6.111 Fall 2016

12-LEAD EKG

Final Project Report

Jeremy Ellison & Stone Montegomery

Contents

1 Introduction ... 3

1.1 Goals ... 4

2 High-Level Technical Overview ... 5

3 Frontend (Stone) ... 4

3.1 Analog ... 6

3.1.1 Input ... 7

3.1.2 Amplification Stage 1 ... 8

3.1.3 Filter Stage 1 ... 8

3.1.4 Analog Multiplexer ... 9

3.1.5 Wilson's Central Terminal ... 10

3.1.6 Amplification 2 ... 11

3.1.7 DC Offset/HP Filter ... 11

3.2 Allocation Subsystem ... 11

3.2.1 ADC ... 12

3.2.2 Recorder ... 12

4 Backend Processing and Display (Jeremy) ... 13

4.1 Wave Container Module ... 13

4.1.1 One Axis ... 13

4.1.2 Two Axes ... 14

4.1.3 Four Axes ... 15

4.1.4 Twelve Axes ... 17

4.2 Wave Module ... 18

4.2.1 EKG Memory ... 18

4.2.2 Wave Display ... 19

4.2.2.1 Reading Memory ... 19

4.2.2.2 Generating Pixel Data ... 20

4.2.2.3 Pixel Connection ... 20

4.3 Heart Rate Module ... 21

4.3.1 Heart Rate Calculator ... 21

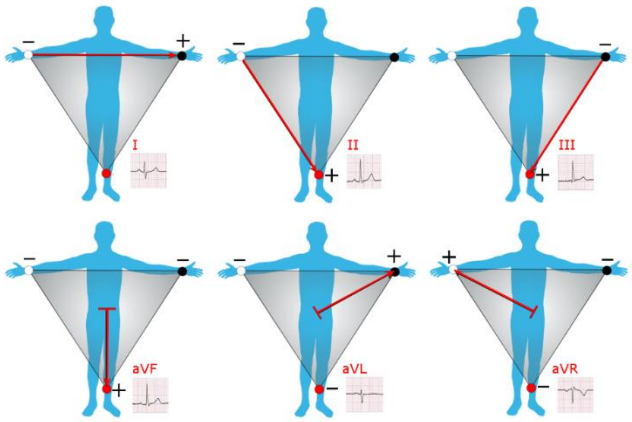
4.3.2 Heart Rate Display ... 21

5 Review and Conclusion **22**

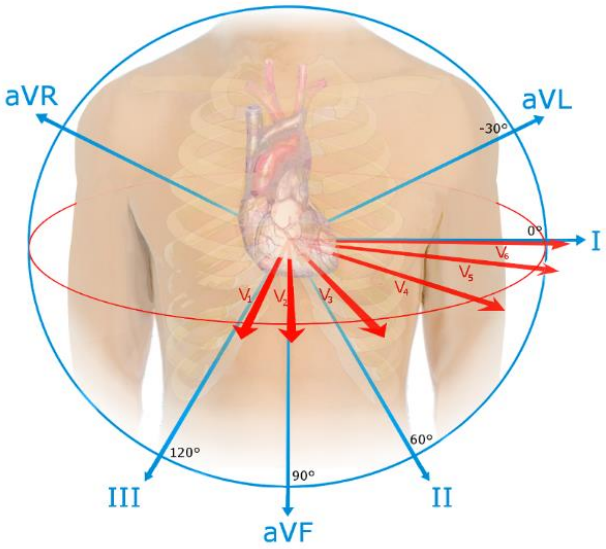
Appendix **23**

1 Introduction

Electrocardiograms (EKG's or ECG's) are used by hospital and pre-hospital medical professionals to diagnose and monitor cardiac problems in patients. EKG's work by measuring uV level signals generated by a patient's heart and filtering/amplifying the signal enough for human interpretation. In this project, we attempted to create an industry standard 12-lead EKG. A 12-lead EKG uses only 10 physical leads and combines them in different directions and configurations to create 12 virtual leads. Below are the placements of the physical leads and the polarity direction of the virtual leads:



Above: Position and direction of the 3 limb leads and augmented leads.



Above: Position and direction of all 12 leads.

Lead Name	Positive Lead	Negative Lead
I	LA	RA
II	LL	RA
III	LL	LA
aVL	RA	V_w
aVR	LA	V_w
aVF	LL	V_w
V1	V1	V_w
V2	V2	V_w
V3	V3	V_w
V4	V4	V_w
V5	V5	V_w
V6	V6	V_w

Many 12-leads EKGs allow the user to view any of the 12 possible leads. However, current models do not have the capability to view an arbitrary number of leads simultaneously and dynamically reconfigure the screen layout to accommodate the number of leads that the user wants to view concurrently. We aim to create a system that simultaneously record data from all 12 leads and allow the user to view one, some, or all leads at once, dynamically changing the screen layout to accommodate the user's current request.

1.1 Goals

Primary Goal: Display a single lead waveform

- Requires properly functioning Leads, Filter, Amplifier, ADC, Recorder Module

- Requires properly functioning waveform display

Secondary Goal: Ability to display patient heart rate and all 12 leads individually

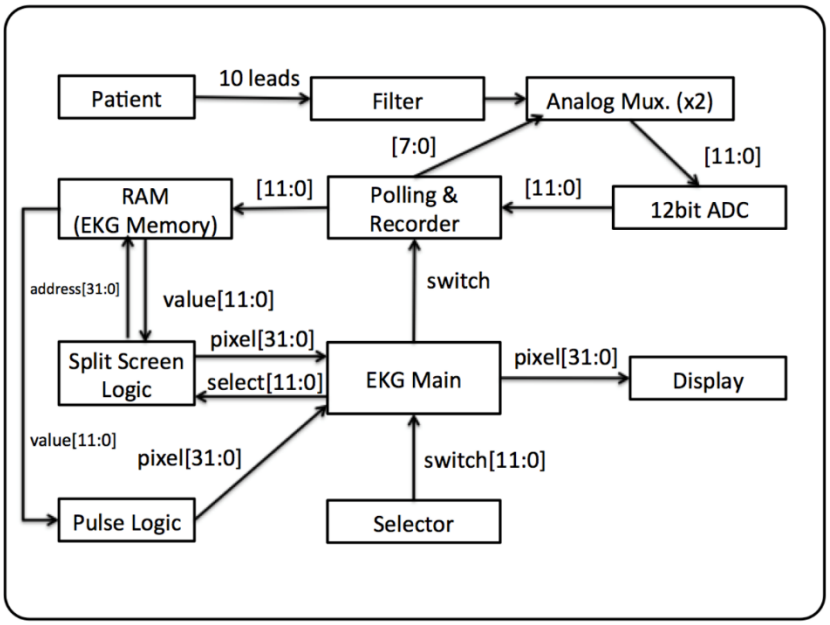
- Requires all functionality from Primary Goal section
- Requires properly functioning Pulse Logic and Selector
- Requires properly functioning Poller and Recorder and Analog Multiplexer

Stretch Goal: Ability to display all 12 leads simultaneously

- Requires all functionality from the Secondary Goal section
- Requires properly functioning Split Screen Logic

2 High-Level Technical Overview

As described in the following sections, our 12-Lead EKG is split into two primary sections. The first section is the Frontend. This section includes ten physical leads from the patient are brought into an analog filtering board by a braided cable. After filtering, the two relevant leads are selected by the two analog multiplexers. The output of the multiplexers is amplified and read by an ADC. Polling and Recorder unit set the select lines and coordinates the transfer of the value from the ADC to the correct place in the EKG Memory RAM.



Project Block Diagram

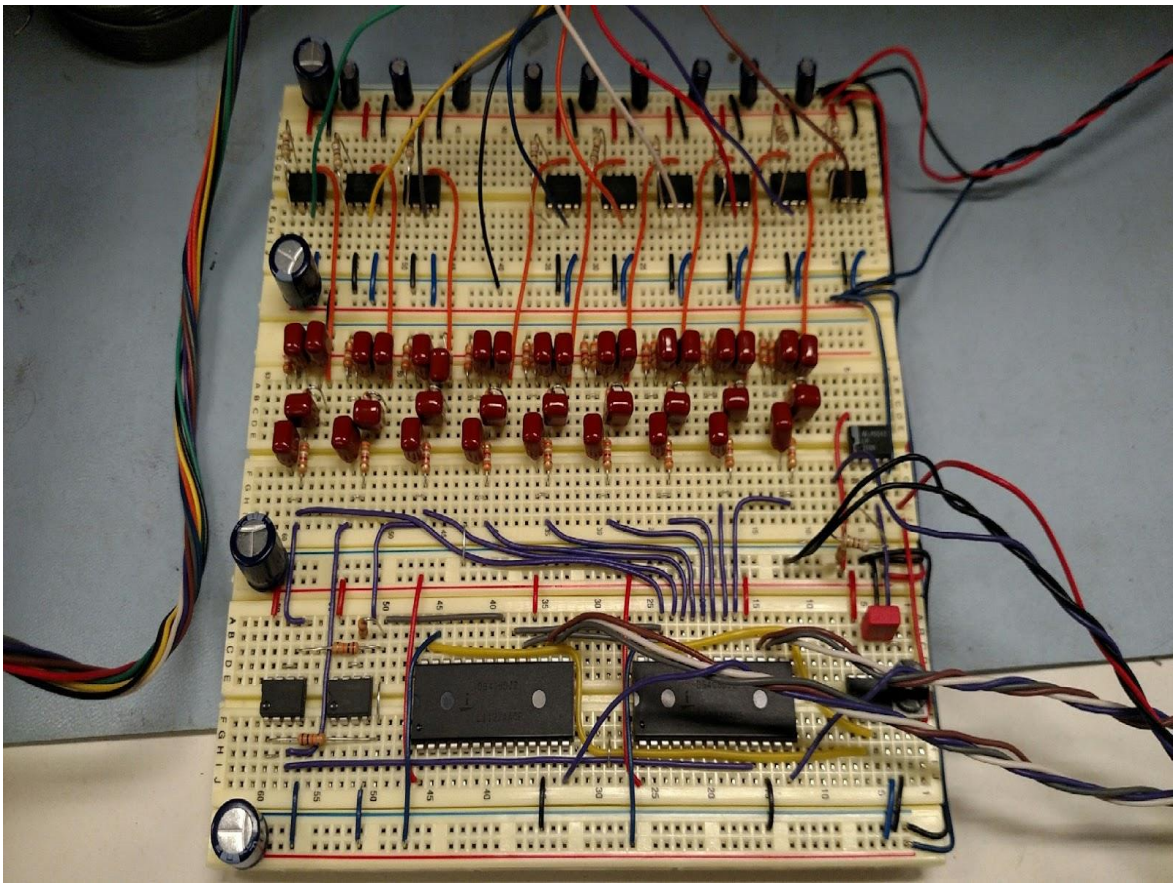
The other section is the Backend and Display. This section includes the EKG Memory, Split Screen Logic, Pulse (Heart Rate) Logic, and the Display. The Split Screen Logic reads data from EKG Memory and processes the data to generate pixel values to make up waveforms on the display. The Pulse Logic takes the value to be written to EKG Memory and processes them to calculate the patient's heart rate. This heart rate is then used to generate digits on the display on the form of pixels.

3 Frontend

The 12-lead EKG system is roughly split into two main systems. The first is the frontend which processes the data from the patient and organizes it, and the backend which takes this organized data and displays it in a readable, user-adjustable format for the medical professional using the device. The first system discussed is the frontend which is further composed of an Analog acquisition subsystem and an allocation subsystem.

3.1 Analog

The analog section of this project served to being in signals from the patient (or in our case, a patient simulator) and condition it so that it could be read by the recorder and displayed by the backend. This section is comprised of several parts: the input, a gain stage, a filter, an analog multiplexer, another gain stage, and another filter. This section of the report will outline each of these stages and their operation. The below picture is the analog filtering and analog selection circuitry.



3.1.1 Input

The input to our system was a 12-lead patient simulator used by EMS professionals to practice analysis and treatment of heart arrhythmias. This was selected over using ourselves or other people as the signal source for several reasons. First, using a simulator prevented using a large number of lead ends, the sticky pads that connect a patient to the EKG leads. In addition, it was more portable, and we could leave the work area without connecting and disconnecting ourselves from the system. It also produced a more consistent reading. Less noise was present in the simulator than when we attempted using our own bodies as the source. Lastly, using a simulator allowed us to view and test the EKG with signals other than normal rhythms. Since we cannot have anyone voluntarily enter an abnormal heart rhythm, viewing one with the simulator was as easy as pressing a button.

3.1.2 Amplification Stage 1

When selecting the amplifiers to be used for our filtering hardware, our largest concern was the offset voltage required to change the output. The patient simulator output signals in the range of 100uV to 1mV, meaning any offset voltage would have to be much smaller. We settled on using an INA126 Instrumentation amplifier. An instrumentation amplifier contains 3 typical operational amplifiers and creates a small, high-gain package, with very low offset voltage. The internal schematic of the INA126 is shown below. Note that the gain of the stage can also be set with a single resistor in the range of 5x - 10000x. Each amp was provided with +12V and -12V power rails.

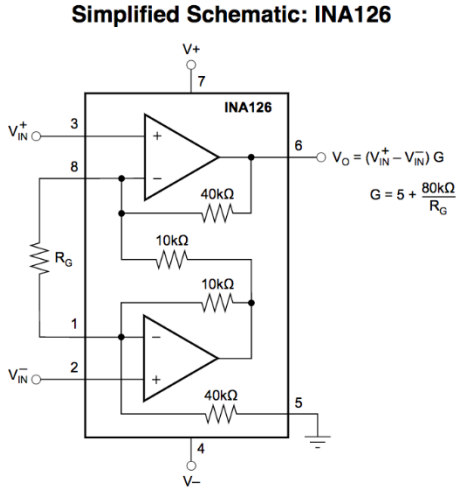


Image from Texas Instruments INA126 Datasheet.

3.1.3 Filter Stage 1

The most prevalent source of noise in our system was 60Hz noise from the mains power line. Since the patient simulator could be kept close to the analog circuitry, wires were able to be kept short. In addition, since all 10 physical leads to the patient were wound together, much of this noise was eliminated before it entered the system. The remaining noise was eliminated using a notch filter, a specific, high-Q factor band stop filter. The values selected for R and C were 680nF and 3.86kOhm respectively resulting in an attenuated frequency of 60Hz. The schematic below shows the schematic of the filter used:

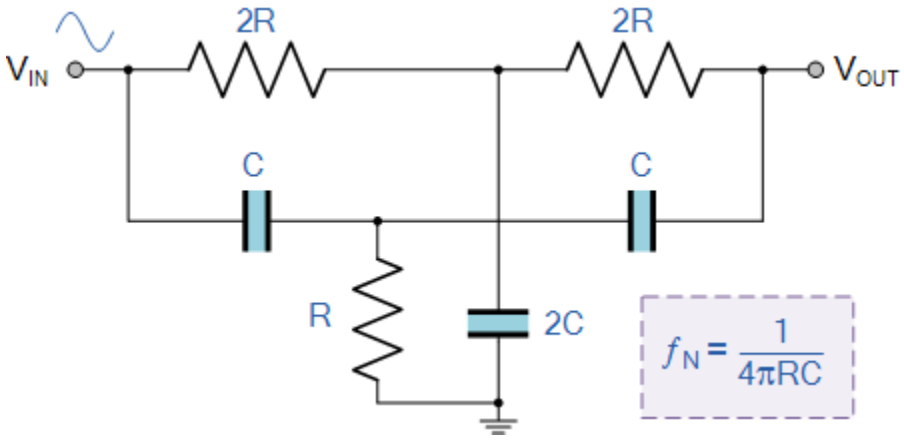
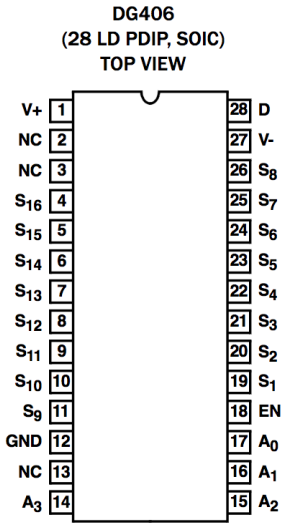


Image from: electronics-tutorials.com

3.1.4 Analog Multiplexer

There are 10 physical leads mentioned above the when combined in different combinations, create the 12 virtual leads that are displayed on the screen. Our goals included simultaneously recording all 12 of these leads so that when the user selects a lead to view, the last 3 seconds are immediately visible and that when all 12 leads are viewed together, they all update smoothly together. In order to make this happen, we needed to poll each lead fast enough that the updates be smooth to a user looking at the signal on the screen. Since we wanted to sample 12 leads, we decided on a sample rate of 4k samples/second leaving 333 samples/second/lead. As described in the chart in the Introduction section, for each virtual lead, one of the physical leads takes the positive and negative input to the next stage of this amplifier. Therefore, we needed to select 1 of 10 leads to be the positive and negative leads and be able to switch this selection quickly. To accomplish this, we used an analog multiplexer. The multiplexer selected was Intersil DG406DJZ since it was available in the desired package and was able to operate above the necessary speed.



Two of these units were used - one to select the high lead and one to select the low lead. The multiplexer controlling the high lead used input 1-9 and the low used inputs 1-3. Both multiplexers had their enable lines ties high and were connected to +12V and -12V rails. The inputs for the address lines A_0, A_1, A_2, A_3 were connected to the JA and JB outputs of the Nexys 4. The method for doing of doing so is described in the Recorder section of this report. We originally had some difficulty where the multiplexer was adding a different DC offset to each lead. When amplified by the succeeding gain stage, this would great unrealistic results. This effect was minimized by loading the output of each multiplexer with a 100kOhm resistor.

3.1.5 Wilson's Central Terminal

Of the 12 virtual leads, 9 of them use an "11th" lead as its negative reference. This lead is the average of the Left Arm, Right Arm, and Left Leg leads. To create this, we place voltage buffers on the output of the appropriate leads after the 60Hz filter and created an average voltage using equi-valued resistors. The schematic is shown below:

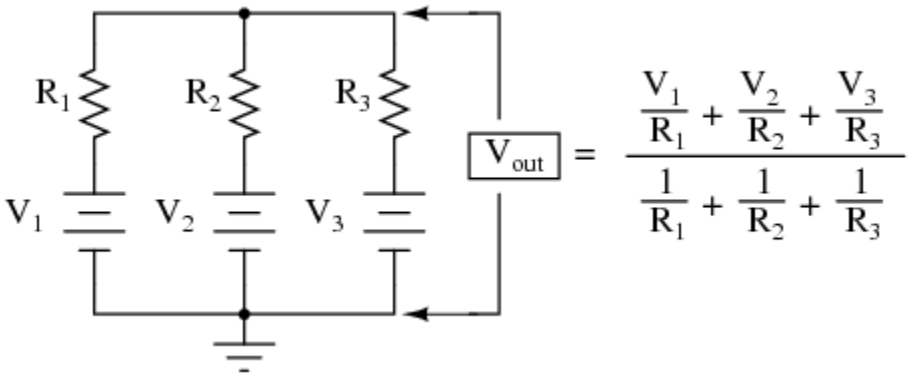


Image from: allaboutcircuits.com

3.1.6 Amplification 2

After selection of the positive and negative physical leads corresponding to the appropriate virtual lead, the signal is again amplified by another INA126. This amplification is lower than the previous gain stages, limiting the amplification of noise or DC offsets introduced by the prior conditioning stages. Since the R_g terminals are left floating, R_g is practically infinite and the gain of this stage can be approximated at 5, given the formula from the datasheet excerpt above.

3.1.7 DC Offset/HP Filter

In order for the EKG signal to be effectively read by the ADC on the Nexys 4, the signal must range from 0V-1V. Normal heart signals, typically have a negative component, which is amplified alongside the positive parts during signal filtering. To ensure that the voltage is always positive, we add a user-defined DC component to the signal. This stage was created by using a low-resistance potentiometer and coupling its output to the output of the 2nd gain stage. This both added a DC component to the signal and created an LC high-pass filter using between the capacitor and the inherent resistance of the potentiometer.

3.2 Allocation Subsystem

The Allocation Subsystem is the second primary part of the front-end system, taking the signal from the analog part of the frontend and collecting and allocation the data in the appropriate format for use by the backend. This subsystem is comprised of two main

components, the Analog to Digital Converter (ADC) and the module called Recorder. The implementation and use of each is described in the following sections.

3.2.1 ADC

The Nexys 4 has a built-in 12-bit Analog to Digital converter. While having several channels built-in, we opted to use external multiplexing since there were not sufficient (12) channels to do the multiplexing entirely on the Nexys. Like mentioned above, we used a 4kHz sample clock. However, the ADC was running a 100kHz sample clock. In this way, we could minimize the likelihood of errors due to reading a value from the ADC the corresponded to the previous lead. We initially had trouble getting the ADC to sample at all, eventually learning that we were attempting to sample below its minimum sample rate. The ADC was configured as a single differential channel and the negative terminal was shorted to GND, allowing a 1-wire interface between the analog filtering board and the Nexys 4.

3.2.2 Recorder

The Recorder module, as it is referred to in both this report and the Verilog, comprised the majority of the digital logic component of the frontend. The Recorder is tasked with reading values from the ADC, assigning the appropriate values to the select lines of the analog multiplexers and directing the converted value to the memory bank corresponding to the appropriate lead. The recorder maintains an internal value called lead which is incremented on each 4 kHz clock pulse. The recorder uses this value in a 12-section case statement to appropriately set the select lines to the analog multiplexers. The on each 4kHz pulse, the ADC value is read, the lines select lines are set, and a write value is set. The write value is a 12-bit value from which each of the BRAMs for the 12 leads takes a single bit for its write value. At most, 1 bit of the write value is set at any one time.

4 Backend Processing and Display

The Backend Processing and Display of the 12-Lead EKG includes the EKG Memory, Split Screen, Heart Rate, and Display modules. From the Frontend, the Backend receives a 12-bit ACD value between 0 and 4096 and a 12-bit write value from the Recorder module at 4KHz. The ACD value is a digital representation of the voltage from the Frontend. The write value dictates which lead the ACD value is associated, with bit 0 pertaining to lead 1 and bit 11 with lead 12. The setting high of the write bit for each lead indicates that the specific lead is ready to write a value to memory.

4.1 Wave Container Module

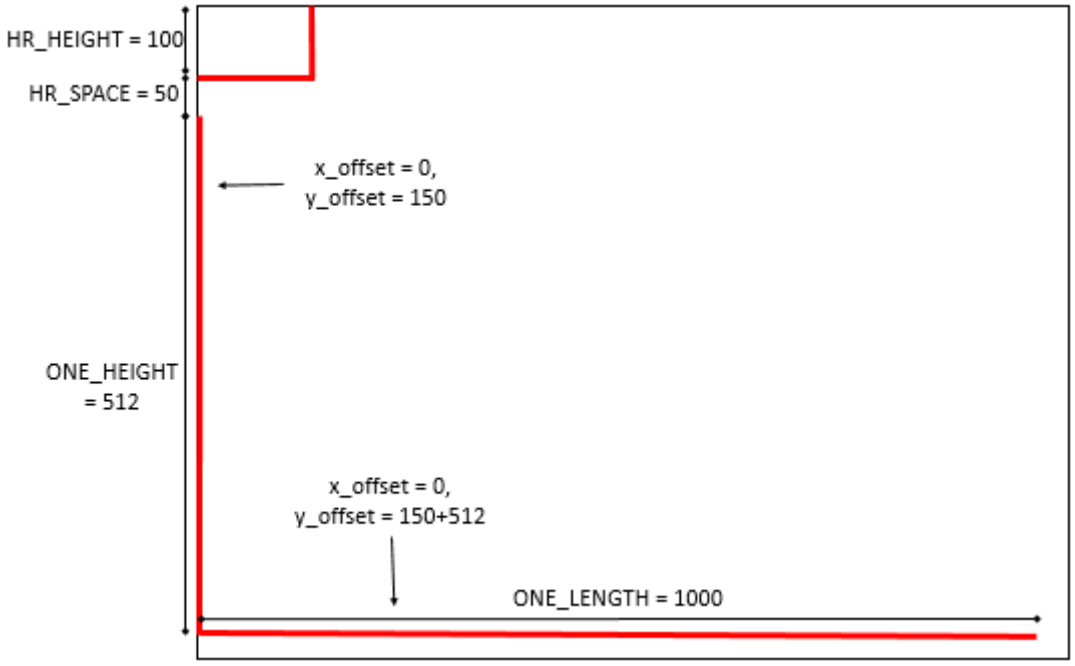
The Wave Container module generates pixel data to display the x and y axes of each wave container on the screen. The number of switches flipped on indicates the number of wave containers to be displayed. This number is represented by the number_of_graphs variable in the verilog code. Depending on the number of graphs to be displayed, the Wave Container module will be in either One Axis, Two Axes, Four Axes, or Twelve Axes mode.

Each axis of each wave container has a length, height, x-offset, and y-offset. The length of each y axis is 2 pixels wide and each height of each x axis of 2 pixels wide. The length of the x axis and height of the y axis are determinant on which mode the Wave Container module is in. The x-offset of each axis is the starting hcount of the axis. Likewise, the y-offset of each axis is the starting vcount of the axis. The first 150 values of vcount are occupied by the Heart Rate display module and thus every axis is shifted down 150 pixels. The x-offset and the y-offset of the y-axis are used to position each wave in its appropriate container as described in Section 4.2.

4.1.1 One Axis

One Axis mode is initiated when one of the switches on the Nexys is turned on. The axis values of the selected lead are set when the corresponding switch is turned on. The x-offset of both values are set to 0 so to align with the left of the display. The y-

offset of the y-axis is set to 150 and the y-offset of the x-axis is set to 150+512 to align it with the bottom of the y-axis. The height of the y-axis is set to 512 because 512 is a factor of 4096, making scaling of the values from memory easy. The length of the x-axis is 1000 so that all 1000 values in memory can be displayed within the container. All values of the other 11 axes not selected are set to 0. The diagram below gives a visual representation of One Axis mode and all of its components:

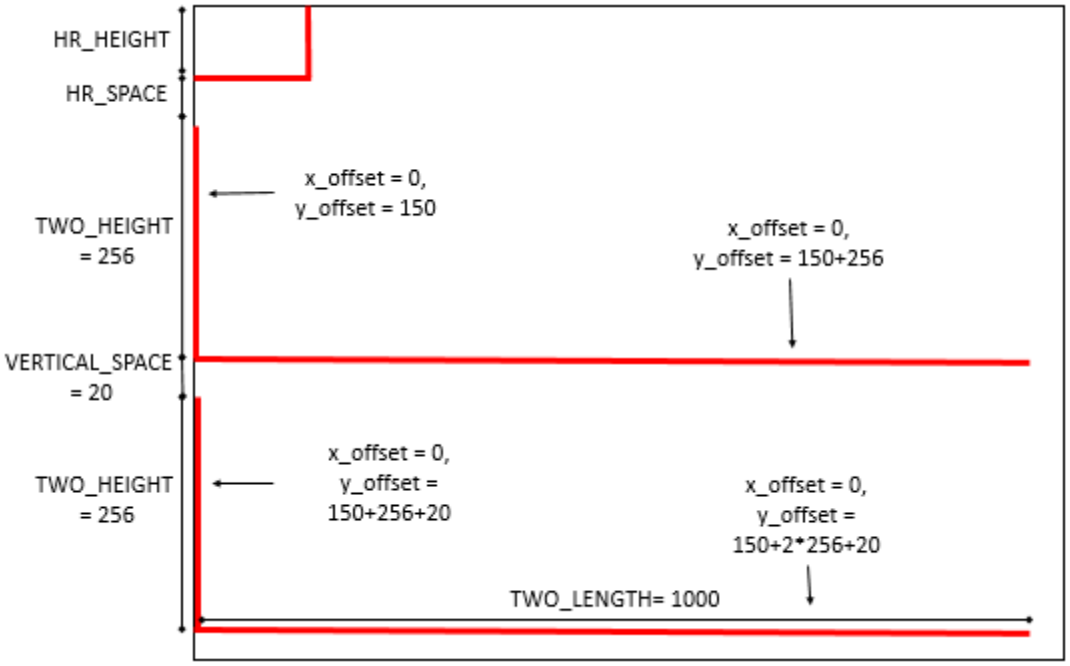


One Axis Mode with Important Values

4.1.2 Two Axes

Two Axes mode is initiated when two of the switches on the Nexys are turned on. In order to set which axis goes in which spot on the screen, the Two Axes mode checks the number of containers that have be filled so far. If no container has been set to display, then the selected container's values are set to reside in the first spot. If one container has been set to display, then the selected container's values are set to reside in the second spot, etc. The algorithm starts with the first lead and finishes with the last lead. This ensures the selected leads will display in the order they are laid out on the switches.

The x-offset of both containers are set to 0 so to align with the left of the display. The y-offset of the first y-axis is set to 150 and the y-offset of the x-axis is set to 150+256 to align it with the bottom of the y-axis. The y-offset of the second y-axis is set to 150+256+20 to be below the first container with additional space and the y-offset of the second x-axis is set to 150+2*256+20 to align it with the bottom of the second y-axis. The height of the y-axis is set to 256 because it is a factor of 4096. Like in the One Axis mode, the length of the x-axis is 1000 so that all 1000 values in memory can be displayed within the container. All values of the other 10 axes not selected are set to 0. The diagram below gives a visual representation of Two Axes mode and all of its components:



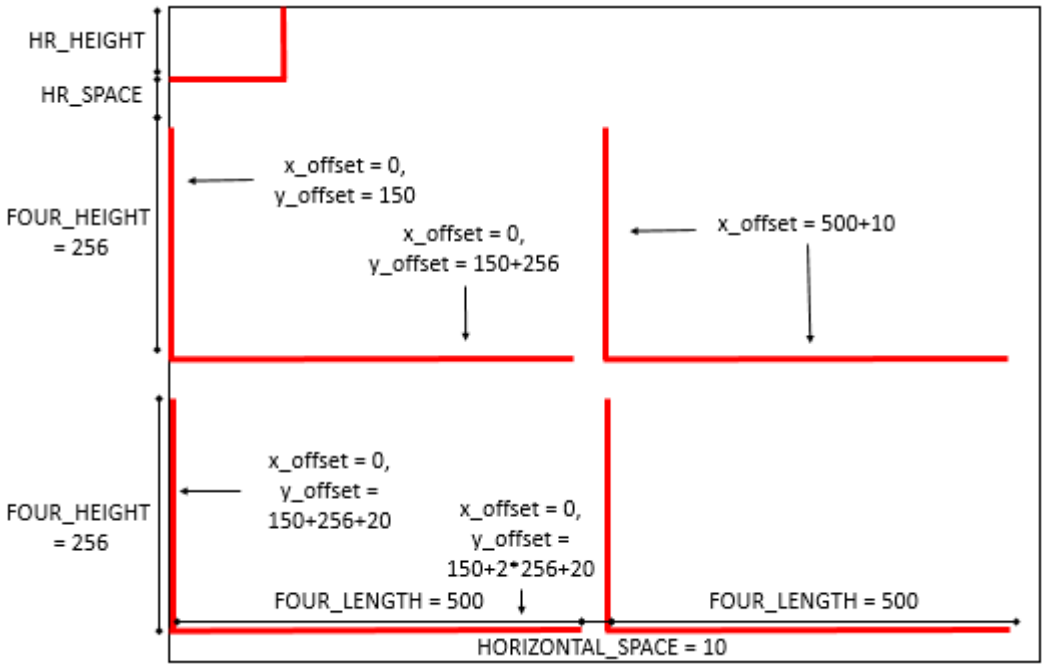
Two Axes Mode with Important Values

4.1.3 Four Axes

Four Axes mode is initiated when three or four of the switches on the Nexys are turned on. In order to set which axis goes in which spot on the screen, the Four Axes mode checks the number of containers that have be filled so far. If no container has been set to display, then the selected container's values are set to reside in the first spot. If one container has been set to display, then the selected container's values are set to reside

in the second spot, etc. If only three switches are selected, the fourth container will not display. Like the Two Axes mode, the algorithm starts with the first lead and finishes with the last lead.

The x-offset of containers one and three are set to 0 so to align with the left of the display. The x-offset of containers two and four are set to 500+10 so to be to the right of the first and third containers and allow for space between containers. The y-offset of the first and second y-axes are set to 150 and the y-offset of the x-axes are set to 150+256 to align it with the bottom of the y-axis. The y-offset of the third and fourth y-axes are set to 150+256+20 to be below the first two containers with additional space and the y-offset of the last two x-axes are set to 150+2*256+20 to align it with the bottom of the second y-axis. Like the Two Axes mode, the height of the y-axis is set to 256 because it is a factor of 4096. The length of the x-axis is 500 so that every other value in memory can be displayed, which allows for easy scaling of the wave. All values of the other axes not selected are set to 0. The diagram below gives a visual representation of Four Axes mode and all of its components:

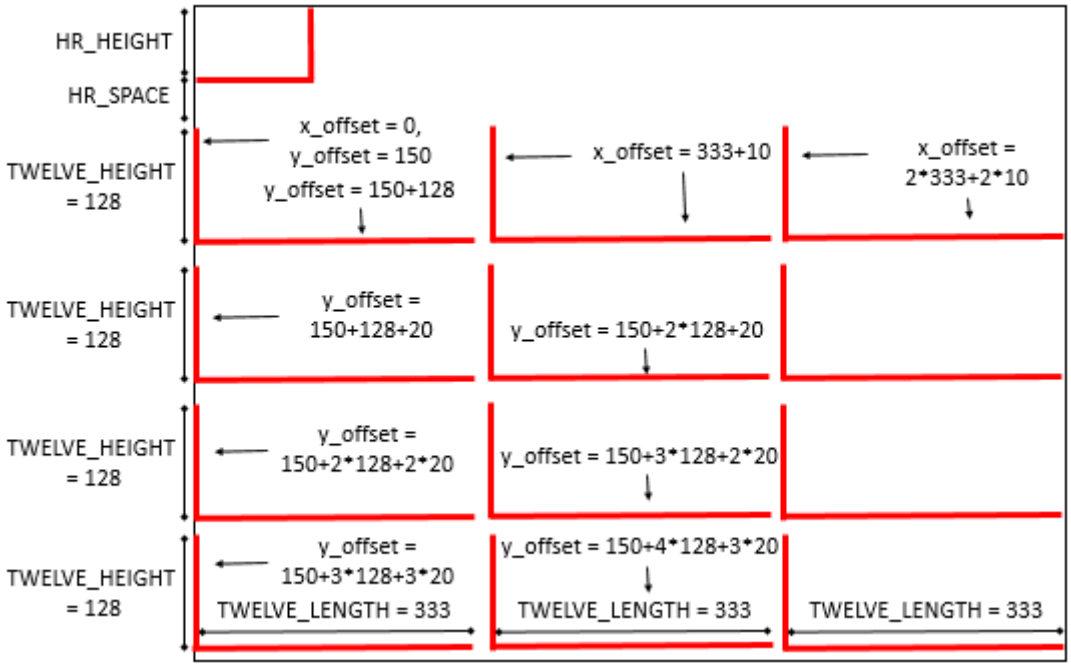


Four Axes Mode with Important Values

4.1.4 Twelve Axes

Twelve Axes mode is initiated when five or more of the switches on the Nexys are turned on. No matter the number of leads selected in this mode, all twelve containers are displayed. If a switch is not turned on, the container associated with it is empty.

The x-offset of containers one and four, seven, and ten are set to 0 so to align with the left of the display. The x-offset of containers two, five, eight, and eleven are set to 333+10 so to be to the right of the previous containers and allow for space between containers. Similarly, the x-offset of containers three, six, nine, and twelve are set to 2*333+2*10. The y-offset of the first three y-axes are set to 150 and the y-offset of the x-axes are set to 150+128 to align it with the bottom of the y-axis. The y-offset of the second three y-axes are set to 150+128+20 and the y-offset of the x-axes are set to 150+2*128+20 to align it with the bottom of the y-axis. The y-offset of the third three y-axes are set to 150+2*256+2*20 and the y-offset of the x-axes are set to 150+3*256+2*20 to align it with the bottom of the y-axis. The y-offset of the last three y-axes are set to 150+3*256+3*20 and the y-offset of the x-axes are set to 150+4*256+3*20 to align it with the bottom of the y-axis. The height of the y-axis is set to 128 because it is a factor of 4096. The length of the x-axis is 333 so that every third value in memory can be displayed. The diagram below gives a visual representation of Four Axes mode and all of its components.



Twelve Axes Mode with Important Values

4.2 Wave Module

The Wave Module contains the EKG Memory and the Wave Display that takes in values from the Recorder, writes the values to memory, and processes the data to be displayed on the monitor.

4.2.1 EKG Memory

Each of the 12 leads has its own BRAM that allows for simultaneous reading and writing. Reading and writing from the BRAM is driven by a 65MHz clock, to ensure writes and read are in sync and to allow for the BRAM to be read at the speed needed to display pixels via XVGA (see Section 4.2.2). Each BRAM is 1000 addresses long with each address containing a 12-bit number. This length and height allows for 3 seconds of wave data to be stored for each lead.

Writing to a lead's memory is initiated by the Recorder setting the write bit of the appropriate lead high. Every 65MHz clock cycle, the write bit is checked to see if Every 4kHz, one of the 12 leads' write bits is set high while the other 11 write bits are low. This means each lead writes at 333Hz. Every time a write bit is set, each lead

memory receives the same ACD value to write to memory. Since every 4kHz the write bit and the write value are updated, each lead writes only its associated value.

In addition to the write bit, write value and 65MHz clock, the BRAM takes in an address indicating a location in memory spanning from 0 to 999. When the address reaches 999, it gets set back to 0 on the next 65MHz clock cycle. Therefore, the wave data writes circularly, resulting in the oldest value in memory always being replaced by the newest value.

4.2.2 Wave Display

The Wave Display module reads values from the EKG Memory and generates a pixel value to send to the display. Each wave has its instance of the Wave Display module and generates its own pixel value. The module is driven by a 65MHz clock in order to generate pixels at the appropriate timing to display via XVGA. The XVGA module provided from Lab 3 generates the inputs of hcount and vcount, which are used to read from memory and generate pixels.

4.2.2.1 Reading Memory

The module starts by reading a value from memory at a selected read address. This read address is dependent on the current hcount and the length of the x-axis the wave corresponds to. If the length of the x-axis is 1000 (as in the One and Two Axes modes), the read address is simply hcount. If the length of the x-axis is 500 (as in the Four Axes mode) and hcount is between the x_offset and x_offset + x_length (within the horizontal bounds of the associated container), the read address is (hcount - x_offset)*2. This shifts the read address to start at the first value of the memory and increment by two every clock cycle. If the length of the x-axis is 333 (as in the Twelve Axes mode) and hcount is between the x_offset and x_offset + x_length, the read address is (hcount - x_offset)*3. This shifts the read address to start at the first value of the memory and increment by three every clock cycle. If hcount is out of bounds of the container, the read address is set to 1000 so that it does not read from memory.

4.2.2.2 Generating Pixel Data

The value read from memory is the value from the ADC reading from 0 to 4096. The value represents the y value in which the pixel should be displayed within the confines of the wave's axes. In order to scale the values down to fit in the container, the value is divided by 4096/y_height. In the case of the y_height equal to 512, the value is bit shifted right by 3. If y_height is 256, the value is bit shifted right by 4. If y_height is 128, the value is bit shifted right by 5. This value is the display value.

In order to know when to display a pixel, the vcount is compared to the display value. In order to shift the vcount to the same range as display value, vcount is subtracted by the y-offset of the y-axis. Additionally, vcount counts up as it goes down the display. In order for the highest display value to be at the lowest vcount value, display value is subtracted from y_height to flip the display value. If vcount minus the y-offset equals y_height minus display value, the pixel is set to display. Otherwise, the pixel is set to 0.

4.2.2.3 Pixel Connection

Using the algorithm in Section 4.2.2.2 results in a smooth wave only if subsequent displayed pixel values are continuous. If the values are not continuous, there are jumps in the displayed wave and neighboring pixels are not connected. In order to fix this, neighboring pixels must be connected. To connect pixels, the previous display value is compared to the current display value. A line of pixels on the current hcount are displayed by setting pixels high with vcount between the two values. However, a line between the most recently written value and the oldest value should not be displayed. In order to remove the line, the hcount value is compared to the current write address. The write address points to the oldest value in memory. However, reading from memory has a delay of 3 cycles. Therefore, when the read address increases by 1 each cycle, when read address minus 3 equals the write address, set the pixel to 0. When the read address increases by 2 each cycle, when read address minus 6 or minus 7 equals the write address, set the pixel to 0. When the read address increases by 3 each cycle, when read address minus 9 to minus 11 equals the write address, set the pixel to 0.

4.3 Heart Rate Module

The Heart Rate module calculates the patient's current heartbeat within 5 beats per minutes. Data from Lead 1 is used to calculate the heartbeat due to its significant peak in voltage every beat. Every 12 seconds, a new heart rate is calculated.

4.3.1 Heart Rate Calculator

The Heart Rate module starts in a low voltage state. Every heartbeat, there is a spike in voltage and thus a spike in the value written to memory. The Heart Rate model looks at the newest value to memory and checks if it is above a high voltage threshold. If it is above a high voltage threshold, the module enters the high voltage state. If the next two values are also above a high boundary that is lower than the high threshold, a heartbeat is recorded, Once the write value goes below the high boundary, the state machine goes back to the low state and waits for the value to go above the high voltage value. After 12 seconds, the heart rate is calculated by taking the number of heartbeats recorded and multiplying it by 5 to get the number of beats in one minute. In order to mitigate errors due to random spikes in voltage that are not heart beats, the state machine ensures a heartbeat is not counted until there has been three consecutive values of high voltage. Additionally, to correct for errors voltages dropping during a heartbeat unexpectedly, the high boundary is set below the initial high threshold.

4.3.2 Heart Rate Display

The Heart Rate Calculator sends Heart Rate data to the Heart Rate Display. The Heart Rate Display takes the heart rate value and divides it by 100 to get the hundreds digit. The remainder is then divided by ten to get the tens digit and the ones digit in the remainder (this division is run by the divider module provided in class). These three digits are then displayed within the heart rate container in the top left corner of the display. Depending on the digit and its value, the display sets the pixel on if the hcount and vcount are within the correct ranges or values to display the digit (ranges of hcount and vcount for each digit and value are in the Appendix in the Heart Rate Module).

Every 12 seconds when the heart rate changes, the display updates with the new heart rate value.

5 Review and Conclusion

Our 12-Lead EKG system achieved its stretch goals and, we believe, could be of use to medical professionals. We were able to use an analog multiplexer to read 12 leads simultaneously, display their waveforms in real time and dynamically adjust screen positioning and layout to suit the user's needs. Applied to real-world systems, we believe these features could prove beneficial to those in the medical field.

Appendix

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
//
// Create Date: 10/1/2015 V1.0
// Design Name:
// Module Name: labkit
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module labkit(
  input CLK100MHZ,
  input[15:0] SW,
  input BTNC, BTNU, BTNL, BTNR, BTND,
  output[3:0] VGA_R,
  output[3:0] VGA_B,
  output[3:0] VGA_G,
  output[7:0] JA,
  output[7:0] JB,
  output VGA_HS,
  output VGA_VS,
  input AD3N,
  input AD3P,
  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
  );


//clocks
          wire clock_25mhz;
          wire clock_65mhz;
          wire pulse_4khz;
```

```
        clock_quarter_divider clockgen(.clk100_mhz(CLK100MHZ),
.clock_25mhz(clock_25mhz));
        clk_wiz_0 clock_65(.clk_in1(CLK100MHZ), .clk_out1(clock_65mhz));
        clock_4khz_pulse clk_4khz(.clock_65mhz(clock_65mhz),.pulse_4khz(pulse_4khz));
//debug visuals
        wire [31:0] data;
        wire [6:0] segments;
        wire [11:0] write_value1;
        wire [7:0] ja_output;
        wire [7:0] heart_rate;
        display_8hex display(.clk(clock_25mhz),.data(data), .seg(segments), .strobe(AN));
        assign SEG[6:0] = segments;
        assign SEG[7] = 1'b1;


        assign data = {36'h0000000000};
        assign LED16_R = BTNL;          // left button -> red led
        assign LED16_G = BTNC;          // center button -> green led
        assign LED16_B = BTNR;          // right button -> blue led
        assign LED17_R = BTNL;
        assign LED17_G = BTNC;
        assign LED17_B = BTNR;



        wire hsync, vsync, blank, phsync, pvsync, pblank;
        wire [9:0] vcount;
        wire [10:0] hcount;
        wire [11:0] pixel;
        reg [11:0] rgb;
        reg b,hs,vs;
        wire [11:0] wave_write, adc_value;
        wire [4:0] selectLow, selectHigh;

        assign JA[7:0] = selectHigh;
        assign JB[7:0] = selectLow;
        recorder recorder(.CLK100MHZ(CLK100MHZ),
.clock_65mhz(clock_65mhz),.pulse_4khz(pulse_4khz),
.selectLow(selectLow),.selectHigh(selectHigh),.AD3N(AD3N), .AD3P(AD3P),
.write(wave_write), .value(adc_value), .sw(SW[11:0]));

        xvga
xvga1(.vclock(clock_65mhz),.hcount(hcount),.vcount(vcount),.hsync(hsync),.vsync(vsync),.blan
k(blank));
        display dp(.clk_100mhz(CLK100MHZ),.vclock(clock_65mhz),.pulse_4khz(pulse_4khz),
        .hcount(hcount),.vcount(vcount),
                .hsync(hsync),.vsync(vsync),.blank(blank),.sw(SW[11:0]),
.write_lead(wave_write),.write_value(adc_value),
        .phsync(phsync),.pvsync(pvsync),.pblank(pblank),.pixel(pixel));

        always @(posedge clock_65mhz) begin
```

```
        hs <= phsync;
        vs <= pvsync;
        b <= pblank;
        rgb <= pixel;
        end

        assign VGA_R = ~blank ? rgb[11:8] : 0;
        assign VGA_G = ~blank ? rgb[7:4] : 0;
        assign VGA_B = ~blank ? rgb[3:0] : 0;
        assign VGA_HS = ~hs;
        assign VGA_VS = ~vs;
endmodule

module display #(parameter HR_HEIGHT = 100, HR_LENGTH = 150, HR_SPACE = 50,
VERTICAL_SPACE = 20, HORIZONTAL_SPACE = 10,
        ONE_HEIGHT = 512, ONE_LENGTH = 1000, TWELVE_HEIGHT = 128,
TWELVE_LENGTH = 333, FOUR_HEIGHT = 256, FOUR_LENGTH = 500, TWO_LENGTH =
1000, TWO_HEIGHT = 256)
  (
  input clk_100mhz,
  input vclock,    // 65MHz clock
  input pulse_4khz,
  input [10:0] hcount,    // horizontal index of current pixel (0..1023)
  input [9:0]     vcount, // vertical index of current pixel (0..767)
  input hsync,              // XVGA horizontal sync signal (active low)
  input vsync,              // XVGA vertical sync signal (active low)
  input blank,    // XVGA blanking (1 means output black pixel)
  input [11:0] sw,
  input [11:0] write_lead,
  input [11:0] write_value,

  output phsync,    // pong game's horizontal sync
  output pvsync,    // pong game's vertical sync
  output pblank,    // pong game's blanking
  output [11:0] pixel
  );



  assign phsync = hsync;
  assign pvsync = vsync;
  assign pblank = blank;
  wire [11:0] hrY_pixel;
  wire [11:0] hrX_pixel;

  reg [10:0] x1_x, y1_x, x2_x, y2_x;
  reg [9:0] x1_y, y1_y;
  reg [10:0] x3_x, x4_x, x5_x, x6_x, x7_x, x8_x, x9_x, x10_x, x11_x, x12_x;
  reg [9:0] x2_y, x3_y, x4_y, x5_y, x6_y, x7_y, x8_y, x9_y, x10_y, x11_y, x12_y;
  reg [10:0] y3_x, y4_x, y5_x , y6_x, y7_x, y8_x, y9_x, y10_x, y11_x, y12_x;
  reg [9:0] y2_y, y3_y, y4_y, y5_y , y6_y, y7_y, y8_y, y9_y, y10_y, y11_y, y12_y;
```

```
  wire [11:0] x1_pixel, x2_pixel, x3_pixel, x4_pixel, x5_pixel, x6_pixel, x7_pixel, x8_pixel,
x9_pixel, x10_pixel, x11_pixel, x12_pixel;
  wire [11:0] y1_pixel, y2_pixel, y3_pixel, y4_pixel, y5_pixel, y6_pixel, y7_pixel, y8_pixel,
y9_pixel, y10_pixel, y11_pixel, y12_pixel;

  reg [10:0] x1_length, x2_length, x3_length, x4_length, x5_length, x6_length, x7_length,
x8_length, x9_length, x10_length, x11_length, x12_length = 0;
  reg [9:0] x1_height, x2_height, x3_height, x4_height, x5_height, x6_height, x7_height,
x8_height, x9_height, x10_height, x11_height, x12_height = 0;
  reg [10:0] y1_length, y2_length, y3_length, y4_length, y5_length, y6_length, y7_length,
y8_length, y9_length, y10_length, y11_length, y12_length = 0;
  reg [9:0] y1_height, y2_height, y3_height, y4_height, y5_height, y6_height, y7_height,
y8_height, y9_height, y10_height, y11_height, y12_height= 0;

  reg [10:0] hcount_start = 0;
  wire [3:0] number_of_graphs;
  reg [3:0] graphs_used = 0;
  assign number_of_graphs = sw[11] + sw[10] + sw[9] + sw[8] + sw[7] + sw[6] + sw[5] + sw[4] +
sw[3] + sw[2] + sw[1] + sw[0];
  always @(negedge vsync) begin
          hcount_start <= hcount_start + 1;
          if (hcount_start >= 1000) hcount_start <=0;
          if (number_of_graphs == 0) begin
          x1_length <= 0;
          x1_height <= 0;
          y1_length <= 0;
          y1_height <= 0;

          x2_length <= 0;
          x2_height <= 0;
          y2_length <= 0;
          y2_height <= 0;

          x3_length <= 0;
          x3_height <= 0;
          y3_length <= 0;
          y3_height <= 0;

          x4_length <= 0;
          x4_height <= 0;
          y4_length <= 0;
          y4_height <= 0;

          x5_length <= 0;
          x5_height <= 0;
          y5_length <= 0;
          y5_height <= 0;

          x6_length <= 0;
          x6_height <= 0;
```

```
y6_length <= 0;
y6_height <= 0;

x7_length <= 0;
x7_height <= 0;
y7_length <= 0;
y7_height <= 0;

x8_length <= 0;
x8_height <= 0;
y8_length <= 0;
y8_height <= 0;

x9_length <= 0;
x9_height <= 0;
y9_length <= 0;
y9_height <= 0;

x10_length <= 0;
x10_height <= 0;
y10_length <= 0;
y10_height <= 0;

x11_length <= 0;
x11_height <= 0;
y11_length <= 0;
y11_height <= 0;

x12_length <= 0;
x12_height <= 0;
y12_length <= 0;
y12_height <= 0;
end
else if (number_of_graphs == 1) begin
x1_length <= ONE_LENGTH;
x1_height <= 2;
y1_length <= 2;
y1_height <= ONE_HEIGHT;
x1_x <= 0;
x1_y <= ONE_HEIGHT + HR_HEIGHT + HR_SPACE;
y1_x <= 0;
y1_y <= HR_HEIGHT + HR_SPACE;

x2_length <= ONE_LENGTH;
x2_height <= 2;
y2_length <= 2;
y2_height <= ONE_HEIGHT;
x2_x <= 0;
x2_y <= ONE_HEIGHT + HR_HEIGHT + HR_SPACE;
y2_x <= 0;
y2_y <= HR_HEIGHT + HR_SPACE;
```

```
x3_length <= ONE_LENGTH;
x3_height <= 2;
y3_length <= 2;
y3_height <= ONE_HEIGHT;
x3_x <= 0;
x3_y <= ONE_HEIGHT + HR_HEIGHT + HR_SPACE;
y3_x <= 0;
y3_y <= HR_HEIGHT + HR_SPACE;

x4_length <= ONE_LENGTH;
x4_height <= 2;
y4_length <= 2;
y4_height <= ONE_HEIGHT;
x4_x <= 0;
x4_y <= ONE_HEIGHT + HR_HEIGHT + HR_SPACE;
y4_x <= 0;
y4_y <= HR_HEIGHT + HR_SPACE;

x5_length <= ONE_LENGTH;
x5_height <= 2;
y5_length <= 2;
y5_height <= ONE_HEIGHT;
x5_x <= 0;
x5_y <= ONE_HEIGHT + HR_HEIGHT + HR_SPACE;
y5_x <= 0;
y5_y <= HR_HEIGHT + HR_SPACE;

x6_length <= ONE_LENGTH;
x6_height <= 2;
y6_length <= 2;
y6_height <= ONE_HEIGHT;
x6_x <= 0;
x6_y <= ONE_HEIGHT + HR_HEIGHT + HR_SPACE;
y6_x <= 0;
y6_y <= HR_HEIGHT + HR_SPACE;

x7_length <= ONE_LENGTH;
x7_height <= 2;
y7_length <= 2;
y7_height <= ONE_HEIGHT;
x7_x <= 0;
x7_y <= ONE_HEIGHT + HR_HEIGHT + HR_SPACE;
y7_x <= 0;
y7_y <= HR_HEIGHT + HR_SPACE;

x8_length <= ONE_LENGTH;
x8_height <= 2;
y8_length <= 2;
y8_height <= ONE_HEIGHT;
x8_x <= 0;
```

```
x8_y <= ONE_HEIGHT + HR_HEIGHT + HR_SPACE;
y8_x <= 0;
y8_y <= HR_HEIGHT + HR_SPACE;


x9_length <= ONE_LENGTH;
x9_height <= 2;
y9_length <= 2;
y9_height <= ONE_HEIGHT;
x9_x <= 0;
x9_y <= ONE_HEIGHT + HR_HEIGHT + HR_SPACE;
y9_x <= 0;
y9_y <= HR_HEIGHT + HR_SPACE;

x10_length <= ONE_LENGTH;
x10_height <= 2;
y10_length <= 2;
y10_height <= ONE_HEIGHT;
x10_x <= 0;
x10_y <= ONE_HEIGHT + HR_HEIGHT + HR_SPACE;
y10_x <= 0;
y10_y <= HR_HEIGHT + HR_SPACE;

x11_length <= ONE_LENGTH;
x11_height <= 2;
y11_length <= 2;
y11_height <= ONE_HEIGHT;
x11_x <= 0;
x11_y <= ONE_HEIGHT + HR_HEIGHT + HR_SPACE;
y11_x <= 0;
y11_y <= HR_HEIGHT + HR_SPACE;

x12_length <= ONE_LENGTH;
x12_height <= 2;
y12_length <= 2;
y12_height <= ONE_HEIGHT;
x12_x <= 0;
x12_y <= ONE_HEIGHT + HR_HEIGHT + HR_SPACE;
y12_x <= 0;
y12_y <= HR_HEIGHT + HR_SPACE;
end
else if (number_of_graphs == 2) begin
if (sw[0]) begin
x1_length <= TWO_LENGTH;
x1_height <= 2;
y1_length <= 2;
y1_height <= TWO_HEIGHT;
x1_x <= 0;
x1_y <= TWO_HEIGHT + HR_HEIGHT + HR_SPACE;
y1_x <= 0;
y1_y <= HR_HEIGHT + HR_SPACE;
```

```
		graphs_used = graphs_used + 1;
		end
		else begin
		x1_length <= 0;
		x1_height <= 0;
		y1_length <= 0;
		y1_height <= 0;
		end
		if (sw[1] && graphs_used < 2) begin
		if (graphs_used == 1) begin
			x2_length <= TWO_LENGTH;
			x2_height <= 2;
			y2_length <= 2;
			y2_height <= TWO_HEIGHT;
			x2_x <= 0;
			x2_y <= TWO_HEIGHT + VERTICAL_SPACE + TWO_HEIGHT + HR_HEIGHT
+ HR_SPACE;
			y2_x <= 0;
			y2_y <= VERTICAL_SPACE + TWO_HEIGHT + HR_HEIGHT + HR_SPACE;
		end
		else if (graphs_used == 0) begin
			x2_length <= TWO_LENGTH;
			x2_height <= 2;
			y2_length <= 2;
			y2_height <= TWO_HEIGHT;
			x2_x <= 0;
			x2_y <= TWO_HEIGHT + HR_HEIGHT + HR_SPACE;
			y2_x <= 0;
			y2_y <= HR_HEIGHT + HR_SPACE;
		end
		graphs_used = graphs_used + 1;
		end
		else begin
		x2_length <= 0;
		x2_height <= 0;
		y2_length <= 0;
		y2_height <= 0;
		end
		if (sw[2] && graphs_used < 2) begin
		if (graphs_used == 1) begin
			x3_length <= TWO_LENGTH;
			x3_height <= 2;
			y3_length <= 2;
			y3_height <= TWO_HEIGHT;
			x3_x <= 0;
			x3_y <= TWO_HEIGHT + VERTICAL_SPACE + TWO_HEIGHT + HR_HEIGHT
+ HR_SPACE;
			y3_x <= 0;
			y3_y <= VERTICAL_SPACE + TWO_HEIGHT + HR_HEIGHT + HR_SPACE;
			end
		else if (graphs_used == 0) begin
```

```
                x3_length <= TWO_LENGTH;
                x3_height <= 2;
                y3_length <= 2;
                y3_height <= TWO_HEIGHT;
                x3_x <= 0;
                x3_y <= TWO_HEIGHT + HR_HEIGHT + HR_SPACE;
                y3_x <= 0;
                y3_y <= HR_HEIGHT + HR_SPACE;
        end
        graphs_used = graphs_used + 1;
        end
        else begin
        x3_length <= 0;
        x3_height <= 0;
        y3_length <= 0;
        y3_height <= 0;
        end
        if (sw[3] && graphs_used < 2) begin
        if (graphs_used == 1) begin
                x4_length <= TWO_LENGTH;
                x4_height <= 2;
                y4_length <= 2;
                y4_height <= TWO_HEIGHT;
                x4_x <= 0;
                x4_y <= TWO_HEIGHT + VERTICAL_SPACE + TWO_HEIGHT + HR_HEIGHT
+ HR_SPACE;
                y4_x <= 0;
                y4_y <= VERTICAL_SPACE + TWO_HEIGHT + HR_HEIGHT + HR_SPACE;
        end
        else if (graphs_used == 0) begin
                x4_length <= TWO_LENGTH;
                x4_height <= 2;
                y4_length <= 2;
                y4_height <= TWO_HEIGHT;
                x4_x <= 0;
                x4_y <= TWO_HEIGHT + HR_HEIGHT + HR_SPACE;
                y4_x <= 0;
                y4_y <= HR_HEIGHT + HR_SPACE;
        end
        graphs_used = graphs_used + 1;
        end
        else begin
        x4_length <= 0;
        x4_height <= 0;
        y4_length <= 0;
        y4_height <= 0;
        end
        if (sw[4] && graphs_used < 2) begin
        if (graphs_used == 1) begin
                x5_length <= TWO_LENGTH;
                x5_height <= 2;
```

```
                y5_length <= 2;
                y5_height <= TWO_HEIGHT;
                x5_x <= 0;
                x5_y <= TWO_HEIGHT + VERTICAL_SPACE + TWO_HEIGHT + HR_HEIGHT
+ HR_SPACE;
                y5_x <= 0;
                y5_y <= VERTICAL_SPACE + TWO_HEIGHT + HR_HEIGHT + HR_SPACE;
        end
        else if (graphs_used == 0) begin
                x5_length <= TWO_LENGTH;
                x5_height <= 2;
                y5_length <= 2;
                y5_height <= TWO_HEIGHT;
                x5_x <= 0;
                x5_y <= TWO_HEIGHT + HR_HEIGHT + HR_SPACE;
                y5_x <= 0;
                y5_y <= HR_HEIGHT + HR_SPACE;
        end
        graphs_used = graphs_used + 1;
        end
        else begin
        x5_length <= 0;
        x5_height <= 0;
        y5_length <= 0;
        y5_height <= 0;
        end
        if (sw[5] && graphs_used < 2) begin
        if (graphs_used == 1) begin
                x6_length <= TWO_LENGTH;
                x6_height <= 2;
                y6_length <= 2;
                y6_height <= TWO_HEIGHT;
                x6_x <= 0;
                x6_y <= TWO_HEIGHT + VERTICAL_SPACE + TWO_HEIGHT + HR_HEIGHT
+ HR_SPACE;
                y6_x <= 0;
                y6_y <= VERTICAL_SPACE + TWO_HEIGHT + HR_HEIGHT + HR_SPACE;
        end
        else if (graphs_used == 0) begin
                x6_length <= TWO_LENGTH;
                x6_height <= 2;
                y6_length <= 2;
                y6_height <= TWO_HEIGHT;
                x6_x <= 0;
                x6_y <= TWO_HEIGHT + HR_HEIGHT + HR_SPACE;
                y6_x <= 0;
                y6_y <= HR_HEIGHT + HR_SPACE;
        end
        graphs_used = graphs_used + 1;
        end
        else begin
```

```
		x6_length <= 0;
		x6_height <= 0;
		y6_length <= 0;
		y6_height <= 0;
		end
		if (sw[6] && graphs_used < 2) begin
		if (graphs_used == 1) begin
			x7_length <= TWO_LENGTH;
			x7_height <= 2;
			y7_length <= 2;
			y7_height <= TWO_HEIGHT;
			x7_x <= 0;
			x7_y <= TWO_HEIGHT + VERTICAL_SPACE + TWO_HEIGHT + HR_HEIGHT
+ HR_SPACE;
			y7_x <= 0;
			y7_y <= VERTICAL_SPACE + TWO_HEIGHT + HR_HEIGHT + HR_SPACE;
		end
		else if (graphs_used == 0) begin
			x7_length <= TWO_LENGTH;
			x7_height <= 2;
			y7_length <= 2;
			y7_height <= TWO_HEIGHT;
			x7_x <= 0;
			x7_y <= TWO_HEIGHT + HR_HEIGHT + HR_SPACE;
			y7_x <= 0;
			y7_y <= HR_HEIGHT + HR_SPACE;
		end
		graphs_used = graphs_used + 1;
		end
		else begin
		x7_length <= 0;
		x7_height <= 0;
		y7_length <= 0;
		y7_height <= 0;
		end
		if (sw[7] && graphs_used < 2) begin
		if (graphs_used == 1) begin
			x8_length <= TWO_LENGTH;
			x8_height <= 2;
			y8_length <= 2;
			y8_height <= TWO_HEIGHT;
			x8_x <= 0;
			x8_y <= TWO_HEIGHT + VERTICAL_SPACE + TWO_HEIGHT + HR_HEIGHT
+ HR_SPACE;
			y8_x <= 0;
			y8_y <= VERTICAL_SPACE + TWO_HEIGHT + HR_HEIGHT + HR_SPACE;
		end
		else if (graphs_used == 0) begin
			x8_length <= TWO_LENGTH;
			x8_height <= 2;
			y8_length <= 2;
```

```
                y8_height <= TWO_HEIGHT;
                x8_x <= 0;
                x8_y <= TWO_HEIGHT + HR_HEIGHT + HR_SPACE;
                y8_x <= 0;
                y8_y <= HR_HEIGHT + HR_SPACE;
        end
        graphs_used = graphs_used + 1;
        end
        else begin
        x8_length <= 0;
        x8_height <= 0;
        y8_length <= 0;
        y8_height <= 0;
        end
        if (sw[8] && graphs_used < 2) begin
        if (graphs_used == 1) begin
                x9_length <= TWO_LENGTH;
                x9_height <= 2;
                y9_length <= 2;
                y9_height <= TWO_HEIGHT;
                x9_x <= 0;
                x9_y <= TWO_HEIGHT + VERTICAL_SPACE + TWO_HEIGHT + HR_HEIGHT
+ HR_SPACE;
                y9_x <= 0;
                y9_y <= VERTICAL_SPACE + TWO_HEIGHT + HR_HEIGHT + HR_SPACE;
        end
        else if (graphs_used == 0) begin
                x9_length <= TWO_LENGTH;
                x9_height <= 2;
                y9_length <= 2;
                y9_height <= TWO_HEIGHT;
                x9_x <= 0;
                x9_y <= TWO_HEIGHT + HR_HEIGHT + HR_SPACE;
                y9_x <= 0;
                y9_y <= HR_HEIGHT + HR_SPACE;
        end
        graphs_used = graphs_used + 1;
        end
        else begin
        x9_length <= 0;
        x9_height <= 0;
        y9_length <= 0;
        y9_height <= 0;
        end
        if (sw[9] && graphs_used < 2) begin
        if (graphs_used == 1) begin
                x10_length <= TWO_LENGTH;
                x10_height <= 2;
                y10_length <= 2;
                y10_height <= TWO_HEIGHT;
                x10_x <= 0;
```

```
                x10_y <= TWO_HEIGHT + VERTICAL_SPACE + TWO_HEIGHT + HR_HEIGHT
+ HR_SPACE;
                y10_x <= 0;
                y10_y <= VERTICAL_SPACE + TWO_HEIGHT + HR_HEIGHT + HR_SPACE;
        end
        else if (graphs_used == 0) begin
                x10_length <= TWO_LENGTH;
                x10_height <= 2;
                y10_length <= 2;
                y10_height <= TWO_HEIGHT;
                x10_x <= 0;
                x10_y <= TWO_HEIGHT + HR_HEIGHT + HR_SPACE;
                y10_x <= 0;
                y10_y <= HR_HEIGHT + HR_SPACE;
        end
        graphs_used = graphs_used + 1;
        end
        else begin
        x10_length <= 0;
        x10_height <= 0;
        y10_length <= 0;
        y10_height <= 0;
        end
        if (sw[10] && graphs_used < 2) begin
        if (graphs_used == 1) begin
                x11_length <= TWO_LENGTH;
                x11_height <= 2;
                y11_length <= 2;
                y11_height <= TWO_HEIGHT;
                x11_x <= 0;
                x11_y <= TWO_HEIGHT + VERTICAL_SPACE + TWO_HEIGHT + HR_HEIGHT
+ HR_SPACE;
                y11_x <= 0;
                y11_y <= VERTICAL_SPACE + TWO_HEIGHT + HR_HEIGHT + HR_SPACE;
        end
        else if (graphs_used == 0) begin
                x11_length <= TWO_LENGTH;
                x11_height <= 2;
                y11_length <= 2;
                y11_height <= TWO_HEIGHT;
                x11_x <= 0;
                x11_y <= TWO_HEIGHT + HR_HEIGHT + HR_SPACE;
                y11_x <= 0;
                y11_y <= HR_HEIGHT + HR_SPACE;
        end
        graphs_used = graphs_used + 1;
        end
        else begin
        x11_length <= 0;
        x11_height <= 0;
        y11_length <= 0;
```

```
		y11_height <= 0;
		end
		if (sw[11] && graphs_used < 2) begin
		if (graphs_used == 1) begin
			x12_length <= TWO_LENGTH;
			x12_height <= 2;
			y12_length <= 2;
			y12_height <= TWO_HEIGHT;
			x12_x <= 0;
			x12_y <= TWO_HEIGHT + VERTICAL_SPACE + TWO_HEIGHT + HR_HEIGHT
+ HR_SPACE;
			y12_x <= 0;
			y12_y <= VERTICAL_SPACE + TWO_HEIGHT + HR_HEIGHT + HR_SPACE;
		end
		else if (graphs_used == 0) begin
			x12_length <= TWO_LENGTH;
			x12_height <= 2;
			y12_length <= 2;
			y12_height <= TWO_HEIGHT;
			x12_x <= 0;
			x12_y <= TWO_HEIGHT + HR_HEIGHT + HR_SPACE;
			y12_x <= 0;
			y12_y <= HR_HEIGHT + HR_SPACE;
		end
		graphs_used = graphs_used + 1;
		end
		else begin
		x12_length <= 0;
		x12_height <= 0;
		y12_length <= 0;
		y12_height <= 0;
		end
		end
		else if (number_of_graphs <= 4) begin
		if (sw[0]) begin
		x1_length <= FOUR_LENGTH;
		x1_height <= 2;
		y1_length <= 2;
		y1_height <= FOUR_HEIGHT;
		x1_x <= 0;
		x1_y <= FOUR_HEIGHT + HR_HEIGHT + HR_SPACE;
		y1_x <= 0;
		y1_y <= HR_HEIGHT + HR_SPACE;
		graphs_used = graphs_used + 1;
		end
		else begin
		x1_length <= 0;
		x1_height <= 0;
		y1_length <= 0;
		y1_height <= 0;
		end
```

```
        if (sw[1] && graphs_used < 4) begin
        if (graphs_used == 1) begin
                x2_length <= FOUR_LENGTH;
                x2_height <= 2;
                y2_length <= 2;
                y2_height <= FOUR_HEIGHT;
                x2_x <= FOUR_LENGTH + HORIZONTAL_SPACE;
                x2_y <= FOUR_HEIGHT + HR_HEIGHT + HR_SPACE;
                y2_x <= FOUR_LENGTH + HORIZONTAL_SPACE;
                y2_y <= HR_HEIGHT + HR_SPACE;
        end
        else if (graphs_used == 0) begin
                x2_length <= FOUR_LENGTH;
                x2_height <= 2;
                y2_length <= 2;
                y2_height <= FOUR_HEIGHT;
                x2_x <= 0;
                x2_y <= FOUR_HEIGHT + HR_HEIGHT + HR_SPACE;
                y2_x <= 0;
                y2_y <= HR_HEIGHT + HR_SPACE;
        end
        graphs_used = graphs_used + 1;
        end
        else begin
        x2_length <= 0;
        x2_height <= 0;
        y2_length <= 0;
        y2_height <= 0;
        end
        if (sw[2] && graphs_used < 4) begin
        if (graphs_used == 2) begin
                x3_length <= FOUR_LENGTH;
                x3_height <= 2;
                y3_length <= 2;
                y3_height <= FOUR_HEIGHT;
                x3_x <= 0;
                x3_y <= FOUR_HEIGHT+ VERTICAL_SPACE +FOUR_HEIGHT + HR_HEIGHT
+ HR_SPACE;
                y3_x <= 0;
                y3_y <= VERTICAL_SPACE +FOUR_HEIGHT + HR_HEIGHT + HR_SPACE;
        end
        else if (graphs_used == 1) begin
                x3_length <= FOUR_LENGTH;
                x3_height <= 2;
                y3_length <= 2;
                y3_height <= FOUR_HEIGHT;
                x3_x <= FOUR_LENGTH + HORIZONTAL_SPACE;
                x3_y <= FOUR_HEIGHT + HR_HEIGHT + HR_SPACE;
                y3_x <= FOUR_LENGTH + HORIZONTAL_SPACE;
                y3_y <= HR_HEIGHT + HR_SPACE;
        end
```

```
        else if (graphs_used == 0) begin
                x3_length <= FOUR_LENGTH;
                x3_height <= 2;
                y3_length <= 2;
                y3_height <= FOUR_HEIGHT;
                x3_x <= 0;
                x3_y <= FOUR_HEIGHT + HR_HEIGHT + HR_SPACE;
                y3_x <= 0;
                y3_y <= HR_HEIGHT + HR_SPACE;

        end
        graphs_used = graphs_used + 1;
        end
        else begin
        x3_length <= 0;
        x3_height <= 0;
        y3_length <= 0;
        y3_height <= 0;
        end
        if (sw[3] && graphs_used < 4) begin
        if (graphs_used == 3) begin
                x4_length <= FOUR_LENGTH;
                x4_height <= 2;
                y4_length <= 2;
                y4_height <= FOUR_HEIGHT;
                x4_x <= FOUR_LENGTH + HORIZONTAL_SPACE;
                x4_y <= FOUR_HEIGHT+ VERTICAL_SPACE +FOUR_HEIGHT + HR_HEIGHT
+ HR_SPACE;
                y4_x <= FOUR_LENGTH + HORIZONTAL_SPACE;
                y4_y <= VERTICAL_SPACE +FOUR_HEIGHT + HR_HEIGHT + HR_SPACE;
        end
        else if (graphs_used == 2) begin
                x4_length <= FOUR_LENGTH;
                x4_height <= 2;
                y4_length <= 2;
                y4_height <= FOUR_HEIGHT;
                x4_x <= 0;
                x4_y <= FOUR_HEIGHT+ VERTICAL_SPACE +FOUR_HEIGHT + HR_HEIGHT
+ HR_SPACE;
                y4_x <= 0;
                y4_y <= VERTICAL_SPACE +FOUR_HEIGHT + HR_HEIGHT + HR_SPACE;
        end
        else if (graphs_used == 1) begin
                x4_length <= FOUR_LENGTH;
                x4_height <= 2;
                y4_length <= 2;
                y4_height <= FOUR_HEIGHT;
                x4_x <= FOUR_LENGTH + HORIZONTAL_SPACE;
                x4_y <= FOUR_HEIGHT + HR_HEIGHT + HR_SPACE;
                y4_x <= FOUR_LENGTH + HORIZONTAL_SPACE;
                y4_y <= HR_HEIGHT + HR_SPACE;
```

```
        end
        else if (graphs_used == 0) begin
                x4_length <= FOUR_LENGTH;
                x4_height <= 2;
                y4_length <= 2;
                y4_height <= FOUR_HEIGHT;
                x4_x <= 0;
                x4_y <= FOUR_HEIGHT + HR_HEIGHT + HR_SPACE;
                y4_x <= 0;
                y4_y <= HR_HEIGHT + HR_SPACE;
        end
        graphs_used = graphs_used + 1;
        end
        else begin
        x4_length <= 0;
        x4_height <= 0;
        y4_length <= 0;
        y4_height <= 0;
        end
        if (sw[4] && graphs_used < 4) begin
        if (graphs_used == 3) begin
                x5_length <= FOUR_LENGTH;
                x5_height <= 2;
                y5_length <= 2;
                y5_height <= FOUR_HEIGHT;
                x5_x <= FOUR_LENGTH + HORIZONTAL_SPACE;
                x5_y <= FOUR_HEIGHT+ VERTICAL_SPACE +FOUR_HEIGHT + HR_HEIGHT
+ HR_SPACE;
                y5_x <= FOUR_LENGTH + HORIZONTAL_SPACE;
                y5_y <= VERTICAL_SPACE +FOUR_HEIGHT + HR_HEIGHT + HR_SPACE;
        end
        else if (graphs_used == 2) begin
                x5_length <= FOUR_LENGTH;
                x5_height <= 2;
                y5_length <= 2;
                y5_height <= FOUR_HEIGHT;
                x5_x <= 0;
                x5_y <= FOUR_HEIGHT+ VERTICAL_SPACE +FOUR_HEIGHT + HR_HEIGHT
+ HR_SPACE;
                y5_x <= 0;
                y5_y <= VERTICAL_SPACE +FOUR_HEIGHT + HR_HEIGHT + HR_SPACE;
        end
        else if (graphs_used == 1) begin
                x5_length <= FOUR_LENGTH;
                x5_height <= 2;
                y5_length <= 2;
                y5_height <= FOUR_HEIGHT;
                x5_x <= FOUR_LENGTH + HORIZONTAL_SPACE;
                x5_y <= FOUR_HEIGHT + HR_HEIGHT + HR_SPACE;
                y5_x <= FOUR_LENGTH + HORIZONTAL_SPACE;
                y5_y <= HR_HEIGHT + HR_SPACE;
```

```
        end
        else if (graphs_used == 0) begin
                x5_length <= FOUR_LENGTH;
                x5_height <= 2;
                y5_length <= 2;
                y5_height <= FOUR_HEIGHT;
                x5_x <= 0;
                x5_y <= FOUR_HEIGHT + HR_HEIGHT + HR_SPACE;
                y5_x <= 0;
                y5_y <= HR_HEIGHT + HR_SPACE;
        end
        graphs_used = graphs_used + 1;
        end
        else begin
        x5_length <= 0;
        x5_height <= 0;
        y5_length <= 0;
        y5_height <= 0;
        end
        if (sw[5] && graphs_used < 4) begin
        if (graphs_used == 3) begin
                x6_length <= FOUR_LENGTH;
                x6_height <= 2;
                y6_length <= 2;
                y6_height <= FOUR_HEIGHT;
                x6_x <= FOUR_LENGTH + HORIZONTAL_SPACE;
                x6_y <= FOUR_HEIGHT+ VERTICAL_SPACE +FOUR_HEIGHT + HR_HEIGHT
+ HR_SPACE;
                y6_x <= FOUR_LENGTH + HORIZONTAL_SPACE;
                y6_y <= VERTICAL_SPACE +FOUR_HEIGHT + HR_HEIGHT + HR_SPACE;
        end
        else if (graphs_used == 2) begin
                x6_length <= FOUR_LENGTH;
                x6_height <= 2;
                y6_length <= 2;
                y6_height <= FOUR_HEIGHT;
                x6_x <= 0;
                x6_y <= FOUR_HEIGHT+ VERTICAL_SPACE +FOUR_HEIGHT + HR_HEIGHT
+ HR_SPACE;
                y6_x <= 0;
                y6_y <= VERTICAL_SPACE +FOUR_HEIGHT + HR_HEIGHT + HR_SPACE;
        end
        else if (graphs_used == 1) begin
                x6_length <= FOUR_LENGTH;
                x6_height <= 2;
                y6_length <= 2;
                y6_height <= FOUR_HEIGHT;
                x6_x <= FOUR_LENGTH + HORIZONTAL_SPACE;
                x6_y <= FOUR_HEIGHT + HR_HEIGHT + HR_SPACE;
                y6_x <= FOUR_LENGTH + HORIZONTAL_SPACE;
                y6_y <= HR_HEIGHT + HR_SPACE;
```

```
		end
		else if (graphs_used == 0) begin
			x6_length <= FOUR_LENGTH;
			x6_height <= 2;
			y6_length <= 2;
			y6_height <= FOUR_HEIGHT;
			x6_x <= 0;
			x6_y <= FOUR_HEIGHT + HR_HEIGHT + HR_SPACE;
			y6_x <= 0;
			y6_y <= HR_HEIGHT + HR_SPACE;
		end
		graphs_used = graphs_used + 1;
		end
		else begin
		x6_length <= 0;
		x6_height <= 0;
		y6_length <= 0;
		y6_height <= 0;
		end
		if (sw[6] && graphs_used < 4) begin
		if (graphs_used == 3) begin
			x7_length <= FOUR_LENGTH;
			x7_height <= 2;
			y7_length <= 2;
			y7_height <= FOUR_HEIGHT;
			x7_x <= FOUR_LENGTH + HORIZONTAL_SPACE;
			x7_y <= FOUR_HEIGHT+ VERTICAL_SPACE +FOUR_HEIGHT + HR_HEIGHT
+ HR_SPACE;
			y7_x <= FOUR_LENGTH + HORIZONTAL_SPACE;
			y7_y <= VERTICAL_SPACE +FOUR_HEIGHT + HR_HEIGHT + HR_SPACE;
		end
		else if (graphs_used == 2) begin
			x7_length <= FOUR_LENGTH;
			x7_height <= 2;
			y7_length <= 2;
			y7_height <= FOUR_HEIGHT;
			x7_x <= 0;
			x7_y <= FOUR_HEIGHT+ VERTICAL_SPACE +FOUR_HEIGHT + HR_HEIGHT
+ HR_SPACE;
			y7_x <= 0;
			y7_y <= VERTICAL_SPACE +FOUR_HEIGHT + HR_HEIGHT + HR_SPACE;
		end
		else if (graphs_used == 1) begin
			x7_length <= FOUR_LENGTH;
			x7_height <= 2;
			y7_length <= 2;
			y7_height <= FOUR_HEIGHT;
			x7_x <= FOUR_LENGTH + HORIZONTAL_SPACE;
			x7_y <= FOUR_HEIGHT + HR_HEIGHT + HR_SPACE;
			y7_x <= FOUR_LENGTH + HORIZONTAL_SPACE;
			y7_y <= HR_HEIGHT + HR_SPACE;
```

```
        end
        else if (graphs_used == 0) begin
                x7_length <= FOUR_LENGTH;
                x7_height <= 2;
                y7_length <= 2;
                y7_height <= FOUR_HEIGHT;
                x7_x <= 0;
                x7_y <= FOUR_HEIGHT + HR_HEIGHT + HR_SPACE;
                y7_x <= 0;
                y7_y <= HR_HEIGHT + HR_SPACE;
        end
        graphs_used = graphs_used + 1;
        end
        else begin
        x7_length <= 0;
        x7_height <= 0;
        y7_length <= 0;
        y7_height <= 0;
        end
        if (sw[7] && graphs_used < 4) begin
        if (graphs_used == 3) begin
                x8_length <= FOUR_LENGTH;
                x8_height <= 2;
                y8_length <= 2;
                y8_height <= FOUR_HEIGHT;
                x8_x <= FOUR_LENGTH + HORIZONTAL_SPACE;
                x8_y <= FOUR_HEIGHT+ VERTICAL_SPACE +FOUR_HEIGHT + HR_HEIGHT
+ HR_SPACE;
                y8_x <= FOUR_LENGTH + HORIZONTAL_SPACE;
                y8_y <= VERTICAL_SPACE +FOUR_HEIGHT + HR_HEIGHT + HR_SPACE;
        end
        else if (graphs_used == 2) begin
                x8_length <= FOUR_LENGTH;
                x8_height <= 2;
                y8_length <= 2;
                y8_height <= FOUR_HEIGHT;
                x8_x <= 0;
                x8_y <= FOUR_HEIGHT+ VERTICAL_SPACE +FOUR_HEIGHT + HR_HEIGHT
+ HR_SPACE;
                y8_x <= 0;
                y8_y <= VERTICAL_SPACE +FOUR_HEIGHT + HR_HEIGHT + HR_SPACE;
        end
        else if (graphs_used == 1) begin
                x8_length <= FOUR_LENGTH;
                x8_height <= 2;
                y8_length <= 2;
                y8_height <= FOUR_HEIGHT;
                x8_x <= FOUR_LENGTH + HORIZONTAL_SPACE;
                x8_y <= FOUR_HEIGHT + HR_HEIGHT + HR_SPACE;
                y8_x <= FOUR_LENGTH + HORIZONTAL_SPACE;
                y8_y <= HR_HEIGHT + HR_SPACE;
```

```
        end
        else if (graphs_used == 0) begin
                x8_length <= FOUR_LENGTH;
                x8_height <= 2;
                y8_length <= 2;
                y8_height <= FOUR_HEIGHT;
                x8_x <= 0;
                x8_y <= FOUR_HEIGHT + HR_HEIGHT + HR_SPACE;
                y8_x <= 0;
                y8_y <= HR_HEIGHT + HR_SPACE;
        end
        graphs_used = graphs_used + 1;
        end
        else begin
        x8_length <= 0;
        x8_height <= 0;
        y8_length <= 0;
        y8_height <= 0;
        end
        if (sw[8] && graphs_used < 4) begin
        if (graphs_used == 3) begin
                x9_length <= FOUR_LENGTH;
                x9_height <= 2;
                y9_length <= 2;
                y9_height <= FOUR_HEIGHT;
                x9_x <= FOUR_LENGTH + HORIZONTAL_SPACE;
                x9_y <= FOUR_HEIGHT+ VERTICAL_SPACE +FOUR_HEIGHT + HR_HEIGHT
+ HR_SPACE;
                y9_x <= FOUR_LENGTH + HORIZONTAL_SPACE;
                y9_y <= VERTICAL_SPACE +FOUR_HEIGHT + HR_HEIGHT + HR_SPACE;
        end
        else if (graphs_used == 2) begin
                x9_length <= FOUR_LENGTH;
                x9_height <= 2;
                y9_length <= 2;
                y9_height <= FOUR_HEIGHT;
                x9_x <= 0;
                x9_y <= FOUR_HEIGHT+ VERTICAL_SPACE +FOUR_HEIGHT + HR_HEIGHT
+ HR_SPACE;
                y9_x <= 0;
                y9_y <= VERTICAL_SPACE +FOUR_HEIGHT + HR_HEIGHT + HR_SPACE;
        end
        else if (graphs_used == 1) begin
                x9_length <= FOUR_LENGTH;
                x9_height <= 2;
                y9_length <= 2;
                y9_height <= FOUR_HEIGHT;
                x9_x <= FOUR_LENGTH + HORIZONTAL_SPACE;
                x9_y <= FOUR_HEIGHT + HR_HEIGHT + HR_SPACE;
                y9_x <= FOUR_LENGTH + HORIZONTAL_SPACE;
                y9_y <= HR_HEIGHT + HR_SPACE;
```

```
        end
        else if (graphs_used == 0) begin
                x9_length <= FOUR_LENGTH;
                x9_height <= 2;
                y9_length <= 2;
                y9_height <= FOUR_HEIGHT;
                x9_x <= 0;
                x9_y <= FOUR_HEIGHT + HR_HEIGHT + HR_SPACE;
                y9_x <= 0;
                y9_y <= HR_HEIGHT + HR_SPACE;
        end
        graphs_used = graphs_used + 1;
        end
        else begin
        x9_length <= 0;
        x9_height <= 0;
        y9_length <= 0;
        y9_height <= 0;
        end
        if (sw[9] && graphs_used < 4) begin
        if (graphs_used == 3) begin
                x10_length <= FOUR_LENGTH;
                x10_height <= 2;
                y10_length <= 2;
                y10_height <= FOUR_HEIGHT;
                x10_x <= FOUR_LENGTH + HORIZONTAL_SPACE;
                x10_y <= FOUR_HEIGHT+ VERTICAL_SPACE +FOUR_HEIGHT +
HR_HEIGHT + HR_SPACE;
                y10_x <= FOUR_LENGTH + HORIZONTAL_SPACE;
                y10_y <= VERTICAL_SPACE +FOUR_HEIGHT + HR_HEIGHT + HR_SPACE;
        end
        else if (graphs_used == 2) begin
                x10_length <= FOUR_LENGTH;
                x10_height <= 2;
                y10_length <= 2;
                y10_height <= FOUR_HEIGHT;
                x10_x <= 0;
                x10_y <= FOUR_HEIGHT+ VERTICAL_SPACE +FOUR_HEIGHT +
HR_HEIGHT + HR_SPACE;
                y10_x <= 0;
                y10_y <= VERTICAL_SPACE +FOUR_HEIGHT + HR_HEIGHT + HR_SPACE;
        end
        else if (graphs_used == 1) begin
                x10_length <= FOUR_LENGTH;
                x10_height <= 2;
                y10_length <= 2;
                y10_height <= FOUR_HEIGHT;
                x10_x <= FOUR_LENGTH + HORIZONTAL_SPACE;
                x10_y <= FOUR_HEIGHT + HR_HEIGHT + HR_SPACE;
                y10_x <= FOUR_LENGTH + HORIZONTAL_SPACE;
                y10_y <= HR_HEIGHT + HR_SPACE;
```

```
        end
        else if (graphs_used == 0) begin
                x10_length <= FOUR_LENGTH;
                x10_height <= 2;
                y10_length <= 2;
                y10_height <= FOUR_HEIGHT;
                x10_x <= 0;
                x10_y <= FOUR_HEIGHT + HR_HEIGHT + HR_SPACE;
                y10_x <= 0;
                y10_y <= HR_HEIGHT + HR_SPACE;
        end
        graphs_used = graphs_used + 1;
        end
        else begin
        x10_length <= 0;
        x10_height <= 0;
        y10_length <= 0;
        y10_height <= 0;
        end
        if (sw[10] && graphs_used < 4) begin
        if (graphs_used == 3) begin
                x11_length <= FOUR_LENGTH;
                x11_height <= 2;
                y11_length <= 2;
                y11_height <= FOUR_HEIGHT;
                x11_x <= FOUR_LENGTH + HORIZONTAL_SPACE;
                x11_y <= FOUR_HEIGHT+ VERTICAL_SPACE +FOUR_HEIGHT +
HR_HEIGHT + HR_SPACE;
                y11_x <= FOUR_LENGTH + HORIZONTAL_SPACE;
                y11_y <= VERTICAL_SPACE +FOUR_HEIGHT + HR_HEIGHT + HR_SPACE;
        end
        else if (graphs_used == 2) begin
                x11_length <= FOUR_LENGTH;
                x11_height <= 2;
                y11_length <= 2;
                y11_height <= FOUR_HEIGHT;
                x11_x <= 0;
                x11_y <= FOUR_HEIGHT+ VERTICAL_SPACE +FOUR_HEIGHT +
HR_HEIGHT + HR_SPACE;
                y11_x <= 0;
                y11_y <= VERTICAL_SPACE +FOUR_HEIGHT + HR_HEIGHT + HR_SPACE;
        end
        else if (graphs_used == 1) begin
                x11_length <= FOUR_LENGTH;
                x11_height <= 2;
                y11_length <= 2;
                y11_height <= FOUR_HEIGHT;
                x11_x <= FOUR_LENGTH + HORIZONTAL_SPACE;
                x11_y <= FOUR_HEIGHT + HR_HEIGHT + HR_SPACE;
                y11_x <= FOUR_LENGTH + HORIZONTAL_SPACE;
                y11_y <= HR_HEIGHT + HR_SPACE;
```

```
        end
        else if (graphs_used == 0) begin
                x11_length <= FOUR_LENGTH;
                x11_height <= 2;
                y11_length <= 2;
                y11_height <= FOUR_HEIGHT;
                x11_x <= 0;
                x11_y <= FOUR_HEIGHT + HR_HEIGHT + HR_SPACE;
                y11_x <= 0;
                y11_y <= HR_HEIGHT + HR_SPACE;
        end
        graphs_used = graphs_used + 1;
        end
        else begin
        x11_length <= 0;
        x11_height <= 0;
        y11_length <= 0;
        y11_height <= 0;
        end
        if (sw[11] && graphs_used < 4) begin
        if (graphs_used == 3) begin
                x12_length <= FOUR_LENGTH;
                x12_height <= 2;
                y12_length <= 2;
                y12_height <= FOUR_HEIGHT;
                x12_x <= FOUR_LENGTH + HORIZONTAL_SPACE;
                x12_y <= FOUR_HEIGHT+ VERTICAL_SPACE +FOUR_HEIGHT +
HR_HEIGHT + HR_SPACE;
                y12_x <= FOUR_LENGTH + HORIZONTAL_SPACE;
                y12_y <= VERTICAL_SPACE +FOUR_HEIGHT + HR_HEIGHT + HR_SPACE;
        end
        else if (graphs_used == 2) begin
                x12_length <= FOUR_LENGTH;
                x12_height <= 2;
                y12_length <= 2;
                y12_height <= FOUR_HEIGHT;
                x12_x <= 0;
                x12_y <= FOUR_HEIGHT+ VERTICAL_SPACE +FOUR_HEIGHT +
HR_HEIGHT + HR_SPACE;
                y12_x <= 0;
                y12_y <= VERTICAL_SPACE +FOUR_HEIGHT + HR_HEIGHT + HR_SPACE;
        end
        else if (graphs_used == 1) begin
                x12_length <= FOUR_LENGTH;
                x12_height <= 2;
                y12_length <= 2;
                y12_height <= FOUR_HEIGHT;
                x12_x <= FOUR_LENGTH + HORIZONTAL_SPACE;
                x12_y <= FOUR_HEIGHT + HR_HEIGHT + HR_SPACE;
                y12_x <= FOUR_LENGTH + HORIZONTAL_SPACE;
                y12_y <= HR_HEIGHT + HR_SPACE;
```

```
end
else if (graphs_used == 0) begin
        x12_length <= FOUR_LENGTH;
        x12_height <= 2;
        y12_length <= 2;
        y12_height <= FOUR_HEIGHT;
        x12_x <= 0;
        x12_y <= FOUR_HEIGHT + HR_HEIGHT + HR_SPACE;
        y12_x <= 0;
        y12_y <= HR_HEIGHT + HR_SPACE;
end
graphs_used = graphs_used + 1;
end
else begin
x12_length <= 0;
x12_height <= 0;
y12_length <= 0;
y12_height <= 0;
end
end
else begin
x1_length <= TWELVE_LENGTH;
x1_height <= 2;
y1_length <= 2;
y1_height <= TWELVE_HEIGHT;
x1_x <= 0;
x1_y <= TWELVE_HEIGHT + HR_HEIGHT + HR_SPACE;
y1_x <= 0;
y1_y <= HR_HEIGHT + HR_SPACE;

x2_length <= TWELVE_LENGTH;
x2_height <= 2;
y2_length <= 2;
y2_height <= TWELVE_HEIGHT;
x2_x <= TWELVE_LENGTH + HORIZONTAL_SPACE;
x2_y <= TWELVE_HEIGHT + HR_HEIGHT + HR_SPACE;
y2_x <= TWELVE_LENGTH + HORIZONTAL_SPACE;
y2_y <= HR_HEIGHT + HR_SPACE;

x3_length <= TWELVE_LENGTH;
x3_height <= 2;
y3_length <= 2;
y3_height <= TWELVE_HEIGHT;
x3_x <= 2*TWELVE_LENGTH + 2*HORIZONTAL_SPACE;
x3_y <= TWELVE_HEIGHT + HR_HEIGHT + HR_SPACE;
y3_x <= 2*TWELVE_LENGTH + 2*HORIZONTAL_SPACE;
y3_y <= HR_HEIGHT + HR_SPACE;

x4_length <= TWELVE_LENGTH;
x4_height <= 2;
y4_length <= 2;
```

```
        y4_height <= TWELVE_HEIGHT;
        x4_x <= 0;
        x4_y <= TWELVE_HEIGHT + VERTICAL_SPACE + TWELVE_HEIGHT + HR_HEIGHT
+ HR_SPACE;
        y4_x <= 0;
        y4_y <= VERTICAL_SPACE + TWELVE_HEIGHT + HR_HEIGHT + HR_SPACE;

        x5_length <= TWELVE_LENGTH;
        x5_height <= 2;
        y5_length <= 2;
        y5_height <= TWELVE_HEIGHT;
        x5_x <= TWELVE_LENGTH + HORIZONTAL_SPACE;
        x5_y <= TWELVE_HEIGHT + VERTICAL_SPACE + TWELVE_HEIGHT + HR_HEIGHT
+ HR_SPACE;
        y5_x <= TWELVE_LENGTH + HORIZONTAL_SPACE;
        y5_y <= VERTICAL_SPACE + TWELVE_HEIGHT + HR_HEIGHT + HR_SPACE;

        x6_length <= TWELVE_LENGTH;
        x6_height <= 2;
        y6_length <= 2;
        y6_height <= TWELVE_HEIGHT;
        x6_x <= 2*TWELVE_LENGTH + 2*HORIZONTAL_SPACE;
        x6_y <= TWELVE_HEIGHT + VERTICAL_SPACE + TWELVE_HEIGHT + HR_HEIGHT
+ HR_SPACE;
        y6_x <= 2*TWELVE_LENGTH + 2*HORIZONTAL_SPACE;
        y6_y <= VERTICAL_SPACE + TWELVE_HEIGHT + HR_HEIGHT + HR_SPACE;

        x7_length <= TWELVE_LENGTH;
        x7_height <= 2;
        y7_length <= 2;
        y7_height <= TWELVE_HEIGHT;
        x7_x <= 0;
        x7_y <= TWELVE_HEIGHT + VERTICAL_SPACE + TWELVE_HEIGHT +
VERTICAL_SPACE + TWELVE_HEIGHT + HR_HEIGHT + HR_SPACE;
        y7_x <= 0;
        y7_y <= VERTICAL_SPACE + TWELVE_HEIGHT + VERTICAL_SPACE +
TWELVE_HEIGHT + HR_HEIGHT + HR_SPACE;

        x8_length <= TWELVE_LENGTH;
        x8_height <= 2;
        y8_length <= 2;
        y8_height <= TWELVE_HEIGHT;
        x8_x <= TWELVE_LENGTH + HORIZONTAL_SPACE;
        x8_y <= TWELVE_HEIGHT + VERTICAL_SPACE + TWELVE_HEIGHT +
VERTICAL_SPACE + TWELVE_HEIGHT + HR_HEIGHT + HR_SPACE;
        y8_x <= TWELVE_LENGTH + HORIZONTAL_SPACE;
        y8_y <= VERTICAL_SPACE + TWELVE_HEIGHT + VERTICAL_SPACE +
TWELVE_HEIGHT + HR_HEIGHT + HR_SPACE;

        x9_length <= TWELVE_LENGTH;
        x9_height <= 2;
```

```
        y9_length <= 2;
        y9_height <= TWELVE_HEIGHT;
        x9_x = 2*TWELVE_LENGTH + 2*HORIZONTAL_SPACE;
        x9_y = TWELVE_HEIGHT + VERTICAL_SPACE + TWELVE_HEIGHT +
VERTICAL_SPACE + TWELVE_HEIGHT + HR_HEIGHT + HR_SPACE;
        y9_x = 2*TWELVE_LENGTH + 2*HORIZONTAL_SPACE;
        y9_y = VERTICAL_SPACE + TWELVE_HEIGHT + VERTICAL_SPACE +
TWELVE_HEIGHT + HR_HEIGHT + HR_SPACE;

        x10_length <= TWELVE_LENGTH;
        x10_height <= 2;
        y10_length <= 2;
        y10_height <= TWELVE_HEIGHT;
        x10_x <= 0;
        x10_y <= TWELVE_HEIGHT + VERTICAL_SPACE + TWELVE_HEIGHT +
VERTICAL_SPACE + TWELVE_HEIGHT + VERTICAL_SPACE + TWELVE_HEIGHT +
HR_HEIGHT + HR_SPACE;
        y10_x <= 0;
        y10_y <= VERTICAL_SPACE + TWELVE_HEIGHT + VERTICAL_SPACE +
TWELVE_HEIGHT + VERTICAL_SPACE + TWELVE_HEIGHT + HR_HEIGHT + HR_SPACE;

        x11_length <= TWELVE_LENGTH;
        x11_height <= 2;
        y11_length <= 2;
        y11_height <= TWELVE_HEIGHT;
        x11_x <= TWELVE_LENGTH + HORIZONTAL_SPACE;
        x11_y <= TWELVE_HEIGHT + VERTICAL_SPACE + TWELVE_HEIGHT +
VERTICAL_SPACE + TWELVE_HEIGHT + VERTICAL_SPACE + TWELVE_HEIGHT +
HR_HEIGHT + HR_SPACE;
        y11_x <= TWELVE_LENGTH + HORIZONTAL_SPACE;
        y11_y <= VERTICAL_SPACE + TWELVE_HEIGHT + VERTICAL_SPACE +
TWELVE_HEIGHT + VERTICAL_SPACE + TWELVE_HEIGHT + HR_HEIGHT + HR_SPACE;


        x12_length <= TWELVE_LENGTH;
        x12_height <= 2;
        y12_length <= 2;
        y12_height <= TWELVE_HEIGHT;
        x12_x <= 2*TWELVE_LENGTH + 2*HORIZONTAL_SPACE;
        x12_y <= TWELVE_HEIGHT + VERTICAL_SPACE + TWELVE_HEIGHT +
VERTICAL_SPACE + TWELVE_HEIGHT + VERTICAL_SPACE + TWELVE_HEIGHT +
HR_HEIGHT + HR_SPACE;
        y12_x <= 2*TWELVE_LENGTH + 2*HORIZONTAL_SPACE;
        y12_y <= VERTICAL_SPACE + TWELVE_HEIGHT + VERTICAL_SPACE +
TWELVE_HEIGHT + VERTICAL_SPACE + TWELVE_HEIGHT + HR_HEIGHT + HR_SPACE;
        end
        graphs_used = 0;
  end

  wire [11:0] wave_pixel1,  wave_pixel2, wave_pixel3, wave_pixel4, wave_pixel5, wave_pixel6,
wave_pixel7, wave_pixel8, wave_pixel9, wave_pixel10, wave_pixel11, wave_pixel12;
```

```
  blob #(.COLOR(12'hF_0_0))
        hrY(.x(HR_LENGTH -
1),.y(10'd0),.width(2),.height(HR_HEIGHT),.hcount(hcount),.vcount(vcount),.pixel(hrY_pixel));
  blob #(.COLOR(12'hF_0_0))
        hrX(.x(11'd0),.y(HR_HEIGHT -
1),.width(HR_LENGTH),.height(2),.hcount(hcount),.vcount(vcount),.pixel(hrX_pixel));

  // Wave 1 axes
  blob #(.COLOR(12'hF_0_0))

        x1(.x(x1_x),.y(x1_y),.width(x1_length),.height(x1_height),.hcount(hcount),.vcount(vcount
),.pixel(x1_pixel));
  blob #(.COLOR(12'hF_0_0))

        y1(.x(y1_x),.y(y1_y),.width(y1_length),.height(y1_height),.hcount(hcount),.vcount(vcount
),.pixel(y1_pixel));

  // Wave 2 axes
  blob #(.COLOR(12'hF_0_0))

        x2(.x(x2_x),.y(x2_y),.width(x2_length),.height(x2_height),.hcount(hcount),.vcount(vcount
),.pixel(x2_pixel));
  blob #(.COLOR(12'hF_0_0))

        y2(.x(y2_x),.y(y2_y),.width(y2_length),.height(y2_height),.hcount(hcount),.vcount(vcount
),.pixel(y2_pixel));

  // Wave 3 axes
  blob #(.COLOR(12'hF_0_0))

        x3(.x(x3_x),.y(x3_y),.width(x3_length),.height(x3_height),.hcount(hcount),.vcount(vcount
),.pixel(x3_pixel));
  blob #(.COLOR(12'hF_0_0))

        y3(.x(y3_x),.y(y3_y),.width(y3_length),.height(y3_height),.hcount(hcount),.vcount(vcount
),.pixel(y3_pixel));

  // Wave 4 axes
  blob #(.COLOR(12'hF_0_0))

        x4(.x(x4_x),.y(x4_y),.width(x4_length),.height(x4_height),.hcount(hcount),.vcount(vcount
),.pixel(x4_pixel));
  blob #(.COLOR(12'hF_0_0))

        y4(.x(y4_x),.y(y4_y),.width(y4_length),.height(y4_height),.hcount(hcount),.vcount(vcount
),.pixel(y4_pixel));

  // Wave 5 axes
  blob #(.COLOR(12'hF_0_0))
```

```
        x5(.x(x5_x),.y(x5_y),.width(x5_length),.height(x5_height),.hcount(hcount),.vcount(vcount
),.pixel(x5_pixel));
  blob #(.COLOR(12'hF_0_0))

        y5(.x(y5_x),.y(y5_y),.width(y5_length),.height(y5_height),.hcount(hcount),.vcount(vcount
),.pixel(y5_pixel));

  // Wave 6 axes
        blob #(.COLOR(12'hF_0_0))

        x6(.x(x6_x),.y(x6_y),.width(x6_length),.height(x6_height),.hcount(hcount),.vcount(vcount
),.pixel(x6_pixel));
        blob #(.COLOR(12'hF_0_0))

        y6(.x(y6_x),.y(y6_y),.width(y6_length),.height(y6_height),.hcount(hcount),.vcount(vcount
),.pixel(y6_pixel));

  // Wave 7 axes
  blob #(.COLOR(12'hF_0_0))

        x7(.x(x7_x),.y(x7_y),.width(x7_length),.height(x7_height),.hcount(hcount),.vcount(vcount
),.pixel(x7_pixel));
  blob #(.COLOR(12'hF_0_0))

        y7(.x(y7_x),.y(y7_y),.width(y7_length),.height(y7_height),.hcount(hcount),.vcount(vcount
),.pixel(y7_pixel));

  // Wave 8 axes
  blob #(.COLOR(12'hF_0_0))

        x8(.x(x8_x),.y(x8_y),.width(x8_length),.height(x8_height),.hcount(hcount),.vcount(vcount
),.pixel(x8_pixel));
  blob #(.COLOR(12'hF_0_0))

        y8(.x(y8_x),.y(y8_y),.width(y8_length),.height(y8_height),.hcount(hcount),.vcount(vcount
),.pixel(y8_pixel));

  // Wave 9 axes
        blob #(.COLOR(12'hF_0_0))

        x9(.x(x9_x),.y(x9_y),.width(x9_length),.height(x9_height),.hcount(hcount),.vcount(vcount
),.pixel(x9_pixel));
        blob #(.COLOR(12'hF_0_0))

        y9(.x(y9_x),.y(y9_y),.width(y9_length),.height(y9_height),.hcount(hcount),.vcount(vcount
),.pixel(y9_pixel));

  // Wave 10 axes
        blob #(.COLOR(12'hF_0_0))
```

```
        x10(.x(x10_x),.y(x10_y),.width(x10_length),.height(x10_height),.hcount(hcount),.vcount(
vcount),.pixel(x10_pixel));
        blob #(.COLOR(12'hF_0_0))

        y10(.x(y10_x),.y(y10_y),.width(y10_length),.height(y10_height),.hcount(hcount),.vcount(
vcount),.pixel(y10_pixel));
        // Wave 11 axes

        blob #(.COLOR(12'hF_0_0))

        x11(.x(x11_x),.y(x11_y),.width(x11_length),.height(x11_height),.hcount(hcount),.vcount(
vcount),.pixel(x11_pixel));
        blob #(.COLOR(12'hF_0_0))

        y11(.x(y11_x),.y(y11_y),.width(y11_length),.height(y11_height),.hcount(hcount),.vcount(
vcount),.pixel(y11_pixel));

        // Wave 12 axes
        blob #(.COLOR(12'hF_0_0))

        x12(.x(x12_x),.y(x12_y),.width(x12_length),.height(x12_height),.hcount(hcount),.vcount(
vcount),.pixel(x12_pixel));
        blob #(.COLOR(12'hF_0_0))

        y12(.x(y12_x),.y(y12_y),.width(y12_length),.height(y12_height),.hcount(hcount),.vcount(
vcount),.pixel(y12_pixel));


  wire sample;


  khz_clock k_clock(.clock100_mhz(clk_100mhz), .clock1k(sample));

  wire heart_rate_ready, divider_finished1, divider_finished2;
  wire [7:0] heart_rate,rem;
  wire [7:0] hundreds;
  wire [7:0] tens;
  wire [7:0] ones;
  wire [11:0] ones_pixel, tens_pixel, hundreds_pixel;

  divider #(.WIDTH(8)) div1(.clk(vclock), .sign(0), .start(heart_rate_ready),
.dividend(heart_rate),.divider(100),.quotient(hundreds),.remainder(rem),.ready(divider_finished1
));
  divider #(.WIDTH(8)) div2(.clk(vclock), .sign(0), .start(divider_finished1),
.dividend(rem),.divider(10),.quotient(tens),.remainder(ones),.ready(divider_finished2));

  wave #(.COLOR(12'h0_F_0))
wave1(.vclock(vclock),.hcount(hcount),.vcount(vcount),.sw(sw[0]),.number_of_graphs(number_
of_graphs),.write(write_lead[0]),.write_value(write_value),.x_offset(x1_x),.x_length(x1_length),.y
```

```
_offset(y1_y),.y_height(y1_height),.pixel(wave_pixel1),
.heart_rate(heart_rate),.heart_rate_ready(heart_rate_ready));
  wave #(.COLOR(12'h0_F_0))
wave2(.vclock(vclock),.hcount(hcount),.vcount(vcount),.sw(sw[1]),.number_of_graphs(number_
of_graphs),.write(write_lead[1]),.write_value(write_value),.x_offset(x2_x),.x_length(x2_length),.y
_offset(y2_y),.y_height(y2_height),.pixel(wave_pixel2), .heart_rate(),.heart_rate_ready());
  wave #(.COLOR(12'h0_F_0))
wave3(.vclock(vclock),.hcount(hcount),.vcount(vcount),.sw(sw[2]),.number_of_graphs(number_
of_graphs),.write(write_lead[2]),.write_value(write_value),.x_offset(x3_x),.x_length(x3_length),.y
_offset(y3_y),.y_height(y3_height),.pixel(wave_pixel3), .heart_rate(),.heart_rate_ready());
  wave #(.COLOR(12'h0_F_0))
wave4(.vclock(vclock),.hcount(hcount),.vcount(vcount),.sw(sw[3]),.number_of_graphs(number_
of_graphs),.write(write_lead[3]),.write_value(write_value),.x_offset(x4_x),.x_length(x4_length),.y
_offset(y4_y),.y_height(y4_height),.pixel(wave_pixel4), .heart_rate(),.heart_rate_ready());
  wave #(.COLOR(12'h0_F_0))
wave5(.vclock(vclock),.hcount(hcount),.vcount(vcount),.sw(sw[4]),.number_of_graphs(number_
of_graphs),.write(write_lead[4]),.write_value(write_value),.x_offset(x5_x),.x_length(x5_length),.y
_offset(y5_y),.y_height(y5_height),.pixel(wave_pixel5), .heart_rate(),.heart_rate_ready());
  wave #(.COLOR(12'h0_F_0))
wave6(.vclock(vclock),.hcount(hcount),.vcount(vcount),.sw(sw[5]),.number_of_graphs(number_
of_graphs),.write(write_lead[5]),.write_value(write_value),.x_offset(x6_x),.x_length(x6_length),.y
_offset(y6_y),.y_height(y6_height),.pixel(wave_pixel6), .heart_rate(),.heart_rate_ready());
  wave #(.COLOR(12'h0_F_0))
wave7(.vclock(vclock),.hcount(hcount),.vcount(vcount),.sw(sw[6]),.number_of_graphs(number_
of_graphs),.write(write_lead[6]),.write_value(write_value),.x_offset(x7_x),.x_length(x7_length),.y
_offset(y7_y),.y_height(y7_height),.pixel(wave_pixel7), .heart_rate(),.heart_rate_ready());
  wave #(.COLOR(12'h0_F_0))
wave8(.vclock(vclock),.hcount(hcount),.vcount(vcount),.sw(sw[7]),.number_of_graphs(number_
of_graphs),.write(write_lead[7]),.write_value(write_value),.x_offset(x8_x),.x_length(x8_length),.y
_offset(y8_y),.y_height(y8_height),.pixel(wave_pixel8), .heart_rate(),.heart_rate_ready());
  wave #(.COLOR(12'h0_F_0))
wave9(.vclock(vclock),.hcount(hcount),.vcount(vcount),.sw(sw[8]),.number_of_graphs(number_
of_graphs),.write(write_lead[8]),.write_value(write_value),.x_offset(x9_x),.x_length(x9_length),.y
_offset(y9_y),.y_height(y9_height),.pixel(wave_pixel9), .heart_rate(),.heart_rate_ready());
  wave #(.COLOR(12'h0_F_0))
wave10(.vclock(vclock),.hcount(hcount),.vcount(vcount),.sw(sw[9]),.number_of_graphs(number
_of_graphs),.write(write_lead[9]),.write_value(write_value),.x_offset(x10_x),.x_length(x10_lengt
h),.y_offset(y10_y),.y_height(y10_height),.pixel(wave_pixel10),
.heart_rate(),.heart_rate_ready());
  wave #(.COLOR(12'h0_F_0))
wave11(.vclock(vclock),.hcount(hcount),.vcount(vcount),.sw(sw[10]),.number_of_graphs(numbe
r_of_graphs),.write(write_lead[10]),.write_value(write_value),.x_offset(x11_x),.x_length(x11_len
gth),.y_offset(y11_y),.y_height(y11_height),.pixel(wave_pixel11),
.heart_rate(),.heart_rate_ready());
  wave #(.COLOR(12'h0_F_0))
wave12(.vclock(vclock),.hcount(hcount),.vcount(vcount),.sw(sw[11]),.number_of_graphs(numbe
r_of_graphs),.write(write_lead[11]),.write_value(write_value),.x_offset(x12_x),.x_length(x12_len
gth),.y_offset(y12_y),.y_height(y12_height),.pixel(wave_pixel12),
.heart_rate(),.heart_rate_ready());
```

```
  heart_rate #(.COLOR(12'hF_0_0)) hr(.hcount(hcount),
.vcount(vcount),.heart_rate(heart_rate),.hundreds(hundreds),.tens(tens),.ones(ones),.ones_pixe
l(ones_pixel),.tens_pixel(tens_pixel), .hundreds_pixel(hundreds_pixel));

  assign pixel = hrY_pixel | hrX_pixel | y1_pixel | x1_pixel |  x2_pixel | y2_pixel |  x3_pixel |
y3_pixel |  x4_pixel | y4_pixel | x5_pixel |  y5_pixel | x6_pixel |  y6_pixel
        | x7_pixel |  y7_pixel | x8_pixel | y8_pixel | x9_pixel |  y9_pixel | x10_pixel | y10_pixel |
x11_pixel |  y11_pixel | x12_pixel | y12_pixel
        | wave_pixel1 | wave_pixel2 | wave_pixel3 | wave_pixel4 | wave_pixel5 | wave_pixel6 |
wave_pixel7 | wave_pixel8 | wave_pixel9 | wave_pixel10 | wave_pixel11 | wave_pixel12
        | ones_pixel | tens_pixel | hundreds_pixel;

endmodule


module clock_quarter_divider(input clk100_mhz, output reg clock_25mhz = 0);
        reg counter = 0;
        always @(posedge clk100_mhz) begin
        counter <= counter + 1;
        if (counter == 0) begin
        clock_25mhz <= ~clock_25mhz;
        end
        end
endmodule


module clock_4khz_pulse(input clock_65mhz, output reg pulse_4khz = 0);
        reg [18:0] counter = 0;
        always @(posedge clock_65mhz) begin
        if(counter == 16250) begin
        pulse_4khz <= 1;
        counter <= 0;
        end
        else begin
        pulse_4khz <= 0;
        counter <= counter + 1;
        end
        end
endmodule

module khz_clock(input clock100_mhz, output reg clock1k = 0);
        reg [17:0] counter = 0;
        always @(posedge clock100_mhz) begin
        counter <= counter + 1;
        if(counter == 50000) clock1k <= 1;
        else clock1k <=  0;
        end
endmodule

module heart_rate #(parameter
```

```
        COLOR = 12'hF_F_F)  // default color: white
  (input [10:0] hcount,
        input [9:0] vcount,
        input [7:0] heart_rate,
        input [7:0] hundreds,
        input [7:0] tens,
        input [7:0] ones,
        output reg [11:0] ones_pixel, tens_pixel, hundreds_pixel);


  always @ * begin
        case(ones)
        0: begin
        if((hcount >= 100 && hcount <= 139 && ((vcount >= 10 && vcount <=13) || (vcount >=90
&& vcount <= 93) ))
                || (((hcount >= 100 && hcount <= 103)||(hcount >=136 && hcount <= 139)) &&
vcount >= 10 && vcount <= 93)) ones_pixel <= COLOR;
        else ones_pixel <= 0;
        end
        1: begin
        if(hcount >= 136 && hcount <= 139 && vcount >= 10 && vcount <= 93) ones_pixel <=
COLOR;
        else ones_pixel <= 0;
        end
        2: begin
        if((hcount >= 100 && hcount <= 139 && ((vcount >= 10 && vcount <=13) || (vcount >= 48
&& vcount <= 51) || (vcount >=90 && vcount <= 93) ))
                || (hcount >= 100 && hcount <= 103 && vcount >= 48 && vcount <= 93) || (hcount
>=136 && hcount <= 139 && vcount >= 10 && vcount <= 48)) ones_pixel <= COLOR;
        else ones_pixel <= 0;
        end
        3: begin
        if((hcount >= 100 && hcount <= 139 && ((vcount >= 10 && vcount <=13) || (vcount >= 48
&& vcount <= 51) || (vcount >=90 && vcount <= 93) ))
                || (hcount >=136 && hcount <= 139 && vcount >= 10 && vcount <= 94))
ones_pixel <= COLOR;
        else ones_pixel <= 0;
        end
        4: begin
        if((hcount >= 100 && hcount <= 139 && vcount >= 48 && vcount <= 51) || (((hcount >=
100 && hcount <=103)
                || (hcount >=136 && hcount <= 139)) && vcount >= 10 && vcount <= 51) ||
(hcount >= 136 && hcount <=139 && vcount >= 48 && vcount <= 93)) ones_pixel <= COLOR;
        else ones_pixel <= 0;
        end
        5: begin
        if ((hcount >= 100 && hcount <= 139 && ((vcount >= 10 && vcount <= 13)||(vcount >= 48
&& vcount <=51)||(vcount >= 90 && vcount <= 93)))
                || (hcount >= 100 && hcount <= 103 && vcount >=10 && vcount <= 51) || (hcount
>= 136 && hcount <= 139 && vcount >= 48 && vcount <= 93)) ones_pixel <= COLOR;
        else ones_pixel <= 0;
```

```
        end
        6: begin
        if((hcount >= 100 && hcount <= 139 && ((vcount >= 10 && vcount <= 13)||(vcount >= 48
&& vcount <=51)||(vcount >= 90 && vcount <= 93)))
                || (hcount >= 100 && hcount <= 103 && vcount >=10 && vcount <= 93) || (hcount
>= 136 && hcount <= 139 && vcount >=48 && vcount <= 93)) ones_pixel <= COLOR;
        else ones_pixel <= 0;
        end
        7: begin
        if ((hcount >= 136 && hcount <= 139 && vcount >=10 && vcount <= 93) ||(vcount >= 10
&& vcount <= 13 && hcount >= 100 && hcount <= 139) ) ones_pixel <= COLOR;
        else ones_pixel <= 0;
        end
        8: begin
        if ((hcount >= 100 && hcount <= 139 && ((vcount >= 10 && vcount <= 13)||(vcount >= 48
&& vcount <=51)||(vcount >= 90 && vcount <= 93)))
                || (((hcount >= 100 && hcount <= 103) || (hcount >= 136 && hcount <= 139)) &&
vcount >=10 && vcount <= 93) ) ones_pixel <= COLOR;
        else ones_pixel <= 0;
        end
        9: begin
        if ((((hcount >= 100 && hcount <= 103) || hcount >= 136 && hcount <= 139) && vcount
>=10 && vcount <= 51)
                || (hcount >= 136 && hcount <= 139 && vcount >= 48 && vcount <= 93)
                || (((vcount >= 10 && vcount <= 13)||(vcount >= 48 && vcount <=51)) && hcount
>= 100 && hcount <= 139)) ones_pixel <= COLOR;
        else ones_pixel <= 0;
        end
        endcase
        case(hundreds)
        0: begin
        hundreds_pixel <= 0;
        end
        1: begin
        if(hcount >=41 && hcount <= 44 && vcount >= 10 && vcount <= 93) hundreds_pixel <=
COLOR;
        else hundreds_pixel <= 0;
        end
        2: begin
        if((hcount >= 5 && hcount <= 44 && ((vcount >= 10 && vcount <=13) || (vcount >= 48 &&
vcount <= 51) || (vcount >=90 && vcount <= 93) ))
                || (hcount >= 5 && hcount <=8 && vcount >= 48 && vcount <= 93) || (hcount >=41
&& hcount <= 44 && vcount >= 10 && vcount <= 48)) hundreds_pixel <= COLOR;
        else hundreds_pixel <= 0;
        end
        3: begin
        if((hcount >= 5 && hcount <= 44 && ((vcount >= 10 && vcount <=13) || (vcount >= 48 &&
vcount <= 51) || (vcount >=90 && vcount <= 93) ))
                || (hcount >=41 && hcount <= 44 && vcount >= 10 && vcount <= 93))
hundreds_pixel <= COLOR;
        else hundreds_pixel <= 0;
```

```
        end
        4: begin
        if((hcount >= 5 && hcount <= 44 && vcount >= 48 && vcount <= 51) || (((hcount >= 5 &&
hcount <=8)
                || (hcount >=41 && hcount <= 44)) && vcount >= 10 && vcount <= 51) || (hcount
>= 41 && hcount <=44 && vcount >= 48 && vcount <= 93)) hundreds_pixel <= COLOR;
        else hundreds_pixel <= 0;
        end
        5: begin
        if ((hcount >= 5 && hcount <= 44 && ((vcount >= 10 && vcount <= 13)||(vcount >= 48 &&
vcount <=51)||(vcount >= 90 && vcount <= 93)))
                || (hcount >= 5 && hcount <= 8 && vcount >=10 && vcount <= 51) || (hcount >=
41 && hcount <= 44 && vcount >= 48 && vcount <= 93)) hundreds_pixel <= COLOR;
        else hundreds_pixel <= 0;
        end
        6: begin
        if((hcount >= 5 && hcount <= 44 && ((vcount >= 10 && vcount <= 13)||(vcount >= 48 &&
vcount <=51)||(vcount >= 90 && vcount <= 93)))
                || (hcount >= 5 && hcount <= 8 && vcount >=10 && vcount <= 93) || (hcount >=
41 && hcount <= 44 && vcount >=48 && vcount <= 93)) hundreds_pixel <= COLOR;
        else hundreds_pixel <= 0;
        end
        7: begin
        if ((hcount >= 41 && hcount <= 44 && vcount >=10 && vcount <= 93) ||(vcount >= 10 &&
vcount <= 13 && hcount >= 5 && hcount <= 44) ) hundreds_pixel <= COLOR;
        else hundreds_pixel <= 0;
        end
        8: begin
        if ((hcount >= 5 && hcount <= 44 && ((vcount >= 10 && vcount <= 13)||(vcount >= 48 &&
vcount <=51)||(vcount >= 90 && vcount <= 93)))
                || (((hcount >= 5 && hcount <= 8) || (hcount >= 41 && hcount <= 44)) && vcount
>=10 && vcount <= 93) ) hundreds_pixel <= COLOR;
        else hundreds_pixel <= 0;
        end
        9: begin
        if ((((hcount >= 5 && hcount <= 8) || hcount >= 41 && hcount <= 44) && vcount >=10 &&
vcount <= 51)
                || (hcount >= 41 && hcount <= 44 && vcount >= 48 && vcount <= 93)
                || (((vcount >= 10 && vcount <= 13)||(vcount >= 48 && vcount <=51)) && hcount
>= 5 && hcount <= 44)) hundreds_pixel <= COLOR;
        else hundreds_pixel <= 0;
        end
        endcase
        case(tens)
        0: begin
        if((hcount >= 55 && hcount <= 94 && ((vcount >= 10 && vcount <=13) || (vcount >=90
&& vcount <= 93) ))
                || (((hcount >= 55 && hcount <= 58)||(hcount >=91 && hcount <= 94)) && vcount
>= 10 && vcount <= 93)) tens_pixel <= COLOR;
        else tens_pixel <= 0;
        end
```

```
        1: begin
        if(hcount >= 91 && hcount <= 94 && vcount >= 10 && vcount <= 93) tens_pixel <=
COLOR;
        else tens_pixel <= 0;
        end
        2: begin
        if((hcount >= 55 && hcount <= 94 && ((vcount >= 10 && vcount <=13) || (vcount >= 48
&& vcount <= 51) || (vcount >=90 && vcount <= 93) ))
                || (hcount >= 55 && hcount <= 58 && vcount >= 48 && vcount <= 93) || (hcount
>=91 && hcount <= 94 && vcount >= 10 && vcount <= 48)) tens_pixel <= COLOR;
        else tens_pixel <= 0;
        end
        3: begin
        if((hcount >= 55 && hcount <= 94 && ((vcount >= 10 && vcount <=13) || (vcount >= 48
&& vcount <= 51) || (vcount >=90 && vcount <= 93) ))
                || (hcount >=91 && hcount <= 94 && vcount >= 10 && vcount <= 94)) tens_pixel
<= COLOR;
        else tens_pixel <= 0;
        end
        4: begin
        if((hcount >= 55 && hcount <= 94 && vcount >= 48 && vcount <= 51) || (((hcount >= 55
&& hcount <=58)
                || (hcount >=91 && hcount <= 94)) && vcount >= 10 && vcount <= 51) || (hcount
>= 91 && hcount <=94 && vcount >= 48 && vcount <= 93)) tens_pixel <= COLOR;
        else tens_pixel <= 0;
        end
        5: begin
        if ((hcount >= 55 && hcount <= 94 && ((vcount >= 10 && vcount <= 13)||(vcount >= 48
&& vcount <=51)||(vcount >= 90 && vcount <= 93)))
                || (hcount >= 55 && hcount <= 58 && vcount >=10 && vcount <= 51) || (hcount >=
91 && hcount <= 94 && vcount >= 48 && vcount <= 93)) tens_pixel <= COLOR;
        else tens_pixel <= 0;
        end
        6: begin
        if((hcount >= 55 && hcount <= 94 && ((vcount >= 10 && vcount <= 13)||(vcount >= 48
&& vcount <=51)||(vcount >= 90 && vcount <= 93)))
                || (hcount >= 55 && hcount <= 58 && vcount >=10 && vcount <= 93) || (hcount >=
91 && hcount <= 94 && vcount >=48 && vcount <= 93)) tens_pixel <= COLOR;
        else tens_pixel <= 0;
        end
        7: begin
        if ((hcount >= 91 && hcount <= 94 && vcount >=10 && vcount <= 93) ||(vcount >= 10 &&
vcount <= 13 && hcount >= 55 && hcount <= 94) ) tens_pixel <= COLOR;
        else tens_pixel <= 0;
        end
        8: begin
        if ((hcount >= 55 && hcount <= 94 && ((vcount >= 10 && vcount <= 13)||(vcount >= 48
&& vcount <=51)||(vcount >= 90 && vcount <= 93)))
                || (((hcount >= 55 && hcount <= 58) || (hcount >= 91 && hcount <= 94)) &&
vcount >=10 && vcount <= 93) ) tens_pixel <= COLOR;
        else tens_pixel <= 0;
```

```
        end
        9: begin
        if ((((hcount >= 55 && hcount <= 58) || hcount >= 91 && hcount <= 94) && vcount >=10
&& vcount <= 51)
                || (hcount >= 91 && hcount <= 94 && vcount >= 48 && vcount <= 93)
                || (((vcount >= 10 && vcount <= 13)||(vcount >= 48 && vcount <=51)) && hcount
>= 55 && hcount <= 94)) tens_pixel <= COLOR;
        else tens_pixel <= 0;
        end
        endcase
  end

endmodule

// The divider module divides one number by another. It
// produces a signal named "ready" when the quotient output
// is ready, and takes a signal named "start" to indicate
// the the input dividend and divider is ready.
// sign -- 0 for unsigned, 1 for twos complement

// It uses a simple restoring divide algorithm.
// http://en.wikipedia.org/wiki/Division_(digital)#Restoring_division

module divider #(parameter WIDTH = 8)
 (input clk, sign, start,
  input [WIDTH-1:0] dividend,
  input [WIDTH-1:0] divider,
  output reg [WIDTH-1:0] quotient,
  output [WIDTH-1:0] remainder,
  output ready);

  reg [WIDTH-1:0]  quotient_temp;
  reg [WIDTH*2-1:0] dividend_copy, divider_copy, diff;
  reg negative_output;

  assign remainder = (!negative_output) ?
        dividend_copy[WIDTH-1:0] : ~dividend_copy[WIDTH-1:0] + 1'b1;

  reg [5:0] bit;
  reg del_ready = 1;
  assign ready = (!bit) & ~del_ready;

  wire [WIDTH-2:0] zeros = 0;
  initial bit = 0;
  initial negative_output = 0;
  always @( posedge clk ) begin
        del_ready <= !bit;
        if( start ) begin

        bit = WIDTH;
```

```
        quotient = 0;
        quotient_temp = 0;
        dividend_copy = (!sign || !dividend[WIDTH-1]) ?
                {1'b0,zeros,dividend} :
                {1'b0,zeros,~dividend + 1'b1};
        divider_copy = (!sign || !divider[WIDTH-1]) ?
                  {1'b0,divider,zeros} :
                  {1'b0,~divider + 1'b1,zeros};

        negative_output = sign &&
                ((divider[WIDTH-1] && !dividend[WIDTH-1])
                ||(!divider[WIDTH-1] && dividend[WIDTH-1]));
        end
        else if ( bit > 0 ) begin
        diff = dividend_copy - divider_copy;
        quotient_temp = quotient_temp << 1;
        if( !diff[WIDTH*2-1] ) begin
        dividend_copy = diff;
        quotient_temp[0] = 1'd1;
        end
        quotient = (!negative_output) ?
                quotient_temp :
                ~quotient_temp + 1'b1;
        divider_copy = divider_copy >> 1;
        bit = bit - 1'b1;
        end
  end
endmodule


module recorder(input CLK100MHZ, clock_65mhz,  pulse_4khz, AD3N, AD3P, output reg[4:0]
selectLow, selectHigh, output reg [11:0] write, output [11:0] value, input [11:0] sw);
        reg conversion_start, conversion_done;
        wire [15:0] adc_data;
        wire eoc;
        reg [3:0] LEAD = 0;
        assign value = adc_data[15:4];
        xadc_wiz_0 LEADadc(
        .dclk_in(CLK100MHZ),   // Master clock for DRP and XADC.
        .di_in(0),         // DRP input info (0 becuase we don't need to write)
        .daddr_in(6'h13),       // The DRP register address for the third analog aux
        .den_in(1),             // DRP enable line high (we want to read)
        .dwe_in(0),             // DRP write enable low (never write)
        .drdy_out(),            // DRP ready signal (unused)
        .do_out(adc_data),      // DRP output from register (the ADC data)
        .vp_in(0),              // dedicated/built in analog channel on bank 0
        .vn_in(0),              // can't use this analog channel b/c of nexys 4 setup
        .vauxp3(AD3P),          // The third analog auxiliary input channel
        .vauxn3(AD3N),          // Choose this one b/c it's on JXADC header 1
        .channel_out(),         // Not useful in sngle channel mode
        .eoc_out(eoc),          // Pulses high on end of ADC conversion
```

```
        .alarm_out(),           // Not useful
        .eos_out(),             // End of sequence pulse, not useful
        .busy_out()             // High when conversion is in progress. unused.
);
        always @(posedge clock_65mhz) begin

        /*if(sw[0]) begin
        selectLow <= 1;
        selectHigh <= 6;
        write <= 12'b000000000001;
        end
        else if(sw[1]) begin
        selectLow <= 1;
        selectHigh <= 8;
        write <= 12'b000000000010;
        end
        else if(sw[2]) begin
        selectLow <= 0;
        selectHigh <= 8;
        write <= 12'b000000000100;
        end
        else if(sw[3]) begin
        selectLow <= 2;
        selectHigh <= 7;
        write <= 12'b000000001000;
        end
        else if(sw[4]) begin
        selectLow <= 2;
        selectHigh <= 6;
        write <= 12'b000000010000;
        end
        else if(sw[5]) begin
        selectLow <= 2;
        selectHigh <= 8;
        write <= 12'b000000100000;
        end
        else if(sw[6]) begin
        selectLow <= 2;
        selectHigh <= 5;
        write <= 12'b000001000000;
        end
        else if(sw[7]) begin
                selectLow <= 2;
        selectHigh <= 4;
        write <= 12'b000010000000;
        end
        else if(sw[8]) begin
        selectLow <= 2;
        selectHigh <= 3;
        write <= 12'b000100000000;
        end
```

```
else if(sw[9]) begin
selectLow <= 2;
selectHigh <= 2;
write <= 12'b001000000000;
end
else if(sw[10]) begin
selectLow <= 2;
selectHigh <= 1;
write <= 12'b010000000000;
end
else if(sw[11]) begin
        selectLow <= 2;
selectHigh <= 0;
write <= 12'b100000000000;
end
else begin
selectLow <= 0;
selectHigh <= 0;
write <= 12'b000000000000;
end
end  */
if(pulse_4khz) begin
case(LEAD)
0:  begin
        selectLow <= 1;
        selectHigh <= 6;
        write <= 12'b000000000001;
        end
1:  begin
        selectLow <= 1;
        selectHigh <= 8;
        write = 12'b000000000010;
        end
2:  begin
        selectLow <= 0;
        selectHigh <= 8;
        write = 12'b000000000100;
        end
3:  begin
        selectLow <= 2;
        selectHigh <= 7;
        write = 12'b000000001000;
        end
4:  begin
        selectLow <= 2;
        selectHigh <= 6;
        write = 12'b000000010000;
        end
5:  begin
        selectLow <= 2;
        selectHigh <= 8;
```

```
                write = 12'b000000100000;
                end
        6:  begin
                selectLow <= 2;
                selectHigh <= 5;
                write = 12'b000001000000;
                end
        7:  begin
                selectLow <= 2;
                selectHigh <= 4;
                write = 12'b000010000000;
                end
        8:  begin
                selectLow <= 2;
                selectHigh <= 3;
                write = 12'b000100000000;
                end
        9:  begin
                selectLow <= 2;
                selectHigh <= 2;
                write = 12'b001000000000;
                end
        10:  begin
                selectLow <= 2;
                selectHigh <= 1;
                write = 12'b010000000000;
                end
        11:  begin
                selectLow <= 2;
                selectHigh <= 0;
                write = 12'b100000000000;
                end
                default: begin
                selectLow <= 0;
                selectHigh <= 0;
                write <= 12'b000000000000;
                end
        endcase
        if(LEAD == 11) LEAD <= 0;
        else LEAD <= LEAD + 1;
        end
        else begin
        write <= 12'b000000000000;
        end
        end
endmodule

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,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
endmodule

module blob
  #(parameter
          COLOR = 12'hF_F_F)  // default color: white
  (input [10:0] x,hcount, width,
          input [9:0] y,vcount, height,
          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
```

```
module wave #(parameter
        COLOR = 12'hF_F_F, // default color: white
        HIGH_VALUE = 2000,
        HIGH_BOUNDARY = 1700)
  (
        input vclock,
        input [10:0] hcount,
        input [9:0] vcount,
        input sw,
        input [3:0] number_of_graphs,
        input write,
        input [11:0] write_value,
        input [10:0] x_offset,
        input [10:0] x_length,
        input [9:0] y_offset,
        input [9:0] y_height,
        output reg [11:0] pixel,
        output reg [7:0] heart_rate,
        output reg heart_rate_ready = 0);

  wire [11:0] out1;
  reg [11:0] display_value = 0;
  reg [9:0] address = 0;
  reg [10:0] read_address = 0;
  reg [2:0] divider = 3;
  reg [11:0] reset_counter = 0;
  reg [9:0] high_counter = 0;
  reg [5:0] heart_rate_counter = 0;
  reg state = 0;
  reg [11:0] old_display_value = 0;
  blk_mem_gen_0 lead1(.clka(vclock), .ena(1), .wea(write), .addra(address), .dina(write_value),
.clkb(vclock), .enb(1), .addrb(read_address[9:0]), .doutb(out1));
  always @(posedge vclock)begin
        if (number_of_graphs <= 1) begin
        divider <= 3;
        if (hcount < 1000) read_address <= hcount;
        else read_address <= 1000;
        end
        else if (number_of_graphs == 2) begin
        divider <= 4;
        if (hcount < 1000) read_address <= hcount;
        else read_address <= 1000;
        end
        else if (number_of_graphs <= 4) begin
        divider <= 4;
        if (hcount >= x_offset && hcount <= x_offset + x_length) read_address <= (hcount -
x_offset)*2;
        else read_address <= 1000;
        end
        else if (number_of_graphs > 4) begin
        divider <= 5;
```

```
        if (hcount >= x_offset && hcount <= x_offset + x_length) read_address <= (hcount -
x_offset)*3;
        else read_address <= 1000;
        end
        display_value <= out1 >> divider;
        if (write) begin
        if (address == 999) address <= 0;
        else address <= address + 1;
        if (reset_counter < 3996) begin
        heart_rate_ready <= 0;
        reset_counter <= reset_counter + 1;
        if (state == 0) begin
                if (write_value > HIGH_VALUE) begin // value above high value for heart beat
                state <= 1;
                end
        end
        else if (state == 1) begin
                if (write_value > HIGH_BOUNDARY) begin  // value above the boundary to stay
high
                high_counter <= high_counter + 1;   //track number of high values while in high
state
                if (high_counter == 2) begin // record a heart beat if number of sequential high
counts is 2
                heart_rate_counter <= heart_rate_counter + 1;
                end
                end
                else begin
                state <= 0; // go back to low value state when value is below high boundary
                high_counter <= 0;
                end
        end
        end
        else begin //after X number of entries are recorded, reset the heart rate counter to start
again
        reset_counter <= 0;
        high_counter <= 0;
        state <= 0;
        heart_rate_counter <= 0;
        heart_rate <= heart_rate_counter * 5; //recording 12 seconds of heart rates so multiply
by 5
        heart_rate_ready <= 1;
        end
        end

        if (sw) begin
        if (number_of_graphs <= 2 && read_address - 3 == address) pixel = 0;
        else if (number_of_graphs <= 4 && (read_address - 6 == address || read_address - 7 ==
address) ) pixel = 0;
        else if (number_of_graphs > 4 && (read_address - 9 == address || read_address - 10 ==
address || read_address - 11 == address) ) pixel = 0;
        else if (hcount > x_offset+4 && hcount < x_offset + x_length) begin
```

```
        if (vcount > y_offset) begin
                if (vcount - y_offset == y_height - display_value) begin
                pixel = COLOR;
                end
                else if (display_value > old_display_value) begin
                if(vcount - y_offset > y_height - display_value && vcount - y_offset < y_height-
old_display_value) pixel = COLOR;
                else pixel = 0;
                end
                else if (display_value < old_display_value) begin
                if(vcount - y_offset < y_height - display_value && vcount - y_offset > y_height-
old_display_value) pixel = COLOR;
                else pixel = 0;
                end
                else pixel = 0;
        end
        else pixel = 0;
        end
        else pixel = 0;
        end
        else pixel = 0;
        old_display_value <= display_value;
  end
endmodule
```