HPS_ENET_TX_EN), //
.hps_io_emac1_inst_TX_CTL
        .hps_io_hps_io_emac1_inst_RX_CLK (HPS_ENET_RX_CLK), //
.hps_io_emac1_inst_RX_CLK
        .hps_io_hps_io_emac1_inst_RXD1   (HPS_ENET_RX_DATA[1]),   //
.hps_io_emac1_inst_RXD1
        .hps_io_hps_io_emac1_inst_RXD2   (HPS_ENET_RX_DATA[2]),   //
.hps_io_emac1_inst_RXD2
        .hps_io_hps_io_emac1_inst_RXD3   (HPS_ENET_RX_DATA[3]),   //
.hps_io_emac1_inst_RXD3
        .hps_io_hps_io_qspi_inst_IO0     (HPS_FLASH_DATA[0]),     //
```

```
.hps_io_qspi_inst_IO0
        .hps_io_hps_io_qspi_inst_IO1     (HPS_FLASH_DATA[1]),     //
.hps_io_qspi_inst_IO1
        .hps_io_hps_io_qspi_inst_IO2     (HPS_FLASH_DATA[2]),     //
.hps_io_qspi_inst_IO2
        .hps_io_hps_io_qspi_inst_IO3     (HPS_FLASH_DATA[3]),     //
.hps_io_qspi_inst_IO3
        .hps_io_hps_io_qspi_inst_SS0     (HPS_FLASH_NCSO),     //
.hps_io_qspi_inst_SS0
        .hps_io_hps_io_qspi_inst_CLK     (HPS_FLASH_DCLK),     //
.hps_io_qspi_inst_CLK
        .hps_io_hps_io_sdio_inst_CMD     (HPS_SD_CMD),     //
.hps_io_sdio_inst_CMD
        .hps_io_hps_io_sdio_inst_D0      (HPS_SD_DATA[0]),      //
.hps_io_sdio_inst_D0
        .hps_io_hps_io_sdio_inst_D1      (HPS_SD_DATA[1]),      //
.hps_io_sdio_inst_D1
        .hps_io_hps_io_sdio_inst_CLK     (HPS_SD_CLK),     //
.hps_io_sdio_inst_CLK
        .hps_io_hps_io_sdio_inst_D2      (HPS_SD_DATA[2]),      //
.hps_io_sdio_inst_D2
        .hps_io_hps_io_sdio_inst_D3      (HPS_SD_DATA[3]),      //
.hps_io_sdio_inst_D3
        .hps_io_hps_io_usb1_inst_D0      (HPS_USB_DATA[0]),      //
.hps_io_usb1_inst_D0
        .hps_io_hps_io_usb1_inst_D1      (HPS_USB_DATA[1]),      //
.hps_io_usb1_inst_D1
        .hps_io_hps_io_usb1_inst_D2      (HPS_USB_DATA[2]),      //
.hps_io_usb1_inst_D2
        .hps_io_hps_io_usb1_inst_D3      (HPS_USB_DATA[3]),      //
.hps_io_usb1_inst_D3
        .hps_io_hps_io_usb1_inst_D4      (HPS_USB_DATA[4]),      //
.hps_io_usb1_inst_D4
        .hps_io_hps_io_usb1_inst_D5      (HPS_USB_DATA[5]),      //
.hps_io_usb1_inst_D5
        .hps_io_hps_io_usb1_inst_D6      (HPS_USB_DATA[6]),      //
.hps_io_usb1_inst_D6
        .hps_io_hps_io_usb1_inst_D7      (HPS_USB_DATA[7]),      //
.hps_io_usb1_inst_D7
        .hps_io_hps_io_usb1_inst_CLK     (HPS_USB_CLKOUT),     //
.hps_io_usb1_inst_CLK
        .hps_io_hps_io_usb1_inst_STP     (HPS_USB_STP),     //
.hps_io_usb1_inst_STP
        .hps_io_hps_io_usb1_inst_DIR     (HPS_USB_DIR),     //
.hps_io_usb1_inst_DIR
        .hps_io_hps_io_usb1_inst_NXT     (HPS_USB_NXT),     //
.hps_io_usb1_inst_NXT
        .hps_io_hps_io_spim0_inst_CLK    (HPS_SPIM_CLK),    //
.hps_io_spim0_inst_CLK
        .hps_io_hps_io_spim0_inst_MOSI   (HPS_SPIM_MOSI),   //
.hps_io_spim0_inst_MOSI
        .hps_io_hps_io_spim0_inst_MISO   (HPS_SPIM_MISO),   //
.hps_io_spim0_inst_MISO
        .hps_io_hps_io_spim0_inst_SS0    (HPS_SPIM_SS),    //
.hps_io_spim0_inst_SS0
        .hps_io_hps_io_spim1_inst_CLK    (HPS_LCM_SPIM_CLK),    //
.hps_io_spim1_inst_CLK
        .hps_io_hps_io_spim1_inst_MOSI   (HPS_LCM_SPIM_MOSI),   //
.hps_io_spim1_inst_MOSI
        .hps_io_hps_io_spim1_inst_MISO   (HPS_SPIM_MISO),   //
```

```
.hps_io_spim1_inst_MISO
        .hps_io_hps_io_spim1_inst_SS0    (HPS_LCM_SPIM_SS),    //
.hps_io_spim1_inst_SS0
        .hps_io_hps_io_uart0_inst_RX     (HPS_UART_RX),     //
.hps_io_uart0_inst_RX
        .hps_io_hps_io_uart0_inst_TX     (HPS_UART_TX),     //
.hps_io_uart0_inst_TX
        .hps_io_hps_io_i2c1_inst_SDA     (HPS_I2C_SDA),     //
.hps_io_i2c1_inst_SDA
        .hps_io_hps_io_i2c1_inst_SCL     (HPS_I2C_CLK),     //
.hps_io_i2c1_inst_SCL
  .vga_R (VGA_R),
.vga_G (VGA_G),
.vga_B (VGA_B),
.vga_CLK (VGA_CLK),
.vga_HS (VGA_HS),
.vga_VS (VGA_VS),
.vga_BLANK_n (VGA_BLANK_n),
.vga_SYNC_n (VGA_SYNC_n),                        //       .SYNC_n

.audio_LEDS (LED),
.audio_audio_clk(audio_clk),
.audio_audio_output (audio_output),
.audio_sample_req (sample_req[1])
);




//=======================================================
//  Structural coding
//=======================================================




endmodule
```

VGA_Piano.sv

```
/*
 * VGA Piano
 */

module VGA_Piano(input logic        clk,
             input logic    reset,
             input logic [31:0]  writedata,
             input logic    write,
             input          chipselect,
             input logic [15:0]  address,

             output logic [7:0] VGA_R, VGA_G, VGA_B,
             output logic   VGA_CLK, VGA_HS, VGA_VS, VGA_BLANK_n,
             output logic   VGA_SYNC_n);
```

```
// RGBA read from sprite memory, sent to Emulator
logic [7:0] R;
logic [7:0] G;
logic [7:0] B;
logic [7:0] A;

logic [31:0] sprite_data; // data written to sprite memory
logic [4:0] pixelx, pixely; // sprite input x and y pixel address selector
logic [4:0] vga_pixelx, vga_pixely; // output from emulator, requested coordinates to
read from sprite mem

// current position of cursor on the screen
logic [15:0] cursorx;
logic [15:0] cursory;
logic [7:0] keypress; // determines which key is being pressed

initial begin
       cursorx = '0;
       cursory = '0;
       keypress = '0;
end

logic swrite; // input to sprite mem, telling it whether to write

always_comb begin
       sprite_data = writedata;

       // if avalon not writing to device, these are the values sent to sprite inputs
       swrite = 0;
       pixelx = vga_pixelx;
       pixely = vga_pixely;

       if (write && chipselect) begin
              if(address <= 16'h03ff) begin // if address in memory's range, write dat
to sprite mem
                     swrite = 1;

                     // split 1D address into 2D sprite address
                     pixelx = address[4:0];
                     pixely = address[9:5];
              end
       end
end

always_ff @(posedge clk) begin
       if (write && chipselect) begin
              if (address == 16'h0400) begin // cursor position write address
                            cursorx = writedata[15:0];
                            cursory = writedata[31:16];
              end else if (address[15:0] == 16'h0401) begin // keypress write address
                            keypress = writedata[7:0];
              end
       end
end




sprite_memory mem(.clk(clk), .reset(reset), .writedata(sprite_data), .x(pixelx),
.y(pixely),
              .write(swrite), .R(R), .G(G), .B(B), .A(A)
              );

VGA_Piano_Emulator piano_emulator(.clk50(clk), .reset(reset), .keypress(keypress),
.xcoord(cursorx), .ycoord(cursory),
              .sR(R), .sG(G), .sB(B), .sA(A), .spixelx(vga_pixelx),
.spixely(vga_pixely),
              .VGA_R(VGA_R), .VGA_G(VGA_G), .VGA_B(VGA_B), .VGA_CLK(VGA_CLK),
.VGA_HS(VGA_HS), .VGA_VS(VGA_VS),
              .VGA_BLANK_n(VGA_BLANK_n), .VGA_SYNC_n(VGA_SYNC_n));

endmodule

/*Handles all memory for the sprite
  Stores the sprite data in memory, and outputs the appropriate pixel if requested*/
module sprite_memory(input logic clk,
              input logic reset,
              input logic [31:0] writedata,
              input logic [4:0] x,
              input logic [4:0] y,
              input logic write,

              output logic [7:0] R,
              output logic [7:0] G,
              output logic [7:0] B,
              output logic [7:0] A
              );


reg [31:0] sprite1 [31:0][31:0]; // 2D array of 32 bit words, each stores ones pixel
reg [31:0] out;

initial begin
       for (int i =0; i<32;i++) begin
              for (int j=0; j<32; j++)
                     sprite1[i][j] = '0;
       end
end

// split up output into 4 values
assign R = out[7:0];
assign G = out[15:8];
assign B = out[23:16];
assign A = out[31:24];

always_ff @(posedge clk) begin
       if(write) begin
              sprite1[x][y] <= writedata;
       end
       out <= sprite1[x][y];
end
endmodule
```

VGA_Piano_Emulator.sv

```
/*
```

```
 * Piano
 */

module VGA_Piano_Emulator(
 input logic     clk50, reset,
 input logic [7:0] keypress,
 input logic [15:0] xcoord, ycoord, // the bottom center point of the cursor
 input logic [7:0] sR, sG, sB, sA,
 output logic [4:0] spixelx, spixely, // address to pixel memory top left = 0, 0
 output logic [7:0] VGA_R, VGA_G, VGA_B,
 output logic           VGA_CLK, VGA_HS, VGA_VS, VGA_BLANK_n, VGA_SYNC_n);

 logic [15:0] xcount, ycount; // pixel location on screen of raster scan

/*
 * 640 X 480 VGA timing for a 50 MHz clock: one pixel every other cycle
 *
 *HCOUNT 1599 0             1279       1599 0
 *            _______________              ________
 * __________|    Video      |____________|  Video
 *
 *
 * |SYNC| BP |<-- HACTIVE -->|FP|SYNC| BP |<-- HACTIVE
 *
 *       _______________________      _____________
 * |____|       VGA_HS          |____|
 */

       /*Parameters for the VGA screen*/
   parameter HACTIVE      = 11'd 1280,
             HFRONT_PORCH = 11'd 32,
             HSYNC        = 11'd 192,
             HBACK_PORCH  = 11'd 96,
             HTOTAL       = HACTIVE + HFRONT_PORCH + HSYNC + HBACK_PORCH; //1600

   parameter VACTIVE      = 10'd 480,
             VFRONT_PORCH = 10'd 10,
             VSYNC        = 10'd 2,
             VBACK_PORCH  = 10'd 33,
             VTOTAL       = VACTIVE + VFRONT_PORCH + VSYNC + VBACK_PORCH; //525

       /*Parameters for the key display*/
       parameter WHITE_KEY_WIDTH  = 10'd 91,
                            BLACK_KEY_WIDTH    = 10'd 30,
                            WHITE_KEY_HEIGHT   = 10'd 400,
                            WHITE_KEY_HEIGHT_PRESSED  = 10'd 430,
                            BLACK_KEY_HEIGHT   = 10'd 380,
                            BLACK_KEY_HEIGHT_PRESSED  = 10'd 395;

       parameter CURSOR_SIZE      = 16'd 32;

   logic [10:0]                         hcount; // Horizontal counter
   logic                                      endOfLine;

   always_ff @(posedge clk50 or posedge reset)
     if (reset)            hcount <= 0;
     else if (endOfLine)   hcount <= 0;
     else                        hcount <= hcount + 11'd 1;

   assign endOfLine = hcount == HTOTAL - 1;
   // Vertical counter
   logic [9:0]                          vcount;
   logic                                endOfField;

   always_ff @(posedge clk50 or posedge reset)
     if (reset)              vcount <= 0;
     else if (endOfLine)
       if (endOfField)       vcount <= 0;
       else                  vcount <= vcount + 10'd 1;

   assign endOfField = vcount == VTOTAL - 1;

   // Horizontal sync: from 0x520 to 0x57F
   // 101 0010 0000 to 101 0111 1111
   assign VGA_HS = !( (hcount[10:7] == 4'b1010) & (hcount[6] | hcount[5]));
   assign VGA_VS = !( vcount[9:1] == (VACTIVE + VFRONT_PORCH) / 2);

   assign VGA_SYNC_n = 1; // For adding sync to video signals; not used for VGA

   // Horizontal active: 0 to 1279     Vertical active: 0 to 479
   // 101 0000 0000  1280          01 1110 0000  480
   // 110 0011 1111  1599          10 0000 1100  524
   assign VGA_BLANK_n = !( hcount[10] & (hcount[9] | hcount[8]) ) &
                  !( vcount[9] | (vcount[8:5] == 4'b1111) );

   assign VGA_CLK = hcount[0]; // 25 MHz clock: pixel latched on rising edge

       assign xcount = hcount >> 1;
       assign ycount = vcount;

       logic on_white_key;              // raster point is on a white key
       logic [7:0]key_horizontal; // index of the white key that the horizontal
position corresponds to, if any
       logic on_border;                         // set to 1 if on border between white
keys
       logic on_black_key;
       logic [7:0]black_key_horizontal; // index of the black key that the horizontal
position corresponds to, if any
       logic [15:0] tempx, tempy;       // used to pick off lower bits of calculated
memory address
       assign spixelx = tempx[4:0];
       assign spixely = tempy[4:0];

       always_comb begin
              key_horizontal = 0;
              on_white_key = 0;
              on_border = 0;
              black_key_horizontal = 0;
              on_black_key = 0;

              /*
               * raster scan painting logic
               */

              // determine which white key the horizontal position corresponds to
              if (xcount < WHITE_KEY_WIDTH)
                     key_horizontal = 1;
              else if (xcount > WHITE_KEY_WIDTH && xcount < WHITE_KEY_WIDTH*2)
                     key_horizontal = 2;
              else if (xcount > WHITE_KEY_WIDTH*2 && xcount < WHITE_KEY_WIDTH*3)
```

```
                        key_horizontal = 3;
                else if (xcount > WHITE_KEY_WIDTH*3 && xcount < WHITE_KEY_WIDTH*4)
                        key_horizontal = 4;
                else if (xcount > WHITE_KEY_WIDTH*4 && xcount < WHITE_KEY_WIDTH*5)
                        key_horizontal = 5;
                else if (xcount > WHITE_KEY_WIDTH*5 && xcount < WHITE_KEY_WIDTH*6)
                        key_horizontal = 6;
                else if (xcount > WHITE_KEY_WIDTH*6)
                        key_horizontal = 7;
                else
                        on_border = ycount > WHITE_KEY_HEIGHT; // if not horizontally
aligned with a white key, must be on a border if vertically within white keys

                // determine if vertically on a white key
                if (key_horizontal == keypress) // if key pressed, check for pressed
white key height
                        on_white_key = ycount > WHITE_KEY_HEIGHT_PRESSED;
                else
                        on_white_key = ycount > WHITE_KEY_HEIGHT;

                // determine which black key the horizontal position corresponds to
                if(xcount > (WHITE_KEY_WIDTH - BLACK_KEY_WIDTH/2) && xcount <
(WHITE_KEY_WIDTH + BLACK_KEY_WIDTH/2))
                        black_key_horizontal = 8;
                else if(xcount > (2*WHITE_KEY_WIDTH - BLACK_KEY_WIDTH/2) && xcount <
(2*WHITE_KEY_WIDTH + BLACK_KEY_WIDTH/2))
                        black_key_horizontal = 9;
                else if(xcount > (4*WHITE_KEY_WIDTH - BLACK_KEY_WIDTH/2) && xcount <
(4*WHITE_KEY_WIDTH + BLACK_KEY_WIDTH/2))
                        black_key_horizontal = 10;
                else if(xcount > (5*WHITE_KEY_WIDTH - BLACK_KEY_WIDTH/2) && xcount <
(5*WHITE_KEY_WIDTH + BLACK_KEY_WIDTH/2))
                        black_key_horizontal = 11;
                else if(xcount > (6*WHITE_KEY_WIDTH - BLACK_KEY_WIDTH/2) && xcount <
(6*WHITE_KEY_WIDTH + BLACK_KEY_WIDTH/2))
                        black_key_horizontal = 12;

                // determine if vertically and horizontally on a black key
                if(black_key_horizontal == keypress && black_key_horizontal != 0) // if
current black key is pressed check for pressed black key height
                        on_black_key = ycount > BLACK_KEY_HEIGHT_PRESSED;
                else if(black_key_horizontal != 0)
                        on_black_key = ycount > BLACK_KEY_HEIGHT;

                /*
                 * Do the actual painting
                 */

                // draw background color, lowest priority
        {VGA_R, VGA_G, VGA_B} = {8'ha8, 8'hba, 8'hea};

                tempx = 0;
                tempy = 0;

                // if in the range of the cursor, calculate pixel address and send to
memory
                if (xcount >= xcoord && xcount < (xcoord + CURSOR_SIZE) && ycount >=
ycoord && ycount < (ycoord + CURSOR_SIZE)) begin
                        tempx = (xcount - xcoord);
                        tempy = (ycount - ycoord);
```

```
             end

             if(on_black_key) // paint black key, low priority
                   {VGA_R, VGA_G, VGA_B} = {8'h0, 8'h0, 8'h0};    //draw black key
             if( on_white_key ) // paint white key, medium priority
                   {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff}; //draw white key
             if (on_border) // paint border, high priority
                   {VGA_R, VGA_G, VGA_B} = {8'h0, 8'h0, 8'h0};    //draw black
border
             if (xcount > xcoord && xcount <= (xcoord + CURSOR_SIZE) && ycount >=
ycoord && ycount < (ycoord + CURSOR_SIZE) && sA != 0) //draw cursor
                   {VGA_R, VGA_G, VGA_B} = {sR, sG, sB}; // paint cursor, highest
priority
   end

endmodule // VGA_Piano_Emulator
```