Gesture Recognition System for Music Playback

Jenny Li, Shana Mathew
{jenli8, smathew}@mit.edu

December 12, 2018

Contents

1 Introduction

We began our ideation process with a common interest in a diverse array of music genres. We regularly used multiple music streaming platforms such as Spotify, Tunein, and Soundcloud, but across all of these platforms were the same control interfaces: the play button and next/last song icons sit on top of the scrolling progress bar with the volume bar located to the right or in the general vicinity of the other controls. As a result, we wanted to create a system that reinvented the way users control their music while decreasing the hassle of having to navigate around in their device to get to the controls. This project was the result of our goals that we felt transforms a user's music playing experience while still remaining intuitive.

We drew inspiration from both lab 5 and 5c (the voice recording and bubble leveler labs) to complete our system. The same breadboard setup of Teensy + MPU-9255 chip from the leveller lab was used for gesture recognition and audio background knowledge from the audio lab was used for audio playback and storage. Inspiration was also drawn from actually marketed products such as the Leap Motion motion tracking devices to interact with virtual objects.

2 Summary

We developed a gesture recognition system that allows users to control their music playing experience solely through hand movements. Users will control play/pause state, volume level, song position, and music selection through our handheld/wearable system that track user's hand acceleration and rotation in 3D space. For example, an upward lifting motion should be interpreted as increasing the volume of the song while a wave of a hand towards the user's right tells our system to fast forward 10 seconds of the current song.

An external board fitted with a Teensy and MPU-9255 (3-axis IMU and gyroscope included) will track user's hand movements (acceleration and rotation) and interpret the user's movements as a specific command to modify music playback. Song data was stored in the memory of a 2 GB microSD card, and the overall module was later altered to also accommodate streaming music through an external Bluetooth Digital Power Amplifier Board (XY-502B) module that was connected via bluetooth to a laptop and hooked up to an FPGA. Song information such as song name and artist was displayed on a monitor display along with the play/pause signs, volume level, and real-time updates on the progress of the song. All our base and expected goals were met throughout the project development phase, with a working gesture recognition system controlling both the songs loaded in an SD card and songs streamed to the FPGA via a custom bluetooth module. Below is a breakdown of our experience building this project, the challenges faced by each of us as well as during the integration phase, and future areas of improvement for our product.

3 System Design: Gesture Recognition

3.1 Overview

Gesture control was achieved with an MPU-9255 IMU chip that sent 3 axes acceleration and gyroscope data to a Teensy which then serially sent the 12 bytes of values to a Nexys 4 Artix-7 FPGA. The Teensy was configured to read raw 16-bit acceleration and gyroscope values which was integrated with a pre-determined loss coefficient of 0.5. The 16-bit two's complement formatted raw values were then serially sent byte by byte to the FPGA at a loop speed of 30ms. Bits were sent at 9600 baud to the FPGA and the complete setup of IMU, Teensy, and wires connected to the FPGA (not shown) is pictured in Figure 1 below. In order to have the system start reading values, users must press the center button on the FPGA at which point users have 1 second to perform a valid gesture. A series of flags were also used to indicate the state of each gesture where the change in state of a flag indicated a certain gesture was performed.

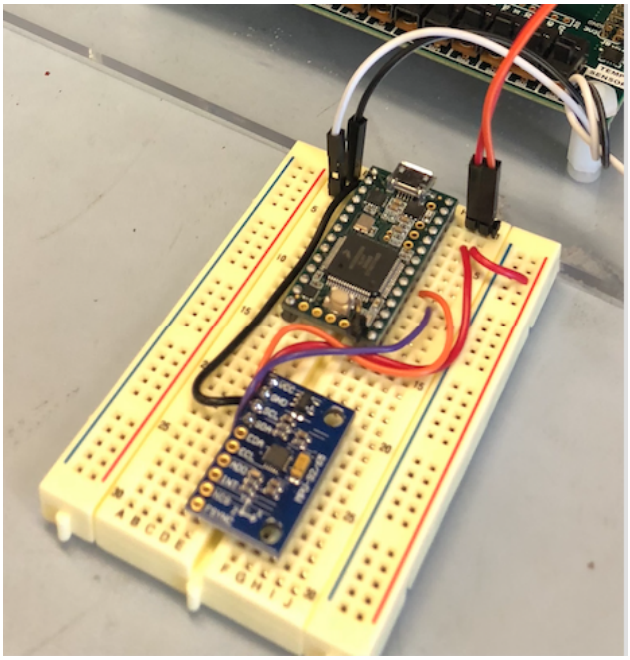


Figure 1: IMU and Teensy board connected to FPGA

3.2 Data Collection

The data collection module continuously takes serial data bits from the Nexys 4 FPGA's JB[0] port as an input and builds up 2 60-bit shift registers to hold all axes of data for acceleration and gyroscope values (6 bytes of data and start/stop bits for each type of data). Data was sent in the following order:

start bit → Accel X byte1 → stop bit → Accel X byte2 ... → start bit → Gyro X byte1 ...

As shown in the data sending order above, the acceleration values were sent first, meaning the 60-bit acceleration shift register was populated first and the 60-bit gyroscope shift register afterwards. Figure 2 below depicts the waveform of one byte's transmission from Teensy to FPGA. After both shift registers received 60 bits of data, a binary 'done' register is asserted to notify other dependent modules of newly completed data available for use.

Figure 2: Waveform for one byte's transmission

3.3 Data Parsing

The parser module was one such module that depended on the assertion of the 'done' register output by the data collection module explained in Section 3.2. Once 'done' is asserted, parser will index into the two shift registers that were built in the data collection module and parse only the relevant data bytes without the start/stop bits. Data values were sent to the FPGA in a two's complement format, requiring us to convert each byte into unsigned binary format with sign information preserved through the most significant bit (MSB): An MSB with hexadecimal 'F' meant the value was negative and an MSB with hexadecimal '0' meant the value was positive. The parsed and sign preserved binary numbers are then output to the filtering module described in the next section.

3.4 Filtering

The filtering module took a running average of the incoming values to stabilize the overall values. For example, if one incoming value was 16'h0025 and the next value was 16'h0028, the new running average ends up as 16'h0027. Averaging is performed for the x-, y-, z- axes values for both acceleration and gyroscope. It is important to note that because the values were sign represented through the MSB, the incoming value had to be compared to the running average and incorporated into the new running average according to it's MSB value. For example, a positive running average value that is incorporating a new incoming negative value greater in magnitude than the running average would have to perform this subtraction:

(incoming negative value - running average)

This difference is bit shifted right by 1 and the negative value's MSB is appended as the MSB of the difference to get the new running average. The filtered values are finally output to the gesture recognition module where interpretation occurs.

3.5 Gesture Recognition

The Gesture Recognition FSM module interprets actions by looking at specific axes' values that were experimentally determined to be more prominent than other axes when a specific gesture is performed. Experimentally determined thresholds were set for both acceleration and gyroscope axes to determine when incoming values from a target axes exceeds the threshold at any point within the one second countdown which begins at the press of the center button on the FPGA. When a gesture is made, values are checked to see if it exceeds threshold conditions. When values are confirmed to exceed thresholds, a new one second timer is started to see when the exceeded axes falls back below it's threshold, indicating the completion of a full gesture. Figure 3 shows a sample Arduino plotter graphing the acceleration values; the evident changes in the z-direction (red graph) suggests this gesture should focus on the z-acceleration and not as much on the x-direction (blue graph)

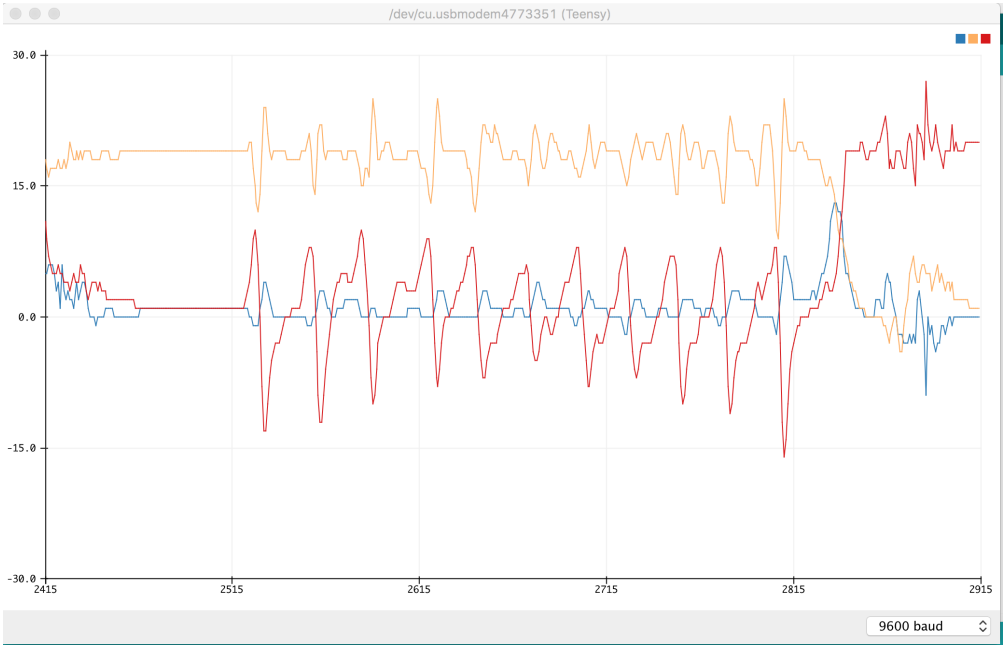


Figure 3: Plotted directional graphs for a gesture

Finally, the gesture recognition module outputs the NOT-ed form of binary gesture flags to reflect the gesture. For example, a play/pause gesture is performed by having a user hold the IMU remote in a vertical upright position with the IMU facing away from the user. When in that starting position, users press the center button and perform a quick pushing motion before holding the same hand position at the end.

When interpreted correctly, the music will stop playing and a monitor displaying song status will show the corresponding icon to indicate paused status. To the FPGA, values change as follows:

play flag= 0 → BTNC= 1 → gesture performed → play flag= play flag= 1

It is important to note that in the FSM, a DOUBLE PROCESS state exists to differentiate between fast forward/next song and rewind/previous song gestures. While both pairs of gestures perform the same starting action, the end hand positions differ, therefore requiring an extra 0.37 seconds where the end hand position is evaluated to determine which gesture was performed. Figure 3 shows a sample Arduino plotter is shown of the acceleration values and the evident changes in the z-direction (red graph) suggests this gesture should focus on the z-acceleration and not as much on the x-direction (blue graph)

3.6 Testing Methods

3.6.1 12 byte transmission

A lab oscilloscope was used to verify that 12 complete bytes of information was sent to the FPGA. Probes were hooked up to the IMU+Teensy breadboard and the incoming bits were displayed on the oscilloscope, producing the waveform shown in Figure 4.

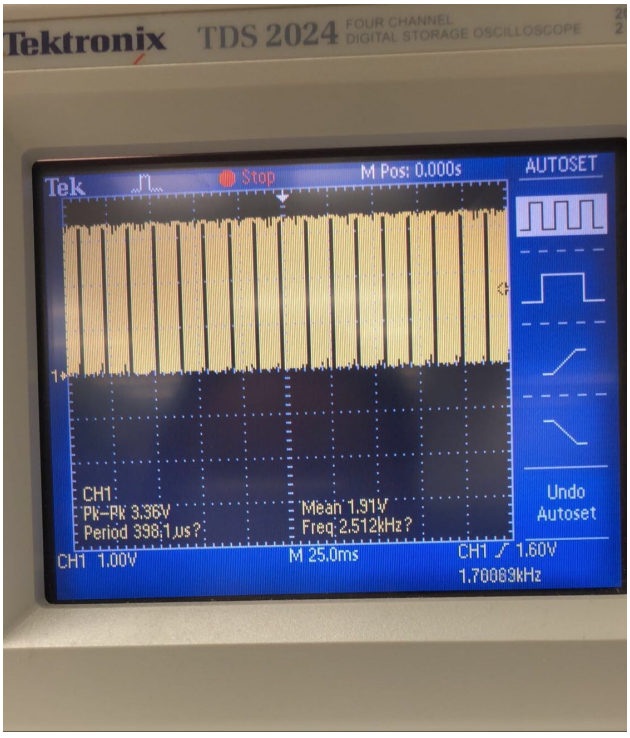


Figure 4: cycles of bit transmission from Teensy to FPGA

3.6.2 Correct Gesture Interpretation

Interpreted gestures were tested using a uniquely colored blob displayed on the lab monitor. Each gesture corresponded to a temporary parameter value which corresponded to a specifically colored blob. The color of the blob indicated the gesture that was interpreted from the gesture read into the system.

4 Audio Control

4.1 Writing to the microSD card

A 2GB microSD card was used to store song data. In order to write to the microSD card, the desired audio files were first converted to a .wav format with an 8-bit depth and 44.1 kHz sample rate in order to ensure that all the songs were consistent. Then they were converted to a binary file using a Matlab script which rescaled the data to an 8-bit unsigned integer. HxD, a hex, disk, and memory editor, was used to open the SD card as a physical device and copy the contents of the bin file into the desired start memory address, and the start and end addresses of the sectors were recorded.

4.2 Reading from the microSD card

Then, we interfaced with a series of lab-provided modules that read the first two 512-bytes sectors of the SD card containing audio data, and store each sector in two separate 8-bit wide 512-byte deep BRAM. Once both BRAMs were full, data is read out from one of the BRAMs at a rate of 44.1KHz and output to the PWM module. Once one of the BRAMs have been completely read, the same process is done for the other BRAM and and the next sector is read from the SD card and put in the first BRAM since SD card sector reads happen at a faster rate than the rate of playing the data. This is done until the end sector of the audio data has been played.

These modules were modified so the main module contains a two length 5 arrays of 32-bit registers, each array corresponding to the start or end sector of a certain audio file. A length of 5 was chosen so that 5 songs could be put on our "playlist"; however, this number can be accordingly modified. The index of the song that should be playing was stored as a 4-bit register and sent to the corresponding start and end sector memory locations to the `sd_handler` module.

When the current SD address has reached the end address, the new song index is output. Its value is the current song index either incremented by one or reset to 0 if all the songs have been played. In the main module, the current song index is set to this new song index.

4.3 Audio Manipulation

4.3.1 Volume Control

We control the volume by right bit-shifting the 8-bit sample data. The default bit shift is 2, so if the volume is raised, we decrement the current bit shift, an internal register, by 1; similarly, if the volume is lowered, we increment the current bit shift by 1. We chose the range of the bit shift to be [0, 4]; a right bit shift greater than 4-bits would result in the audio being too low to hear. We change the audio sample using this bit shift and input this new sample to the PWM module.

4.3.2 Rewinding and Fast Forwarding

To rewind the song, we check if the current SD address that is being read is more than 443392 bits, or approximately 10 seconds of data, past the start address; this is to ensure that the song can indeed be rewinded 10 seconds. If this is the case, we then set: current SD address = the current SD address − 443392 bits.

Likewise, to fast forward the song, if the current SD address is more than 443392 bits, behind the end address, the current SD address is set to 443392 bits after the current SD address, effectively fast forwarding the song by 10 seconds.

4.3.3 Skipping songs

When the signal to go to the previous song is received, the current song index is decremented by 1, or set it to the index of the last song if the current song index is 0. We also create a pulse delayed by one clock cycle to tell the `audio_handler` to load the song at this new SD card start address.

When we receive the signal to go to the previous song, we decrement the current song index by 1 or if the current song index is 0, we set it to the index of the last song. We also create a pulse delayed by one clock cycle to tell the `audio_handler` to load the song at this new SD card start address which is passed in at the next clock cycle.

4.3.4 Play/Pause

The 1-bit `pause_flag` wire initially starts off with a value of 0; when a play/pause gesture is performed the bit is flipped such that the first time the signal is given, the wire representing the 1-bit `pause_flag` is 1. The switch to signal that songs should be loaded and-ed with the bit-wise "not" of the `pause_flag` wire is passed into the cycle to tell the `audio_handler` module as an input called `song_playing`.

If `song_playing` is 1, then only will music be played from BRAM. This is controlled by either incrementing address of the sample playing from BRAM or switching to the address of the other BRAM if we've reached the end of the current BRAM.

4.4 Bluetooth Streaming

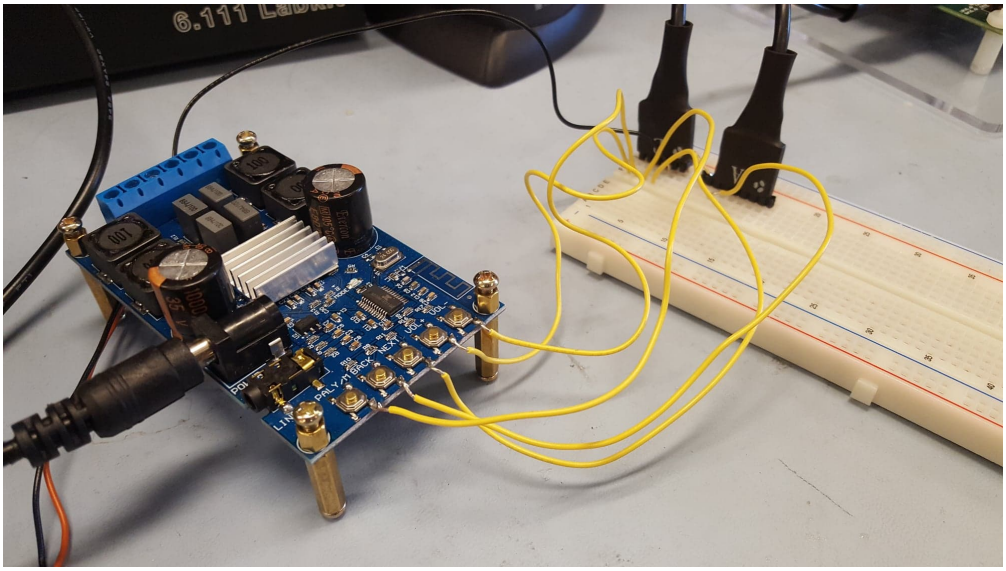


Figure 5: Bluetooth module setup for streaming music

The bluetooth module had functionalities to control the volume, playing/pausing, and skipping to the previous/next song. A high-Z corresponded to no change in these actions, while a pulse low corresponded to that action being triggered. We used a change in the action flags to trigger a low pulse that lasted $\frac{1}{4}$ of a second that was output to the appropriate JC ports connected to the bluetooth module (see yellow wires on the above picture). The black wire from the bluetooth module was connected to GND on the FPGA.

4.5 Testing

Testing was done by assigning the values of the flags to different switches on the Nexys so that the audio modules could be tested separately from the gesture recognition part of the project. This allowed us to sequentially and parallelly try different combinations of actions test the interface and integration of the SD and audio manipulation modules.

5 XVGA Monitor Display

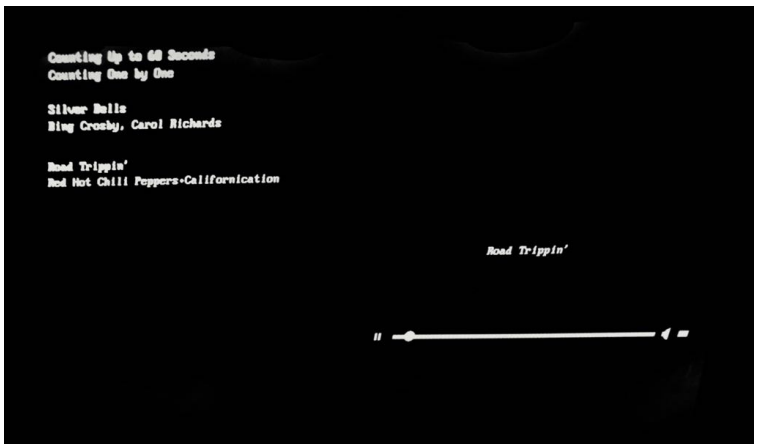


Figure 6: XVGA Monitor Display Setup

5.1 Displaying Text

A text module was written in order to display strings of text on the monitor using XVGA resolution.

The text module takes in a 400-bit input that representing the string of 8-bit characters to display. This can hold up to a maximum of 50 characters. Each character of this input is sequentially indexed into and sent in 8-bit chunks to the `ascii_to_dots` module.

The `ascii_to_dots` module contains the pixel data for each ASCII character. Each character has a map of 8×16 pixels; each character's pixel data can be indexed in by row. If the character exists at a certain location in the 8×16 grid, then that pixel will have a value of 1; otherwise if it is blank, then it will have a value of 0.

Depending on the current hcount and vcount values, the character's location in the string, and where the string starts (the top left of the string is denoted by the 11-bit x and 10-bit y inputs), the `ascii_to_dots` module returns the appropriate 8-bit binary value of the row of the character currently being displayed. Using an internal register that serves as a counter as to which pixel in the row is being displayed, we index into that particular value in the pixel map row and set the pixel value to either the color parameter input or 0 accordingly.

Using two length 5 arrays of 400-bit registers, we stored the song title information of the songs we put in the SD card in one and the song artist/album information in the other. These were sequentially indexed into and displayed as a playlist on the left-hand side of the screen. On the right-hand side above the song progress bar, we displayed the currently playing song by indexing into the array of song titles with the register that stored the value of the current song index.

5.2 Song Progress Bar

In our main module, we also in a length 5 array store 11-bit registers with the value of how many pixels correspond to one second of each song, given that our progress bar is 350 px. When the song has started and is playing (the `pause_flag` wire has a value of 0), we use a 1 Hz enable signal to adjust the x position of the `circle_blob` to move the circle along the progress bar. If a rewind or a fast forward happens (and is acted upon), then we move the `circle_blob` back or forward the pixel amount corresponding to 10 seconds.

We know if a rewind or fast forward signal has been acted on since we have two registers in the `sd_handler` that flips their bits every time we actually rewind or fast forward (if more than 10 seconds have passed in the song or if there are more than 10 seconds remaining). These registers are assigned to the values of output wires of this module accordingly.

We store the value of the previous registers of these wires; if the values have changed, we know that a rewind or a fast forward has just happened and can change the progress bar circle accordingly.

5.3 Volume Display

We display an speaker icon using the `character` module, which works similarly to the text module in calling the `ascii_to_dots` module but takes in only one 8-bit character instead of a 400-bit string. Next to this icon, a dynamic rectangular blob corresponding to the volume level is displayed. This comes from an 8-bit wire output of the `audio_handler` module that subtracts the current bit shift register from 4 (since a smaller bit shift means the volume is higher) and multiplies this value by 4 by left bit-shifting by 2. This value is passed into the blob module as the input corresponding to the width of the rectangular blob that indicates the volume level.

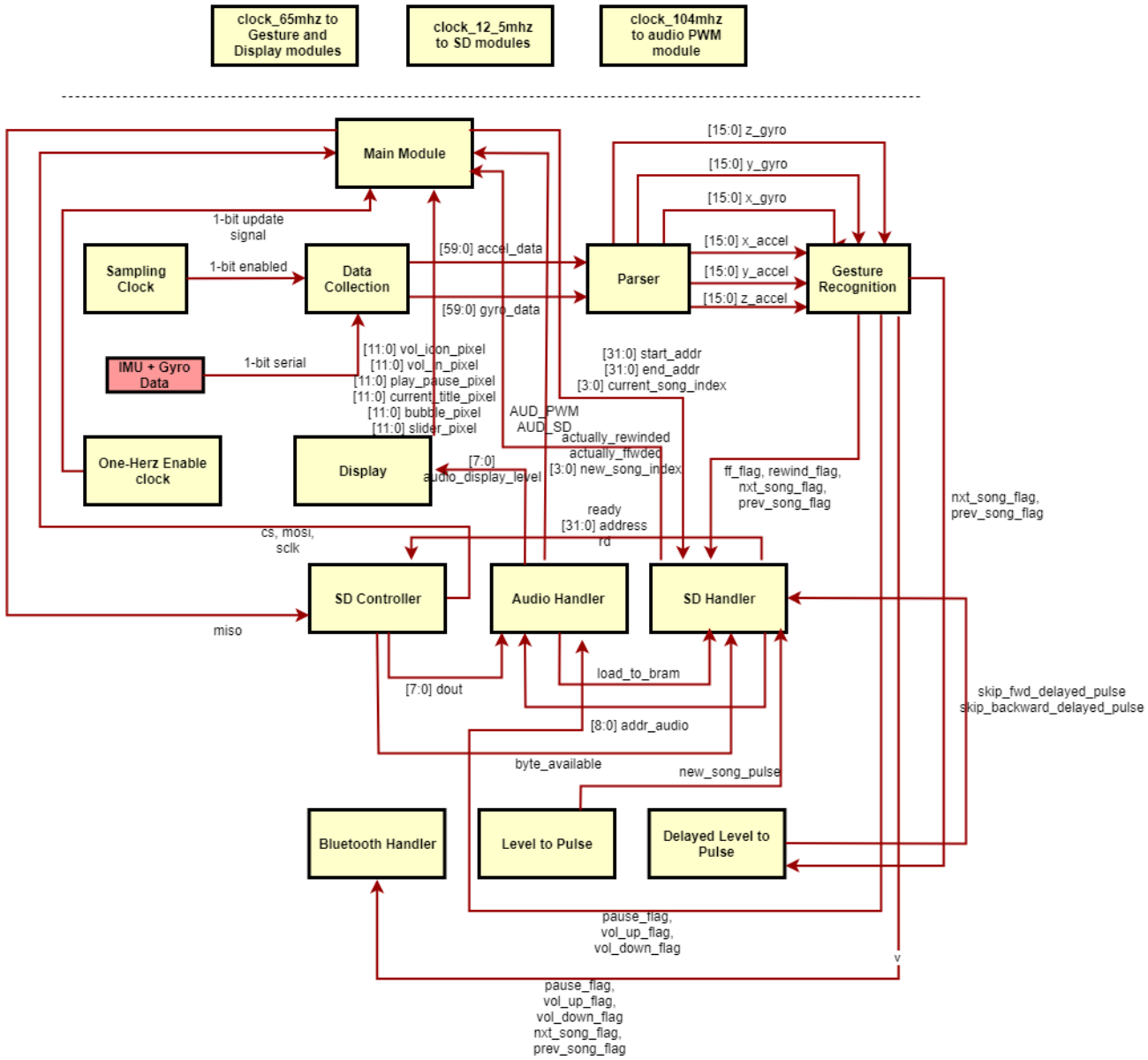
5.4 Play/Pause Display

Depending on the value of `pause_flag`, the character module is called and a pixel icon corresponding to a play arrow or a pause symbol is displayed.

6 Integration Testing

After integrating the gesture recognition and audio handling sides into one cohesive project, testing was done with a completed monitor interface that showed the necessary icons for song playback and LEDs on the FPGA were assigned to each gesture to show which gesture was performed at any given time. Additionally, the seven segment displays on the FPGA were used to display acceleration values from the IMU remote.

7 Block Diagram



8 Challenges

The biggest challenge to gesture recognition was being able to clearly distinguish which axes would accurately differentiate between gestures. While each gesture was different in the way it is performed, the gyroscope and acceleration values were the same. As a result, the orientation of the board was required as well as specific thresholds that separated gestures. A lot of time was spent on fine tuning each gesture and despite the fine tuning, there was still noise that caused certain gestures to blend with others (i.e last song with volume down).

Another challenge was implementing both last song and rewind song gestures. Due to the similarities in icons of the rewind and last songs on common music platforms, we wanted to implement gestures that were similar enough to be intuitive for users. However, the original idea of implementing a double swipe for rewind proved challenging because implementing a pause period in between the gesture reading period was unreliable and caused too much error to be differentiated correctly. As a result, last song was accomplished by having players swipe left and then laying their hand flat to zero out the axes while rewinding a song is accomplished by swiping left and keeping their hand in the same position.

Also, the manipulation of audio volume was another challenge. While right-shifting the 8-bit samples worked well, left-shifting the 8-bit samples so as to increase their value introduced a lot of noise because resolution was being lost. We considered various options, such as adding an external analog circuit that would increase/decrease the amplitude of the outgoing audio accordingly. However, we eventually settled on right bit-shifting the 8-bit samples from the very start and raising the volume by decreasing the amount by which it is right bit-shifted.

9 Future Work

A potential area for future improvement is to disconnect the IMU + Teensy remote from the FPGA so that users can reliably and freely perform the gestures without being limited by wires. The setup, as shown in Figure 5, included 2 Bluetooth modules that were synced to send values to each other. While our final implementation did not use the disconnected remote yet, progress was made on the Bluetooth communication such that data values were being sent successfully. However, the actual values jumped too erratically for an unknown reason that resulted in our inability to integrate the Bluetooth remote into the final system.

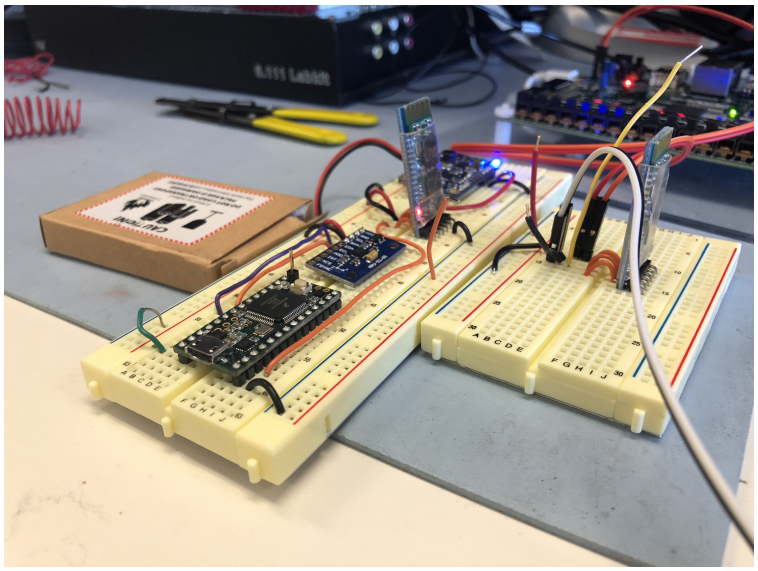


Figure 7: Bluetooth module setup for IMU remote

Another improvement would be to continuously integrate a set number of data points on the Verilog side instead of the Arduino side so that the values would accurately represent a change in velocity over time and better represent a gesture being performed. This would be reliable and feasible because the system is set to give users a 1 second gesture performance period. Therefore, change in time for a gesture is always known and given a 30ms loop speed, approximately 30 data points can be integrated together and analyzed at certain intervals to obtain a clearer understanding of the gesture. This in tandem with the orientation of the board may better distinguish gestures in future iterations.

In the future, we would like to expand the bluetooth streaming functionality by querying the Spotify Web API to get information about the currently playing song and playlist to display on the monitor.

Also, throughout our integration testing, we noticed our system experienced lag between a gesture being performed and the actual response reflected in the display monitor and song. This bug was revealed when the lag between seeing a gesture get recognized and the resulting response was long enough that we would try to perform the gesture again which resulted in the wrong song name being displayed compared to the song that was playing. In the future, better pipelining our system so as to fix these timing issues would be an improvement.

10 Acknowledgements

We would like to acknowledge 6.111 professors Gim Hom and Joe Steinmeyer for dedicating their time to helping us past obstacles in the project phase and fostering our growth as engineers. We would also like to extend our gratitude to the 6.111 LAs/TAs who provided invaluable insight to how key components of our project worked.

11 Appendix

```
`timescale 1ns / 1ps
//`default_nettype none

module labkit(
   input CLK100MHZ,
   input[15:0] SW,
   input SD_CD,
   input BTNC, BTNU, BTNL, BTNR, BTND,
   output[3:0] VGA_R,
   output[3:0] VGA_B,
   output[3:0] VGA_G,
   input[7:0] JB,
   inout[7:0] JC,
   output SD_RESET, SD_SCK, SD_CMD,
   inout [3:0] SD_DAT,
   output AUD_PWM, AUD_SD,
   output VGA_HS,
   output VGA_VS,
   output LED16_B, LED16_G, LED16_R,
   output LED17_B, LED17_G, LED17_R,
   output[15:0] LED,
   output[7:0] SEG,  // segments A-G (0-6), DP (7)
   output[7:0] AN    // Display 0-7
   );

   // SETUP CLOCKS
   // 12.5Mhz clock as the main SD clock
   // 104Mhz clock for the PWM Module

   wire clock_104mhz;
   wire clock_12_5mhz;
   wire clock_65mhz;


// create 65mhz system clock for 1024 x 768 XVGA timing
    clk_wiz_2 clockgen(.clk_in1(CLK100MHZ),
.clk_out1(clock_12_5mhz),.clk_out2(clock_104mhz), .clk_out3(clock_65mhz));

//  instantiate 7-segment display;
    wire [31:0] data;
    wire [6:0] segments;
    display_8hex display(.clk(clock_65mhz),.data(data), .seg(segments), .strobe(AN));

// ** DEBOUNCE MODULE **
    debounce db_btnc (.reset(SW[15]), .clock(clock_65mhz), .noisy(BTNC),
.clean(clean_BTNC));

    assign SEG[6:0] = segments;
    assign SEG[7] = 1'b1;


    wire look_for_start, enabled, herz_enabled, found_start, done;
    wire [3:0] current_state, param_val;
    wire [2:0] gc_state;
```

```
    wire [5:0] accelz_msb;
    wire [2:0] gr_state;
    wire times_performed;
    wire [4:0] assert_seq;
    wire [59:0] accel_array, gyro_array;  //60-bit shift register to hold incoming IMU
data.

    //Parsed x,y,z components
    wire [15:0] accel_x, accel_y, accel_z, gyro_x, gyro_y, gyro_z;
    wire [15:0] filtered_ax, filtered_ay, filtered_az, filtered_gx, filtered_gy,
filtered_gz;
    wire hsync, vsync, at_display_area;
    wire [1:0] cmd;

// ** Action Modules Flags

    wire vol_up_flag_wire, vol_down_flag_wire, pause_flag_wire, rewind_flag_wire,
ff_flag_wire, nxt_song_flag_wire, last_song_flag_wire;

// ** MODULE INSTANTIATIONS **
    herz_clk one_sec_clk (.clk(clock_65mhz), .enabled(herz_enabled));
    recovery_period two_ms_period(.clk(clock_65mhz), .data(JB[0]),
.look_for_start(look_for_start));
    sample_clock sample_clk (.clk(clock_65mhz), .enabled(enabled));
    data_collection dc ( .clk(clock_65mhz), .enabled(enabled), .data(JB[0]),
.look_for_start(look_for_start),
                        .accel_array(accel_array), .gyro_array(gyro_array), .done(done),
.state(current_state));
    parser parse_coords ( .clk(clock_65mhz), .done(done), .accel_array(accel_array),
.gyro_array(gyro_array),
                          .accel_x(accel_x), .accel_y(accel_y), .accel_z(accel_z),
.gyro_x(gyro_x), .gyro_y(gyro_y), .gyro_z(gyro_z));
    filter_mod FM(.clk(clock_65mhz), .vsync(vsync), .clean_BTNC(clean_BTNC),
.accel_x(accel_x), .accel_y(accel_y), .accel_z(accel_z),
                  .gyro_x(gyro_x), .gyro_y(gyro_y), .gyro_z(gyro_z),
.filtered_ax(filtered_ax), .filtered_ay(filtered_ay),
                  .filtered_az(filtered_az), .filtered_gx(filtered_gx),
.filtered_gy(filtered_gy), .filtered_gz(filtered_gz));
    gesture_rec GC(.clk(clock_65mhz), .btnc_clean(clean_BTNC),
                    .filtered_ax(filtered_ax), .filtered_ay(filtered_ay),
                    .filtered_az(filtered_az), .filtered_gx(filtered_gx),
.filtered_gy(filtered_gy), .filtered_gz(filtered_gz),
                    .param_val(param_val), .assert_seq(assert_seq),
.active_axes(accelz_msb), .state(gc_state), .performed(times_performed),
                    .vol_up_flag_wire(vol_up_flag_wire),
.vol_down_flag_wire(vol_down_flag_wire), .pause_flag_wire(pause_flag_wire),
                    .rewind_flag_wire(rewind_flag_wire), .ff_flag_wire(ff_flag_wire),
.nxt_song_flag_wire(nxt_song_flag_wire), .last_song_flag_wire(last_song_flag_wire));

    assign data = {filtered_az, 12'b0, param_val};  //Data displayed on 7-segment displays

   assign SD_DAT[2] = 1;
   assign SD_DAT[3] = spiCS;
   assign SD_CMD = spiMosi;
   assign SD_RESET = 0;
```

```
   assign SD_SCK = spiClk;
   assign spiMiso = SD_DAT[0];
   assign SD_DAT[1] = 1;

   wire rd;
   reg wr = 0;
   reg [7:0] din = 0;
   wire [7:0] dout;
   wire byte_available;
   wire ready;
   wire ready_for_next_byte;
   wire [31:0] adr;
   wire [4:0] state;

   //SD SPI controller instantiation
   sd_controller sdcont(.cs(spiCS), .mosi(spiMosi), .miso(spiMiso),
                        .sclk(spiClk), .rd(rd), .wr(wr), .reset(rst),
                        .din(din), .dout(dout), .byte_available(byte_available),
                        .ready(ready), .address(adr),
                        .ready_for_next_byte(ready_for_next_byte), .clk(clock_12_5mhz),
                        .status(state));

   //Pulse used to start playing song
   wire new_song_pulse, skip_fwd_delayed_pulse, skip_backward_delayed_pulse;
   level_to_pulse new_song(.clk(clock_12_5mhz), .level(SW[0]), .pulse(new_song_pulse));
   delayed_level_to_pulse delayed_skip_fwd(.clk(clock_12_5mhz),
.level(nxt_song_flag_wire|SW[6]), .pulse(skip_fwd_delayed_pulse));
   delayed_level_to_pulse delayed_skip_backward(.clk(clock_12_5mhz),
.level(last_song_flag_wire|SW[5]), .pulse(skip_backward_delayed_pulse));

   wire load_to_bram;
   wire [8:0] addr_audio;

   assign LED[0] = SW[0]; //play audio
   assign LED[1] = vol_down_flag_wire; //volume down
   assign LED[2] = vol_up_flag_wire; //volume up
   assign LED[3] = rewind_flag_wire; //rewind
   assign LED[4] = ff_flag_wire; //fast forward
   assign LED[5] = last_song_flag_wire; //skip back
   assign LED[6] = nxt_song_flag_wire; //skip forward
   assign LED[7] = pause_flag_wire; //pause/play
   assign LED[15] = SW[15]; //reset


    reg [31:0] start_sectors [4:0];
    reg [31:0] end_sectors [4:0];
    reg [8:0] song_lengths [4:0]; //length of song in seconds
    reg [0:399] song_titles [4:0];
    reg [0:399] song_info [4:0];
    reg [10:0] song_move [4:0];  //~ how much to move the progress circle per second
    initial begin

        start_sectors[0] = 32'h6400; //counting to 60 seconds - at sector 50
        start_sectors[1] = 32'h2A3000; //silver bells - at sector 5400
        start_sectors[2] = 32'h00ABE000; //road trippin' - at sector 22000
```

```
        end_sectors[0] = 32'h297800; //counting to 60 seconds - at sector 5308
        end_sectors[1] = 32'hA90C00; //silver bells - at sector 21638
        end_sectors[2] = 32'h01353000; //road trippin' - at sector 39576

        song_lengths[0] = 9'd61;
        song_lengths[1] = 9'd189;
        song_lengths[2] = 9'd204;

        song_move[0] = 9'd5;
        song_move[1] = 9'd2;
        song_move[2] = 9'd2;

        song_titles[0] =  "Counting Up to 60 Seconds                          ";
        song_titles[1] =  "Silver Bells                                       ";
        song_titles[2] =  "Road Trippin'                                      ";

         song_info[0] =  "Counting One by One                                ";
         song_info[1] =  "Bing Crosby, Carol Richards                        ";
         song_info[2] =  {"Red Hot Chili Peppers",8'h07,"Californication             "};
     end

    reg [3:0] current_song_index = 4'd0;
    wire [3:0] new_song_index;
    wire actually_ffwded, actually_rewinded;
    //Reads audio from SD card using SPI controller and stores in audio buffers,
    //handled by audio handler

    sd_handler SD_to_buffer(.clk(clock_12_5mhz), .ready(ready),
                            .start_addr(start_sectors[current_song_index]),
                            .end_addr(end_sectors[current_song_index]),
                            .current_song_index(current_song_index),
                            .rewind(rewind_flag_wire|SW[3]),
                            .fast_forward(ff_flag_wire|SW[4]),
                            .skip_back(last_song_flag_wire|SW[5]),
                            .skip_forward(nxt_song_flag_wire|SW[6]),
                            .load_song(SW[0]),
                            .load_new_song_pulse(new_song_pulse),
                            .skip_fwd_delayed_pulse(skip_fwd_delayed_pulse),
                            .skip_backward_delayed_pulse(skip_backward_delayed_pulse),
                            .load_to_bram(load_to_bram),
                            .byte_available(byte_available),
                            .address_SD(adr),
                            .rea_SD(rd),
                            .byte_count(addr_audio),
                            .new_song_index(new_song_index),
                            .actually_rewinded(actually_rewinded),
                            .actually_ffwded(actually_ffwded)
                            );
    always @(*) current_song_index = new_song_index;

    wire [7:0] audio_display_level;
    //Handles the SD reads and storage in appropriate BRAM when required
    audio_handler SD_to_PWM(.clock_SD(clock_12_5mhz), .song_playing(SW[0]&&!
(pause_flag_wire|SW[7])),
```

```
                              .data_w(dout), .address_w(addr_audio),
.volume_up(SW[2]|vol_up_flag_wire), .volume_down(SW[1]|vol_down_flag_wire),
                              .load_bram(load_to_bram),
                              .PWM(AUD_PWM), .SD(AUD_SD), .clk_pwm(clock_104mhz),
.audio_display_level(audio_display_level)
   );

   ////////////////////////////////////////////////////////////////////////////
   //
   //
   // AUDIO HANDLER ENDS HERE
   //
   //
   ////////////////////////////////////////////////////////////////////////////
   wire jc0, jc1, jc2, jc3, jc4;
       bluetooth_handler (.clock(clock_12_5mhz), .vol_up_flag(vol_up_flag_wire),
.vol_down(vol_down_flag_wire),
                          .pause_flag(pause_flag_wire),
.nxt_song_flag(nxt_song_flag_wire), .last_song_flag(last_song_flag_wire),
                          .jc0(jc0), .jc1(jc1), .jc2(jc2), .jc3(jc3), .jc4(jc4));
   assign JC[0] = jc0?1'bz:0; //high Z if not active
   assign JC[1] = jc1?1'bz:0;
   assign JC[2] = jc2?1'bz:0;
   assign JC[3] = jc3?1'bz:0;
   assign JC[4] = jc4?1'bz:0;
   ////////////////////////////////////////////////////////////////////////////
   // sample Verilog to generate color bars or border

   wire [10:0] hcount;
   wire [9:0] vcount;
   xvga xvga1(.vclock(clock_65mhz),.hcount(hcount),.vcount(vcount),
         .hsync(hsync),.vsync(vsync),.blank(blank));
   wire [11:0] rgb;   // rgb is 12 bits

   wire border = (hcount==0 | hcount==1023 | vcount==0 | vcount==767 |
                  hcount == 512 | vcount == 384);

   wire [11:0] bubble_pixel, slider_pixel;
   wire [11:0] song0_title_pixel, song0_info_pixel, song1_title_pixel, song1_info_pixel,
song2_title_pixel, song2_info_pixel,
   song3_title_pixel, song3_info_pixel, song4_title_pixel, song4_info_pixel,
current_title_pixel, play_pause_pixel, vol_in_pixel, vol_icon_pixel;


   character vol_icon_disp (910, hcount, 469, vcount, 8'h01 , vol_icon_pixel);
   volume_blob #(.COLOR(12'h9cd)) volume_inside (.clk(clock_65mhz),
.x(930),.hcount(hcount), .y(473),
                                                 .vcount(vcount),
.change_in_x(audio_display_level), .pixel(vol_in_pixel));

   text song0_title_disp (clock_65mhz, 70, hcount, 100, vcount, song_titles[0],
song0_title_pixel);
   text #(.COLOR(12'h9cd)) song0_info_disp (clock_65mhz, 70, hcount, 120, vcount,
song_info[0], song0_info_pixel);
```

```
   text song1_title_disp (clock_65mhz, 70, hcount, 160, vcount, song_titles[1],
song1_title_pixel);
   text #(.COLOR(12'h9cd)) song1_info_disp (clock_65mhz, 70, hcount, 180, vcount,
song_info[1] , song1_info_pixel);

   text song2_title_disp (clock_65mhz, 70, hcount, 230, vcount, song_titles[2],
song2_title_pixel);
   text #(.COLOR(12'h9cd)) song2_info_disp (clock_65mhz, 70, hcount, 250, vcount,
song_info[2] , song2_info_pixel);

   text currently_playing_song_disp (clock_65mhz, 650, hcount, 350, vcount,
song_titles[current_song_index], current_title_pixel);

   play_pause_text #(.COLOR(12'h9cd)) play_pause_disp (clock_65mhz, 525, hcount, 470,
vcount, pause_flag_wire|SW[7], SW[0], play_pause_pixel);


   blob #(.WIDTH(350), .HEIGHT(5)) slider (550, hcount, 475, vcount, slider_pixel);

   wire circle_enabled;
   song_herz_clk song_second(clock_65mhz, pause_flag_wire, circle_enabled);

   reg [10:0] x_pos = 11'd546; //default x-pos

   reg prev_actually_rewinded_reg = 0;
   reg prev_actually_ffwded_reg = 0;

   reg [3:0] prev_song_index = 0;
   always @(posedge circle_enabled) begin
       if (SW[0] && ~(pause_flag_wire|SW[7])) begin //if loaded and not paused
           if ((x_pos +song_move[current_song_index]) <= 11'd893) begin //if rewind
actually happened
               if (prev_actually_rewinded_reg != actually_rewinded) begin
                 x_pos <= x_pos-(song_move[current_song_index]*10);
                 prev_actually_rewinded_reg <= actually_rewinded;
               end
               else if (prev_actually_ffwded_reg != actually_ffwded) begin //if ffwd
actually happened
                 x_pos <= x_pos+(song_move[current_song_index]*10);
                 prev_actually_ffwded_reg <= actually_ffwded;
               end
               else x_pos <= x_pos+song_move[current_song_index];
           end
        end
       else if (!SW[0]) begin //if not loaded, start from beginning of progress bar
       x_pos <= 11'd546;
       end

       if (prev_song_index!=current_song_index) begin //if new song, start from beginning
of progress bar
            prev_song_index <= current_song_index;
            x_pos <= 11'd546;
       end
   end
```

```
   circle_blob bubble (clock_65mhz, x_pos, hcount, 469, vcount, bubble_pixel);

   assign pixel = outline_pixel |slider_pixel | bubble_pixel | song0_title_pixel |
           song0_info_pixel | song1_title_pixel | song1_info_pixel | song2_title_pixel |
song2_info_pixel | song3_title_pixel | song3_info_pixel |
   song4_title_pixel | song4_info_pixel | current_title_pixel | play_pause_pixel |
vol_in_pixel | vol_icon_pixel;

   assign rgb = pixel;

// the following lines are required for the Nexys4 VGA circuit
   assign VGA_R = ~blank ? rgb[11:8]: 0;
   assign VGA_G = ~blank ? rgb[7:4] : 0;
   assign VGA_B = ~blank ? rgb[3:0] : 0;

   synchronize syn1 (.clk(clock_65mhz), .in(hsync), .out(hsync2));
   synchronize syn2 (.clk(clock_65mhz), .in(vsync), .out(vsync2));

   assign VGA_HS = ~hsync2;
   assign VGA_VS = ~vsync2;

endmodule
```

```
////////////////////////////////////////////////////////////////////////////////////
// Flags for all modules
// - Each flag will only flip when the action is performed. Stays high/low otherwise.
////////////////////////////////////////////////////////////////////////////////////


//////////////////////////////////////////////////////////////////////
// data_collection_FSM: collect x,y,z accel and gyro data in shift register
//////////////////////////////////////////////////////////////////////
module data_collection(input clk, enabled, data, look_for_start,
                       output reg [59:0] accel_array, gyro_array, output done, output reg
[3:0] state);
    reg [5:0] cycles = 0;
    reg [5:0] accel_index = 0;
    reg [5:0] gyro_index = 0;
    reg [3:0] current_state;
    reg internal_LFS = 0;  //Will remain high even after look_for_star signal pulses
    reg collecting = 0;
    reg [15:0] internal_x, internal_y;

    //States in FSM
    parameter [3:0] FIRST_BIT_COLLECT = 4'b0000;
    parameter [3:0] ACCEL_COLLECT = 4'b0001;
    parameter [3:0] GYRO_COLLECT = 4'b0010;
    parameter [3:0] RESET_STATE = 4'b0011;
    parameter [3:0] IDLE_STATE = 4'b0100;
    parameter [3:0] WAIT_STATE = 4'b0110;
    parameter [3:0] FINAL_STATE = 4'b1000;

    always @(posedge clk) begin
        state <= current_state;
        if(look_for_start && (current_state != ACCEL_COLLECT) && (current_state !=
GYRO_COLLECT)) internal_LFS <= 1;  //2ms high period satisfied wait for falling edge
        else if(internal_LFS && data == 0) begin  //Found falling edge
            internal_LFS <= 0;
            current_state <= FIRST_BIT_COLLECT;
        end
        else begin
            case(current_state)
                FIRST_BIT_COLLECT: begin  //Collect first bit which is middle of start bit
duration
                    if(enabled) begin
                        if(cycles < 7) begin
                            cycles <= cycles + 1;
                            current_state <= FIRST_BIT_COLLECT;
                        end
                        else if(cycles == 7) begin  //Middle of start bit, collect a data
bit
                            accel_array <= {data, accel_array[59:1]};
                            collecting <= 1;
                            cycles <= 0;
                            accel_index <= accel_index + 1;
                            current_state <= ACCEL_COLLECT;
                        end
```

```
                end
            end
            ACCEL_COLLECT: begin  //Collect bits 2-60
                if(enabled) begin
                    if(accel_index == 59) begin
                        current_state <= GYRO_COLLECT;
                        cycles <= 0;
                    end
                    else if(cycles < 15) begin
                        collecting <= 1;
                        cycles <= cycles + 1;
                        current_state <= ACCEL_COLLECT;
                    end
                    else if(cycles == 15) begin
                        accel_array <= {data, accel_array[59:1]};
                        collecting <= 1;
                        cycles <= 0;
                        accel_index <= accel_index + 1;
                        current_state <= ACCEL_COLLECT;
                    end
                end
            end
            GYRO_COLLECT: begin  //Collect bits 2-60
                if(enabled) begin
                    if(gyro_index == 59) current_state <= RESET_STATE;
                    else if(cycles < 15) begin
                        collecting <= 1;
                        cycles <= cycles + 1;
                        current_state <= GYRO_COLLECT;
                    end
                    else if(cycles == 15) begin
                        gyro_array <= {data, gyro_array[59:1]};
                        collecting <= 1;
                        cycles <= 0;
                        gyro_index <= gyro_index + 1;
                        current_state <= GYRO_COLLECT;
                    end
                end
            end
            RESET_STATE: begin
                internal_LFS <= 0;
                accel_index <= 0;
                gyro_index <= 0;
                collecting <= 0;
                current_state <= WAIT_STATE;
            end
            WAIT_STATE: begin  //Intermediate waiting state
                current_state <= WAIT_STATE;
            end
            default: ;
        endcase
    end
end
assign done = (gyro_index == 59) ? 1:0;  //If gyroscope shift register filled 59 bits,
all data ready
```

```
endmodule

//////////////////////////////////////////////////////////////////
// parsing_mod: Parses the 16 bits for x and y coordinates
//////////////////////////////////////////////////////////////////
module parser(input clk, done, input [59:0] accel_array, gyro_array,
                  output reg [15:0] accel_x, accel_y, accel_z, gyro_x, gyro_y, gyro_z);

   // Parse all axes of data without start/stop bits
    wire [15:0] internal_ax = {accel_array[18:11], accel_array[8:1]};
    wire [15:0] internal_ay = {accel_array[38:31], accel_array[28:21]};
    wire [15:0] internal_az = {accel_array[58:51], accel_array[48:41]};

    wire[15:0] internal_gx = {gyro_array[18:11], gyro_array[8:1]};
    wire[15:0] internal_gy = {gyro_array[38:31], gyro_array[28:21]};
    wire[15:0] internal_gz = {gyro_array[58:51], gyro_array[48:41]};

    reg [15:0] temp_ax, temp_gx, temp_ay, temp_gy, temp_az, temp_gz;
    always @(posedge clk) begin
        if(done) begin  //Collection done, start parsing out x,y acceleration + gyro vals
            //Differentiating SIGNS for all data
            if(internal_ax[15] == 1) begin
                temp_ax <= ~(internal_ax[15:0]- 16'd1);
                accel_x <= {4'hF,temp_ax[12:1]};  //Deciding whether value is positive or
negative
            end
            else accel_x <= {4'h0, internal_ax[12:1]};
            if(internal_ay[15] == 1) begin
                temp_ay <= ~(internal_ay[15:0]- 16'd1);
                accel_y <= {4'hF, temp_ay[11:0]};
            end
            else accel_y <= {4'h0, internal_ay[11:0]};
            if(internal_az[15] == 1) begin
                temp_az <= ~(internal_az[15:0]-1);
                accel_z <= {4'hF,temp_az[11:0]};  //Deciding whether value is positive or
negative
            end
            else accel_z <= {4'h0, internal_az[11:0]};

            //Differentiating SIGNS in gyro data
            if(internal_gx[15] == 1) begin
                temp_gx <= ~(internal_gx[15:0]-1);
                gyro_x <= {4'hF,temp_gx[15:4]};  //Deciding whether value is positive or
negative
            end
            else gyro_x <= {4'h0, internal_gx[15:4]};
            if(internal_gy[15] == 1) begin
                temp_gy <= ~(internal_gy[15:0]-1);
                gyro_y <= {4'hF, temp_gy[15:4]};
            end
            else gyro_y <= {4'h0, internal_gy[15:4]};
            if(internal_gz[15] == 1) begin
                temp_gz <= ~(internal_gz[15:0]-1);
                gyro_z <= {4'hF,temp_gz[15:4]};  //Deciding whether value is positive or
negative
```

```
            end
            else gyro_z <= {4'h0, internal_gz[15:4]};
        end
    end
endmodule

////////////////////////////////////////////////////////////////////////////////////
/////
// filter_mod: Filtering module that keeps running average of acceleration and gyro values
////////////////////////////////////////////////////////////////////////////////////
////
module filter_mod(input clk, vsync, clean_BTNC, input [15:0] accel_x, accel_y, accel_z,
gyro_x, gyro_y, gyro_z,
                  output reg [15:0] filtered_ax, filtered_ay, filtered_az, filtered_gx,
filtered_gy, filtered_gz);
    reg [15:0] prev_ax = 0;
    reg [15:0] prev_ay =0;
    reg [15:0] prev_az = 0;
    reg [15:0] prev_gx = 0;
    reg [15:0] prev_gy = 0;
    reg [15:0] prev_gz = 0;
    always @(posedge clk) begin
        // Take running average of data
        if (filtered_ax[15:12] != accel_x[15:12]) begin  //sign(Incoming value) !=
sign(previous value), treat accordingly.
            if (filtered_ax[11:0] > accel_x[11:0]) filtered_ax <= {filtered_ax[15:12],
(filtered_ax[11:0] - accel_x[11:0]) >> 1};
            else filtered_ax <= {accel_x[15:12], (accel_x[11:0] - filtered_ax[11:0]) >>
1};
        end
        else filtered_ax <= {filtered_ax[15:12], (filtered_ax[11:0] + accel_x[11:0]) >>
1};  //sign(Incoming value) == sign(previous value) just add.

        if (filtered_ay[15:12] != accel_y[15:12]) begin
            if (filtered_ay[11:0] > accel_y[11:0]) filtered_ay <= {filtered_ay[15:12],
(filtered_ay[11:0] - accel_y[11:0]) >> 1};
            else filtered_ay <= {accel_y[15:12], (accel_y[11:0] - filtered_ay[11:0]) >>
1};
        end
        else filtered_ay <= {filtered_ay[15:12], (filtered_ay[11:0] + accel_y[11:0]) >>
1};

        if (filtered_az[15:12] != accel_z[15:12]) begin
            if (filtered_az[11:0] > accel_z[11:0]) filtered_az <= {filtered_az[15:12],
(filtered_az[11:0] - accel_z[11:0]) >> 1};
            else filtered_az <= {accel_z[15:12], (accel_z[11:0] - filtered_az[11:0]) >>
1};
        end
        else filtered_az <= {filtered_az[15:12], (filtered_az[11:0] + accel_z[11:0]) >>
1};

        if (filtered_gx[15:12] != gyro_x[15:12]) begin
                    if (filtered_gx[11:0] > gyro_x[11:0]) filtered_gx <=
{filtered_gx[15:12], (filtered_gx[11:0] - gyro_x[11:0]) >> 1};
                    else filtered_gx <= {gyro_x[15:12], (gyro_x[11:0] - filtered_gx[11:0])
```

```
>> 1};
                end
                else filtered_gx <= {filtered_gx[15:12], (filtered_gx[11:0] +
gyro_x[11:0]) >> 1};
        filtered_gy <= gyro_y;
        filtered_gz <= gyro_z;
    end
endmodule

//////////////////////////////////////////////////////////////////////
// Gesture_Recognition: Look at filtered values to determine gesture done.
// Output: Parameter value hard coded to specific action module AND assert
//         signal sequence denoting which action modules should activate.
//////////////////////////////////////////////////////////////////////
module gesture_rec(input clk, btnc_clean,
                   input [15:0] filtered_ax, filtered_ay, filtered_az, filtered_gx,
filtered_gy, filtered_gz,
                   output reg [3:0] param_val, output reg [6:0] assert_seq, output reg
[5:0] active_axes,
                   output [2:0] state, output[2:0] performed,
                   output vol_up_flag_wire, vol_down_flag_wire, pause_flag_wire,
rewind_flag_wire, ff_flag_wire, nxt_song_flag_wire, last_song_flag_wire);

   reg vol_up_flag, vol_down_flag, pause_flag, rewind_flag, ff_flag, nxt_song_flag,
last_song_flag;

   //Experimentally determined threshold values
   parameter [11:0] gx_threshold = 12'd100;
   parameter [11:0] az_threshold = 12'd100;
   parameter [11:0] ay_threshold = 12'd30;
   parameter [11:0] ax_threshold = 12'd15;
   parameter [11:0] play_threshold = 12'd100;
   parameter [11:0] next_song_threshold = 12'd10;
   parameter[11:0] ff_threshold = 12'h100;

   //STATES
   parameter IDLE_STATE = 3'b000;
   parameter READING_STATE = 3'b001;
   parameter PROCESS_STATE = 3'b011;
   parameter DOUBLE_PROCESS = 3'b010;
   parameter RESET_STATE = 3'b100;

   //GESTURES AXES
   parameter VOLUME_UP = 6'b100000;
   parameter VOLUME_DOWN = 6'b000001;
   parameter PLAY_PAUSE = 6'b000011;
   parameter NEXT_SONG = 6'b000100;
   parameter FAST_FORWARD = 6'b000111;
   parameter LAST_SONG = 6'b100100;
   parameter REWIND = 6'b010101;

   reg [2:0] current_state = IDLE_STATE;
   reg prev_btnc = 1'b0;
   reg [25:0] index = 0;
   reg [25:0] index2 = 0;
```

```
   reg [25:0] index3 = 0;
   reg times_performed = 0;  //Register to determine whether gesture was fast forward/next
song OR rewind/prev song

   always @(posedge clk) begin
        if (btnc_clean == 1 && prev_btnc == 0) begin   //When BTNC pressed, starts 1
second timer for user to perform gesture
            current_state <= READING_STATE;
            prev_btnc <= 1;
        end

        else begin
            case(current_state)
                IDLE_STATE: current_state <= IDLE_STATE;   //Wait suntil next button press
to start reading
                READING_STATE: begin  //Starts 1 second timer to listen for gesture
                    if(index == 26'd64000000) current_state <= RESET_STATE;

                    else if(filtered_az[15:12] == 4'b0000 && filtered_gx[11:0] >=
gx_threshold && filtered_ax[11:0] < ax_threshold) begin  // ** VOLUME UP **
                        active_axes <= VOLUME_UP;
                        current_state <= PROCESS_STATE;
                    end

                    else if(filtered_az[15:12] == 4'b1111 && filtered_gx[11:0] >=
gx_threshold && filtered_ax[11:0] < ax_threshold) begin  // ** VOLUME DOWN **
                        active_axes <= VOLUME_DOWN;
                        current_state <= PROCESS_STATE;
                    end

                    else if(filtered_ay[11:0] >= ay_threshold && filtered_ax[11:0] < 12'd5
&& filtered_az[11:0] <= 12'd5) begin  // ** PLAY/PAUSE **
                        active_axes <= PLAY_PAUSE;
                        current_state <= PROCESS_STATE;
                    end

                    else if(filtered_gx[11:0] >= gx_threshold && filtered_gx[15:12] ==
4'hF && filtered_ax[11:0] >= ax_threshold && filtered_az[11:0] >= next_song_threshold)
begin   //  ** NEXT SONG **
                        times_performed <= 0;
                        current_state <= DOUBLE_PROCESS;
                    end

                    else if(filtered_ax[11:0] >= ax_threshold && filtered_gx[15:12] ==
4'h0 && filtered_az[11:0] >= next_song_threshold && filtered_gx[11:0] >= gx_threshold)
begin  // ** LAST SONG **
                        times_performed <= 1;
                        current_state <= DOUBLE_PROCESS;
                    end

                    else begin
                        index <= index + 1;
                        current_state <= READING_STATE;
                    end
                end
```

```
                PROCESS_STATE: begin  //Reading values done, process values to verify
gesture completed
                    case(active_axes)
                        VOLUME_UP: begin
                            if(index2 == 26'd64000000) current_state <= RESET_STATE;
                            // ** VOLUME UP **
                            else if (filtered_gx[11:0] < gx_threshold &&
filtered_az[15:12] == 4'h0) begin  //DEACTIVATED
                                vol_up_flag <= ~vol_up_flag;
                                current_state <= RESET_STATE;
                            end
                        end

                        VOLUME_DOWN: begin  // ** VOLUME DOWN **
                            if(index2 == 26'd64000000) current_state <= RESET_STATE;
                            else if(filtered_az[15:12] == 4'b1111 && filtered_gx[11:0] <
gx_threshold) begin  //DEACTIVATED
                                vol_down_flag <= ~vol_down_flag;
                                current_state <= RESET_STATE;
                            end
                            else index2 <= index2 + 1;
                        end
                        PLAY_PAUSE: begin
                            if(index2 == 26'd64000000) current_state <= RESET_STATE;
                            // **PLAY/PAUSE **
                            else if(filtered_ay[11:0] < ay_threshold && filtered_az[11:0]
< az_threshold)  begin  //DEACTIVATED
                                pause_flag <= ~pause_flag;
                                current_state <= RESET_STATE;
                            end
                            else index2 <= index2 + 1;
                        end

                        LAST_SONG: begin
                            if(index2 == 26'd64000000) current_state <= RESET_STATE;
                            // ** LAST SONG **
                            else if(filtered_gx[11:0] < gx_threshold && filtered_gx[15:12]
== 4'h0) begin  //DEACTIVATED
                                last_song_flag <= ~last_song_flag;
                                current_state <= RESET_STATE;
                            end
                            else index2 <= index2 + 1;
                        end

                        FAST_FORWARD: begin
                            if(index2 == 26'd64000000) current_state <= RESET_STATE;
                            // ** FAST FORWARD **
                            else if(filtered_az[11:0] < ff_threshold) begin  //DEACTIVATED
                                ff_flag <= ~ff_flag;
                                current_state <= RESET_STATE;
                            end
                            else index2 <= index2 + 1;
                        end

                        NEXT_SONG: begin  // ** NEXT SONG **
```

```
                            if(index2 == 26'd64000000) current_state <= RESET_STATE;
                            else if(filtered_gx[11:0] < gx_threshold && filtered_gx[15:12]
== 4'hF) begin  //DEACTIVATED
                                    nxt_song_flag <= ~nxt_song_flag;
                                    current_state <= RESET_STATE;
                            end
                            else index2 <= index2 + 1;
                        end

                        REWIND: begin  // ** REWIND **
                            if(index2 == 26'd64000000) current_state <= RESET_STATE;
                            else if(filtered_gx[11:0] < gx_threshold && filtered_gx[15:12]
== 4'h0) begin
                                    rewind_flag <= ~rewind_flag;
                                    current_state <= RESET_STATE;
                            end
                            else index2 <= index2 + 1;
                        end
                        default: begin
                            index2 <= index2 + 1;
                            current_state <= RESET_STATE;
                        end
                    endcase
                end
                DOUBLE_PROCESS: begin
                    if(index3 == 26'd24000000 && times_performed == 0) begin
                        if(filtered_ax >= ax_threshold) begin  // ** FAST FORWARD **
                            active_axes <= FAST_FORWARD;
                            current_state <= PROCESS_STATE;
                        end
                        else begin
                            active_axes <= NEXT_SONG;  // ** NEXT SONG **
                            current_state <= PROCESS_STATE;
                        end
                    end
                    else if(index3 == 26'd24000000 && times_performed == 1) begin
                        if(filtered_ax >= ax_threshold) begin  // ** REWIND **
                            active_axes <= REWIND;
                            current_state <= PROCESS_STATE;
                        end
                        else begin  // ** LAST SONG **
                            active_axes <= LAST_SONG;
                            current_state <= PROCESS_STATE;
                        end
                    end
                    else index3 <= index3 + 1;
                end
                RESET_STATE: begin
                    prev_btnc <= 0;
                    index <= 0;
                    index2 <= 0;
                    index3 <= 0;
                    times_performed <= 0;
                    active_axes <= 6'b000000;
                    current_state <= IDLE_STATE;
```

```
                end
             endcase
        end
   end
   assign state = current_state;
   assign performed = times_performed;

   assign vol_up_flag_wire = vol_up_flag;
   assign vol_down_flag_wire = vol_down_flag;
   assign pause_flag_wire = pause_flag;
   assign rewind_flag_wire = rewind_flag;
   assign ff_flag_wire = ff_flag;
   assign nxt_song_flag_wire = nxt_song_flag;
   assign last_song_flag_wire = last_song_flag;

endmodule

//////////////////////////////////////////////////////////////////////////////////
////
// recovery_period: Wait for 2ms period of dataline high before starting to look for start
bit
//////////////////////////////////////////////////////////////////////////////////
////
module recovery_period(input clk, data, output look_for_start);
    reg [16:0] counter = 0;
    reg dataline_high = 1;
    always @(posedge clk) begin
        if(counter < 130000 && data && dataline_high) begin   //Checks that dataline is
high for 130,000 cycles (130,000/65 million = 2ms)
            counter <= counter + 1;
            dataline_high <= dataline_high & data;
        end
        else if(counter == 130000 && dataline_high) begin
            counter <= 0;
        end
        else if(!dataline_high) begin
            counter <= 0;
        end
    end
    assign look_for_start = (counter == 130000);
endmodule
```

```
////////////////////////////////////////////////////////////////////////////////
//
//  SD SPI controller
//  SD Card controller module. Allows reading from and writing to a microSD card
//  through SPI mode.
//
////////////////////////////////////////////////////////////////////////////////

module sd_controller(
    output reg cs, // Connect to SD_DAT[3].
    output mosi, // Connect to SD_CMD.
    input miso, // Connect to SD_DAT[0].
    output sclk, // Connect to SD_SCK.
                 // For SPI mode, SD_DAT[2] and SD_DAT[1] should be held HIGH.
                 // SD_RESET should be held LOW.

    input rd,   // Read-enable. When [ready] is HIGH, asseting [rd] will
                // begin a 512-byte READ operation at [address].
                // [byte_available] will transition HIGH as a new byte has been
                // read from the SD card. The byte is presented on [dout].
    output reg [7:0] dout, // Data output for READ operation.
    output reg byte_available, // A new byte has been presented on [dout].

    input wr,   // Write-enable. When [ready] is HIGH, asserting [wr] will
                // begin a 512-byte WRITE operation at [address].
                // [ready_for_next_byte] will transition HIGH to request that
                // the next byte to be written should be presentaed on [din].
    input [7:0] din, // Data input for WRITE operation.
    output reg ready_for_next_byte, // A new byte should be presented on [din].

    input reset, // Resets controller on assertion.
    output ready, // HIGH if the SD card is ready for a read or write operation.
    input [31:0] address,   // Memory address for read/write operation. This MUST
                            // be a multiple of 512 bytes, due to SD sectoring.
    input clk,  // 25 MHz clock.
    output [4:0] status // For debug purposes: Current state of controller.
);

    parameter RST = 0;
    parameter INIT = 1;
    parameter CMD0 = 2;
    parameter CMD55 = 3;
    parameter CMD41 = 4;
    parameter POLL_CMD = 5;

    parameter IDLE = 6;
    parameter READ_BLOCK = 7;
    parameter READ_BLOCK_WAIT = 8;
    parameter READ_BLOCK_DATA = 9;
    parameter READ_BLOCK_CRC = 10;
    parameter SEND_CMD = 11;
    parameter RECEIVE_BYTE_WAIT = 12;
    parameter RECEIVE_BYTE = 13;
    parameter WRITE_BLOCK_CMD = 14;
```

```
parameter WRITE_BLOCK_INIT = 15;
parameter WRITE_BLOCK_DATA = 16;
parameter WRITE_BLOCK_BYTE = 17;
parameter WRITE_BLOCK_WAIT = 18;

parameter WRITE_DATA_SIZE = 515;

reg [4:0] state = RST;
assign status = state;
reg [4:0] return_state;
reg sclk_sig = 0;
reg [55:0] cmd_out;
reg [7:0] recv_data;
reg cmd_mode = 1;
reg [7:0] data_sig = 8'hFF;

reg [9:0] byte_counter;
reg [9:0] bit_counter;

reg [26:0] boot_counter = 27'd100_000_000;
always @(posedge clk) begin
    if(reset == 1) begin
        state <= RST;
        sclk_sig <= 0;
        boot_counter <= 27'd100_000_000;
    end
    else begin
        case(state)
            RST: begin
                if(boot_counter == 0) begin
                    sclk_sig <= 0;
                    cmd_out <= {56{1'b1}};
                    byte_counter <= 0;
                    byte_available <= 0;
                    ready_for_next_byte <= 0;
                    cmd_mode <= 1;
                    bit_counter <= 160;
                    cs <= 1;
                    state <= INIT;
                end
                else begin
                    boot_counter <= boot_counter - 1;
                end
            end
            INIT: begin
                if(bit_counter == 0) begin
                    cs <= 0;
                    state <= CMD0;
                end
                else begin
                    bit_counter <= bit_counter - 1;
                    sclk_sig <= ~sclk_sig;
                end
            end
            CMD0: begin
```

```
    cmd_out <= 56'hFF_40_00_00_00_00_95;
    bit_counter <= 55;
    return_state <= CMD55;
    state <= SEND_CMD;
end
CMD55: begin
    cmd_out <= 56'hFF_77_00_00_00_00_01;
    bit_counter <= 55;
    return_state <= CMD41;
    state <= SEND_CMD;
end
CMD41: begin
    cmd_out <= 56'hFF_69_00_00_00_00_01;
    bit_counter <= 55;
    return_state <= POLL_CMD;
    state <= SEND_CMD;
end
POLL_CMD: begin
    if(recv_data[0] == 0) begin
        state <= IDLE;
    end
    else begin
        state <= CMD55;
    end
end
IDLE: begin
    if(rd == 1) begin
        state <= READ_BLOCK;
    end
    else if(wr == 1) begin
        state <= WRITE_BLOCK_CMD;
    end
    else begin
        state <= IDLE;
    end
end
READ_BLOCK: begin
    cmd_out <= {16'hFF_51, address, 8'hFF};
    bit_counter <= 55;
    return_state <= READ_BLOCK_WAIT;
    state <= SEND_CMD;
end
READ_BLOCK_WAIT: begin
    if(sclk_sig == 1 && miso == 0) begin
        byte_counter <= 511;
        bit_counter <= 7;
        return_state <= READ_BLOCK_DATA;
        state <= RECEIVE_BYTE;
    end
    sclk_sig <= ~sclk_sig;
end
READ_BLOCK_DATA: begin
    dout <= recv_data;
    byte_available <= 1;
    if (byte_counter == 0) begin
```

```
            bit_counter <= 7;
            return_state <= READ_BLOCK_CRC;
            state <= RECEIVE_BYTE;
        end
        else begin
            byte_counter <= byte_counter - 1;
            return_state <= READ_BLOCK_DATA;
            bit_counter <= 7;
            state <= RECEIVE_BYTE;
        end
    end
    READ_BLOCK_CRC: begin
        bit_counter <= 7;
        return_state <= IDLE;
        state <= RECEIVE_BYTE;
    end
    SEND_CMD: begin
        if (sclk_sig == 1) begin
            if (bit_counter == 0) begin
                state <= RECEIVE_BYTE_WAIT;
            end
            else begin
                bit_counter <= bit_counter - 1;
                cmd_out <= {cmd_out[54:0], 1'b1};
            end
        end
        sclk_sig <= ~sclk_sig;
    end
    RECEIVE_BYTE_WAIT: begin
        if (sclk_sig == 1) begin
            if (miso == 0) begin
                recv_data <= 0;
                bit_counter <= 6;
                state <= RECEIVE_BYTE;
            end
        end
        sclk_sig <= ~sclk_sig;
    end
    RECEIVE_BYTE: begin
        byte_available <= 0;
        if (sclk_sig == 1) begin
            recv_data <= {recv_data[6:0], miso};
            if (bit_counter == 0) begin
                state <= return_state;
            end
            else begin
                bit_counter <= bit_counter - 1;
            end
        end
        sclk_sig <= ~sclk_sig;
    end
    WRITE_BLOCK_CMD: begin
        cmd_out <= {16'hFF_58, address, 8'hFF};
        bit_counter <= 55;
        return_state <= WRITE_BLOCK_INIT;
```

```
                state <= SEND_CMD;
            ready_for_next_byte <= 1;
            end
            WRITE_BLOCK_INIT: begin
                cmd_mode <= 0;
                byte_counter <= WRITE_DATA_SIZE;
                state <= WRITE_BLOCK_DATA;
                ready_for_next_byte <= 0;
            end
            WRITE_BLOCK_DATA: begin
                if (byte_counter == 0) begin
                    state <= RECEIVE_BYTE_WAIT;
                    return_state <= WRITE_BLOCK_WAIT;
                end
                else begin
                    if ((byte_counter == 2) || (byte_counter == 1)) begin
                        data_sig <= 8'hFF;
                    end
                    else if (byte_counter == WRITE_DATA_SIZE) begin
                        data_sig <= 8'hFE;
                    end
                    else begin
                        data_sig <= din;
                        ready_for_next_byte <= 1;
                    end
                    bit_counter <= 7;
                    state <= WRITE_BLOCK_BYTE;
                    byte_counter <= byte_counter - 1;
                end
            end
            WRITE_BLOCK_BYTE: begin
                if (sclk_sig == 1) begin
                    if (bit_counter == 0) begin
                        state <= WRITE_BLOCK_DATA;
                        ready_for_next_byte <= 0;
                    end
                    else begin
                        data_sig <= {data_sig[6:0], 1'b1};
                        bit_counter <= bit_counter - 1;
                    end;
                end;
                sclk_sig <= ~sclk_sig;
            end
            WRITE_BLOCK_WAIT: begin
                if (sclk_sig == 1) begin
                    if (miso == 1) begin
                        state <= IDLE;
                        cmd_mode <= 1;
                    end
                end
                sclk_sig = ~sclk_sig;
            end
        endcase
    end
end
```

```
    assign sclk = sclk_sig;
    assign mosi = cmd_mode ? cmd_out[55] : data_sig[7];
    assign ready = (state == IDLE);
endmodule
```

```
//////////////////////////////////////////////////////////////////////////////////
//
//  SD Handler
//  Reads audio from SD card using SPI controller and stores in two different
//  audio buffers. SD read when all samples in one of the brams played
//
//////////////////////////////////////////////////////////////////////////////////
module sd_handler(input clk, ready,
                  input [31:0] start_addr,
                  input [31:0] end_addr,
                  input [3:0] current_song_index,
                  input rewind,
                  input fast_forward,
                  input skip_back,
                  input skip_forward,
                  input load_song,
                  input load_new_song_pulse,
                  input skip_fwd_delayed_pulse,
                  input skip_backward_delayed_pulse,
                  input load_to_bram,
                  input byte_available,
                  output reg [31:0] address_SD,
                  output reg rea_SD,
                  output reg [8:0] byte_count, //Used as audio buffer address
                  output [3:0] new_song_index,
                  output actually_rewinded,
                  output actually_ffwded
                  );
          parameter NUM_SONGS = 3;
          reg prev_rewind = 0; //rewind
          reg prev_ffwd = 0; //fast forward
          reg prev_skip_back = 0; //skip back
          reg prev_skip_ffwd = 0; //skip forward

          reg get_song = 0;
          reg loading_bram = 0;

          reg [31:0] start_address = 0;
          reg [3:0] new_song_index_reg;
          reg actually_rewinded_reg=0;
          reg actually_ffwded_reg=0;

          always @(posedge clk)begin
            //READING AUDIO

            if(load_new_song_pulse & ready)begin
                get_song <= 1;
                address_SD <= start_addr;
                rea_SD <= 1;
                byte_count <= 0;
            end
            else if(skip_fwd_delayed_pulse & ready)begin
                get_song <= 1;
                address_SD <= start_addr;
```

```
                rea_SD <= 1;
                byte_count <= 0;
            end
            else if(skip_backward_delayed_pulse & ready)begin
                get_song <= 1;
                address_SD <= start_addr;
                rea_SD <= 1;
                byte_count <= 0;
            end
            else if (get_song) begin
                //If song over, go to next song
                if (address_SD == end_addr) begin
                    if (current_song_index == NUM_SONGS-1) new_song_index_reg <= 0;
                    else new_song_index_reg <= current_song_index+1;

                end
                if(load_song)begin
                    //Start reading from SD when one of audio buffers played out
                    if(load_to_bram)begin //Assume load_to_bram is a pulse
                        rea_SD <= 1;
                        loading_bram <= 1;
                    end

                    if(loading_bram)begin
                        if(byte_available)begin
                            byte_count <= byte_count + 1; //Keeps track of position in
block, also used as song address
                            if(byte_count == 9'd511)begin //Detects end of block
                                address_SD <= address_SD + 512; //Read next block
                                byte_count <= 0;
                                rea_SD <= 0;
                                loading_bram <= 0;
                            end
                        end
                    end
                end
                else get_song <= 0;
            end

            if (prev_rewind != rewind) begin
                if (address_SD >= start_addr + 443392) begin //corresponds to ~10 seconds
                    address_SD <= address_SD - 443392; //only rewind if at least 10
seconds have passed
                    actually_rewinded_reg <= ~actually_rewinded_reg;
                end

                prev_rewind <= ~prev_rewind;
            end
            if (prev_ffwd != fast_forward) begin
                if (address_SD + 443392 <= end_addr) begin
                    address_SD <= address_SD + 443392; //only ffwd if there are more than
10 seconds remaining
                    actually_ffwded_reg <= ~actually_ffwded_reg;
                end
```

```
                prev_ffwd <= ~prev_ffwd;

            end

            if (prev_skip_back != skip_back) begin
                if (current_song_index == 0) new_song_index_reg <= NUM_SONGS-1;
                else new_song_index_reg <= current_song_index-1;

                prev_skip_back <= ~prev_skip_back;
            end
            if (prev_skip_ffwd != skip_forward) begin
                if (current_song_index == NUM_SONGS-1) new_song_index_reg <= 0;
                else new_song_index_reg <= current_song_index+1;

                prev_skip_ffwd <= ~prev_skip_ffwd;
            end
        end

        assign new_song_index = new_song_index_reg;
        assign actually_ffwded = actually_ffwded_reg;
        assign actually_rewinded = actually_rewinded_reg;


endmodule
```

```
//////////////////////////////////////////////////////////////////////////////////
//
//  Audio Handler
//  Reads audio from SD card using SPI controller and stores in two different
//  audio buffers. SD read when all samples in one of the brams played
//
//////////////////////////////////////////////////////////////////////////////////
module audio_handler(input clock_SD, clk_pwm,
                     input song_playing,
                     input [7:0] data_w,
                     input [8:0] address_w,
                     input volume_up,
                     input volume_down,
                     output reg load_bram,
                     output PWM,
                     output SD,
                     output [7:0] audio_display_level
                    );


    wire song_playing_pulse;
    wire load_bram_pulse;
    reg clock_audio = 0;
    reg [8:0] counter_clk_audio = 0;
    reg wea_1 = 0;
    reg wea_2 = 0;

    reg [8:0] addr_sample;
    reg [7:0] sample;
    wire [7:0] sample1;
    wire [7:0] sample2;
    reg bram1_full;
    reg bram2_full;

    parameter DEFAULT_BIT_SHIFT = 2; //default volume level
    reg [7:0] current_bit_shift = DEFAULT_BIT_SHIFT;
    //When song_play is asserted, convert it to pulse
    level_to_pulse
song_playing_p(.clk(clock_SD),.level(song_playing),.pulse(song_playing_pulse));
    level_to_pulse load_bram_p(.clk(clock_SD),.level(load_bram),.pulse(load_bram_pulse));

    reg prev_SW_1 = 0;
    reg prev_SW_2 = 0;

    always @(posedge clock_SD)begin
        if(song_playing_pulse)begin //Initially, both brams empty
            bram1_full <= 0;
            bram2_full <= 0;
            addr_sample <= 0;
        end

        //Generate 44.326 KHZ clock
        if (counter_clk_audio < 141) counter_clk_audio <= counter_clk_audio + 1;
        else begin
```

```
            counter_clk_audio <= 0;
            clock_audio <= ~clock_audio;
            if (clock_audio) begin //At posedge, increment sample address
                if (song_playing & bram1_full & bram2_full) begin
                    addr_sample <= addr_sample + 1;
                    if(wea_1) sample <= sample2; //If bram1 being written to, play from
bram2
                    else if (wea_2) sample <= sample1; //If bram2 being written to, play
from bram1
                end
            end
        end

        //Tells SD_handler to read block from SD and write to buffer
        if( (~bram1_full || ~bram2_full) || (addr_sample == 9'd511))load_bram <= 1;
        else load_bram <= 0;

        //Write enable logic
        if (~bram1_full & ~bram2_full) begin
            wea_1 <= 1;
            if(address_w == 9'd511) bram1_full <= 1;
        end
        else if (bram1_full & ~bram2_full )begin
            wea_1 <= 0;
            wea_2 <= 1;
            if(address_w == 9'd511) bram2_full <= 1;
        end
        else if (load_bram_pulse & bram1_full & bram2_full) begin
            wea_1 <= wea_2;
            wea_2 <= wea_1;
        end

        //decrease volume if change in volume_down_flag
        if ((prev_SW_1!=volume_down) && (current_bit_shift <= 3))begin //set upper limit
of range to 4 inclusive
            prev_SW_1 = !prev_SW_1;
            current_bit_shift <= current_bit_shift + 1;
        end
        //increase volume if change in volume_up_flag
        if (prev_SW_2!=volume_up && (current_bit_shift > 0))begin //cannot be less than 0
            prev_SW_2 = !prev_SW_2;
            current_bit_shift <= current_bit_shift - 1;
        end

    end

    //Audio buffers
    audio_buffer buffer1(.clka(clock_SD), .wea(wea_1), .addra(address_w), .dina(data_w),
.clkb(clock_SD), .addrb(addr_sample), .doutb(sample1));
    audio_buffer buffer2(.clka(clock_SD), .wea(wea_2), .addra(address_w), .dina(data_w),
.clkb(clock_SD), .addrb(addr_sample), .doutb(sample2));


    //Output to PWM
    pwm10 pwm_out( .clk(clk_pwm),
```

```
                    .PWM_in( sample >> current_bit_shift),
                    .PWM_out(PWM),
                    .PWM_sd(SD)
                    );
    assign audio_display_level = (7'd4-current_bit_shift) << 2; //for displaying to the
monitor

endmodule
```

```
//////////////////////////////////////////////
Interfaces with XY-502B Bluetooth Amplifier.
Converts changes in the wires to a 1/4 second
pulse -- high when not active and low when active.
//////////////////////////////////////////////

module bluetooth_handler (input clock, input vol_up_flag, input vol_down,
                          input pause_flag, input nxt_song_flag, input last_song_flag,
                          output jc0, jc1, jc2, jc3, jc4);


    ble_level_to_pulse (.clk(clock), .level(vol_up_flag), .pulse(jc0));
    ble_level_to_pulse (.clk(clock), .level(vol_down), .pulse(jc1));
    ble_level_to_pulse (.clk(clock), .level(pause_flag), .pulse(jc2));
    ble_level_to_pulse (.clk(clock), .level(nxt_song_flag), .pulse(jc3));
    ble_level_to_pulse (.clk(clock), .level(last_song_flag), .pulse(jc4));

endmodule
```

```
//////////////////////////////////////////////////////////////////////////////
//
// PWM Module
// 10 bit PWM audio out is reasonable because otherwise, the PWM frequency would
// drop close to the audible and unfiltered range. 10bits -> 104Mhz/2^10=101.6Khz
//
//////////////////////////////////////////////////////////////////////////////
module pwm10 (
    input wire clk,
    input wire [9:0] PWM_in,
    output reg PWM_out,
    output wire PWM_sd
    );

    reg [9:0] new_pwm=0;
    reg [9:0] PWM_ramp=0;
    always @(posedge clk) begin
        if (PWM_ramp==0) new_pwm <= PWM_in;
        PWM_ramp <= PWM_ramp + 1'b1;
        PWM_out <= (new_pwm>PWM_ramp);
    end
    assign PWM_sd = 1;
endmodule


//////////////////////////////////////////////////////////////////////////////
//
// pulse synchronizer Module
//
//////////////////////////////////////////////////////////////////////////////
module synchronize #(parameter NSYNC = 2)  // number of sync flops.  must be >= 2
                   (input clk,in,
                    output reg out);

  reg [NSYNC-2:0] sync;

  always @ (posedge clk)
  begin
    {out,sync} <= {sync[NSYNC-2:0],in};
  end
endmodule

//////////////////////////////////////////////////////////////////////////////
//
// Switch Debounce Module
// use your system clock for the clock input
// to produce a synchronous, debounced output
//
//////////////////////////////////////////////////////////////////////////////
module debounce #(parameter DELAY=270000)   // .01 sec with a 27Mhz clock    NOT BEING
USED RIGHT NOW
          (input reset, clock, noisy,
           output reg clean);
```

```
   reg [18:0] count;
   reg new;

   always @(posedge clock)
     if (reset)
       begin
    count <= 0;
    new <= noisy;
    clean <= noisy;
       end
     else if (noisy != new)
       begin
    new <= noisy;
    count <= 0;
       end
     else if (count == DELAY)
       clean <= new;
     else
       count <= count+1;
endmodule

////////////////////////////////////////////////////////////////////////////////
//
// Level to Pulse
// Converts a level to a pulse
//
////////////////////////////////////////////////////////////////////////////////

module level_to_pulse (
    input wire clk,
    input wire level,
    output wire pulse);

    reg last_level;
    always @(posedge clk) begin
        last_level <= level;
    end
    assign pulse = level & ~last_level;

endmodule

////////////////////////////////////////////////////////////////////////////////
// Converts a change in level to a 1/4 second low pulse.
////////////////////////////////////////////////////////////////////////////////

module ble_level_to_pulse (
    input wire clk,
    input wire level,
    output wire pulse);

    reg last_level;
    reg [21:0] count=0;
    always @(posedge clk) begin
        if (level == last_level)
            count<=22'b0;
```

```
        else
            count<=count+1;

        if (count == 3125000)
            last_level <= level;
    end
    assign pulse = level^~last_level; //xnor - will be low when both are different (low =
on for bluetooth)

endmodule

////////////////////////////////////////////////////////////////////////////////
// Change in level generates a pulse delayed by a clock cycle.
////////////////////////////////////////////////////////////////////////////////

module delayed_level_to_pulse (
    input wire clk,
    input wire level,
    output wire pulse);

    reg last_level = 0;
    reg pulse_reg = 0;
    always @(posedge clk) begin
        last_level <= level;
        pulse_reg <= level ^ last_level; //xor - as long as they're different - high
    end
    assign pulse = pulse_reg;

endmodule




////////////////////////////////////////////////////////////////////////////////
// (slightly less than) 1/2_HZ clock: enables every less than 1/2
// second
////////////////////////////////////////////////////////////////////////////////
module herz_clk(input clk, output enabled);
    reg [25:0] counter = 0;
    always @(posedge clk) begin
        if(counter < 26'd32000000) counter <= counter + 1;
        else if(counter == 26'd32000000) begin
            counter <= 0;
        end
    end
    assign enabled = (counter == 32000000);
endmodule


////////////////////////////////////////////////////////////////////////////////
// Sample_Clock: Makes a sample clock at 9600*16 baud sampling rate
////////////////////////////////////////////////////////////////////////////////
module sample_clock(input clk, output reg enabled);
    reg [12:0] counter = 0;
    always @(posedge clk) begin
```

```
        if(counter < 424) begin
            counter <= counter + 1;
            enabled <= 0;
        end
        else if(counter == 424) begin
            enabled <= 1;
            counter <= 0;
        end
    end
endmodule

//////////////////////////////////////////////////////////////////////////////////
// 1_HZ clock: enables every 1 second. Takes in 65 mHz clock.
//////////////////////////////////////////////////////////////////////////////////
module song_herz_clk(input clk, input pause_flag_wire, output enabled);
    reg [25:0] counter = 0;
    always @(posedge clk) begin
        //if (~pause_flag_wire)begin
            if(counter < 26'd65000000) counter <= counter + 1;
            else if(counter == 26'd65000000) begin
                counter <= 0;
            end
        //end
    end
    assign enabled = (counter == 65000000);
endmodule
```

```
//////////////////////////////////////////////////////////////////////
// Blob: generate rectangle on screen
//////////////////////////////////////////////////////////////////////
module blob
   #(parameter WIDTH = 64,            // default width: 64 pixels
               HEIGHT = 64,           // default height: 64 pixels
               COLOR = 12'hFFF)  // default color: white
   (input [10:0] x,hcount,
    input [9:0] y,vcount,
    output reg [11:0] pixel);

   always @ * begin
      if ((hcount >= x && hcount < (x+WIDTH)) &&
         (vcount >= y && vcount < (y+HEIGHT)))
        pixel = COLOR;
      else pixel = 0;
   end
endmodule

//////////////////////////////////////////////////////////////////////
// Circle Blob: generate circle on screen
//////////////////////////////////////////////////////////////////////
module circle_blob
   #(parameter RADIUS = 7,           // default height: 64 pixels
               COLOR = 12'hFFF)  // default color: red
   (input clk, input [10:0] x,hcount,
    input [9:0] y,vcount,
    output reg [11:0] pixel);

    reg [24:0] r_sqd;//=RADIUS * RADIUS;
    reg [10:0] deltax_reg;
    reg [9:0] deltay_reg;
    reg [23:0] deltax_sqd;
    reg [22:0] deltay_sqd;
    always @ (posedge clk)  begin
        r_sqd <= RADIUS * RADIUS;
        deltax_reg <= (hcount > (x+RADIUS)) ?(hcount-(x+RADIUS)) : ((x+RADIUS)-hcount);
//Cutset input to multiplier
        deltay_reg <= (vcount > (y+RADIUS)) ?(vcount-(y+RADIUS)) : ((y+RADIUS)-vcount);
//Cutset input to multiplier
        deltax_sqd <= deltax_reg * deltax_reg;  //Cutset output of multiplier
        deltay_sqd <= deltay_reg * deltay_reg;  //Cutset output of multiplier
        if(deltax_sqd + deltay_sqd <= r_sqd) pixel <= COLOR;
        else pixel <= 12'd0;
    end
endmodule
//////////////////////////////////////////////////////////////////////
// Volume Blob: generate rectangle on screen
//////////////////////////////////////////////////////////////////////
module volume_blob
   #(parameter HEIGHT = 8, COLOR = 12'h0_FF)  // default color: cyan
   (input clk, input [10:0] x,hcount,
    input [9:0] y,vcount, change_in_x, output reg [11:0] pixel);
```

```
   always @ (posedge clk) begin
      if ((hcount >= (x) && hcount < (x+4+change_in_x)) &&
         (vcount >= (y) && vcount < (y+HEIGHT)))
        pixel = COLOR;
          else pixel = 0;
      end
endmodule


//////////////////////////////////////////////////////////////////////
// Outline Blob: generate outline on screen
//////////////////////////////////////////////////////////////////////
module outline_blob
   #(parameter RADIUS = 38,           // default height: 64 pixels
               COLOR = 12'hFFF)  // default color: white
   (input clk, input [10:0] x,hcount,
    input [9:0] y,vcount,
    output reg [11:0] pixel);
    reg [24:0] r_sqd=RADIUS * RADIUS;
    reg [24:0] outer_r = (RADIUS+3) * (RADIUS+3);
    reg [10:0] deltax_reg;
    reg [9:0] deltay_reg;
    reg [23:0] deltax_sqd;
    reg [22:0] deltay_sqd;
    always @ (posedge clk)  begin
        deltax_reg <= (hcount > (x+RADIUS)) ?(hcount-(x+RADIUS)) : ((x+RADIUS)-hcount);
//Cutset input to multiplier
        deltay_reg <= (vcount > (y+RADIUS)) ?(vcount-(y+RADIUS)) : ((y+RADIUS)-vcount);
//Cutset input to multiplier
        deltax_sqd <= deltax_reg * deltax_reg;  //Cutset output of multiplier
        deltay_sqd <= deltay_reg * deltay_reg;  //Cutset output of multiplier
        if(deltax_sqd + deltay_sqd <= outer_r && deltax_sqd + deltay_sqd >= r_sqd) pixel
<= COLOR;  //Gives range of coordinates to color in
        else pixel <= 12'd0;
    end
endmodule
```

```
//////////////////////////////////////////////
Takes in a 400-bit str and parses each 8-bit
character. Displays str starting from given x, y
input.
//////////////////////////////////////////////
module text
   #(parameter WIDTH = 8,            // default width: 8 pixels
               HEIGHT = 16,           // default height: 16 pixels
               COLOR = 12'hFFF)  // default color: white
   (input clk, input [10:0] x,hcount,
    input [9:0] y,vcount,
    input [0:399] str, //max 50 characters
    output reg [11:0] pixel);

    wire [7:0] display;
    reg [8:0] str_index = 0;
    ascii_to_dots({str[str_index], str[str_index+1], str[str_index+2], str[str_index+3],
str[str_index+4],
    str[str_index+5], str[str_index+6], str[str_index+7]}, vcount-y, display);

    reg [2:0] counter = 0;
    always @(posedge clk) begin
        if ((hcount >= (x+(str_index)) && hcount < (x+(str_index)+WIDTH)) && (vcount >= y
&& vcount < (y+HEIGHT))) begin
            pixel <= display[WIDTH-1-counter] ? COLOR:0;
            if (counter == 7) begin
                if (str_index==392) str_index <= 0;
                else str_index <= str_index+8;
            end
            counter <= (counter==7)?0:counter+1;
        end
        else pixel <= 0;

    end
endmodule

//////////////////////////////////////////////////////////
Similar to text module but takes in paused and loaded inputs.
If song is paused, display play symbol. If song is not loaded,
display play symbol. Else, display pause symbol (the song is
playing).
//////////////////////////////////////////////////////////

module play_pause_text
   #(parameter WIDTH = 8,            // default width: 8 pixels
               HEIGHT = 16,           // default height: 16 pixels
               COLOR = 12'hFFF)  // default color: white
   (input clk, input [10:0] x,hcount,
    input [9:0] y,vcount,
    input paused,
    input loaded,
    output reg [11:0] pixel);

    wire [7:0] display;
```

```
    wire [7:0] char = (~loaded | paused)?8'h10:8'h13;
    ascii_to_dots(char, vcount-y, display);

    always @ * begin
        if ((hcount >= x && hcount < (x+WIDTH)) && (vcount >= y && vcount < (y+HEIGHT)))
            pixel = display[WIDTH-1-(hcount-x)]?COLOR:0;
        else pixel = 0;
    end
endmodule

//////////////////////////////////////////////////////
Similar to text module but only one character as input.
//////////////////////////////////////////////////////
module character
   #(parameter WIDTH = 8,            // default width: 8 pixels
               HEIGHT = 16,           // default height: 16 pixels
               COLOR = 12'hFFF)  // default color: white
   (input [10:0] x,hcount,
    input [9:0] y,vcount,
    input [7:0] char,
    output reg [11:0] pixel);

    wire [7:0] display;

    ascii_to_dots(char, vcount-y, display);

    always @ * begin
        if ((hcount >= x && hcount < (x+WIDTH)) && (vcount >= y && vcount < (y+HEIGHT)))
            pixel = display[WIDTH-1-(hcount-x)]?COLOR:0;
        else pixel = 0;
    end
endmodule
```

```
\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
Takes in 8-bit char input and row # of char that is being
displayed, and returns the 8-bit bit pixel map of that row
\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
module ascii_to_dots(
  input wire [7:0] char,
  input wire [10:0] row,
  output reg [7:0] pixel
);

    always @(*) begin
        case (char)
            8'h00:
                begin
                    case (row)
                        0:  pixel = 8'b00000000; // -- 0
                        1:  pixel = 8'b00000000; // -- 1
                        2:  pixel = 8'b00000000; // -- 2
                        3:  pixel = 8'b00000000; // -- 3
                        4:  pixel = 8'b00000000; // -- 4
                        5:  pixel = 8'b00000000; // -- 5
                        6:  pixel = 8'b00000000; // -- 6
                        7:  pixel = 8'b00000000; // -- 7
                        8:  pixel = 8'b00000000; // -- 8
                        9:  pixel = 8'b00000000; // -- 9
                        10: pixel = 8'b00000000; // -- a
                        11: pixel = 8'b00000000; // -- b
                        12: pixel = 8'b00000000; // -- c
                        13: pixel = 8'b00000000; // -- d
                        14: pixel = 8'b00000000; // -- e
                        15: pixel = 8'b00000000; // -- f
                    endcase
                end
            8'h01:
                begin
                    case (row)
                        0:  pixel = 8'b00000001; // -- 0        *
                        1:  pixel = 8'b00000111; // -- 1      ***
                        2:  pixel = 8'b00001111; // -- 2     ****
                        3:  pixel = 8'b00011111; // -- 3    *****
                        4:  pixel = 8'b00011111; // -- 4    *****
                        5:  pixel = 8'b11111111; // -- 5 ********
                        6:  pixel = 8'b11111111; // -- 6 ********
                        7:  pixel = 8'b11111111; // -- 7 ********
                        8:  pixel = 8'b11111111; // -- 8 ********
                        9:  pixel = 8'b11111111; // -- 9 ********
                        10: pixel = 8'b11111111; // -- a ********
                        11: pixel = 8'b00011111; // -- b    *****
                        12: pixel = 8'b00011111; // -- c    *****
                        13: pixel = 8'b00001111; // -- d     ****
                        14: pixel = 8'b00000111; // -- e      ***
                        15: pixel = 8'b00000001; // -- f        *
                    endcase
                end
```

```
8'h02:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b01111110; // -- 2  ******
            3:  pixel = 8'b11111111; // -- 3 ********
            4:  pixel = 8'b11011011; // -- 4 ** ** **
            5:  pixel = 8'b11111111; // -- 5 ********
            6:  pixel = 8'b11111111; // -- 6 ********
            7:  pixel = 8'b11000011; // -- 7 **    **
            8:  pixel = 8'b11100111; // -- 8 ***  ***
            9:  pixel = 8'b11111111; // -- 9 ********
            10: pixel = 8'b11111111; // -- a ********
            11: pixel = 8'b01111110; // -- b  ******
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h03:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b00000000; // -- 2
            3:  pixel = 8'b00000000; // -- 3
            4:  pixel = 8'b01101100; // -- 4  ** **
            5:  pixel = 8'b11111110; // -- 5 *******
            6:  pixel = 8'b11111110; // -- 6 *******
            7:  pixel = 8'b11111110; // -- 7 *******
            8:  pixel = 8'b11111110; // -- 8 *******
            9:  pixel = 8'b01111100; // -- 9  *****
            10: pixel = 8'b00111000; // -- a   ***
            11: pixel = 8'b00010000; // -- b    *
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h04:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b00000000; // -- 2
            3:  pixel = 8'b00000000; // -- 3
            4:  pixel = 8'b00010000; // -- 4    *
            5:  pixel = 8'b00111000; // -- 5   ***
            6:  pixel = 8'b01111100; // -- 6  *****
            7:  pixel = 8'b11111110; // -- 7 *******
            8:  pixel = 8'b01111100; // -- 8  *****
            9:  pixel = 8'b00111000; // -- 9   ***
```

```
            10: pixel = 8'b00010000; // --    *
            11: pixel = 8'b00000000; // -- b
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h05:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b00000000; // -- 2
            3:  pixel = 8'b00011000; // --    **
            4:  pixel = 8'b00111100; // --   ****
            5:  pixel = 8'b00111100; // --   ****
            6:  pixel = 8'b11100111; // -- 6 ***  ***
            7:  pixel = 8'b11100111; // -- 7 ***  ***
            8:  pixel = 8'b11100111; // -- 8 ***  ***
            9:  pixel = 8'b00011000; // --    **
            10: pixel = 8'b00011000; // --    **
            11: pixel = 8'b00111100; // --   ****
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h06:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b00000000; // -- 2
            3:  pixel = 8'b00011000; // --    **
            4:  pixel = 8'b00111100; // --   ****
            5:  pixel = 8'b01111110; // --  ******
            6:  pixel = 8'b11111111; // -- 6 ********
            7:  pixel = 8'b11111111; // -- 7 ********
            8:  pixel = 8'b01111110; // --  ******
            9:  pixel = 8'b00011000; // --    **
            10: pixel = 8'b00011000; // --    **
            11: pixel = 8'b00111100; // --   ****
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h07:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
```

```
            2:  pixel = 8'b00000000; // -- 2
            3:  pixel = 8'b00000000; // -- 3
            4:  pixel = 8'b00000000; // -- 4
            5:  pixel = 8'b00000000; // -- 5
            6:  pixel = 8'b00011000; // --    **
            7:  pixel = 8'b00111100; // --   ****
            8:  pixel = 8'b00111100; // --   ****
            9:  pixel = 8'b00011000; // --    **
            10: pixel = 8'b00000000; // -- a
            11: pixel = 8'b00000000; // -- b
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h08:
    begin
        case (row)
            0:  pixel = 8'b11111111; // -- 0 ********
            1:  pixel = 8'b11111111; // -- 1 ********
            2:  pixel = 8'b11111111; // -- 2 ********
            3:  pixel = 8'b11111111; // -- 3 ********
            4:  pixel = 8'b11111111; // -- 4 ********
            5:  pixel = 8'b11111111; // -- 5 ********
            6:  pixel = 8'b11100111; // -- 6 ***  ***
            7:  pixel = 8'b11000011; // -- 7 **    **
            8:  pixel = 8'b11000011; // -- 8 **    **
            9:  pixel = 8'b11100111; // -- 9 ***  ***
            10: pixel = 8'b11111111; // -- a ********
            11: pixel = 8'b11111111; // -- b ********
            12: pixel = 8'b11111111; // -- c ********
            13: pixel = 8'b11111111; // -- d ********
            14: pixel = 8'b11111111; // -- e ********
            15: pixel = 8'b11111111; // -- f ********
        endcase
    end
8'h09:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b00000000; // -- 2
            3:  pixel = 8'b00000000; // -- 3
            4:  pixel = 8'b00000000; // -- 4
            5:  pixel = 8'b00111100; // --   ****
            6:  pixel = 8'b01100110; // --  **  **
            7:  pixel = 8'b01000010; // --  *    *
            8:  pixel = 8'b01000010; // --  *    *
            9:  pixel = 8'b01100110; // --  **  **
            10: pixel = 8'b00111100; // --   ****
            11: pixel = 8'b00000000; // -- b
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
```

```
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h0a:
    begin
        case (row)
            0:  pixel = 8'b11111111; // -- 0 ********
            1:  pixel = 8'b11111111; // -- 1 ********
            2:  pixel = 8'b11111111; // -- 2 ********
            3:  pixel = 8'b11111111; // -- 3 ********
            4:  pixel = 8'b11111111; // -- 4 ********
            5:  pixel = 8'b11000011; // -- 5 **    **
            6:  pixel = 8'b10011001; // -- 6 *  **  *
            7:  pixel = 8'b10111101; // -- 7 * **** *
            8:  pixel = 8'b10111101; // -- 8 * **** *
            9:  pixel = 8'b10011001; // -- 9 *  **  *
            10: pixel = 8'b11000011; // -- a **    **
            11: pixel = 8'b11111111; // -- b ********
            12: pixel = 8'b11111111; // -- c ********
            13: pixel = 8'b11111111; // -- d ********
            14: pixel = 8'b11111111; // -- e ********
            15: pixel = 8'b11111111; // -- f ********
        endcase
    end
8'h0b:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b00011110; // -- 2    ****
            3:  pixel = 8'b00001110; // -- 3     ***
            4:  pixel = 8'b00011010; // -- 4    ** *
            5:  pixel = 8'b00110010; // -- 5   **  *
            6:  pixel = 8'b01111000; // -- 6  ****
            7:  pixel = 8'b11001100; // -- 7 **  **
            8:  pixel = 8'b11001100; // -- 8 **  **
            9:  pixel = 8'b11001100; // -- 9 **  **
            10: pixel = 8'b11001100; // -- a **  **
            11: pixel = 8'b01111000; // -- b  ****
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h0c:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b00111100; // -- 2   ****
            3:  pixel = 8'b01100110; // -- 3  **  **
            4:  pixel = 8'b01100110; // -- 4  **  **
            5:  pixel = 8'b01100110; // -- 5  **  **
            6:  pixel = 8'b01100110; // -- 6  **  **
```

```
            7:  pixel = 8'b00111100; // -- 7   ****
            8:  pixel = 8'b00011000; // -- 8    **
            9:  pixel = 8'b01111110; // -- 9  ******
            10: pixel = 8'b00011000; // -- a    **
            11: pixel = 8'b00011000; // -- b    **
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h0d:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b00111111; // -- 2   ******
            3:  pixel = 8'b00110011; // -- 3   **  **
            4:  pixel = 8'b00111111; // -- 4   ******
            5:  pixel = 8'b00110000; // -- 5   **
            6:  pixel = 8'b00110000; // -- 6   **
            7:  pixel = 8'b00110000; // -- 7   **
            8:  pixel = 8'b00110000; // -- 8   **
            9:  pixel = 8'b01110000; // -- 9  ***
            10: pixel = 8'b11110000; // -- a ****
            11: pixel = 8'b11100000; // -- b ***
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h0e:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b01111111; // -- 2  *******
            3:  pixel = 8'b01100011; // -- 3  **   **
            4:  pixel = 8'b01111111; // -- 4  *******
            5:  pixel = 8'b01100011; // -- 5  **   **
            6:  pixel = 8'b01100011; // -- 6  **   **
            7:  pixel = 8'b01100011; // -- 7  **   **
            8:  pixel = 8'b01100011; // -- 8  **   **
            9:  pixel = 8'b01100111; // -- 9  **  ***
            10: pixel = 8'b11100111; // -- a ***  ***
            11: pixel = 8'b11100110; // -- b ***  **
            12: pixel = 8'b11000000; // -- c **
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h0f:
    begin
```

```
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b00000000; // -- 2
            3:  pixel = 8'b00011000; // -- 3    **
            4:  pixel = 8'b00011000; // -- 4    **
            5:  pixel = 8'b11011011; // -- 5 ** ** **
            6:  pixel = 8'b00111100; // -- 6   ****
            7:  pixel = 8'b11100111; // -- 7 ***  ***
            8:  pixel = 8'b00111100; // -- 8   ****
            9:  pixel = 8'b11011011; // -- 9 ** ** **
            10: pixel = 8'b00011000; // -- a    **
            11: pixel = 8'b00011000; // -- b    **
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h10:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b10000000; // -- 1 *
            2:  pixel = 8'b11000000; // -- 2 **
            3:  pixel = 8'b11100000; // -- 3 ***
            4:  pixel = 8'b11110000; // -- 4 ****
            5:  pixel = 8'b11111000; // -- 5 *****
            6:  pixel = 8'b11111110; // -- 6 *******
            7:  pixel = 8'b11111000; // -- 7 *****
            8:  pixel = 8'b11110000; // -- 8 ****
            9:  pixel = 8'b11100000; // -- 9 ***
            10: pixel = 8'b11000000; // -- a **
            11: pixel = 8'b10000000; // -- b *
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h11:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000010; // -- 1       *
            2:  pixel = 8'b00000110; // -- 2      **
            3:  pixel = 8'b00001110; // -- 3     ***
            4:  pixel = 8'b00011110; // -- 4    ****
            5:  pixel = 8'b00111110; // -- 5   *****
            6:  pixel = 8'b11111110; // -- 6 *******
            7:  pixel = 8'b00111110; // -- 7   *****
            8:  pixel = 8'b00011110; // -- 8    ****
            9:  pixel = 8'b00001110; // -- 9     ***
            10: pixel = 8'b00000110; // -- a      **
            11: pixel = 8'b00000010; // -- b       *
```

```
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h12:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b00011000; // -- 2    **
            3:  pixel = 8'b00111100; // -- 3   ****
            4:  pixel = 8'b01111110; // -- 4  ******
            5:  pixel = 8'b00011000; // -- 5    **
            6:  pixel = 8'b00011000; // -- 6    **
            7:  pixel = 8'b00011000; // -- 7    **
            8:  pixel = 8'b01111110; // -- 8  ******
            9:  pixel = 8'b00111100; // -- 9   ****
            10: pixel = 8'b00011000; // -- a    **
            11: pixel = 8'b00000000; // -- b
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h13:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b11100111; // -- 2  **  **
            3:  pixel = 8'b11100111; // -- 3  **  **
            4:  pixel = 8'b11100111; // -- 4  **  **
            5:  pixel = 8'b11100111; // -- 5  **  **
            6:  pixel = 8'b11100111; // -- 6  **  **
            7:  pixel = 8'b11100111; // -- 7  **  **
            8:  pixel = 8'b11100111; // -- 8  **  **
            9:  pixel = 8'b11100111; // -- 9  **  **
            10: pixel = 8'b11100111; // -- a  **  **
            11: pixel = 8'b11100111; // -- b  **  **
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h14:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b01111111; // -- 2  *******
            3:  pixel = 8'b11011011; // -- 3 ** ** **
```

```
            4:  pixel = 8'b11011011; // -- 4 ** ** **
            5:  pixel = 8'b11011011; // -- 5 ** ** **
            6:  pixel = 8'b01111011; // -- 6  **** **
            7:  pixel = 8'b00011011; // -- 7    ** **
            8:  pixel = 8'b00011011; // -- 8    ** **
            9:  pixel = 8'b00011011; // -- 9    ** **
            10: pixel = 8'b00011011; // -- a    ** **
            11: pixel = 8'b00011011; // -- b    ** **
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h15:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b01111100; // -- 1  *****
            2:  pixel = 8'b11000110; // -- 2 **   **
            3:  pixel = 8'b01100000; // -- 3  **
            4:  pixel = 8'b00111000; // -- 4   ***
            5:  pixel = 8'b01101100; // -- 5  ** **
            6:  pixel = 8'b11000110; // -- 6 **   **
            7:  pixel = 8'b11000110; // -- 7 **   **
            8:  pixel = 8'b01101100; // -- 8  ** **
            9:  pixel = 8'b00111000; // -- 9   ***
            10: pixel = 8'b00001100; // -- a     **
            11: pixel = 8'b11000110; // -- b **   **
            12: pixel = 8'b01111100; // -- c  *****
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h16:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b00000000; // -- 2
            3:  pixel = 8'b00000000; // -- 3
            4:  pixel = 8'b00000000; // -- 4
            5:  pixel = 8'b00000000; // -- 5
            6:  pixel = 8'b00000000; // -- 6
            7:  pixel = 8'b00000000; // -- 7
            8:  pixel = 8'b11111110; // -- 8 *******
            9:  pixel = 8'b11111110; // -- 9 *******
            10: pixel = 8'b11111110; // -- a *******
            11: pixel = 8'b11111110; // -- b *******
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
```

```
    end
8'h17:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b00011000; // --  2    **
            3:  pixel = 8'b00111100; // --  3   ****
            4:  pixel = 8'b01111110; // --  4  ******
            5:  pixel = 8'b00011000; // --  5    **
            6:  pixel = 8'b00011000; // --  6    **
            7:  pixel = 8'b00011000; // --  7    **
            8:  pixel = 8'b01111110; // --  8  ******
            9:  pixel = 8'b00111100; // --  9   ****
            10: pixel = 8'b00011000; // --  a    **
            11: pixel = 8'b01111110; // --  b  ******
            12: pixel = 8'b00110000; // --  c
            13: pixel = 8'b00000000; // --  d
            14: pixel = 8'b00000000; // --  e
            15: pixel = 8'b00000000; // --  f
        endcase
    end
8'h18:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b00011000; // --  2    **
            3:  pixel = 8'b00111100; // --  3   ****
            4:  pixel = 8'b01111110; // --  4  ******
            5:  pixel = 8'b00011000; // --  5    **
            6:  pixel = 8'b00011000; // --  6    **
            7:  pixel = 8'b00011000; // --  7    **
            8:  pixel = 8'b00011000; // --  8    **
            9:  pixel = 8'b00011000; // --  9    **
            10: pixel = 8'b00011000; // --  a    **
            11: pixel = 8'b00011000; // --  b    **
            12: pixel = 8'b00000000; // --  c
            13: pixel = 8'b00000000; // --  d
            14: pixel = 8'b00000000; // --  e
            15: pixel = 8'b00000000; // --  f
        endcase
    end
8'h19:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b00011000; // --  2    **
            3:  pixel = 8'b00011000; // --  3    **
            4:  pixel = 8'b00011000; // --  4    **
            5:  pixel = 8'b00011000; // --  5    **
            6:  pixel = 8'b00011000; // --  6    **
            7:  pixel = 8'b00011000; // --  7    **
            8:  pixel = 8'b00011000; // --  8    **
```

```
            9:  pixel = 8'b01111110; // -- 9  ******
            10: pixel = 8'b00111100; // -- a   ****
            11: pixel = 8'b00011000; // -- b    **
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h1a:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b00000000; // -- 2
            3:  pixel = 8'b00000000; // -- 3
            4:  pixel = 8'b00000000; // -- 4
            5:  pixel = 8'b00011000; // -- 5    **
            6:  pixel = 8'b00001100; // -- 6     **
            7:  pixel = 8'b11111110; // -- 7 *******
            8:  pixel = 8'b00001100; // -- 8     **
            9:  pixel = 8'b00011000; // -- 9    **
            10: pixel = 8'b00000000; // -- a
            11: pixel = 8'b00000000; // -- b
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h1b:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b00000000; // -- 2
            3:  pixel = 8'b00000000; // -- 3
            4:  pixel = 8'b00000000; // -- 4
            5:  pixel = 8'b00110000; // -- 5   **
            6:  pixel = 8'b01100000; // -- 6  **
            7:  pixel = 8'b11111110; // -- 7 *******
            8:  pixel = 8'b01100000; // -- 8  **
            9:  pixel = 8'b00110000; // -- 9   **
            10: pixel = 8'b00000000; // -- a
            11: pixel = 8'b00000000; // -- b
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h1c:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
```

```
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b00000000; // -- 2
            3:  pixel = 8'b00000000; // -- 3
            4:  pixel = 8'b00000000; // -- 4
            5:  pixel = 8'b00000000; // -- 5
            6:  pixel = 8'b11000000; // -- 6 **
            7:  pixel = 8'b11000000; // -- 7 **
            8:  pixel = 8'b11000000; // -- 8 **
            9:  pixel = 8'b11111110; // -- 9 *******
            10: pixel = 8'b00000000; // -- a
            11: pixel = 8'b00000000; // -- b
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h1d:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b00000000; // -- 2
            3:  pixel = 8'b00000000; // -- 3
            4:  pixel = 8'b00000000; // -- 4
            5:  pixel = 8'b00100100; // -- 5   *  *
            6:  pixel = 8'b01100110; // -- 6  **  **
            7:  pixel = 8'b11111111; // -- 7 ********
            8:  pixel = 8'b01100110; // -- 8  **  **
            9:  pixel = 8'b00100100; // -- 9   *  *
            10: pixel = 8'b00000000; // -- a
            11: pixel = 8'b00000000; // -- b
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h1e:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b00000000; // -- 2
            3:  pixel = 8'b00000000; // -- 3
            4:  pixel = 8'b00010000; // -- 4    *
            5:  pixel = 8'b00111000; // -- 5   ***
            6:  pixel = 8'b00111000; // -- 6   ***
            7:  pixel = 8'b01111100; // -- 7  *****
            8:  pixel = 8'b01111100; // -- 8  *****
            9:  pixel = 8'b11111110; // -- 9 *******
            10: pixel = 8'b11111110; // -- a *******
            11: pixel = 8'b00000000; // -- b
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
```

```
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h1f:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b00000000; // -- 2
            3:  pixel = 8'b00000000; // -- 3
            4:  pixel = 8'b11111110; // -- 4 *******
            5:  pixel = 8'b11111110; // -- 5 *******
            6:  pixel = 8'b01111100; // -- 6  *****
            7:  pixel = 8'b01111100; // -- 7  *****
            8:  pixel = 8'b00111000; // -- 8   ***
            9:  pixel = 8'b00111000; // -- 9   ***
            10: pixel = 8'b00010000; // -- a    *
            11: pixel = 8'b00000000; // -- b
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h20:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b00000000; // -- 2
            3:  pixel = 8'b00000000; // -- 3
            4:  pixel = 8'b00000000; // -- 4
            5:  pixel = 8'b00000000; // -- 5
            6:  pixel = 8'b00000000; // -- 6
            7:  pixel = 8'b00000000; // -- 7
            8:  pixel = 8'b00000000; // -- 8
            9:  pixel = 8'b00000000; // -- 9
            10: pixel = 8'b00000000; // -- a
            11: pixel = 8'b00000000; // -- b
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h21:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b00011000; // -- 2    **
            3:  pixel = 8'b00111100; // -- 3   ****
            4:  pixel = 8'b00111100; // -- 4   ****
            5:  pixel = 8'b00111100; // -- 5   ****
```

```
            6:  pixel = 8'b00011000; // -- 6    **
            7:  pixel = 8'b00011000; // -- 7    **
            8:  pixel = 8'b00011000; // -- 8    **
            9:  pixel = 8'b00000000; // -- 9
            10: pixel = 8'b00011000; // -- a    **
            11: pixel = 8'b00011000; // -- b    **
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h22:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b01100110; // -- 1  **  **
            2:  pixel = 8'b01100110; // -- 2  **  **
            3:  pixel = 8'b01100110; // -- 3  **  **
            4:  pixel = 8'b00100100; // -- 4   *  *
            5:  pixel = 8'b00000000; // -- 5
            6:  pixel = 8'b00000000; // -- 6
            7:  pixel = 8'b00000000; // -- 7
            8:  pixel = 8'b00000000; // -- 8
            9:  pixel = 8'b00000000; // -- 9
            10: pixel = 8'b00000000; // -- a
            11: pixel = 8'b00000000; // -- b
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h23:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b00000000; // -- 2
            3:  pixel = 8'b01101100; // -- 3  ** **
            4:  pixel = 8'b01101100; // -- 4  ** **
            5:  pixel = 8'b11111110; // -- 5 *******
            6:  pixel = 8'b01101100; // -- 6  ** **
            7:  pixel = 8'b01101100; // -- 7  ** **
            8:  pixel = 8'b01101100; // -- 8  ** **
            9:  pixel = 8'b11111110; // -- 9 *******
            10: pixel = 8'b01101100; // -- a  ** **
            11: pixel = 8'b01101100; // -- b  ** **
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h24:
```

```
    begin
        case (row)
            0:  pixel = 8'b00011000; // -- 0     **
            1:  pixel = 8'b00011000; // -- 1     **
            2:  pixel = 8'b01111100; // -- 2   *****
            3:  pixel = 8'b11000110; // -- 3  **   **
            4:  pixel = 8'b11000010; // -- 4  **    *
            5:  pixel = 8'b11000000; // -- 5  **
            6:  pixel = 8'b01111100; // -- 6   *****
            7:  pixel = 8'b00000110; // -- 7       **
            8:  pixel = 8'b00000110; // -- 8       **
            9:  pixel = 8'b10000110; // -- 9  *    **
            10: pixel = 8'b11000110; // -- a  **   **
            11: pixel = 8'b01111100; // -- b   *****
            12: pixel = 8'b00011000; // -- c     **
            13: pixel = 8'b00011000; // -- d     **
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h25:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b00000000; // -- 2
            3:  pixel = 8'b00000000; // -- 3
            4:  pixel = 8'b11000010; // -- 4 **    *
            5:  pixel = 8'b11000110; // -- 5 **   **
            6:  pixel = 8'b00001100; // -- 6     **
            7:  pixel = 8'b00011000; // -- 7    **
            8:  pixel = 8'b00110000; // -- 8   **
            9:  pixel = 8'b01100000; // -- 9  **
            10: pixel = 8'b11000110; // -- a **   **
            11: pixel = 8'b10000110; // -- b *    **
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h26:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b00111000; // -- 2   ***
            3:  pixel = 8'b01101100; // -- 3  ** **
            4:  pixel = 8'b01101100; // -- 4  ** **
            5:  pixel = 8'b00111000; // -- 5   ***
            6:  pixel = 8'b01110110; // -- 6  *** **
            7:  pixel = 8'b11011100; // -- 7 ** ***
            8:  pixel = 8'b11001100; // -- 8 **  **
            9:  pixel = 8'b11001100; // -- 9 **  **
            10: pixel = 8'b11001100; // -- a **  **
```

```
            11: pixel = 8'b01110110; // -- b  *** **
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h27:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00110000; // -- 1   **
            2:  pixel = 8'b00110000; // -- 2   **
            3:  pixel = 8'b00110000; // -- 3   **
            4:  pixel = 8'b01100000; // -- 4  **
            5:  pixel = 8'b00000000; // -- 5
            6:  pixel = 8'b00000000; // -- 6
            7:  pixel = 8'b00000000; // -- 7
            8:  pixel = 8'b00000000; // -- 8
            9:  pixel = 8'b00000000; // -- 9
            10: pixel = 8'b00000000; // -- a
            11: pixel = 8'b00000000; // -- b
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h28:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b00001100; // -- 2     **
            3:  pixel = 8'b00011000; // -- 3    **
            4:  pixel = 8'b00110000; // -- 4   **
            5:  pixel = 8'b00110000; // -- 5   **
            6:  pixel = 8'b00110000; // -- 6   **
            7:  pixel = 8'b00110000; // -- 7   **
            8:  pixel = 8'b00110000; // -- 8   **
            9:  pixel = 8'b00110000; // -- 9   **
            10: pixel = 8'b00011000; // -- a    **
            11: pixel = 8'b00001100; // -- b     **
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h29:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b00110000; // -- 2   **
```

```
            3:  pixel = 8'b00011000; // --  3    **
            4:  pixel = 8'b00001100; // --  4     **
            5:  pixel = 8'b00001100; // --  5     **
            6:  pixel = 8'b00001100; // --  6     **
            7:  pixel = 8'b00001100; // --  7     **
            8:  pixel = 8'b00001100; // --  8     **
            9:  pixel = 8'b00001100; // --  9     **
            10: pixel = 8'b00011000; // --  a    **
            11: pixel = 8'b00110000; // --  b   **
            12: pixel = 8'b00000000; // --  c
            13: pixel = 8'b00000000; // --  d
            14: pixel = 8'b00000000; // --  e
            15: pixel = 8'b00000000; // --  f
        endcase
    end
8'h2a:
    begin
        case (row)
            0:  pixel = 8'b00000000; // --  0
            1:  pixel = 8'b00000000; // --  1
            2:  pixel = 8'b00000000; // --  2
            3:  pixel = 8'b00000000; // --  3
            4:  pixel = 8'b00000000; // --  4
            5:  pixel = 8'b01100110; // --  5  **  **
            6:  pixel = 8'b00111100; // --  6   ****
            7:  pixel = 8'b11111111; // --  7 ********
            8:  pixel = 8'b00111100; // --  8   ****
            9:  pixel = 8'b01100110; // --  9  **  **
            10: pixel = 8'b00000000; // --  a
            11: pixel = 8'b00000000; // --  b
            12: pixel = 8'b00000000; // --  c
            13: pixel = 8'b00000000; // --  d
            14: pixel = 8'b00000000; // --  e
            15: pixel = 8'b00000000; // --  f
        endcase
    end
8'h2b:
    begin
        case (row)
            0:  pixel = 8'b00000000; // --  0
            1:  pixel = 8'b00000000; // --  1
            2:  pixel = 8'b00000000; // --  2
            3:  pixel = 8'b00000000; // --  3
            4:  pixel = 8'b00000000; // --  4
            5:  pixel = 8'b00011000; // --  5    **
            6:  pixel = 8'b00011000; // --  6    **
            7:  pixel = 8'b01111110; // --  7  ******
            8:  pixel = 8'b00011000; // --  8    **
            9:  pixel = 8'b00011000; // --  9    **
            10: pixel = 8'b00000000; // --  a
            11: pixel = 8'b00000000; // --  b
            12: pixel = 8'b00000000; // --  c
            13: pixel = 8'b00000000; // --  d
            14: pixel = 8'b00000000; // --  e
            15: pixel = 8'b00000000; // --  f
```

```
        endcase
    end
8'h2c:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b00000000; // -- 2
            3:  pixel = 8'b00000000; // -- 3
            4:  pixel = 8'b00000000; // -- 4
            5:  pixel = 8'b00000000; // -- 5
            6:  pixel = 8'b00000000; // -- 6
            7:  pixel = 8'b00000000; // -- 7
            8:  pixel = 8'b00000000; // -- 8
            9:  pixel = 8'b00011000; // -- 9    **
            10: pixel = 8'b00011000; // -- a    **
            11: pixel = 8'b00011000; // -- b    **
            12: pixel = 8'b00110000; // -- c   **
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h2d:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b00000000; // -- 2
            3:  pixel = 8'b00000000; // -- 3
            4:  pixel = 8'b00000000; // -- 4
            5:  pixel = 8'b00000000; // -- 5
            6:  pixel = 8'b00000000; // -- 6
            7:  pixel = 8'b01111110; // -- 7  ******
            8:  pixel = 8'b00000000; // -- 8
            9:  pixel = 8'b00000000; // -- 9
            10: pixel = 8'b00000000; // -- a
            11: pixel = 8'b00000000; // -- b
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h2e:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b00000000; // -- 2
            3:  pixel = 8'b00000000; // -- 3
            4:  pixel = 8'b00000000; // -- 4
            5:  pixel = 8'b00000000; // -- 5
            6:  pixel = 8'b00000000; // -- 6
            7:  pixel = 8'b00000000; // -- 7
```

```
            8:  pixel = 8'b00000000; // -- 8
            9:  pixel = 8'b00000000; // -- 9
            10: pixel = 8'b00011000; // -- a    **
            11: pixel = 8'b00011000; // -- b    **
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h2f:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b00000000; // -- 2
            3:  pixel = 8'b00000000; // -- 3
            4:  pixel = 8'b00000010; // -- 4       *
            5:  pixel = 8'b00000110; // -- 5      **
            6:  pixel = 8'b00001100; // -- 6     **
            7:  pixel = 8'b00011000; // -- 7    **
            8:  pixel = 8'b00110000; // -- 8   **
            9:  pixel = 8'b01100000; // -- 9  **
            10: pixel = 8'b11000000; // -- a **
            11: pixel = 8'b10000000; // -- b *
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h30:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b01111100; // -- 2  *****
            3:  pixel = 8'b11000110; // -- 3 **   **
            4:  pixel = 8'b11000110; // -- 4 **   **
            5:  pixel = 8'b11001110; // -- 5 **  ***
            6:  pixel = 8'b11011110; // -- 6 ** ****
            7:  pixel = 8'b11110110; // -- 7 **** **
            8:  pixel = 8'b11100110; // -- 8 ***  **
            9:  pixel = 8'b11000110; // -- 9 **   **
            10: pixel = 8'b11000110; // -- a **   **
            11: pixel = 8'b01111100; // -- b  *****
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h31:
    begin
        case (row)
```

```
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b00011000; // --  2
            3:  pixel = 8'b00111000; // --  3
            4:  pixel = 8'b01111000; // --  4    **
            5:  pixel = 8'b00011000; // --  5   ***
            6:  pixel = 8'b00011000; // --  6  ****
            7:  pixel = 8'b00011000; // --  7    **
            8:  pixel = 8'b00011000; // --  8    **
            9:  pixel = 8'b00011000; // --  9    **
            10: pixel = 8'b00011000; // --  a    **
            11: pixel = 8'b01111110; // --  b    **
            12: pixel = 8'b00000000; // --  c    **
            13: pixel = 8'b00000000; // --  d  ******
            14: pixel = 8'b00000000; // --  e
            15: pixel = 8'b00000000; // --  f
        endcase
    end
8'h32:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b01111100; // --  2  *****
            3:  pixel = 8'b11000110; // --  3 **   **
            4:  pixel = 8'b00000110; // --  4      **
            5:  pixel = 8'b00001100; // --  5     **
            6:  pixel = 8'b00011000; // --  6    **
            7:  pixel = 8'b00110000; // --  7   **
            8:  pixel = 8'b01100000; // --  8  **
            9:  pixel = 8'b11000000; // --  9 **
            10: pixel = 8'b11000110; // --  a **   **
            11: pixel = 8'b11111110; // --  b *******
            12: pixel = 8'b00000000; // --  c
            13: pixel = 8'b00000000; // --  d
            14: pixel = 8'b00000000; // --  e
            15: pixel = 8'b00000000; // --  f
        endcase
    end
8'h33:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b01111100; // --  2  *****
            3:  pixel = 8'b11000110; // --  3 **   **
            4:  pixel = 8'b00000110; // --  4      **
            5:  pixel = 8'b00000110; // --  5      **
            6:  pixel = 8'b00111100; // --  6   ****
            7:  pixel = 8'b00000110; // --  7      **
            8:  pixel = 8'b00000110; // --  8      **
            9:  pixel = 8'b00000110; // --  9      **
            10: pixel = 8'b11000110; // --  a **   **
            11: pixel = 8'b01111100; // --  b  *****
            12: pixel = 8'b00000000; // --  c
```

```
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h34:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b00001100; // -- 2     **
            3:  pixel = 8'b00011100; // -- 3    ***
            4:  pixel = 8'b00111100; // -- 4   ****
            5:  pixel = 8'b01101100; // -- 5  ** **
            6:  pixel = 8'b11001100; // -- 6 **  **
            7:  pixel = 8'b11111110; // -- 7 *******
            8:  pixel = 8'b00001100; // -- 8     **
            9:  pixel = 8'b00001100; // -- 9     **
            10: pixel = 8'b00001100; // -- a     **
            11: pixel = 8'b00011110; // -- b    ****
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h35:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b11111110; // -- 2 *******
            3:  pixel = 8'b11000000; // -- 3 **
            4:  pixel = 8'b11000000; // -- 4 **
            5:  pixel = 8'b11000000; // -- 5 **
            6:  pixel = 8'b11111100; // -- 6 ******
            7:  pixel = 8'b00000110; // -- 7      **
            8:  pixel = 8'b00000110; // -- 8      **
            9:  pixel = 8'b00000110; // -- 9      **
            10: pixel = 8'b11000110; // -- a **   **
            11: pixel = 8'b01111100; // -- b  *****
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h36:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b00111000; // -- 2   ***
            3:  pixel = 8'b01100000; // -- 3  **
            4:  pixel = 8'b11000000; // -- 4 **
```

```
            5:  pixel = 8'b11000000; // -- 5 **
            6:  pixel = 8'b11111100; // -- 6 ******
            7:  pixel = 8'b11000110; // -- 7 **   **
            8:  pixel = 8'b11000110; // -- 8 **   **
            9:  pixel = 8'b11000110; // -- 9 **   **
            10: pixel = 8'b11000110; // -- a **   **
            11: pixel = 8'b01111100; // -- b  *****
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h37:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b11111110; // -- 2 *******
            3:  pixel = 8'b11000110; // -- 3 **   **
            4:  pixel = 8'b00000110; // -- 4      **
            5:  pixel = 8'b00000110; // -- 5      **
            6:  pixel = 8'b00001100; // -- 6     **
            7:  pixel = 8'b00011000; // -- 7    **
            8:  pixel = 8'b00110000; // -- 8   **
            9:  pixel = 8'b00110000; // -- 9   **
            10: pixel = 8'b00110000; // -- a   **
            11: pixel = 8'b00110000; // -- b   **
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h38:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b01111100; // -- 2  *****
            3:  pixel = 8'b11000110; // -- 3 **   **
            4:  pixel = 8'b11000110; // -- 4 **   **
            5:  pixel = 8'b11000110; // -- 5 **   **
            6:  pixel = 8'b01111100; // -- 6  *****
            7:  pixel = 8'b11000110; // -- 7 **   **
            8:  pixel = 8'b11000110; // -- 8 **   **
            9:  pixel = 8'b11000110; // -- 9 **   **
            10: pixel = 8'b11000110; // -- a **   **
            11: pixel = 8'b01111100; // -- b  *****
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
```

```
8'h39:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b01111100; // -- 2  *****
            3:  pixel = 8'b11000110; // -- 3 **   **
            4:  pixel = 8'b11000110; // -- 4 **   **
            5:  pixel = 8'b11000110; // -- 5 **   **
            6:  pixel = 8'b01111110; // -- 6  ******
            7:  pixel = 8'b00000110; // -- 7      **
            8:  pixel = 8'b00000110; // -- 8      **
            9:  pixel = 8'b00000110; // -- 9      **
            10: pixel = 8'b00001100; // -- a     **
            11: pixel = 8'b01111000; // -- b  ****
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h3a:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b00000000; // -- 2
            3:  pixel = 8'b00000000; // -- 3
            4:  pixel = 8'b00011000; // -- 4    **
            5:  pixel = 8'b00011000; // -- 5    **
            6:  pixel = 8'b00000000; // -- 6
            7:  pixel = 8'b00000000; // -- 7
            8:  pixel = 8'b00000000; // -- 8
            9:  pixel = 8'b00011000; // -- 9    **
            10: pixel = 8'b00011000; // -- a    **
            11: pixel = 8'b00000000; // -- b
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h3b:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b00000000; // -- 2
            3:  pixel = 8'b00000000; // -- 3
            4:  pixel = 8'b00011000; // -- 4    **
            5:  pixel = 8'b00011000; // -- 5    **
            6:  pixel = 8'b00000000; // -- 6
            7:  pixel = 8'b00000000; // -- 7
            8:  pixel = 8'b00000000; // -- 8
            9:  pixel = 8'b00011000; // -- 9    **
```

```
            10: pixel = 8'b00011000; // -- a    **
            11: pixel = 8'b00110000; // -- b   **
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h3c:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b00000000; // -- 2
            3:  pixel = 8'b00000110; // -- 3      **
            4:  pixel = 8'b00001100; // -- 4     **
            5:  pixel = 8'b00011000; // -- 5    **
            6:  pixel = 8'b00110000; // -- 6   **
            7:  pixel = 8'b01100000; // -- 7  **
            8:  pixel = 8'b00110000; // -- 8   **
            9:  pixel = 8'b00011000; // -- 9    **
            10: pixel = 8'b00001100; // -- a     **
            11: pixel = 8'b00000110; // -- b      **
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h3d:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b00000000; // -- 2
            3:  pixel = 8'b00000000; // -- 3
            4:  pixel = 8'b00000000; // -- 4
            5:  pixel = 8'b01111110; // -- 5  ******
            6:  pixel = 8'b00000000; // -- 6
            7:  pixel = 8'b00000000; // -- 7
            8:  pixel = 8'b01111110; // -- 8  ******
            9:  pixel = 8'b00000000; // -- 9
            10: pixel = 8'b00000000; // -- a
            11: pixel = 8'b00000000; // -- b
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h3e:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
```

```
            2:  pixel = 8'b00000000; // -- 2
            3:  pixel = 8'b01100000; // -- 3  **
            4:  pixel = 8'b00110000; // -- 4   **
            5:  pixel = 8'b00011000; // -- 5    **
            6:  pixel = 8'b00001100; // -- 6     **
            7:  pixel = 8'b00000110; // -- 7      **
            8:  pixel = 8'b00001100; // -- 8     **
            9:  pixel = 8'b00011000; // -- 9    **
            10: pixel = 8'b00110000; // -- a   **
            11: pixel = 8'b01100000; // -- b  **
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h3f:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b01111100; // -- 2  *****
            3:  pixel = 8'b11000110; // -- 3 **   **
            4:  pixel = 8'b11000110; // -- 4 **   **
            5:  pixel = 8'b00001100; // -- 5     **
            6:  pixel = 8'b00011000; // -- 6    **
            7:  pixel = 8'b00011000; // -- 7    **
            8:  pixel = 8'b00011000; // -- 8    **
            9:  pixel = 8'b00000000; // -- 9
            10: pixel = 8'b00011000; // -- a    **
            11: pixel = 8'b00011000; // -- b    **
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h40:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b01111100; // -- 2  *****
            3:  pixel = 8'b11000110; // -- 3 **   **
            4:  pixel = 8'b11000110; // -- 4 **   **
            5:  pixel = 8'b11000110; // -- 5 **   **
            6:  pixel = 8'b11011110; // -- 6 ** ****
            7:  pixel = 8'b11011110; // -- 7 ** ****
            8:  pixel = 8'b11011110; // -- 8 ** ****
            9:  pixel = 8'b11011100; // -- 9 ** ***
            10: pixel = 8'b11000000; // -- a **
            11: pixel = 8'b01111100; // -- b  *****
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
```

```
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h41:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b00010000; // -- 2    *
            3:  pixel = 8'b00111000; // -- 3   ***
            4:  pixel = 8'b01101100; // -- 4  ** **
            5:  pixel = 8'b11000110; // -- 5 **   **
            6:  pixel = 8'b11000110; // -- 6 **   **
            7:  pixel = 8'b11111110; // -- 7 *******
            8:  pixel = 8'b11000110; // -- 8 **   **
            9:  pixel = 8'b11000110; // -- 9 **   **
            10: pixel = 8'b11000110; // -- a **   **
            11: pixel = 8'b11000110; // -- b **   **
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h42:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b11111100; // -- 2 ******
            3:  pixel = 8'b01100110; // -- 3  **  **
            4:  pixel = 8'b01100110; // -- 4  **  **
            5:  pixel = 8'b01100110; // -- 5  **  **
            6:  pixel = 8'b01111100; // -- 6  *****
            7:  pixel = 8'b01100110; // -- 7  **  **
            8:  pixel = 8'b01100110; // -- 8  **  **
            9:  pixel = 8'b01100110; // -- 9  **  **
            10: pixel = 8'b01100110; // -- a  **  **
            11: pixel = 8'b11111100; // -- b ******
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h43:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b00111100; // -- 2   ****
            3:  pixel = 8'b01100110; // -- 3  **  **
            4:  pixel = 8'b11000010; // -- 4 **    *
            5:  pixel = 8'b11000000; // -- 5 **
            6:  pixel = 8'b11000000; // -- 6 **
```

```
            7:  pixel = 8'b11000000; // -- 7 **
            8:  pixel = 8'b11000000; // -- 8 **
            9:  pixel = 8'b11000010; // -- 9 **    *
            10: pixel = 8'b01100110; // -- a  **  **
            11: pixel = 8'b00111100; // -- b   ****
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h44:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b11111000; // -- 2 *****
            3:  pixel = 8'b01101100; // -- 3  ** **
            4:  pixel = 8'b01100110; // -- 4  **  **
            5:  pixel = 8'b01100110; // -- 5  **  **
            6:  pixel = 8'b01100110; // -- 6  **  **
            7:  pixel = 8'b01100110; // -- 7  **  **
            8:  pixel = 8'b01100110; // -- 8  **  **
            9:  pixel = 8'b01100110; // -- 9  **  **
            10: pixel = 8'b01101100; // -- a  ** **
            11: pixel = 8'b11111000; // -- b *****
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h45:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b11111110; // -- 2 *******
            3:  pixel = 8'b01100110; // -- 3  **  **
            4:  pixel = 8'b01100010; // -- 4  **   *
            5:  pixel = 8'b01101000; // -- 5  ** *
            6:  pixel = 8'b01111000; // -- 6  ****
            7:  pixel = 8'b01101000; // -- 7  ** *
            8:  pixel = 8'b01100000; // -- 8  **
            9:  pixel = 8'b01100010; // -- 9  **   *
            10: pixel = 8'b01100110; // -- a  **  **
            11: pixel = 8'b11111110; // -- b *******
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h46:
    begin
```

```
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b11111110; // -- 2 *******
            3:  pixel = 8'b01100110; // -- 3  **  **
            4:  pixel = 8'b01100010; // -- 4  **   *
            5:  pixel = 8'b01101000; // -- 5  ** *
            6:  pixel = 8'b01111000; // -- 6  ****
            7:  pixel = 8'b01101000; // -- 7  ** *
            8:  pixel = 8'b01100000; // -- 8  **
            9:  pixel = 8'b01100000; // -- 9  **
            10: pixel = 8'b01100000; // -- a  **
            11: pixel = 8'b11110000; // -- b ****
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h47:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b00111100; // -- 2   ****
            3:  pixel = 8'b01100110; // -- 3  **  **
            4:  pixel = 8'b11000010; // -- 4 **    *
            5:  pixel = 8'b11000000; // -- 5 **
            6:  pixel = 8'b11000000; // -- 6 **
            7:  pixel = 8'b11011110; // -- 7 ** ****
            8:  pixel = 8'b11000110; // -- 8 **   **
            9:  pixel = 8'b11000110; // -- 9 **   **
            10: pixel = 8'b01100110; // -- a  **  **
            11: pixel = 8'b00111010; // -- b   *** *
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h48:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b11000110; // -- 2 **   **
            3:  pixel = 8'b11000110; // -- 3 **   **
            4:  pixel = 8'b11000110; // -- 4 **   **
            5:  pixel = 8'b11000110; // -- 5 **   **
            6:  pixel = 8'b11111110; // -- 6 *******
            7:  pixel = 8'b11000110; // -- 7 **   **
            8:  pixel = 8'b11000110; // -- 8 **   **
            9:  pixel = 8'b11000110; // -- 9 **   **
            10: pixel = 8'b11000110; // -- a **   **
            11: pixel = 8'b11000110; // -- b **   **
```

```
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h49:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b00111100; // -- 2   ****
            3:  pixel = 8'b00011000; // -- 3    **
            4:  pixel = 8'b00011000; // -- 4    **
            5:  pixel = 8'b00011000; // -- 5    **
            6:  pixel = 8'b00011000; // -- 6    **
            7:  pixel = 8'b00011000; // -- 7    **
            8:  pixel = 8'b00011000; // -- 8    **
            9:  pixel = 8'b00011000; // -- 9    **
            10: pixel = 8'b00011000; // -- a    **
            11: pixel = 8'b00111100; // -- b   ****
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h4a:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b00011110; // -- 2    ****
            3:  pixel = 8'b00001100; // -- 3     **
            4:  pixel = 8'b00001100; // -- 4     **
            5:  pixel = 8'b00001100; // -- 5     **
            6:  pixel = 8'b00001100; // -- 6     **
            7:  pixel = 8'b00001100; // -- 7     **
            8:  pixel = 8'b11001100; // -- 8 **  **
            9:  pixel = 8'b11001100; // -- 9 **  **
            10: pixel = 8'b11001100; // -- a **  **
            11: pixel = 8'b01111000; // -- b  ****
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h4b:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b11100110; // -- 2 ***  **
            3:  pixel = 8'b01100110; // -- 3  **  **
```

```
            4:  pixel = 8'b01100110; // -- 4  **  **
            5:  pixel = 8'b01101100; // -- 5  ** **
            6:  pixel = 8'b01111000; // -- 6  ****
            7:  pixel = 8'b01111000; // -- 7  ****
            8:  pixel = 8'b01101100; // -- 8  ** **
            9:  pixel = 8'b01100110; // -- 9  **  **
            10: pixel = 8'b01100110; // -- a  **  **
            11: pixel = 8'b11100110; // -- b ***  **
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h4c:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b11110000; // -- 2 ****
            3:  pixel = 8'b01100000; // -- 3  **
            4:  pixel = 8'b01100000; // -- 4  **
            5:  pixel = 8'b01100000; // -- 5  **
            6:  pixel = 8'b01100000; // -- 6  **
            7:  pixel = 8'b01100000; // -- 7  **
            8:  pixel = 8'b01100000; // -- 8  **
            9:  pixel = 8'b01100010; // -- 9  **   *
            10: pixel = 8'b01100110; // -- a  **  **
            11: pixel = 8'b11111110; // -- b *******
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h4d:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b11000011; // -- 2 **    **
            3:  pixel = 8'b11100111; // -- 3 ***  ***
            4:  pixel = 8'b11111111; // -- 4 ********
            5:  pixel = 8'b11111111; // -- 5 ********
            6:  pixel = 8'b11011011; // -- 6 ** ** **
            7:  pixel = 8'b11000011; // -- 7 **    **
            8:  pixel = 8'b11000011; // -- 8 **    **
            9:  pixel = 8'b11000011; // -- 9 **    **
            10: pixel = 8'b11000011; // -- a **    **
            11: pixel = 8'b11000011; // -- b **    **
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
```

```
    end
8'h4e:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b11000110; // -- 2 **   **
            3:  pixel = 8'b11100110; // -- 3 ***  **
            4:  pixel = 8'b11110110; // -- 4 **** **
            5:  pixel = 8'b11111110; // -- 5 *******
            6:  pixel = 8'b11011110; // -- 6 ** ****
            7:  pixel = 8'b11001110; // -- 7 **  ***
            8:  pixel = 8'b11000110; // -- 8 **   **
            9:  pixel = 8'b11000110; // -- 9 **   **
            10: pixel = 8'b11000110; // -- a **   **
            11: pixel = 8'b11000110; // -- b **   **
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h4f:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b01111100; // -- 2  *****
            3:  pixel = 8'b11000110; // -- 3 **   **
            4:  pixel = 8'b11000110; // -- 4 **   **
            5:  pixel = 8'b11000110; // -- 5 **   **
            6:  pixel = 8'b11000110; // -- 6 **   **
            7:  pixel = 8'b11000110; // -- 7 **   **
            8:  pixel = 8'b11000110; // -- 8 **   **
            9:  pixel = 8'b11000110; // -- 9 **   **
            10: pixel = 8'b11000110; // -- a **   **
            11: pixel = 8'b01111100; // -- b  *****
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h50:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b11111100; // -- 2 ******
            3:  pixel = 8'b01100110; // -- 3  **  **
            4:  pixel = 8'b01100110; // -- 4  **  **
            5:  pixel = 8'b01100110; // -- 5  **  **
            6:  pixel = 8'b01111100; // -- 6  *****
            7:  pixel = 8'b01100000; // -- 7  **
            8:  pixel = 8'b01100000; // -- 8  **
```

```
            9:  pixel = 8'b01100000; // -- 9  **
            10: pixel = 8'b01100000; // -- a  **
            11: pixel = 8'b11110000; // -- b ****
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h510:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b01111100; // -- 2  *****
            3:  pixel = 8'b11000110; // -- 3 **   **
            4:  pixel = 8'b11000110; // -- 4 **   **
            5:  pixel = 8'b11000110; // -- 5 **   **
            6:  pixel = 8'b11000110; // -- 6 **   **
            7:  pixel = 8'b11000110; // -- 7 **   **
            8:  pixel = 8'b11000110; // -- 8 **   **
            9:  pixel = 8'b11010110; // -- 9 ** * **
            10: pixel = 8'b11011110; // -- a ** ****
            11: pixel = 8'b01111100; // -- b  *****
            12: pixel = 8'b00001100; // -- c     **
            13: pixel = 8'b00001110; // -- d     ***
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h52:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b11111100; // -- 2 ******
            3:  pixel = 8'b01100110; // -- 3  **  **
            4:  pixel = 8'b01100110; // -- 4  **  **
            5:  pixel = 8'b01100110; // -- 5  **  **
            6:  pixel = 8'b01111100; // -- 6  *****
            7:  pixel = 8'b01101100; // -- 7  ** **
            8:  pixel = 8'b01100110; // -- 8  **  **
            9:  pixel = 8'b01100110; // -- 9  **  **
            10: pixel = 8'b01100110; // -- a  **  **
            11: pixel = 8'b11100110; // -- b ***  **
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h53:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
```

```
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b01111100; // -- 2  *****
            3:  pixel = 8'b11000110; // -- 3 **   **
            4:  pixel = 8'b11000110; // -- 4 **   **
            5:  pixel = 8'b01100000; // -- 5  **
            6:  pixel = 8'b00111000; // -- 6   ***
            7:  pixel = 8'b00001100; // -- 7     **
            8:  pixel = 8'b00000110; // -- 8      **
            9:  pixel = 8'b11000110; // -- 9 **   **
            10: pixel = 8'b11000110; // -- a **   **
            11: pixel = 8'b01111100; // -- b  *****
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h54:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b11111111; // -- 2 ********
            3:  pixel = 8'b11011011; // -- 3 ** ** **
            4:  pixel = 8'b10011001; // -- 4 *  **  *
            5:  pixel = 8'b00011000; // -- 5    **
            6:  pixel = 8'b00011000; // -- 6    **
            7:  pixel = 8'b00011000; // -- 7    **
            8:  pixel = 8'b00011000; // -- 8    **
            9:  pixel = 8'b00011000; // -- 9    **
            10: pixel = 8'b00011000; // -- a    **
            11: pixel = 8'b00111100; // -- b   ****
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h55:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b11000110; // -- 2 **   **
            3:  pixel = 8'b11000110; // -- 3 **   **
            4:  pixel = 8'b11000110; // -- 4 **   **
            5:  pixel = 8'b11000110; // -- 5 **   **
            6:  pixel = 8'b11000110; // -- 6 **   **
            7:  pixel = 8'b11000110; // -- 7 **   **
            8:  pixel = 8'b11000110; // -- 8 **   **
            9:  pixel = 8'b11000110; // -- 9 **   **
            10: pixel = 8'b11000110; // -- a **   **
            11: pixel = 8'b01111100; // -- b  *****
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
```

```
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h56:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b11000011; // -- 2 **    **
            3:  pixel = 8'b11000011; // -- 3 **    **
            4:  pixel = 8'b11000011; // -- 4 **    **
            5:  pixel = 8'b11000011; // -- 5 **    **
            6:  pixel = 8'b11000011; // -- 6 **    **
            7:  pixel = 8'b11000011; // -- 7 **    **
            8:  pixel = 8'b11000011; // -- 8 **    **
            9:  pixel = 8'b01100110; // -- 9  **  **
            10: pixel = 8'b00111100; // -- a   ****
            11: pixel = 8'b00011000; // -- b    **
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h57:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b11000011; // -- 2 **    **
            3:  pixel = 8'b11000011; // -- 3 **    **
            4:  pixel = 8'b11000011; // -- 4 **    **
            5:  pixel = 8'b11000011; // -- 5 **    **
            6:  pixel = 8'b11000011; // -- 6 **    **
            7:  pixel = 8'b11011011; // -- 7 ** ** **
            8:  pixel = 8'b11011011; // -- 8 ** ** **
            9:  pixel = 8'b11111111; // -- 9 ********
            10: pixel = 8'b01100110; // -- a  **  **
            11: pixel = 8'b01100110; // -- b  **  **
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end


8'h58:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b11000011; // -- 2 **    **
            3:  pixel = 8'b11000011; // -- 3 **    **
```

```
            4:  pixel = 8'b01100110; // -- 4  **  **
            5:  pixel = 8'b00111100; // -- 5   ****
            6:  pixel = 8'b00011000; // -- 6    **
            7:  pixel = 8'b00011000; // -- 7    **
            8:  pixel = 8'b00111100; // -- 8   ****
            9:  pixel = 8'b01100110; // -- 9  **  **
            10: pixel = 8'b11000011; // -- a **    **
            11: pixel = 8'b11000011; // -- b **    **
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h59:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b11000011; // -- 2 **    **
            3:  pixel = 8'b11000011; // -- 3 **    **
            4:  pixel = 8'b11000011; // -- 4 **    **
            5:  pixel = 8'b01100110; // -- 5  **  **
            6:  pixel = 8'b00111100; // -- 6   ****
            7:  pixel = 8'b00011000; // -- 7    **
            8:  pixel = 8'b00011000; // -- 8    **
            9:  pixel = 8'b00011000; // -- 9    **
            10: pixel = 8'b00011000; // -- a    **
            11: pixel = 8'b00111100; // -- b   ****
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h5a:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b11111111; // -- 2 ********
            3:  pixel = 8'b11000011; // -- 3 **    **
            4:  pixel = 8'b10000110; // -- 4 *    **
            5:  pixel = 8'b00001100; // -- 5     **
            6:  pixel = 8'b00011000; // -- 6    **
            7:  pixel = 8'b00110000; // -- 7   **
            8:  pixel = 8'b01100000; // -- 8  **
            9:  pixel = 8'b11000001; // -- 9 **     *
            10: pixel = 8'b11000011; // -- a **    **
            11: pixel = 8'b11111111; // -- b ********
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
```

```
    end
8'h5b:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b00111100; // -- 2   ****
            3:  pixel = 8'b00110000; // -- 3   **
            4:  pixel = 8'b00110000; // -- 4   **
            5:  pixel = 8'b00110000; // -- 5   **
            6:  pixel = 8'b00110000; // -- 6   **
            7:  pixel = 8'b00110000; // -- 7   **
            8:  pixel = 8'b00110000; // -- 8   **
            9:  pixel = 8'b00110000; // -- 9   **
            10: pixel = 8'b00110000; // -- a   **
            11: pixel = 8'b00111100; // -- b   ****
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h5c:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b00000000; // -- 2
            3:  pixel = 8'b10000000; // -- 3 *
            4:  pixel = 8'b11000000; // -- 4 **
            5:  pixel = 8'b11100000; // -- 5 ***
            6:  pixel = 8'b01110000; // -- 6  ***
            7:  pixel = 8'b00111000; // -- 7   ***
            8:  pixel = 8'b00011100; // -- 8    ***
            9:  pixel = 8'b00001110; // -- 9     ***
            10: pixel = 8'b00000110; // -- a      **
            11: pixel = 8'b00000010; // -- b       *
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h5d:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b00111100; // -- 2   ****
            3:  pixel = 8'b00001100; // -- 3     **
            4:  pixel = 8'b00001100; // -- 4     **
            5:  pixel = 8'b00001100; // -- 5     **
            6:  pixel = 8'b00001100; // -- 6     **
            7:  pixel = 8'b00001100; // -- 7     **
            8:  pixel = 8'b00001100; // -- 8     **
```

```
            9:  pixel = 8'b00001100; // -- 9     **
            10: pixel = 8'b00001100; // -- a     **
            11: pixel = 8'b00111100; // -- b   ****
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h5e:
    begin
        case (row)
            0:  pixel = 8'b00010000; // -- 0    *
            1:  pixel = 8'b00111000; // -- 1   ***
            2:  pixel = 8'b01101100; // -- 2  ** **
            3:  pixel = 8'b11000110; // -- 3 **   **
            4:  pixel = 8'b00000000; // -- 4
            5:  pixel = 8'b00000000; // -- 5
            6:  pixel = 8'b00000000; // -- 6
            7:  pixel = 8'b00000000; // -- 7
            8:  pixel = 8'b00000000; // -- 8
            9:  pixel = 8'b00000000; // -- 9
            10: pixel = 8'b00000000; // -- a
            11: pixel = 8'b00000000; // -- b
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h5f:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b00000000; // -- 2
            3:  pixel = 8'b00000000; // -- 3
            4:  pixel = 8'b00000000; // -- 4
            5:  pixel = 8'b00000000; // -- 5
            6:  pixel = 8'b00000000; // -- 6
            7:  pixel = 8'b00000000; // -- 7
            8:  pixel = 8'b00000000; // -- 8
            9:  pixel = 8'b00000000; // -- 9
            10: pixel = 8'b00000000; // -- a
            11: pixel = 8'b00000000; // -- b
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b11111111; // -- d ********
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h60:
    begin
        case (row)
            0:  pixel = 8'b00110000; // -- 0   **
```

```
            1:  pixel = 8'b00110000; // -- 1   **
            2:  pixel = 8'b00011000; // -- 2    **
            3:  pixel = 8'b00000000; // -- 3
            4:  pixel = 8'b00000000; // -- 4
            5:  pixel = 8'b00000000; // -- 5
            6:  pixel = 8'b00000000; // -- 6
            7:  pixel = 8'b00000000; // -- 7
            8:  pixel = 8'b00000000; // -- 8
            9:  pixel = 8'b00000000; // -- 9
            10: pixel = 8'b00000000; // -- a
            11: pixel = 8'b00000000; // -- b
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h61:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b00000000; // -- 2
            3:  pixel = 8'b00000000; // -- 3
            4:  pixel = 8'b00000000; // -- 4
            5:  pixel = 8'b01111000; // -- 5  ****
            6:  pixel = 8'b00001100; // -- 6     **
            7:  pixel = 8'b01111100; // -- 7  *****
            8:  pixel = 8'b11001100; // -- 8 **  **
            9:  pixel = 8'b11001100; // -- 9 **  **
            10: pixel = 8'b11001100; // -- a **  **
            11: pixel = 8'b01110110; // -- b  *** **
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h62:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b11100000; // -- 2  ***
            3:  pixel = 8'b01100000; // -- 3   **
            4:  pixel = 8'b01100000; // -- 4   **
            5:  pixel = 8'b01111000; // -- 5   ****
            6:  pixel = 8'b01101100; // -- 6   ** **
            7:  pixel = 8'b01100110; // -- 7   **  **
            8:  pixel = 8'b01100110; // -- 8   **  **
            9:  pixel = 8'b01100110; // -- 9   **  **
            10: pixel = 8'b01100110; // -- a   **  **
            11: pixel = 8'b01111100; // -- b   *****
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
```

```
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h63:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b00000000; // -- 2
            3:  pixel = 8'b00000000; // -- 3
            4:  pixel = 8'b00000000; // -- 4
            5:  pixel = 8'b01111100; // -- 5  *****
            6:  pixel = 8'b11000110; // -- 6 **   **
            7:  pixel = 8'b11000000; // -- 7 **
            8:  pixel = 8'b11000000; // -- 8 **
            9:  pixel = 8'b11000000; // -- 9 **
            10: pixel = 8'b11000110; // -- a **   **
            11: pixel = 8'b01111100; // -- b  *****
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h64:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b00011100; // -- 2    ***
            3:  pixel = 8'b00001100; // -- 3     **
            4:  pixel = 8'b00001100; // -- 4     **
            5:  pixel = 8'b00111100; // -- 5   ****
            6:  pixel = 8'b01101100; // -- 6  ** **
            7:  pixel = 8'b11001100; // -- 7 **  **
            8:  pixel = 8'b11001100; // -- 8 **  **
            9:  pixel = 8'b11001100; // -- 9 **  **
            10: pixel = 8'b11001100; // -- a **  **
            11: pixel = 8'b01110110; // -- b  *** **
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h65:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b00000000; // -- 2
            3:  pixel = 8'b00000000; // -- 3
            4:  pixel = 8'b00000000; // -- 4
            5:  pixel = 8'b01111100; // -- 5  *****
```

```
            6:  pixel = 8'b11000110; // -- 6 **   **
            7:  pixel = 8'b11111110; // -- 7 *******
            8:  pixel = 8'b11000000; // -- 8 **
            9:  pixel = 8'b11000000; // -- 9 **
            10: pixel = 8'b11000110; // -- a **   **
            11: pixel = 8'b01111100; // -- b  *****
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h66:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b00111000; // -- 2   ***
            3:  pixel = 8'b01101100; // -- 3  ** **
            4:  pixel = 8'b01100100; // -- 4  **  *
            5:  pixel = 8'b01100000; // -- 5  **
            6:  pixel = 8'b11110000; // -- 6 ****
            7:  pixel = 8'b01100000; // -- 7  **
            8:  pixel = 8'b01100000; // -- 8  **
            9:  pixel = 8'b01100000; // -- 9  **
            10: pixel = 8'b01100000; // -- a  **
            11: pixel = 8'b11110000; // -- b ****
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h67:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b00000000; // -- 2
            3:  pixel = 8'b00000000; // -- 3
            4:  pixel = 8'b00000000; // -- 4
            5:  pixel = 8'b01110110; // -- 5  *** **
            6:  pixel = 8'b11001100; // -- 6 **  **
            7:  pixel = 8'b11001100; // -- 7 **  **
            8:  pixel = 8'b11001100; // -- 8 **  **
            9:  pixel = 8'b11001100; // -- 9 **  **
            10: pixel = 8'b11001100; // -- a **  **
            11: pixel = 8'b01111100; // -- b  *****
            12: pixel = 8'b00001100; // -- c     **
            13: pixel = 8'b11001100; // -- d **  **
            14: pixel = 8'b01111000; // -- e  ****
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h68:
```

```
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b11100000; // -- 2 ***
            3:  pixel = 8'b01100000; // -- 3  **
            4:  pixel = 8'b01100000; // -- 4  **
            5:  pixel = 8'b01101100; // -- 5  ** **
            6:  pixel = 8'b01110110; // -- 6  *** **
            7:  pixel = 8'b01100110; // -- 7  **  **
            8:  pixel = 8'b01100110; // -- 8  **  **
            9:  pixel = 8'b01100110; // -- 9  **  **
            10: pixel = 8'b01100110; // -- a  **  **
            11: pixel = 8'b11100110; // -- b *** **
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h69:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b00011000; // -- 2    **
            3:  pixel = 8'b00011000; // -- 3    **
            4:  pixel = 8'b00000000; // -- 4
            5:  pixel = 8'b00111000; // -- 5   ***
            6:  pixel = 8'b00011000; // -- 6    **
            7:  pixel = 8'b00011000; // -- 7    **
            8:  pixel = 8'b00011000; // -- 8    **
            9:  pixel = 8'b00011000; // -- 9    **
            10: pixel = 8'b00011000; // -- a    **
            11: pixel = 8'b00111100; // -- b   ****
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h6a:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b00000110; // -- 2      **
            3:  pixel = 8'b00000110; // -- 3      **
            4:  pixel = 8'b00000000; // -- 4
            5:  pixel = 8'b00001110; // -- 5     ***
            6:  pixel = 8'b00000110; // -- 6      **
            7:  pixel = 8'b00000110; // -- 7      **
            8:  pixel = 8'b00000110; // -- 8      **
            9:  pixel = 8'b00000110; // -- 9      **
            10: pixel = 8'b00000110; // -- a      **
```

```
            11: pixel = 8'b00000110; // -- b      **
            12: pixel = 8'b01100110; // -- c  **  **
            13: pixel = 8'b01100110; // -- d  **  **
            14: pixel = 8'b00111100; // -- e   ****
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h6b:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b11100000; // -- 2 ***
            3:  pixel = 8'b01100000; // -- 3  **
            4:  pixel = 8'b01100000; // -- 4  **
            5:  pixel = 8'b01100110; // -- 5  **  **
            6:  pixel = 8'b01101100; // -- 6  ** **
            7:  pixel = 8'b01111000; // -- 7  ****
            8:  pixel = 8'b01111000; // -- 8  ****
            9:  pixel = 8'b01101100; // -- 9  ** **
            10: pixel = 8'b01100110; // -- a  **  **
            11: pixel = 8'b11100110; // -- b ***  **
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h6c:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b00111000; // -- 2   ***
            3:  pixel = 8'b00011000; // -- 3    **
            4:  pixel = 8'b00011000; // -- 4    **
            5:  pixel = 8'b00011000; // -- 5    **
            6:  pixel = 8'b00011000; // -- 6    **
            7:  pixel = 8'b00011000; // -- 7    **
            8:  pixel = 8'b00011000; // -- 8    **
            9:  pixel = 8'b00011000; // -- 9    **
            10: pixel = 8'b00011000; // -- a    **
            11: pixel = 8'b00111100; // -- b   ****
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h6d:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b00000000; // -- 2
```

```
            3:  pixel = 8'b00000000; // -- 3
            4:  pixel = 8'b00000000; // -- 4
            5:  pixel = 8'b11100110; // -- 5 ***  **
            6:  pixel = 8'b11111111; // -- 6 ********
            7:  pixel = 8'b11011011; // -- 7 ** ** **
            8:  pixel = 8'b11011011; // -- 8 ** ** **
            9:  pixel = 8'b11011011; // -- 9 ** ** **
            10: pixel = 8'b11011011; // -- a ** ** **
            11: pixel = 8'b11011011; // -- b ** ** **
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h6e:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b00000000; // -- 2
            3:  pixel = 8'b00000000; // -- 3
            4:  pixel = 8'b00000000; // -- 4
            5:  pixel = 8'b11011100; // -- 5 ** ***
            6:  pixel = 8'b01100110; // -- 6  **  **
            7:  pixel = 8'b01100110; // -- 7  **  **
            8:  pixel = 8'b01100110; // -- 8  **  **
            9:  pixel = 8'b01100110; // -- 9  **  **
            10: pixel = 8'b01100110; // -- a  **  **
            11: pixel = 8'b01100110; // -- b  **  **
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h6f:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b00000000; // -- 2
            3:  pixel = 8'b00000000; // -- 3
            4:  pixel = 8'b00000000; // -- 4
            5:  pixel = 8'b01111100; // -- 5  *****
            6:  pixel = 8'b11000110; // -- 6 **   **
            7:  pixel = 8'b11000110; // -- 7 **   **
            8:  pixel = 8'b11000110; // -- 8 **   **
            9:  pixel = 8'b11000110; // -- 9 **   **
            10: pixel = 8'b11000110; // -- a **   **
            11: pixel = 8'b01111100; // -- b  *****
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
```

```
        endcase
    end
8'h70:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b00000000; // -- 2
            3:  pixel = 8'b00000000; // -- 3
            4:  pixel = 8'b00000000; // -- 4
            5:  pixel = 8'b11011100; // -- 5 ** ***
            6:  pixel = 8'b01100110; // -- 6  **  **
            7:  pixel = 8'b01100110; // -- 7  **  **
            8:  pixel = 8'b01100110; // -- 8  **  **
            9:  pixel = 8'b01100110; // -- 9  **  **
            10: pixel = 8'b01100110; // -- a  **  **
            11: pixel = 8'b01111100; // -- b  *****
            12: pixel = 8'b01100000; // -- c  **
            13: pixel = 8'b01100000; // -- d  **
            14: pixel = 8'b11110000; // -- e ****
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h71:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b00000000; // -- 2
            3:  pixel = 8'b00000000; // -- 3
            4:  pixel = 8'b00000000; // -- 4
            5:  pixel = 8'b01110110; // -- 5  *** **
            6:  pixel = 8'b11001100; // -- 6 **  **
            7:  pixel = 8'b11001100; // -- 7 **  **
            8:  pixel = 8'b11001100; // -- 8 **  **
            9:  pixel = 8'b11001100; // -- 9 **  **
            10: pixel = 8'b11001100; // -- a **  **
            11: pixel = 8'b01111100; // -- b  *****
            12: pixel = 8'b00001100; // -- c     **
            13: pixel = 8'b00001100; // -- d     **
            14: pixel = 8'b00011110; // -- e    ****
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h72:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b00000000; // -- 2
            3:  pixel = 8'b00000000; // -- 3
            4:  pixel = 8'b00000000; // -- 4
            5:  pixel = 8'b11011100; // -- 5 ** ***
            6:  pixel = 8'b01110110; // -- 6  *** **
            7:  pixel = 8'b01100110; // -- 7  **  **
```

```
            8:  pixel = 8'b01100000; // -- 8  **
            9:  pixel = 8'b01100000; // -- 9  **
            10: pixel = 8'b01100000; // -- a  **
            11: pixel = 8'b11110000; // -- b ****
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h73:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b00000000; // -- 2
            3:  pixel = 8'b00000000; // -- 3
            4:  pixel = 8'b00000000; // -- 4
            5:  pixel = 8'b01111100; // -- 5  *****
            6:  pixel = 8'b11000110; // -- 6 **   **
            7:  pixel = 8'b01100000; // -- 7  **
            8:  pixel = 8'b00111000; // -- 8   ***
            9:  pixel = 8'b00001100; // -- 9     **
            10: pixel = 8'b11000110; // -- a **   **
            11: pixel = 8'b01111100; // -- b  *****
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h74:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b00010000; // -- 2    *
            3:  pixel = 8'b00110000; // -- 3   **
            4:  pixel = 8'b00110000; // -- 4   **
            5:  pixel = 8'b11111100; // -- 5 ******
            6:  pixel = 8'b00110000; // -- 6   **
            7:  pixel = 8'b00110000; // -- 7   **
            8:  pixel = 8'b00110000; // -- 8   **
            9:  pixel = 8'b00110000; // -- 9   **
            10: pixel = 8'b00110110; // -- a   ** **
            11: pixel = 8'b00011100; // -- b    ***
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h75:
    begin
        case (row)
```

```
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b00000000; // -- 2
            3:  pixel = 8'b00000000; // -- 3
            4:  pixel = 8'b00000000; // -- 4
            5:  pixel = 8'b11001100; // -- 5 **  **
            6:  pixel = 8'b11001100; // -- 6 **  **
            7:  pixel = 8'b11001100; // -- 7 **  **
            8:  pixel = 8'b11001100; // -- 8 **  **
            9:  pixel = 8'b11001100; // -- 9 **  **
            10: pixel = 8'b11001100; // -- a **  **
            11: pixel = 8'b01110110; // -- b  *** **
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h76:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b00000000; // -- 2
            3:  pixel = 8'b00000000; // -- 3
            4:  pixel = 8'b00000000; // -- 4
            5:  pixel = 8'b11000011; // -- 5 **    **
            6:  pixel = 8'b11000011; // -- 6 **    **
            7:  pixel = 8'b11000011; // -- 7 **    **
            8:  pixel = 8'b11000011; // -- 8 **    **
            9:  pixel = 8'b01100110; // -- 9  **  **
            10: pixel = 8'b00111100; // -- a   ****
            11: pixel = 8'b00011000; // -- b    **
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h77:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b00000000; // -- 2
            3:  pixel = 8'b00000000; // -- 3
            4:  pixel = 8'b00000000; // -- 4
            5:  pixel = 8'b11000011; // -- 5 **    **
            6:  pixel = 8'b11000011; // -- 6 **    **
            7:  pixel = 8'b11000011; // -- 7 **    **
            8:  pixel = 8'b11011011; // -- 8 ** ** **
            9:  pixel = 8'b11011011; // -- 9 ** ** **
            10: pixel = 8'b11111111; // -- a ********
            11: pixel = 8'b01100110; // -- b  **  **
            12: pixel = 8'b00000000; // -- c
```

```
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h78:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b00000000; // -- 2
            3:  pixel = 8'b00000000; // -- 3
            4:  pixel = 8'b00000000; // -- 4
            5:  pixel = 8'b11000011; // -- 5 **    **
            6:  pixel = 8'b01100110; // -- 6  **  **
            7:  pixel = 8'b00111100; // -- 7   ****
            8:  pixel = 8'b00011000; // -- 8    **
            9:  pixel = 8'b00111100; // -- 9   ****
            10: pixel = 8'b01100110; // -- a  **  **
            11: pixel = 8'b11000011; // -- b **    **
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h79:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b00000000; // -- 2
            3:  pixel = 8'b00000000; // -- 3
            4:  pixel = 8'b00000000; // -- 4
            5:  pixel = 8'b11000110; // -- 5 **   **
            6:  pixel = 8'b11000110; // -- 6 **   **
            7:  pixel = 8'b11000110; // -- 7 **   **
            8:  pixel = 8'b11000110; // -- 8 **   **
            9:  pixel = 8'b11000110; // -- 9 **   **
            10: pixel = 8'b11000110; // -- a **   **
            11: pixel = 8'b01111110; // -- b  ******
            12: pixel = 8'b00000110; // -- c      **
            13: pixel = 8'b00001100; // -- d     **
            14: pixel = 8'b11111000; // -- e *****
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h7a:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b00000000; // -- 2
            3:  pixel = 8'b00000000; // -- 3
            4:  pixel = 8'b00000000; // -- 4
```

```
            5:  pixel = 8'b11111110; // -- 5 *******
            6:  pixel = 8'b11001100; // -- 6 **  **
            7:  pixel = 8'b00011000; // -- 7    **
            8:  pixel = 8'b00110000; // -- 8   **
            9:  pixel = 8'b01100000; // -- 9  **
            10: pixel = 8'b11000110; // -- a **   **
            11: pixel = 8'b11111110; // -- b *******
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h7b:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b00001110; // -- 2     ***
            3:  pixel = 8'b00011000; // -- 3    **
            4:  pixel = 8'b00011000; // -- 4    **
            5:  pixel = 8'b00011000; // -- 5    **
            6:  pixel = 8'b01110000; // -- 6  ***
            7:  pixel = 8'b00011000; // -- 7    **
            8:  pixel = 8'b00011000; // -- 8    **
            9:  pixel = 8'b00011000; // -- 9    **
            10: pixel = 8'b00011000; // -- a    **
            11: pixel = 8'b00001110; // -- b     ***
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h7c:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b00011000; // -- 2    **
            3:  pixel = 8'b00011000; // -- 3    **
            4:  pixel = 8'b00011000; // -- 4    **
            5:  pixel = 8'b00011000; // -- 5    **
            6:  pixel = 8'b00000000; // -- 6
            7:  pixel = 8'b00011000; // -- 7    **
            8:  pixel = 8'b00011000; // -- 8    **
            9:  pixel = 8'b00011000; // -- 9    **
            10: pixel = 8'b00011000; // -- a    **
            11: pixel = 8'b00011000; // -- b    **
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
```

```
8'h7d:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b01110000; // -- 2  ***
            3:  pixel = 8'b00011000; // -- 3    **
            4:  pixel = 8'b00011000; // -- 4    **
            5:  pixel = 8'b00011000; // -- 5    **
            6:  pixel = 8'b00001110; // -- 6     ***
            7:  pixel = 8'b00011000; // -- 7    **
            8:  pixel = 8'b00011000; // -- 8    **
            9:  pixel = 8'b00011000; // -- 9    **
            10: pixel = 8'b00011000; // -- a    **
            11: pixel = 8'b01110000; // -- b  ***
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h7e:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b01110110; // -- 2  *** **
            3:  pixel = 8'b11011100; // -- 3 ** ***
            4:  pixel = 8'b00000000; // -- 4
            5:  pixel = 8'b00000000; // -- 5
            6:  pixel = 8'b00000000; // -- 6
            7:  pixel = 8'b00000000; // -- 7
            8:  pixel = 8'b00000000; // -- 8
            9:  pixel = 8'b00000000; // -- 9
            10: pixel = 8'b00000000; // -- a
            11: pixel = 8'b00000000; // -- b
            12: pixel = 8'b00000000; // -- c
            13: pixel = 8'b00000000; // -- d
            14: pixel = 8'b00000000; // -- e
            15: pixel = 8'b00000000; // -- f
        endcase
    end
8'h7f:
    begin
        case (row)
            0:  pixel = 8'b00000000; // -- 0
            1:  pixel = 8'b00000000; // -- 1
            2:  pixel = 8'b00000000; // -- 2
            3:  pixel = 8'b00000000; // -- 3
            4:  pixel = 8'b00010000; // -- 4    *
            5:  pixel = 8'b00111000; // -- 5   ***
            6:  pixel = 8'b01101100; // -- 6  ** **
            7:  pixel = 8'b11000110; // -- 7 **   **
            8:  pixel = 8'b11000110; // -- 8 **   **
            9:  pixel = 8'b11000110; // -- 9 **   **
```

```
                    10: pixel = 8'b11111110; // -- a *******
                    11: pixel = 8'b00000000; // -- b
                    12: pixel = 8'b00000000; // -- c
                    13: pixel = 8'b00000000; // -- d
                    14: pixel = 8'b00000000; // -- e
                    15: pixel = 8'b00000000; // -- f
                endcase
               end
            default:pixel=8'd0;
        endcase
    end
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 10/02/2015 02:05:19 AM; comments added 7/24/2018
// Design Name:
// Module Name: xvga
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Revision:
// Revision 1.0 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////////
//
// xvga: Generate XVGA display signals (1024 x 768 @ 60Hz)
// vga: 640x480 verilog is also included by commented out
//
//////////////////////////////////////////////////////////////////////////////


module xvga(input vclock,
            output reg [10:0] hcount,    // pixel number on current line
            output reg [9:0] vcount,     // line number
            output reg vsync,hsync,
            output reg blank);

   // horizontal: 1344 pixels total
   // display 1024 pixels per line
   reg hblank,vblank;
   wire hsyncon,hsyncoff,hreset,hblankon;
   assign hblankon = (hcount == 1023);
   assign hsyncon = (hcount == 1047);
   assign hsyncoff = (hcount == 1183);
   assign hreset = (hcount == 1343);

   // vertical: 806 lines total
   // display 768 lines
   wire vsyncon,vsyncoff,vreset,vblankon;
   assign vblankon = hreset & (vcount == 767);
   assign vsyncon = hreset & (vcount == 776);
   assign vsyncoff = hreset & (vcount == 782);
   assign vreset = hreset & (vcount == 805);

   // sync and blanking
   wire next_hblank,next_vblank;
   assign next_hblank = hreset ? 0 : hblankon ? 1 : hblank;
   assign next_vblank = vreset ? 0 : vblankon ? 1 : vblank;
```

```
   always @(posedge vclock) begin
      hcount <= hreset ? 0 : hcount + 1;
      hblank <= next_hblank;
      hsync <= hsyncon ? 0 : hsyncoff ? 1 : hsync;  // active low

      vcount <= hreset ? (vreset ? 0 : vcount + 1) : vcount;
      vblank <= next_vblank;
      vsync <= vsyncon ? 0 : vsyncoff ? 1 : vsync;  // active low

      blank <= next_vblank | (next_hblank & ~hreset);
   end

//   assign at_display_area = ((hcount >= 0) && (hcount < 1024) && (vcount >= 0) &&
(vcount < 768));
endmodule

/*
module clock_quarter_divider(input clk100_mhz, output reg clock_25mhz = 0);
    reg counter = 0;

    // VERY BAD VERILOG
    // VERY BAD VERILOG
    // VERY BAD VERILOG
    // But it's a quick and dirty way to create a 25Mhz clock
    // Please use the IP Clock Wizard under FPGA Features/Clocking
    //
    // For 1 Hz pulse, it's okay to use a counter to create the pulse as in
    // assign onehz = (counter == 100_000_000);
    // be sure to have the right number of bits.

    always @(posedge clk100_mhz) begin
        counter <= counter + 1;
        if (counter == 0) begin
            clock_25mhz <= ~clock_25mhz;
        end
    end
endmodule

module vga(input vclock,   // clock is now 25Mhz for 640x480
            output reg [10:0] hcount = 0,   // extra bit for compatibility
            output reg [9:0] vcount = 0,    // line number
            output reg vsync, hsync,
            output reg blank);

   // Comments applies to XVGA 1024x768, left in for reference
   // horizontal: 1344 pixels total
   // display 1024 pixels per line
   reg hblank,vblank;
   wire hsyncon,hsyncoff,hreset,hblankon;
   assign hblankon = (hcount == 639);    // active H  1023
   assign hsyncon = (hcount == 655);     // active H + FP 1047
   assign hsyncoff = (hcount == 751);    // active H + fp + sync  1183
   assign hreset = (hcount == 799);      // active H + fp + sync + bp 1343

   // vertical: 806 lines total
```

```
   // display 768 lines
   wire vsyncon,vsyncoff,vreset,vblankon;
   assign vblankon = hreset & (vcount == 479);    // active V   767
   assign vsyncon = hreset & (vcount ==490 );     // active V + fp   776
   assign vsyncoff = hreset & (vcount == 492);    // active V + fp + sync  783
   assign vreset = hreset & (vcount == 523);      // active V + fp + sync + bp 805

   // sync and blanking
   wire next_hblank,next_vblank;
   assign next_hblank = hreset ? 0 : hblankon ? 1 : hblank;
   assign next_vblank = vreset ? 0 : vblankon ? 1 : vblank;
   always @(posedge vclock) begin
      hcount <= hreset ? 0 : hcount + 1;
      hblank <= next_hblank;
      hsync <= hsyncon ? 0 : hsyncoff ? 1 : hsync;  // active low

      vcount <= hreset ? (vreset ? 0 : vcount + 1) : vcount;
      vblank <= next_vblank;
      vsync <= vsyncon ? 0 : vsyncoff ? 1 : vsync;  // active low

      blank <= next_vblank | (next_hblank & ~hreset);
   end


endmodule
*/
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:   g.p.hom
// Engineer:
//
// Create Date:    18:18:59 04/21/2013
// Module Name:    display_8hex

// Description:  Display 8 hex numbers on 7 segment display
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module display_8hex(
    input wire clk,                 // system clock
    input wire [31:0] data,         // 8 hex numbers, msb first
    output reg [6:0] seg,      // seven segment display output
    output reg [7:0] strobe    // digit strobe
    );

    localparam bits = 13;

    reg [bits:0] counter = 0;  // clear on power up

    wire [6:0] segments[15:0]; // 16 7 bit memorys
    assign segments[0]  = 7'b100_0000;
    assign segments[1]  = 7'b111_1001;
    assign segments[2]  = 7'b010_0100;
    assign segments[3]  = 7'b011_0000;
    assign segments[4]  = 7'b001_1001;
    assign segments[5]  = 7'b001_0010;
    assign segments[6]  = 7'b000_0010;
    assign segments[7]  = 7'b111_1000;
    assign segments[8]  = 7'b000_0000;
    assign segments[9]  = 7'b001_1000;
    assign segments[10] = 7'b000_1000;
    assign segments[11] = 7'b000_0011;
    assign segments[12] = 7'b010_0111;
    assign segments[13] = 7'b010_0001;
    assign segments[14] = 7'b000_0110;
    assign segments[15] = 7'b000_1110;

    always @(posedge clk) begin
      counter <= counter + 1;
      case (counter[bits:bits-2])
          3'b000: begin
                  seg <= segments[data[31:28]];
                  strobe <= 8'b0111_1111 ;
                 end

          3'b001: begin
                  seg <= segments[data[27:24]];
```

```
                strobe <= 8'b1011_1111 ;
               end

          3'b010: begin
                   seg <= segments[data[23:20]];
                   strobe <= 8'b1101_1111 ;
                  end
          3'b011: begin
                  seg <= segments[data[19:16]];
                  strobe <= 8'b1110_1111;
                 end
          3'b100: begin
                  seg <= segments[data[15:12]];
                  strobe <= 8'b1111_0111;
                 end

          3'b101: begin
                  seg <= segments[data[11:8]];
                  strobe <= 8'b1111_1011;
                 end

          3'b110: begin
                   seg <= segments[data[7:4]];
                   strobe <= 8'b1111_1101;
                  end
          3'b111: begin
                  seg <= segments[data[3:0]];
                  strobe <= 8'b1111_1110;
                 end



        endcase
       end

endmodule
```