Leap Motion Piano: Design

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

Overview

In our project, we are implementing a virtual piano using a Leap Motion device. The Leap Motion hardware contains a camera system that enables software to use video processing techniques to determine hand and finger positions, as shown in the picture. Since the Leap Motion needs to run on x86 architecture, we cannot 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.

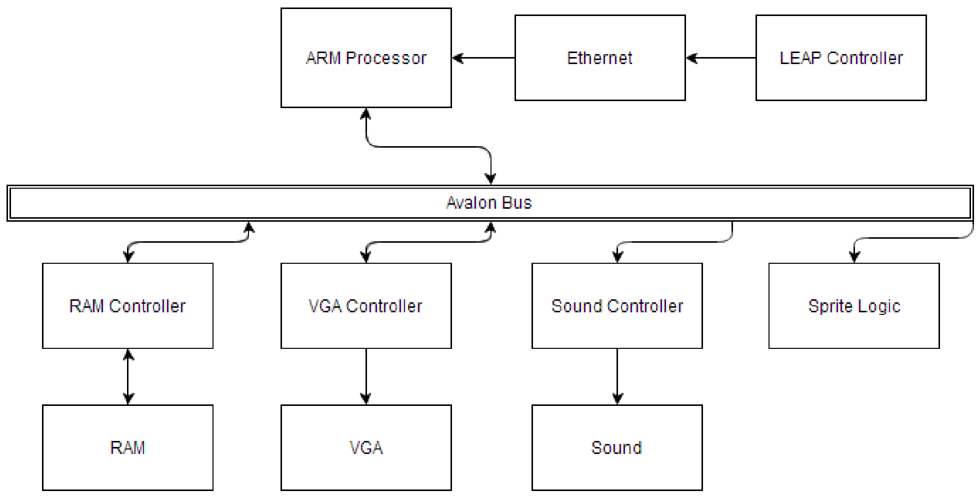
Another major component of the project is a display controller that can interface with the monitor to display the piano. The display will also track finger movement using a cursor displayed by small sprites. Since this piano will be able to play sounds, we also need an audio controller to speak to the audio codec on the board. The audio and video components will be designed in hardware, as shown in the block diagram below.

To play this piano, the user will hover their fingers over the Leap Motion and press one finger down to simulate a piano key press. Whichever note is pressed will be played through the 3.5mm audio out line.

Block Diagram

As shown in the diagram below, the ARM Processor interacts with Ethernet and various controllers through the Avalon Bus. Its primary functions are as follows:

- receiving coordinate data from the Leap Motion device through Ethernet communication
- reading sprite data for the finger cursor from the RAM through the RAM Controller
- displaying piano keys and simulating key presses on the VGA display through the VGA Controller
- producing corresponding audio notes through the Sound Controller



Software
The software side of the project includes 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 will be done using sockets and a UDP connection in Java, using a packet structure that includes the finger position data on the workstation. This will be received on the HPS through a C program. This position data must be converted from physical measurements (millimeters) to pixels.

The second component will then use the position data to calculate when a key is pressed based on the velocity in a direction, and which key is pressed. This will invoke the video and audio components which will write to the areas of memory where the components are mapped.

Memory Requirements
The memory requirements of our project will mainly come from support of the VGA display. The VGA display consists of two components: the sprite indicating the cursor that follows the finger's position, and the background piano display. The sprite for the cursor will be made of an 8x8 pixel region. Each sprite/cursor consists of both sprite data and sprite attributes. This is stored in 4 bytes of RAM, with 1 byte dedicated to the sprite attributes. The pattern table must also store 8*8 = 8 bytes. With a maximum of ten fingers as input, this will lead to at most (4+8)*10 = 120 bytes of sprite data. The video framebuffer functions differently, and is where we will be displaying the piano. We have two options for displaying the keys: drawing a series of 2D rectangles, or drawing a more elaborate design using sprites. If we decide to draw a series of rectangles, everything can be hardcoded, so no memory will be needed. Drawing the piano in

this way, there will be a larger and a smaller sized key, for the white and black keys, respectively. With a 640 x 480 screen, it would be too difficult for to fit an entire piano, so we will be only displaying one octave (at least initially). This will translate to 90 pixels in width per key. If we decide to use sprites, however, we will need to store memory. The larger keys will be 90 x 480, while the smaller keys will be around 45 x 300. Beginning with only one octave, we will have seven large keys and five small keys. With 12 more sprites, the information would be stored in 4*12 = 48 bytes of RAM. The pattern tables would take up 90*480 + 45*300 = 7088 bytes. Thus, the total memory needed would be 7088 + 48 + 120 = **7256 bytes**. Note that this is only an estimate and is subject to change as our design changes.
We will not be needing any memory to support the audio portion of our project, because we will be able to use audio files stored in the chip's memory.

Peripherals

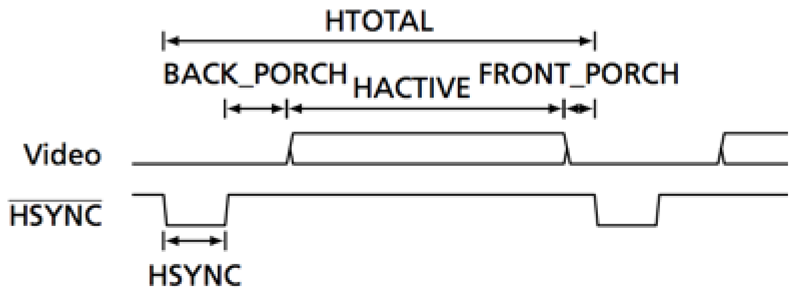
VGA:

The VGA controller will include VGA output logic and a custom frame buffer that renders the piano keys. The controller interface allows additional images to be added to the buffer, such as the cursor. The sprite controller reads the sprite information from RAM, determines the location of the image and sends it to the frame buffer. The sprite controller is sent coordinates from the processor that tell it where to display the cursor.

Software-Hardware:
We will need three registers for the software-hardware VGA interface, consisting of an x-coordinate, a y-coordinate, and a y-velocity. Each register is an integer and will therefore be made up of 4 bytes.

Hardware-Hardware:

Timing Diagram



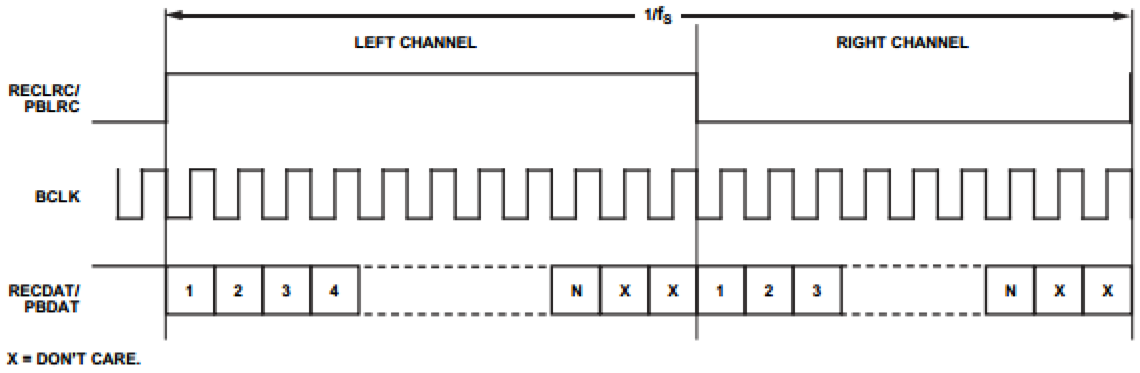
Inputs to our VGA controller:
- Clock
- Framebuffer that holds image data

Outputs to the monitor:
- VGA monitor signals that go to the VGA DAC

Ethernet:
On a separate computer, raw input from the Leap Motion device is sent through Ethernet to the SoCKIT board. Because the VGA display will be constantly updated with high frequency, some packet loss and error can be tolerated. Thus UDP will be used. The structure of the UDP data is simply the x-coordinate, followed by the y-coordinate, followed by the y-velocity. Each value consists of 4 bytes, for a total of 12 bytes per data delivery. Software on the board converts these raw input coordinates to VGA display coordinates and determines where to render the cursor on the screen (translating from mm to pixels), as well as whether a key has been pressed.

Audio:
When the program determines that a key has been pressed it chooses the appropriate audio file. It then writes the file to the audio controller, which sends it to the SoCKIT board's audio device.

Hardware-Hardware:
Below is the timing diagram for the interface between the audio controller and the audio codec device.

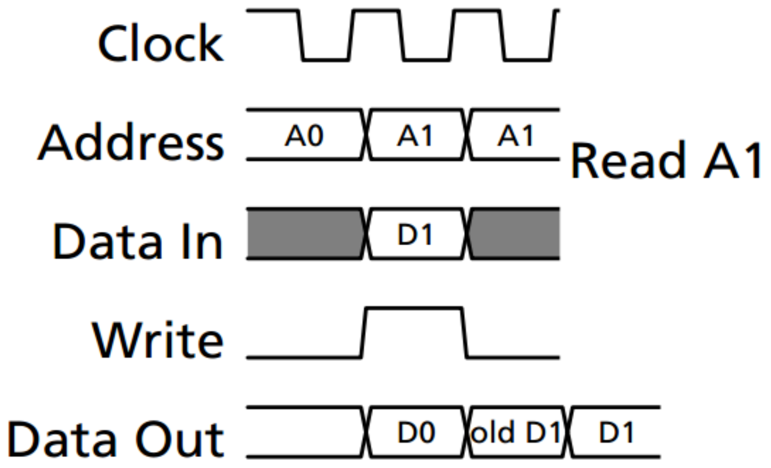


PBLRC is the digital audio frame clock which is used to determine whether the audio data stream is left channel data or right channel data. BCLK is the digital audio bit clock, which informs the device when a new bit is on the data stream. PBDAT contains the time domain multiplexed left and right channel audio stream data.

Software-Hardware:
The register we will be using is a "buffer" for the audio file. The software will write the audio file into this buffer, which will then be sent to the audio codec. There will also be another register used as a start signal, so the device will know when to begin playing the file.

Memory:
The sprite data needed for the cursor will be stored in memory. This data is accessed by the sprite hardware when drawing to the screen. The memory controller takes read and write requests, and then passes them on to the memory.

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 write is 1, the memory reads data from Data In into the specified address. When write is 0, the memory writes data from the specified address onto Data Out.

<u>**Milestones**</u>

<u>Milestone 1:</u>
Write software that runs on an x86 machine that reads data from the Leap API and sends it to the SoCKIT board via Ethernet. For this first milestone, we will start with input from only one finger. We will also write code that runs on the SoCKIT board that reads this data and converts it into coordinates that we then use to display a simple symbol (a ball) on the VGA display. This will involve a basic framebuffer and VGA controller that can send output to a VGA display.

<u>Milestone 2:</u>
Get cursor on VGA display to track finger movement, replacing the symbol from milestone 1 with a sprite. This will require the sprite hardware that will send the cursor image to the framebuffer. Draw the piano to the VGA display.

<u>Milestone 3:</u>
Get sound and proper key press animation based on where cursor is when finger is moved downwards. The framebuffer must be enhanced to display keys and animate a key press. The audio controller must be written to act as an interface between the processor and the board's audio device. The software must be able to recognize a key press and determine which key is being pressed. With this information it informs the framebuffer which key to display as pressed and it must send the correct the audio file to the audio controller.