Final Project Report - CS/EE4840
Monitoring and Processing Stock Market Data in Real-Time Using the Cyclone V FPGA

Nathan Abrams(nca2123) Alexander Gazman(ag3529) Hang Guan(hg2388)

1. Introduction

The general premise of our project is to design a system where hardware and software work together to monitor real-time stock data for many stocks and implement an algorithm to analyze the market to inform a user on whether they should buy, sell, or hold the stock. The stock's fluctuations in price will also be displayed on a screen with a confidence level associated with the suggested action.

2. High Frequency Trading Algorithm

The decision making algorithm is based on high-frequency trading (HFT) where each new incoming data is being analyze in real time. The particular approach being adopted and implemented in this project is based on a comparing exponential moving averages [1]. At each time when a new data being registered at the hardware memory block, three additional points are generated: three exponential moving averages of 128, 32 and 8 points (figure 1).

The algorithm first compares two largest moving averages points, and the results acts as a "compass" for the final decision. If the moving average of 32 is larger than the moving average of 128, the compass will suggest either to Buy or Hold, and vice versa. The second comparison between the real value and the 8 points moving average will determine the final decision. If the value is in the same direction as the compass, the algorithm will suggest to make the action of the compass (buy/sell), otherwise, the decision at the point would be to hold (don't buy or sell).

Figure 1: HFT [1] implementation. Four curves moving simultaneously of real data and exponential averages which predict in real time whether the trader should buy, sell or hold a particular stock.

3. General Design

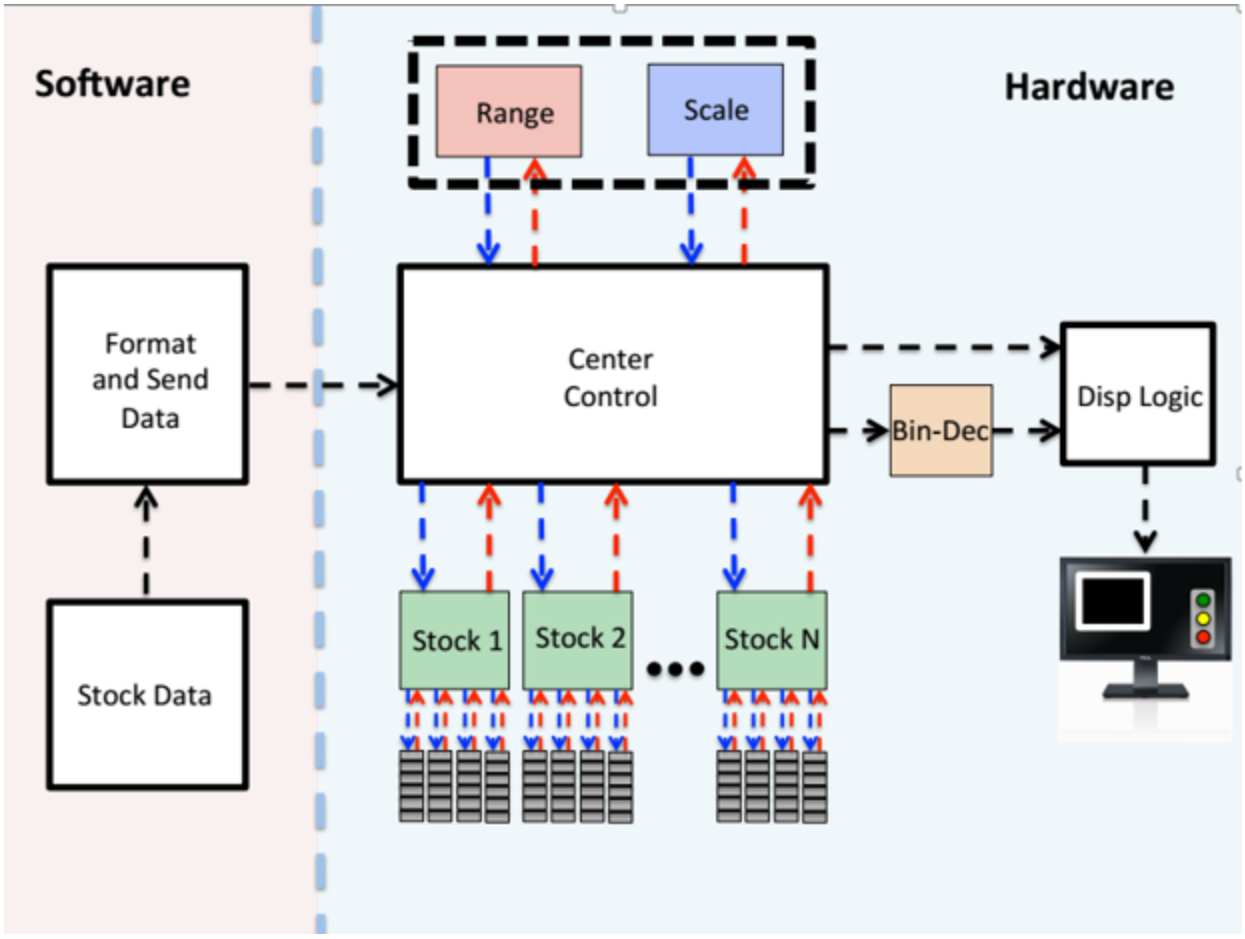


Figure 2: General layout of the project.

The general process is as follows. The software side reads in the stock data from the .txt file containing the relevant stock's values. The software contains two threads. One thread takes in user input. The user can input changes to what stock is displaying, whether the range is fixed or flexible, and restart the data (more on this later). The other thread reads the stock data and sends 32 bit messages to the hardware. The thread also accesses variables that can be altered by the user input thread. The 32 bit message is as follows. The first bit is for indicating whether the message is for sending data to the hardware or requesting the display. Bits 30 to 24 are not used. Bits 23 to 17 are used for the stock ID. If the first bit is set to send data, the stock ID is that of what stock module the data is going to. If the first bit is set to request a display, the stock ID is the stock that should be displayed. The last 17 bits are for the stock data. If the first bit is set to send data, that is the stock value. If the first bit is for requesting a display, the 17 bits may be used for dictating a fixed range (more on this later), or could go unused.

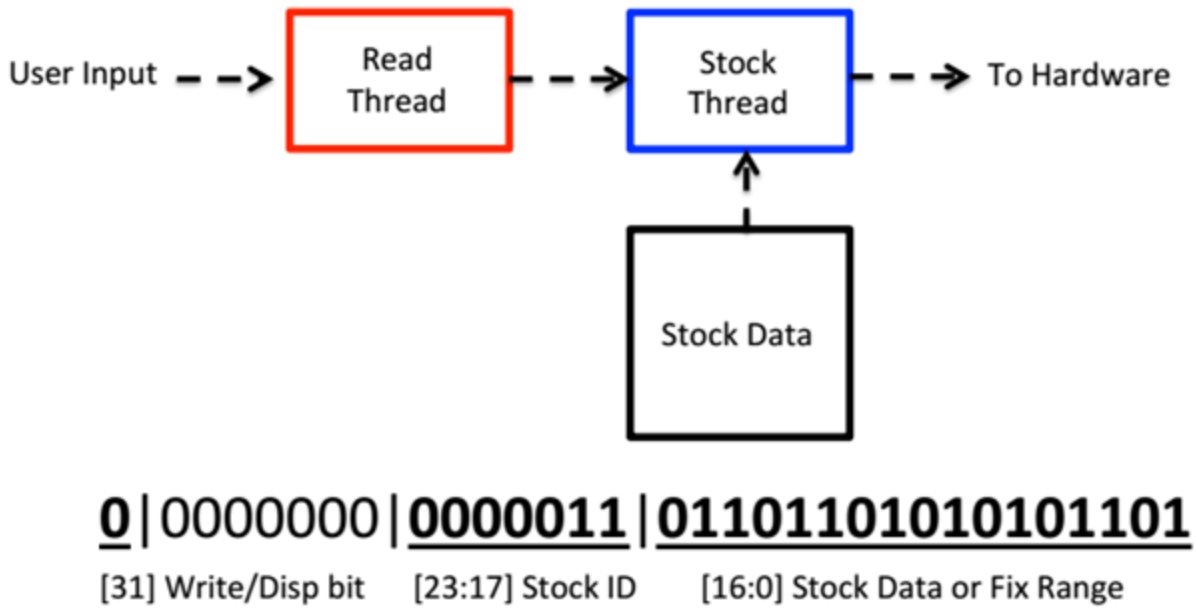


Figure 3: Software block diagram and message format.

The center control on the hardware sends the new stock data to the appropriate stock module when the request is to send data. Once the new data is read in, the following steps are taken. First, the module updates the exponential moving averages. The three exponential moving average equations are given below.

$$EMA\text{-}4_t <= (61/64)*EMA\text{-}4_{t-1} + (3/64)*Stock$$

$$EMA\text{-}32_t <= (63/64)*EMA\text{-}32_{t-1} + (1/64)*Stock$$

$$EMA\text{-}128_t <= (127/128)*EMA\text{-}128_{t-1} + (1/128)*Stock$$

Figure 4: Exponential moving average equations used..

The alpha values of these exponential moving averages were chosen to mimic moving averages of 4 points, 32 points, and 128 points respectively (as determined through simulations in MATLAB). The new stock price is weighted significantly less than the old EMA value, and the slower averages have more weight placed on the old EMA. The values were also chosen with the intent of being a power of two for the denominator. This meant we did not need the expensive division operator, we could just shift the registers the appropriate number of bits to correspond to dividing by 64 or 128. We did use multiplication for the numerator, but were able to accomplish that in a single clock cycle without issue. The module also performed rounding by checking the most significant bit that was not included in the shift; if it was a one then one was added to the

shifted result. Once the EMAs were updated, these values were written to memory in a location called “last stock flag”. This flag corresponds to the oldest point stored in memory. Once the memory was updated, the last stock flag is then shifted down the stack to the next oldest memory.

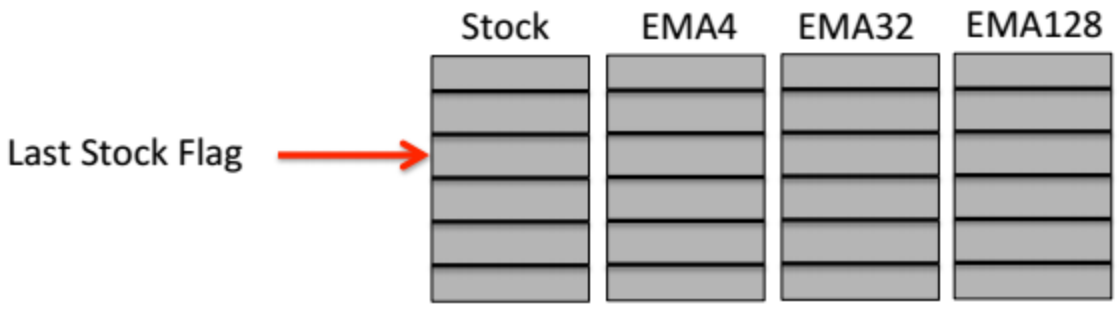


Figure 5: Memory layout for a single stock module.

If the software sends a request display bit in its message, then the center control sends a flag to the appropriate stock module to send the data to the center control. When that occurs, for each clock cycle the appropriate stock module will send the stored stock value, EMA4, EMA32, and EMA128 to the center control, starting with the oldest values first. There, they are stored in registers (400 registers for each of the 4 values, each value containing a 17 bit number for the stock value—which corresponds to values up to $1310.72).

When the software sends a display request, the next step is send the registers to the range and scale modules. While the data is being sent to the center control from the stock modules, it is also being sent to the range module which compares each new value against the previous minimum and maximum values. This way, when the data is uploaded to the center control, the range, min, and max are also determined. The next step is to send each value to the scale module. The scale module’s purpose is to turn the stock values to pixel values. A range is passed in to the scale module (either the calculated range or the hardcoded range sent from the software if the range is selected to be fixed). Then, groups of four values are passed in together every clock cycle to the scale module (the stock value, the EMA4, the EMA32, and the EMA128). The calculation is to shift it from the stock value to a pixel value between 0 and 300 for the display. The input value subtracts in the min, and that result is multiplied by 300. Next the range is rounded up to a power of two (so the register can be shifted instead of using a division block), and the range shift occurs. Finally, that result is rounded as necessary and subtracted from 320 (we want the lowest values to be at pixel 320 vertically. The following pseudo codes were used.

```
Temp <= 300*(input – min)
Mid <= Temp shifted by appropriate range
Output <= 320 – (mid + round)
```

Figure 6: Pseudo code for converting stock values to pixel values.

The next step was to send the values to the display logic. The general layout of the display logic is to read in objects and put a write okay flag for the pixel when it meets the condition below. The object was thought of as an array of registers for us. For the drawing of the graph, each line is represented by an array of registers of length 400 (number of points), where each register was 10 bits with the value corresponding to the vertical pixel location.

```
Write_okay = (vcount == OBJECT[hcount])
```

Figure 7: General drawing check for an object.

There is also a module for converting binary values to decimal values. The decimal values needed are for the maximum value on the y-axis, the minimum value on the y-axis, and the stock price on the bottom of the screen. The binary to decimal algorithm was based on the following pseudo code and translated into an always flip flop module that could handle 17 bit binary numbers.

```
for(i=0; i<8; i++) {
        //check all columns for >= 5
        for each column {
                if (column >= 5)
                        column += 3;

        //shift all binary digits left 1
        Hundreds <<= 1;
        Hundreds[0] = Tens[3];
        Tens << = 1;
        Tens[0] = Ones[3];
        Ones << = 1;
        Ones[0] = Binary[7];
        Binary <<= 1;
}
```

Figure 8: Psuedo code for binary to decimal conversion/

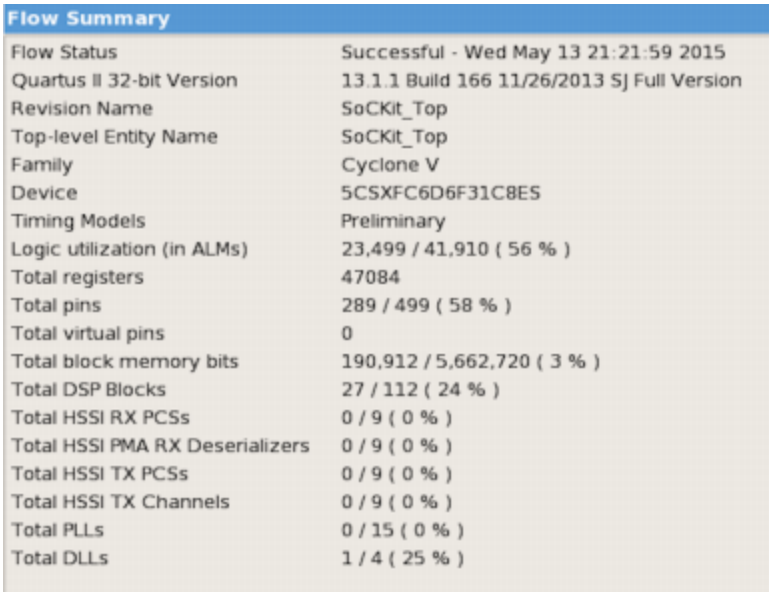
User Interaction

The user interaction allows for the user to change the stock being displayed, to restart the display, and to change the range from a flexible range display to a fixed range display. The fixed range takes in the max and min from the overall stock data as the range, and the flexible range calculates the range for the given points displayed on the screen at any given time. Restarting involves setting the file back to the initial setting (i = 0), and pumping zeros for all the values in memory for all of the stocks and their EMAs.

Scaling

We had initially intended to track an entire index's worth of stocks. We did not attempt to maximize the number of stocks tracked, but we found that at the current implementation we were using 56% of the available logic and 3% of the available memory bits. We had thought our

bottleneck would be because of memory usage, but it appears to be the logic utilization. We could improve on this by using a central module instead of a single module for each stock. Additionally, we could pass less points between modules (we had intended to pass only 100 points worth for each stock and do a line drawing between the points, but ended up having to pass 400 points).



Flow Summary	
Flow Status	Successful - Wed May 13 21:21:59 2015
Quartus II 32-bit Version	13.1.1 Build 166 11/26/2013 SJ Full Version
Revision Name	SoCKit_Top
Top-level Entity Name	SoCKit_Top
Family	Cyclone V
Device	5CSXFC6D6F31C8ES
Timing Models	Preliminary
Logic utilization (in ALMs)	23,499 / 41,910 (56 %)
Total registers	47084
Total pins	289 / 499 (58 %)
Total virtual pins	0
Total block memory bits	190,912 / 5,662,720 (3 %)
Total DSP Blocks	27 / 112 (24 %)
Total HSSI RX PCSs	0 / 9 (0 %)
Total HSSI PMA RX Deserializers	0 / 9 (0 %)
Total HSSI TX PCSs	0 / 9 (0 %)
Total HSSI TX Channels	0 / 9 (0 %)
Total PLLs	0 / 15 (0 %)
Total DLLs	1 / 4 (25 %)

Figure 9: Final compilation results.

4. Data generation

The stock market data that is used in the hardware analysis algorithm is saved as a text file in the Linux Kernel of the Cyclone V board. The text file is saved in the following order:

[stock name_1] [value]
[stock name_2] [value]
[stock name_3] [value] …

Figure 10: Data format in the text file.

and read from the C code. The data is generated using web-scraping from a dedicated python library called Yahoo-Finance [1]. The API allows downloading for each stock symbol the following data sets: stock volume (number of buys and sells per day), opening and closing price, high and low values and the date. The information is saved in a Python dictionary data structure and parsed in order to get the values in interest. Python code for generating the stock market data is attached in the code appendix section.

For simplicity, for each stock symbol about 1400 values are saved which represent a daily closing price between between a span of four years. In our algorithm implementation, we regard this daily data stream as “live” data. The computation of the three moving averages were first calculated and plotted in python before the hardware implementation. Plot 1 shows the three stock symbols: TEVA, IBM and INTC plotted versus days. The red curve is the actual data while the smoother lines correspond to three moving averages. The final data file consist of eleven stock symbol with 4000 data points for each.

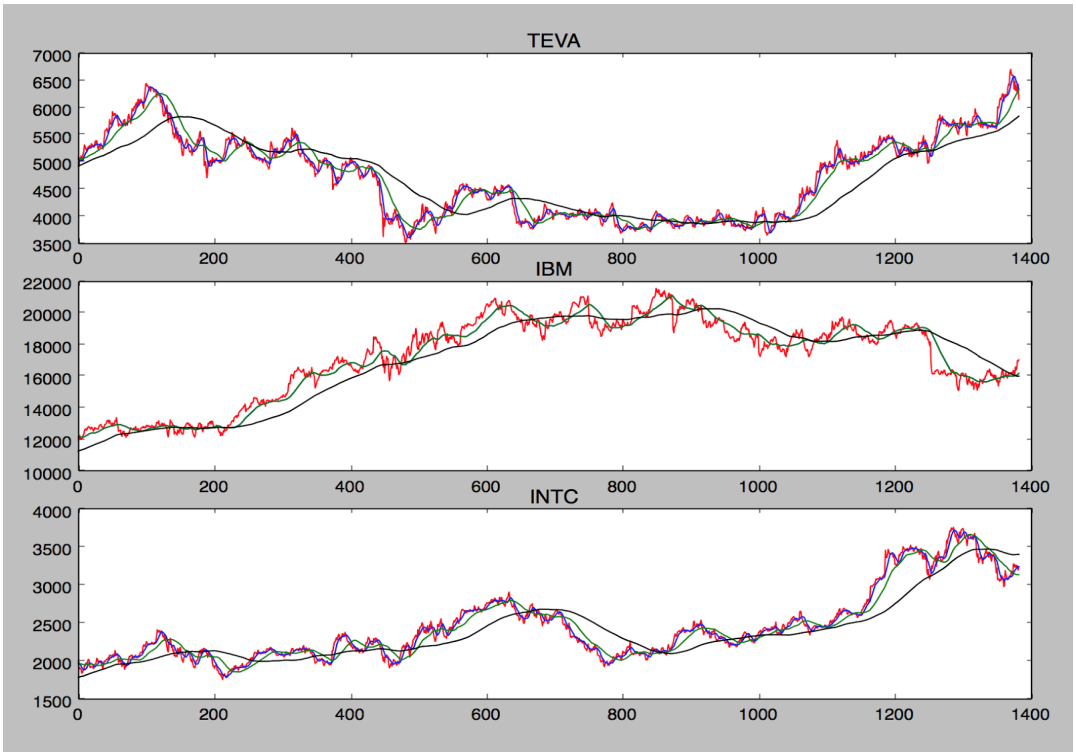


Figure 11: Plotted web-scrapped stock market data of TEVA IBM and INTC.

5. Sprites Generation

Two symbol sprites types were generated in python to include several features in the screen. Numbers are used to represent current stock value prices and y-axis coordinates, while letters are used to show the current stock symbol being analyzes and the appropriate action - buy/sell/hold. An image of the output screen is shown in figure 2.

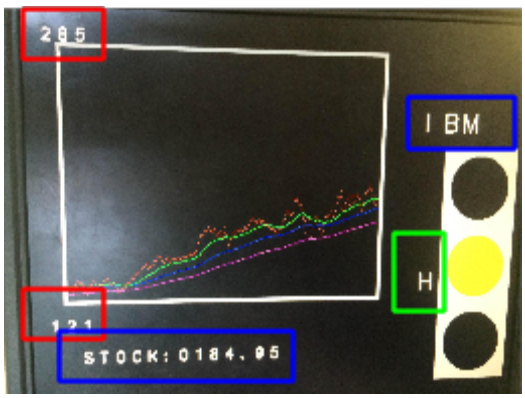
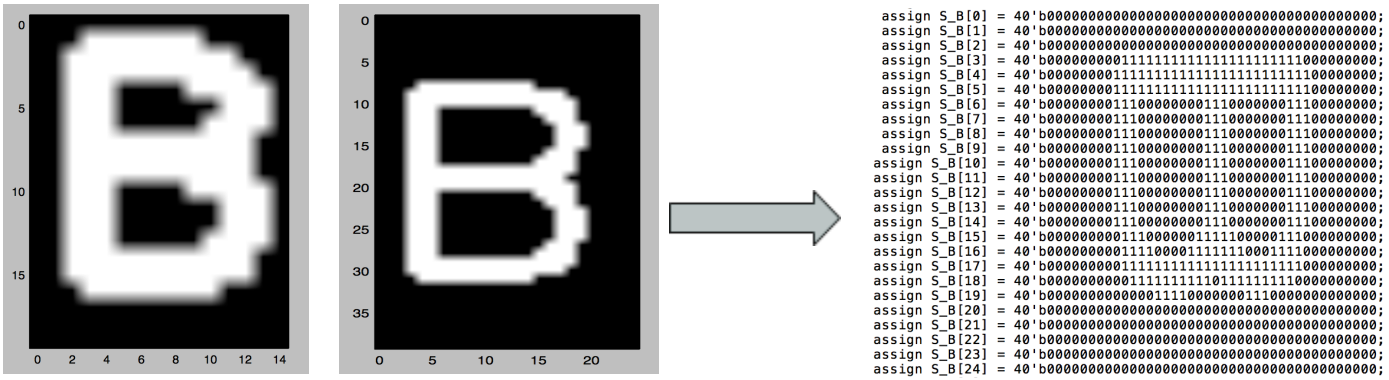


Figure 12: Plotted web-scrapped stock market data of TEVA IBM and INTC.

The sprites consists of the alphabet, numbers and punctuation symbols. The two types corresponds to two sizes, one of 40 x 25 and other 20 x 15 pixels. The code is written using an image processing library called OpenCV. An array of all symbols in interested are generated to black and white images shown in figure 3. Additional filtering stage was applied to achieve a fully binary image which was then converted to a 40 or 15 (depends on the symbol size) registers. The code of the sprites generation is added to the code appendix section.



(a) **(b)**

Figure 13: (a) Example of character B images in two sprite sizes - 40 x 25 and 20x 15 (b) Translated image to a VerilogSystems code.

6. Bresenham's Line Algorithm for Stock Value Display

Since the stock values can change very rapidly even over a very small time interval, if we directly plot the stock value vs. time figure on the screen, the users will end up with seeing lots of discrete dots on the screen. To make our display look nicer, we adopted the so-called Bresenham's line drawing algorithm here. Bresenham's line algorithm is used to determines the points of an n-dimensional raster that should be selected in order to form a close approximation to a straight line between two points. It is commonly used to draw lines on a computer screen, as it uses only integer addition, subtraction and bit shifting, all of which are very cheap operations in standard computer architectures [2].

The block diagram for the display part is shown in Figure 4. It contains four chunks of stock value memories, a controller , a Bresenham's line drawing module, a frame buffer and a raster scanner. The stock value memories contain the stock value, and three exponential moving averages from the current time to 99 sampled time (every 75 ms) before. The controller is implemented using a finite state machine, as shown in Figure 5. The main function is to fetch the data from the memory and input the data into the Bresenham's line drawing module. To synchronize with Bresenham's module, two hand shaking signals (start/done) are used. The generated coordinates will be save in the frame buffer. The frame buffer is a two dimensional

register array. The size of this frame buffer is 640 * 480 * 3. We use 3 bits to represent 5 different colors (black, red, green, blue, magenta)

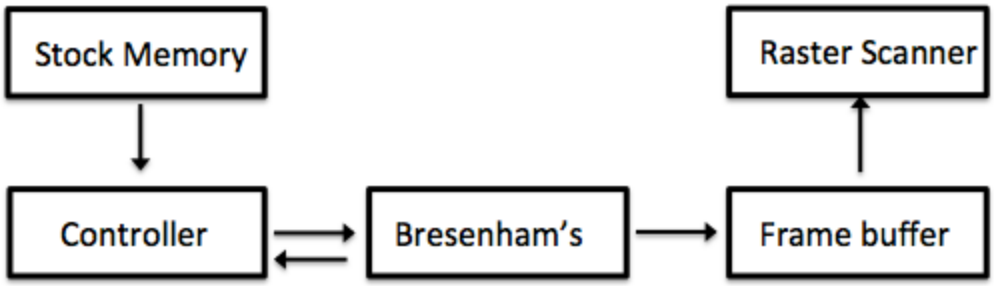


Figure 14: Simplified FSM for the control of Bresenham's algorithm

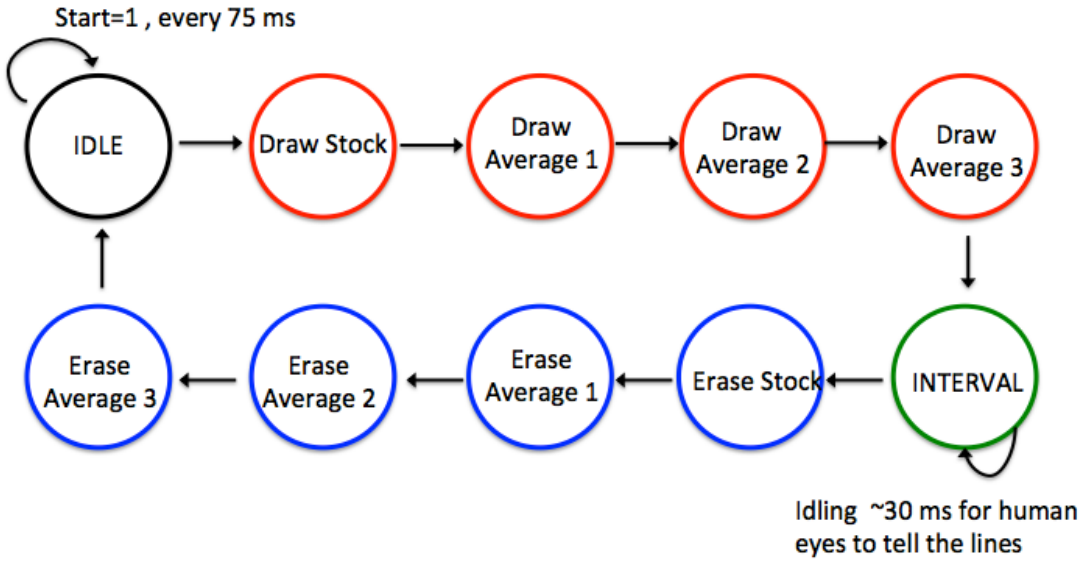


Figure 15: Simplified FSM for the control of Bresenham's algorithm

7. What We Learned/ Advice

One major tool we leaned on heavily was the use of SignalTap. We were able to compile the logic elements much faster than would occur in the regular setup because we could focus on just one module. By the end of the project, compile times were about 35 minutes for the entire project but sections for SignalTap could be compiled in a few minutes. Additionally, SignalTap made debugging much easier. Below is a sample timing diagram from SignalTap for a stock module.

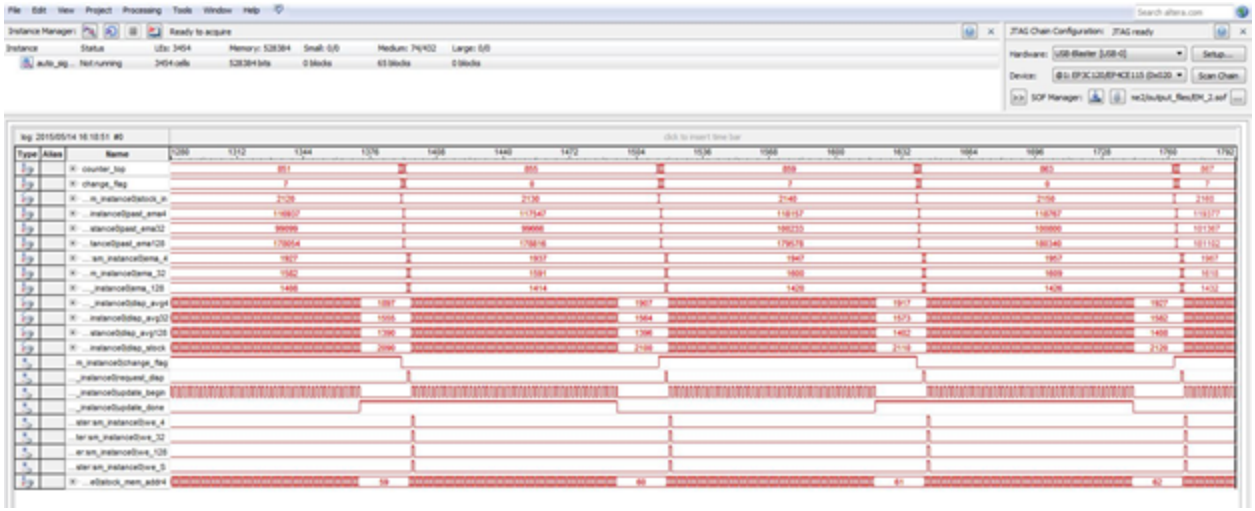


Figure 16: SignalTap for stock module example.

8. Group members roles in the project

Alexander: worked on a software code to generate stock market data that would be analyzed by the hardware, generated sprite matrices and implemented them on the monitor.

Nathan: implemented the software for parsing and sending data to the hardware, implemented the center control module, the stock modules, the range and scale modules, the binary to decimal module, and drawing the non line drawing version of the stocks.

Hang: integrated Bresenham's line drawing algorithm for stock display, simulated the short-term trading algorithm in software.

9. Future Work

The final version of the project implemented to analyze simultaneously seven stock symbols which correspond to 56% utilization of the logic and 3% memory usage of the Cyclone-V FPGA. Increasing the number of stocks that the hardware algorithm could analyze will mostly increase the logic usage, since each for each stock symbol we currently generate a new module. This potentially could further optimized by creating a central module which could handle a large amount of stock. On the other hand, this could come in the price of larger memory usage.

Additionally the data stream is currently read from a text file. In order to implement a faster implementation which could mimic real stock market analysis, the data needs to arrive from a faster and a secure protocol such as TCP/IP.

Lastly, we are extremely interested to see the efficiency of stock market analysis to generate profits. It would be interesting to perform a simulation of various stock to determine the effectiveness of this short-term approach, and whether the profits are actually realized.

10. Reference

[1] “How to Trade Short-Term (Day-Trade)." DailyFX. James Stanley
[2] http://en.wikipedia.org/wiki/Bresenham%27s_line_algorithm
[3] http://www.cs.columbia.edu/~sedwards/classes/2015/4840/lines.pdf

11. Code Appendix

Python Code to Generate Stock Market Data:

```
import numpy as np
from yahoo_finance import Share
from pprint import pprint
import matplotlib.pyplot as plt
import numpy as np
import socket, time

#stock_names =
['TFSC','TFSCR','TFSCU','TFSCW','PIH','FLWS','FCTY','FCCY','SRCE','VNET','TWOU','DGLD','JOBS','SIXD','EGHT','AVHI','SHLM','AAON','ABAX','ABY','ABGB','ABMD','AXAS','ACTG','ACHC','ACAD','ACST','AXDX','XLRN','ANCX','ARAY','ACRX','ACET','AKAO','ACHN','ACIW','ACNB','ACOR','ACFN','ACTS','ACPW','ATVI','ACTA','ACUR','ACXM','ADMS','ADMP','ADAP','ADUS','AEY','ADEP','ADMA','ADBE','ADTN','ADRO','AAAP','AEIS','AMD','ADXS','ADXSW','ADVS','MULT','YPRO','AEGR','AEGN','AEHR','AMTX','AEPI','AERI','AVAV','AEZS','AFMD','AFFX','AGEN','AGRX','AGYS','AGIO','AIRM','AIRT','ATSG','AMCN','AIXG','AKAM','AKBA','AKER','AKRX','ALSK','AMRI','ABDC','ADHD','ALDR','ALDX','ALXN','ALXA','ALCO','ALGN','ALIM','ALKS','ALGT','ALLB','AFOP','AIQ']

# Define function to read stock market data by dates ('2010-04-27','2015-05-5')
# and save as the results as an array of strings
def stockArray(name):
    stock1 = Share(name)
    x = stock1.get_historical('2010-04-27','2015-05-5');
    #print x

    #Volume , Symbol, Adj_Close, High, Low, Date, Close, Open
    close = []
```

```
    for n in range(0,len(x)):
        y = x[n]
        #print '****'
        #print 'Date', y['Date']
        #print 'Close', y['Close']
        close.insert(0,y['Close'])

    close = np.array(close)
    close = map(float,close)
    Times100 = np.array([100])
    close = close * Times100
    close = close.astype(int)

    return close

IBM = stockArray('IBM')
print len(IBM)
TEVA = stockArray('TEVA')
print len(TEVA)
INTC = stockArray('INTC')
print len(INTC)
ED = stockArray('ED')
print len(ED)
KO = stockArray('KO')
print len(KO)
DBD = stockArray('DBD')
print len(DBD)
PG = stockArray('PG')
print len(PG)
ABT = stockArray('ABT')
print len(PG)
AIG = stockArray('AIG')
print len(PG)
DOW = stockArray('DOW')
print len(PG)
HSY = stockArray('HSY')
print len(HSTY)
M = stockArray('M')
print len(M)

file = open("StockData.txt", "w")
for n in range(0,len(IBM)):
    file.write('IBM ' + str(IBM[n])+'\n')
```

```
    file.write('TEVA ' + str(TEVA[n])+'\n')
    file.write('INTC ' + str(INTC[n])+'\n')
    file.write('ED ' + str(ED[n])+'\n')
    file.write('KO ' + str(KO[n])+'\n')
    file.write('DBD ' + str(DBD[n])+'\n')
    file.write('PG ' + str(PG[n])+'\n')
    file.write('ABT ' + str(ABT[n])+'\n')
    file.write('AIG ' + str(AIG[n])+'\n')
    file.write('DOW ' + str(DOW[n])+'\n')
    file.write('M ' + str(M[n])+'\n')
file.close()
```

Python Code to Generate Character Sprites:

```
import numpy as np
import string
from matplotlib import pyplot as plt
import cv2
import scipy.io

#symbols to generate in Upper Case Symbols - 40 x 25 pixels

UpperCaseLetters = string.ascii_uppercase +':'+ ' 123456789'
file = open("Symbols_Data_Flip.txt", "a")
for l in range(0,len(UpperCaseLetters)):
    file.write('logic[39:0] S_'+UpperCaseLetters[l]+'[24:0];\n')

l=0
for l in range(0,len(UpperCaseLetters)):

    img = np.zeros((40,25,3), np.uint8)
    font = cv2.FONT_HERSHEY_SIMPLEX
    cv2.putText(img, UpperCaseLetters[l] ,(0,30), font, 1,(255,255,255),2)
    #plt.imshow(img)
    #plt.show()
    #cv2.imwrite('M.png',img)
    img = np.asarray(img)
    bin_img = np.zeros((40,25))
    for i in range(0,40):
        for j in range(0,25):
            if img[i][j][0] > 120:
                bin_img[i][j] = 1
```

```
            else:
                bin_img[i][j] = 0
    bin_img = bin_img[::-1] #flip array
    #Add letter matrices to a file
    file = open("Symbols_Data_Flip.txt", "a")
    rows = []
    for n in range(0,25):
        for x in range(0,40):
            rows.append(int(bin_img[x][n]))

        file.write("assign S_"+ UpperCaseLetters[l]  +"["+str(n)+"] = 40'b" +
str("".join(map(str,rows))) +';' '\n')
        rows = []

#symbols to generate in Lower Case Symbols - 20 x 15 pixels

UpperCaseLetters = string.ascii_uppercase +'.:'+' 123456789'
file = open("Symbols_Data_Flip.txt", "a")
for l in range(0,len(UpperCaseLetters)):
    file.write('logic[19:0] s_'+UpperCaseLetters[l]+'[14:0];\n')
l=0
n=0
x=0
for l in range(0,len(UpperCaseLetters)):
#for l in range(0,2):
    img = np.zeros((20,15,3), np.uint8)
    font = cv2.FONT_HERSHEY_SIMPLEX
    cv2.putText(img, UpperCaseLetters[l] ,(1,15), font, .6,(255,255,255),2)
    #plt.imshow(img)
    #plt.show()
    #cv2.imwrite('M.png',img)
    img = np.asarray(img)
    bin_img = np.zeros((20,15))
    for i in range(0,20):
        for j in range(0,12):
            if img[i][j][0] > 100:
                bin_img[i][j] = 1
            else:
                bin_img[i][j] = 0

    bin_img = bin_img[::-1] #flip array
    #Add letter matrices to a file
    file = open("Symbols_Data_Flip.txt", "a")
```

```
    rows = []
    for n in range(0,15):
        for x in range(0,20):
            rows.append(int(bin_img[x][n]))

        file.write("assign s_"+ UpperCaseLetters[l]  +"["+str(n)+"] = 20'b" +
str("".join(map(str,rows))) +';' '\n')
        rows = []
```

hello.c File for Software

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

#include <stdio.h>
#include <stdlib.h>
#include "vga_led.h"
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <string.h>
#include <math.h>
#include <unistd.h>
#include <pthread.h>

int vga_led_fd;
int shared_stock_id;
int restart;
int hard_range;
pthread_t ReadUser;
void *ReadUser_f(void *);

void disp_final(unsigned int write_read, unsigned int stock_id, unsigned int data)
{
```

```
        final_stock fs;
        fs.write_read = write_read;
        fs.stock_id = stock_id;
        fs.data = data;

        if(ioctl(vga_led_fd, VGA_LED_FINAL_STOCK, &fs)) {
                perror("ioctl(VGA_LED_STOCK_DISP) failed");
        }

}

int main()
{

        //--------------------------------

          vga_led_arg_t vla;
  int i = 0;
        int z = 0;
  int x;
        int y;

        int line_num = 0;
        char *p;
        char *token[5];
        int m = 0;

  int radius=3;
  int no_zero = 0;
        double temp;
        int temp1;
  int last_line;
  double stock[30000];


        char line [64];
        static const char filename[] = "/dev/vga_led";

  FILE *fp;
  fp = fopen("data.txt", "r");
        //printf("%f", atof("\n"));
  if ( fp != NULL )
        {
```

```
//printf("The data file has been opened successfully! \n");

while(fgets(line, sizeof(line), fp) != NULL)
{
//printf("Read in one line! \n");

                m = 0;
                line_num = line_num + 1;
                //printf("The line: %s\n", line);

                        //printf("A\n");
                        p = strtok(line, " ");
                        while(p != NULL)
                        {
                                //printf("B\n");
                                token[m++] = p;
                                p = strtok(NULL, " ");
                                //printf("C\n");
                        }
                        //printf("D\n");
                        //printf("The token[1]: %s\n", token[1]);
                        stock[z] = atof(token[1]);
                        temp1 = (int)(stock[z]);
                        //printf("Temp1: %d\n", temp1);
                        if(no_zero == 0 && temp1 == 0)
                        {
                                last_line = z;
                                no_zero = 1;
                                printf("Last Line: %d\n", last_line);
                        }
                        //printf("The value saved: %f\n", stock[z]);
                        //printf("E\n");
                        //avg1[z] = atof(token[2]);
                        //avg2[z] = atof(token[3]);
                        //avg3[z] = atof(token[4]);
                        z = z + 1;

                //printf("C\n");
                //stock[i] = atof(line);
                //last_line = i;
                i = i + 1;
}
```

```
        }
        last_line = z;
        //printf("Z: %d\n", last_line);
fclose(fp);

        //printf("Stock 0: %f\n", stock[0]);
        //printf("Stock 1: %f\n", stock[1]);

        //for number of stocks!

        int min0;
        int max0;
        int min1;
        int max1;
        int min2;
        int max2;
        int min3;
        int max3;
        int min4;
        int max4;
        int min5;
        int max5;
        int min6;
        int max6;
        int range[6];

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

        min0 = stock[0];
        max0 = stock[0];
        min1 = stock[1];
        max1 = stock[1];
        min2 = stock[2];
        max2 = stock[2];
        min3 = stock[3];
        max3 = stock[3];
        min4 = stock[4];
        max4 = stock[4];
        min5 = stock[5];
        max5 = stock[5];
```

```
min6 = stock[6];
max6 = stock[6];

i = 0;
while (i < last_line)
{
	if (i%7 == 0)
	{
		if (stock[i] > max0)
		{

			max0 = stock[i];
		}
		if (stock[i] < min0)
		{
			min0 = stock[i];
		}
	}
	if (i%7 == 1)
	{
		if (stock[i] > max1)
		{
			max1 = stock[i];
		}
		if (stock[i] < min1)
		{
			min1 = stock[i];
		}
	}
	if (i%7 == 2)
	{
		if (stock[i] > max2)
		{
			max2 = stock[i];
		}
		if (stock[i] < min2)
		{
			min2 = stock[i];
		}
	}
	if (i%7 == 3)
	{
		if (stock[i] > max3)
```

```
                {
                        max3 = stock[i];
                }
                if (stock[i] < min3)
                {
                        min3 = stock[i];
                }
        }
        if (i%7 == 4)
        {
                if (stock[i] > max4)
                {
                        max4 = stock[i];
                }
                if (stock[i] < min4)
                {
                        min4 = stock[i];
                }
        }
        if (i%7 == 5)
        {
                if (stock[i] > max5)
                {
                        max5 = stock[i];
                }
                if (stock[i] < min5)
                {
                        min5 = stock[i];
                }
        }
        if (i%7 == 6)
        {
                if (stock[i] > max6)
                {
                        max6 = stock[i];
                }
                if (stock[i] < min6)
                {
                        min6 = stock[i];
                }
        }
        i = i + 1;
}
```

```
range[0] = max0 - min0;
range[1] = max1 - min1;
range[2] = max2 - min2;
range[3] = max3 - min3;
range[4] = max4 - min4;
range[5] = max5 - min5;
range[6] = max6 - min6;
//printf("Max: %d\n", max0);
//printf("Min: %d\n", min0);
//printf("Range: %d\n", range[0]);
int test = (int)(range[0]/2) + 65536;
//printf("Test data: %d\n", test);
i = 0;
int j = 0;
//FINAL STUFF
int write_read = 0;
int stock_id = 0;
int data = 0;

pthread_create(&ReadUser, NULL, ReadUser_f, NULL);
while (1)
{
                i = 0;
                j = 0;
                while (i < last_line)
                {
                        if (restart == 1)
                        {
                                restart = 0;
                                i = 0;
                                j = 0;
                                while (j < 401)
                                {
                                        data = 0;
                                        write_read = 0;
                                        stock_id = 0;
                                  disp_final(write_read, stock_id, data);
                                        stock_id = 1;
                                  disp_final(write_read, stock_id, data);
                                        stock_id = 2;
                                  disp_final(write_read, stock_id, data);
                                        stock_id = 3;
                                  disp_final(write_read, stock_id, data);
```

```
                stock_id = 4;
            disp_final(write_read, stock_id, data);
                stock_id = 5;
            disp_final(write_read, stock_id, data);
                stock_id = 6;
            disp_final(write_read, stock_id, data);

                j = j + 1;
        }
}

if (i%7 == 0)
{
        data = stock[i];
        write_read = 0;
        stock_id = 0;
        disp_final(write_read, stock_id, data);
        //send, then wait shortly
        usleep(1000);
}
if (i%7 == 1)
{
        data = stock[i];
        write_read = 0;
        stock_id = 1;
        disp_final(write_read, stock_id, data);
        usleep(1000);
}
if (i%7 == 2)
{
        data = stock[i];
        write_read = 0;
        stock_id = 2;
        disp_final(write_read, stock_id, data);
        usleep(1000);
}
if (i%7 == 3)
{
        data = stock[i];
        write_read = 0;
        stock_id = 3;
        disp_final(write_read, stock_id, data);
        usleep(1000);
```

```
        }
        if (i%7 == 4)
        {
                data = stock[i];
                write_read = 0;
                stock_id = 4;
                disp_final(write_read, stock_id, data);
                usleep(1000);
        }
        if (i%7 == 5)
        {
                data = stock[i];
                write_read = 0;
                stock_id = 5;
                disp_final(write_read, stock_id, data);
                usleep(1000);
        }

        if (i%7 == 6)
        {
                data = stock[i];
                write_read = 0;
                stock_id = 6;
                disp_final(write_read, stock_id, data);
                usleep(1000);

                stock_id = shared_stock_id;
                if (hard_range == 0)
                {
                        data = 0;
                }
                if (hard_range == 1)
                {
                        data = range[stock_id]/2 + 65536;
                }
                write_read = 1;
                disp_final(write_read, stock_id, data);
                usleep(65000);
        }


        i = i + 1;
}
```

```
        }



  //printf("VGA LED Userspace program terminating\n");
        pthread_exit(NULL);
  return 0;
}



void *ReadUser_f(void *ignored)
{

        char usr_input [10];
        char IBM[] = "IBM";
        char TEVA[] = "TEVA";
        char INTC[] = "INTC";
        char ED[] = "ED";
        char KO[] = "KO";
        char DBD[] = "DBD";
        char PG[] = "PG";
        char Restart[] = "Restart";
        char Fix_Range[] = "Fix";
        char Flex_Range[] = "Flex";
        printf ("The stocks being tracked are: IBM, TEVA, INTC, ED, KO, DBD, and PG. \n");
        restart = 1;
        while (1)
 {
                printf ("Enter stock name or command: ");
                scanf ("%s", usr_input);
                if (strcmp(usr_input, IBM) == 0)
                {
                        shared_stock_id = 0;
                }
                if (strcmp(usr_input, TEVA) == 0)
                {
                        shared_stock_id = 1;
                }
                if (strcmp(usr_input, INTC) == 0)
                {
```

```
                shared_stock_id = 2;
            }
            if (strcmp(usr_input, ED) == 0)
            {
                shared_stock_id = 3;
            }
            if (strcmp(usr_input, KO) == 0)
            {
                shared_stock_id = 4;
            }
            if (strcmp(usr_input, DBD) == 0)
            {
                shared_stock_id = 5;
            }
            if (strcmp(usr_input, PG) == 0)
            {
                shared_stock_id = 6;
            }
            if (strcmp(usr_input, Restart) == 0)
            {
                restart = 1;
            }
            if (strcmp(usr_input, Fix_Range) == 0)
            {
                hard_range = 1;
            }
            if (strcmp(usr_input, Flex_Range) == 0)
            {
                hard_range = 0;
            }
        }
        pthread_exit(NULL);
}
```

vga_led.c

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

#define DRIVER_NAME "vga_led"

/*
 * Information about our device
 */
struct vga_led_dev {
	struct resource res; /* Resource: our registers */
	void __iomem *virtbase; /* Where registers can be accessed in memory */
	u8 segments[VGA_LED_DIGITS];
} dev;

/*
 * Write segments of a single digit
 * Assumes digit is in range and the device information has been set up
 */
static void write_digit(int digit, u8 segments)
{
	iowrite8(segments, dev.virtbase + digit);
	dev.segments[digit] = segments;
}

//HEEEEERRRREEEEEEE
static void write_stock(unsigned int price, unsigned int address, unsigned int change_check,
unsigned int display, unsigned int we)
```

```
{
	iowrite32(( we << 27 | display << 26 |change_check << 25 | address << 20 | price),
dev.virtbase); //DONE



}
//HEERRREEEEEEE
static void disp_stock(unsigned int print_bit, unsigned int stock_id, unsigned int graph_id,
unsigned int x_graph, unsigned int y_graph)
{
	iowrite32((print_bit << 31 | stock_id << 29 | graph_id << 27 | x_graph << 10 | y_graph),
dev.virtbase);
}

static void disp_final(unsigned int write_read, unsigned int stock_id, unsigned int data)
{
	iowrite32((write_read << 31 | stock_id << 17 | data), dev.virtbase);
}



/*
 * Handle ioctl() calls from userspace:
 * Read or write the segments on single digits.
 * Note extensive error checking of arguments
 */
static long vga_led_ioctl(struct file *f, unsigned int cmd, unsigned long arg)
{
	vga_led_arg_t vla;
	stock_market sm;
	stock_disp_struct sd;
	final_stock fs;

	switch (cmd) {
	case VGA_LED_WRITE_DIGIT:
		if (copy_from_user(&vla, (vga_led_arg_t *) arg,
					sizeof(vga_led_arg_t)))
			return -EACCES;
		if (vla.digit > 8)
			return -EINVAL;
		write_digit(vla.digit, vla.segments);
		break;
```

```
	case VGA_LED_READ_DIGIT:
		if (copy_from_user(&vla, (vga_led_arg_t *) arg,
				   sizeof(vga_led_arg_t)))
			return -EACCES;
		if (vla.digit > 8)
			return -EINVAL;
		vla.segments = dev.segments[vla.digit];
		if (copy_to_user((vga_led_arg_t *) arg, &vla,
				 sizeof(vga_led_arg_t)))
			return -EACCES;
		break;
	//HEEEEERRREEEE
	case VGA_LED_STOCK_MARKET:
		if (copy_from_user(&sm, (stock_market *) arg, sizeof(stock_market)))
			return -EACCES;
		write_stock(sm.price, sm.address, sm.change_check, sm.display, sm.we);
//others!!!
		break;
	case VGA_LED_STOCK_DISP:
		if(copy_from_user(&sd, (stock_disp_struct *) arg, sizeof(stock_disp_struct)))
			return -EACCES;
		disp_stock(sd.print_bit, sd.stock_id, sd.graph_id, sd.x_graph, sd.y_graph);
		break;
	case VGA_LED_FINAL_STOCK:
		if(copy_from_user(&fs, (final_stock *) arg, sizeof(final_stock)))
			return -EACCES;
		disp_final(fs.write_read, fs.stock_id, fs.data);
		break;
	//HEEERREEEEE
	default:
		return -EINVAL;
	}

	return 0;
}

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

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

/*
 * Initialization code: get resources (registers) and display
 * a welcome message
 */
static int __init vga_led_probe(struct platform_device *pdev)
{
	static unsigned char welcome_message[VGA_LED_DIGITS] = {
		0x3E, 0x7D, 0x77, 0x08, 0x38, 0x79, 0x5E, 0x00};
	int i, ret;

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

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

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

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

	/* Display a welcome message */
	for (i = 0; i < VGA_LED_DIGITS; i++)
```

```
                write_digit(i, welcome_message[i]);

        return 0;

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

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

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

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

/* Called when the module is loaded: set things up */
static int __init vga_led_init(void)
{
        pr_info(DRIVER_NAME ": init\n");
```

```
	return platform_driver_probe(&vga_led_driver, vga_led_probe);
}

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

module_init(vga_led_init);
module_exit(vga_led_exit);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Stephen A. Edwards, Columbia University");
MODULE_DESCRIPTION("VGA 7-segment LED Emulator");
```

vga_led.h

```
#ifndef _VGA_LED_H
#define _VGA_LED_H

#include <linux/ioctl.h>

#define VGA_LED_DIGITS 8

typedef struct {
  unsigned char digit;    /* 0, 1, .. , VGA_LED_DIGITS - 1 */
  unsigned char segments; /* LSB is segment a, MSB is decimal point */
} vga_led_arg_t;

//HEEEEEEREEEEE
typedef struct {
	unsigned int price;
	unsigned int address;
	unsigned int change_check;
	unsigned int display;
	unsigned int we;
} stock_market;
//HEEEERRREEE
```

```
typedef struct {
        unsigned int print_bit;
        unsigned int stock_id;
        unsigned int graph_id;
        unsigned int x_graph;
        unsigned int y_graph;
} stock_disp_struct;

typedef struct {
        unsigned int write_read;
        unsigned int stock_id;
        unsigned int data; //either stock value or time!!
} final_stock;



#define VGA_LED_MAGIC 'q'

/* ioctls and their arguments */
#define VGA_LED_WRITE_DIGIT _IOW(VGA_LED_MAGIC, 1, vga_led_arg_t *)
#define VGA_LED_READ_DIGIT  _IOWR(VGA_LED_MAGIC, 2, vga_led_arg_t *)
#define VGA_LED_STOCK_MARKET _IOW(VGA_LED_MAGIC, 3, vga_led_arg_t *)
#define VGA_LED_STOCK_DISP _IOW(VGA_LED_MAGIC, 4, vga_led_arg_t *)
#define VGA_LED_FINAL_STOCK _IOW(VGA_LED_MAGIC, 5, vga_led_arg_t *)

#endif
```

Makefile

```
ifneq (${KERNELRELEASE},)

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

else

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

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

```
default: module hello

hello:
		gcc -o hello hello.c -lpthread

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

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

endif
```

VGA_LED_Emulator.sv

```
module VGA_LED_Emulator(
 input logic          clk50, reset,
 input logic [9:0] x_coord,
 input logic [9:0] y_coord,
 input logic [9:0] radius,
 //---------------------------------------
 input logic [9:0] disp_stock_x [399:0],
 input logic [9:0] disp_stock_a1 [399:0],
 input logic [9:0] disp_stock_a2 [399:0],
 input logic [9:0] disp_stock_a3 [399:0],
 input logic send_to_vga,
 input logic [6:0] stock_name,
 input logic [1:0] bhs,
 //---------------------------------------
input logic [3:0] S_hundredthousand,
input logic [3:0] S_tenthousand,
input logic [3:0] S_thousand,
input logic [3:0] S_hundred,
input logic [3:0] S_ten,
input logic [3:0] S_one,

input logic [3:0] E4_hundredthousand,
input logic [3:0] E4_tenthousand,
input logic [3:0] E4_thousand,
input logic [3:0] E4_hundred,
```

```
input logic [3:0] E4_ten,
input logic [3:0] E4_one,

input logic [3:0] E32_hundredthousand,
input logic [3:0] E32_tenthousand,
input logic [3:0] E32_thousand,
input logic [3:0] E32_hundred,
input logic [3:0] E32_ten,
input logic [3:0] E32_one,

input logic [3:0] E128_hundredthousand,
input logic [3:0] E128_tenthousand,
input logic [3:0] E128_thousand,
input logic [3:0] E128_hundred,
input logic [3:0] E128_ten,
input logic [3:0] E128_one,

 input logic [31:0] display,
 output logic [7:0] VGA_R, VGA_G, VGA_B,
 output logic       VGA_CLK, VGA_HS, VGA_VS, VGA_BLANK_n, VGA_SYNC_n);

/*
 * 640 X 480 VGA timing for a 50 MHz clock: one pixel every other cycle
 *
 * HCOUNT 1599 0            1279       1599 0
 *           _________________           _________
 * ____________|    Video      |_____________|  Video
 *
 *
 * |SYNC| BP |<-- HACTIVE -->|FP|SYNC| BP |<-- HACTIVE
 *       _______________________     _____________
 * |____|       VGA_HS          |____|
 */
  // Parameters for hcount
  parameter HACTIVE      = 11'd 1280,
            HFRONT_PORCH = 11'd 32,
            HSYNC        = 11'd 192,
            HBACK_PORCH  = 11'd 96,
            HTOTAL       = HACTIVE + HFRONT_PORCH + HSYNC + HBACK_PORCH; // 1600

  // Parameters for vcount
  parameter VACTIVE      = 10'd 480,
            VFRONT_PORCH = 10'd 10,
```

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

logic [10:0]                    hcount; // Horizontal counter
                                // Hcount[10:1] indicates pixel column (0-639)
logic                       endOfLine;

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

assign endOfLine = hcount == HTOTAL - 1;

// Vertical counter
logic [9:0]                     vcount;
logic                       endOfField;

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

assign endOfField = vcount == VTOTAL - 1;

// Horizontal sync: from 0x520 to 0x5DF (0x57F)
// 101 0010 0000 to 101 1101 1111
assign VGA_HS = !( (hcount[10:8] == 3'b101) & !(hcount[7:5] == 3'b111));
assign VGA_VS = !( vcount[9:1] == (VACTIVE + VFRONT_PORCH) / 2);

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

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

/* VGA_CLK is 25 MHz
 *
 * clk50    __|  |__|  |__|
```

```
 *
 *           _____      __
 * hcount[0]__|     |_____|
 */
assign VGA_CLK = hcount[0]; // 25 MHz clock: pixel latched on rising edge

        logic write_okay_r; //whether we should draw this pixel
        logic write_okay_b;
        logic write_okay_g;
        logic write_okay_o;
        logic [31:0] full_x;
        logic [31:0] full_y;
        logic [31:0] full_r;
        logic [31:0] multi_x;
        logic [31:0] multi_y;
        logic [31:0] multi_tot;
        logic [31:0] multi_rad;
        assign full_x[31:10] = 22'b0;
        assign full_y[31:10] = 22'b0;
        assign full_r[31:10] = 22'b0;


//Write okay conditions for buy/hold/sell letters
        logic write_okay_sn1;
        logic [9:0] x_letter1;
        logic [9:0] y_letter1;
        assign x_letter1 = 500;
        assign y_letter1 = 100;

        logic write_okay_sn2;
        logic [9:0] x_letter2;
        logic [9:0] y_letter2;
        assign x_letter2 = 525;
        assign y_letter2 = 100;

        logic write_okay_sn3;
        logic [9:0] x_letter3;
        logic [9:0] y_letter3;
        assign x_letter3 = 550;
        assign y_letter3 = 100;

        logic write_okay_sn4;
        logic [9:0] x_letter4;
```

```
	logic [9:0] y_letter4;
	assign x_letter4 = 575;
	assign y_letter4 = 100;

	logic sn_pos1;
	logic sn_pos2;
	logic sn_pos3;
	logic sn_pos4;

	//////////////////////
	// Draw Stock Name //
	//////////////////////

	// Write in location only ones
	assign sn_pos1 = ((hcount[10:1] < 525)&&(hcount[10:1] > 500)&&(vcount[9:0] >
100)&&(vcount[9:0] < 140));
	assign sn_pos2 = ((hcount[10:1] < 550)&&(hcount[10:1] > 525)&&(vcount[9:0] >
100)&&(vcount[9:0] < 140));
	assign sn_pos3 = ((hcount[10:1] < 575)&&(hcount[10:1] > 550)&&(vcount[9:0] >
100)&&(vcount[9:0] < 140));
	assign sn_pos4 = ((hcount[10:1] < 600)&&(hcount[10:1] > 575)&&(vcount[9:0] >
100)&&(vcount[9:0] < 140));

	always_comb begin


	if (stock_name == 7'b01) begin // TEVA 1
			write_okay_sn1 = ((S_T[hcount[10:1] - x_letter1][vcount[9:0] -
y_letter1]));
			write_okay_sn2 = ((S_E[hcount[10:1] - x_letter2][vcount[9:0] -
y_letter2]));
			write_okay_sn3 = ((S_V[hcount[10:1] - x_letter3][vcount[9:0] -
y_letter3]));
			write_okay_sn4 = ((S_A[hcount[10:1] - x_letter4][vcount[9:0] -
y_letter4]));
		end

		else if (stock_name == 7'b00) begin // IBM 0
			write_okay_sn1 = ((S_I[hcount[10:1] - x_letter1][vcount[9:0] -
y_letter1]));
			write_okay_sn2 = ((S_B[hcount[10:1] - x_letter2][vcount[9:0] -
y_letter2]));
```

```
                write_okay_sn3 = ((S_M[hcount[10:1] - x_letter3][vcount[9:0] -
y_letter3]));
                write_okay_sn4 = ((S_blank[hcount[10:1] - x_letter3][vcount[9:0] -
y_letter3]));

            end

            else if (stock_name == 7'b10) begin // INTC 2
                write_okay_sn1 = ((S_I[hcount[10:1] - x_letter1][vcount[9:0] -
y_letter1]));
                write_okay_sn2 = ((S_N[hcount[10:1] - x_letter2][vcount[9:0] -
y_letter2]));
                write_okay_sn3 = ((S_T[hcount[10:1] - x_letter3][vcount[9:0] -
y_letter3]));
                write_okay_sn4 = ((S_C[hcount[10:1] - x_letter4][vcount[9:0] -
y_letter4]));
            end

            else if (stock_name == 7'b11) begin // ED 3
                write_okay_sn1 = ((S_E[hcount[10:1] - x_letter1][vcount[9:0] -
y_letter1]));
                write_okay_sn2 = ((S_D[hcount[10:1] - x_letter2][vcount[9:0] -
y_letter2]));
                write_okay_sn3 = ((S_blank[hcount[10:1] - x_letter3][vcount[9:0] -
y_letter3]));
                write_okay_sn4 = ((S_blank[hcount[10:1] - x_letter4][vcount[9:0] -
y_letter4]));
            end

            else if (stock_name == 7'b100) begin // KO 4
                write_okay_sn1 = ((S_K[hcount[10:1] - x_letter1][vcount[9:0] -
y_letter1]));
                write_okay_sn2 = ((S_O[hcount[10:1] - x_letter2][vcount[9:0] -
y_letter2]));
                write_okay_sn3 = ((S_blank[hcount[10:1] - x_letter3][vcount[9:0] -
y_letter3]));
                write_okay_sn4 = ((S_blank[hcount[10:1] - x_letter4][vcount[9:0] -
y_letter4]));
            end

            else if (stock_name == 7'b101) begin // BDB 5
                write_okay_sn1 = ((S_D[hcount[10:1] - x_letter1][vcount[9:0] -
y_letter1]));
```

```
                write_okay_sn2 = ((S_B[hcount[10:1] - x_letter2][vcount[9:0] -
y_letter2]));
                write_okay_sn3 = ((S_D[hcount[10:1] - x_letter3][vcount[9:0] -
y_letter3]));
                write_okay_sn4 = ((S_blank[hcount[10:1] - x_letter4][vcount[9:0] -
y_letter4]));
            end

            else if (stock_name == 7'b110) begin // PG
                write_okay_sn1 = ((S_P[hcount[10:1] - x_letter1][vcount[9:0] -
y_letter1]));
                write_okay_sn2 = ((S_G[hcount[10:1] - x_letter2][vcount[9:0] -
y_letter2]));
                write_okay_sn3 = ((S_blank[hcount[10:1] - x_letter3][vcount[9:0] -
y_letter3]));
                write_okay_sn4 = ((S_blank[hcount[10:1] - x_letter4][vcount[9:0] -
y_letter4]));
            end

            else begin
                write_okay_sn1 = ((S_blank[hcount[10:1] - x_letter1][vcount[9:0] -
y_letter1]));
                write_okay_sn2 = ((S_blank[hcount[10:1] - x_letter2][vcount[9:0] -
y_letter2]));
                write_okay_sn3 = ((S_blank[hcount[10:1] - x_letter3][vcount[9:0] -
y_letter3]));
                write_okay_sn4 = ((S_blank[hcount[10:1] - x_letter4][vcount[9:0] -
y_letter4]));
            end
        end

        //
        //
        //For prices on y axis
        logic write_okay_h1;
        logic write_okay_h2;
        logic write_okay_h3;
        logic write_okay_l1;
        logic write_okay_l2;
        logic write_okay_l3;

        logic left_h1;
        logic left_h2;
```

```
        logic left_h3;
        logic left_l1;
        logic left_l2;
        logic left_l3;

        assign left_h1 = ((hcount[10:1] <= 35)&&(hcount[10:1] >= 20)&&(vcount[9:0] >=
20)&&(vcount[9:0] < 40));
        assign left_h2 = ((hcount[10:1] <= 55)&&(hcount[10:1] >= 40)&&(vcount[9:0] >=
20)&&(vcount[9:0] < 40));
        assign left_h3 = ((hcount[10:1] <= 75)&&(hcount[10:1] >= 60)&&(vcount[9:0] >=
20)&&(vcount[9:0] < 40));
        assign left_l1 = ((hcount[10:1] <= 35)&&(hcount[10:1] >= 20)&&(vcount[9:0] >=
360)&&(vcount[9:0] < 380));
        assign left_l2 = ((hcount[10:1] <= 55)&&(hcount[10:1] >= 40)&&(vcount[9:0] >=
360)&&(vcount[9:0] < 380));
        assign left_l3 = ((hcount[10:1] <= 75)&&(hcount[10:1] >= 60)&&(vcount[9:0] >=
360)&&(vcount[9:0] < 380));

        logic [9:0] lx1;
        logic [9:0] ly1;
        logic [9:0] lx2;
        logic [9:0] ly2;
        logic [9:0] lx3;
        logic [9:0] ly3;

        logic [9:0] hx1;
        logic [9:0] hy1;
        logic [9:0] hx2;
        logic [9:0] hy2;
        logic [9:0] hx3;
        logic [9:0] hy3;

        assign lx1 = 20;
        assign lx2 = 40;
        assign lx3 = 60;
        assign hx1 = 20;
        assign hx2 = 40;
        assign hx3= 60;

        assign hy1 = 20;
        assign hy2 = 20;
        assign hy3 = 20;
        assign ly1 = 360;
```

```
assign ly2 = 360;
assign ly3 = 360;

always_comb begin
	if (E32_tenthousand == 0) begin
		write_okay_h1 = ((s_O[hcount[10:1] - hx1][vcount[9:0] - hy1]));
	end
	else if (E32_tenthousand == 1) begin
		write_okay_h1 = ((s_1[hcount[10:1] - hx1][vcount[9:0] - hy1]));
	end
	else if (E32_tenthousand == 2) begin
		write_okay_h1 = ((s_2[hcount[10:1] - hx1][vcount[9:0] - hy1]));
	end
	else if (E32_tenthousand == 3) begin
		write_okay_h1 = ((s_3[hcount[10:1] - hx1][vcount[9:0] - hy1]));
	end
	else if (E32_tenthousand == 4) begin
		write_okay_h1 = ((s_4[hcount[10:1] - hx1][vcount[9:0] - hy1]));
	end
	else if (E32_tenthousand == 5) begin
		write_okay_h1 = ((s_5[hcount[10:1] - hx1][vcount[9:0] - hy1]));
	end
	else if (E32_tenthousand == 6) begin
		write_okay_h1 = ((s_6[hcount[10:1] - hx1][vcount[9:0] - hy1]));
	end
	else if (E32_tenthousand == 7) begin
		write_okay_h1 = ((s_7[hcount[10:1] - hx1][vcount[9:0] - hy1]));
	end
	else if (E32_tenthousand == 8) begin
		write_okay_h1 = ((s_8[hcount[10:1] - hx1][vcount[9:0] - hy1]));
	end
	else if (E32_tenthousand == 9) begin
		write_okay_h1 = ((s_9[hcount[10:1] - hx1][vcount[9:0] - hy1]));
	end
	else begin
		write_okay_h1 = ((s_blank[hcount[10:1] - hx1][vcount[9:0] - hy1]));
	end
end

always_comb begin
	if (E32_thousand == 0) begin
		write_okay_h2 = ((s_O[hcount[10:1] - hx2][vcount[9:0] - hy2]));
	end
```

```
        else if (E32_thousand == 1) begin
                write_okay_h2 = ((s_1[hcount[10:1] - hx2][vcount[9:0] - hy2]));
        end
        else if (E32_thousand == 2) begin
                write_okay_h2 = ((s_2[hcount[10:1] - hx2][vcount[9:0] - hy2]));
        end
        else if (E32_thousand == 3) begin
                write_okay_h2 = ((s_3[hcount[10:1] - hx2][vcount[9:0] - hy2]));
        end
        else if (E32_thousand == 4) begin
                write_okay_h2 = ((s_4[hcount[10:1] - hx2][vcount[9:0] - hy2]));
        end
        else if (E32_thousand == 5) begin
                write_okay_h2 = ((s_5[hcount[10:1] - hx2][vcount[9:0] - hy2]));
        end
        else if (E32_thousand == 6) begin
                write_okay_h2 = ((s_6[hcount[10:1] - hx2][vcount[9:0] - hy2]));
        end
        else if (E32_thousand == 7) begin
                write_okay_h2 = ((s_7[hcount[10:1] - hx2][vcount[9:0] - hy2]));
        end
        else if (E32_thousand == 8) begin
                write_okay_h2 = ((s_8[hcount[10:1] - hx2][vcount[9:0] - hy2]));
        end
        else if (E32_thousand == 9) begin
                write_okay_h2 = ((s_9[hcount[10:1] - hx2][vcount[9:0] - hy2]));
        end
        else begin
                write_okay_h2 = ((s_blank[hcount[10:1] - hx2][vcount[9:0] - hy2]));
        end
end

always_comb begin
        if (E32_hundred == 0) begin
                write_okay_h3 = ((s_O[hcount[10:1] - hx3][vcount[9:0] - hy3]));
        end
        else if (E32_hundred == 1) begin
                write_okay_h3 = ((s_1[hcount[10:1] - hx3][vcount[9:0] - hy3]));
        end
        else if (E32_hundred == 2) begin
                write_okay_h3 = ((s_2[hcount[10:1] - hx3][vcount[9:0] - hy3]));
        end
        else if (E32_hundred == 3) begin
```

```
            write_okay_h3 = ((s_3[hcount[10:1] - hx3][vcount[9:0] - hy3]));
        end
        else if (E32_hundred == 4) begin
            write_okay_h3 = ((s_4[hcount[10:1] - hx3][vcount[9:0] - hy3]));
        end
        else if (E32_hundred == 5) begin
            write_okay_h3 = ((s_5[hcount[10:1] - hx3][vcount[9:0] - hy3]));
        end
        else if (E32_hundred == 6) begin
            write_okay_h3 = ((s_6[hcount[10:1] - hx3][vcount[9:0] - hy3]));
        end
        else if (E32_hundred == 7) begin
            write_okay_h3 = ((s_7[hcount[10:1] - hx3][vcount[9:0] - hy3]));
        end
        else if (E32_hundred == 8) begin
            write_okay_h3 = ((s_8[hcount[10:1] - hx3][vcount[9:0] - hy3]));
        end
        else if (E32_hundred == 9) begin
            write_okay_h3 = ((s_9[hcount[10:1] - hx3][vcount[9:0] - hy3]));
        end
        else begin
            write_okay_h3 = ((s_blank[hcount[10:1] - hx3][vcount[9:0] - hy3]));
        end
end
//
        always_comb begin
        if (E4_tenthousand == 0) begin
            write_okay_l1 = ((s_O[hcount[10:1] - lx1][vcount[9:0] - ly1]));
        end
        else if (E4_tenthousand == 1) begin
            write_okay_l1 = ((s_1[hcount[10:1] - lx1][vcount[9:0] - ly1]));
        end
        else if (E4_tenthousand == 2) begin
            write_okay_l1 = ((s_2[hcount[10:1] - lx1][vcount[9:0] - ly1]));
        end
        else if (E4_tenthousand == 3) begin
            write_okay_l1 = ((s_3[hcount[10:1] - lx1][vcount[9:0] - ly1]));
        end
        else if (E4_tenthousand == 4) begin
            write_okay_l1 = ((s_4[hcount[10:1] - lx1][vcount[9:0] - ly1]));
        end
        else if (E4_tenthousand == 5) begin
            write_okay_l1 = ((s_5[hcount[10:1] - lx1][vcount[9:0] - ly1]));
```

```
        end
        else if (E4_tenthousand == 6) begin
                write_okay_l1 = ((s_6[hcount[10:1] - lx1][vcount[9:0] - ly1]));
        end
        else if (E4_tenthousand == 7) begin
                write_okay_l1 = ((s_7[hcount[10:1] - lx1][vcount[9:0] - ly1]));
        end
        else if (E4_tenthousand == 8) begin
                write_okay_l1 = ((s_8[hcount[10:1] - lx1][vcount[9:0] - ly1]));
        end
        else if (E4_tenthousand == 9) begin
                write_okay_l1 = ((s_9[hcount[10:1] - lx1][vcount[9:0] - ly1]));
        end
        else begin
                write_okay_l1 = ((s_blank[hcount[10:1] - lx1][vcount[9:0] - ly1]));
        end
end

always_comb begin
        if (E4_thousand == 0) begin
                write_okay_l2 = ((s_O[hcount[10:1] - lx2][vcount[9:0] - ly2]));
        end
        else if (E4_thousand == 1) begin
                write_okay_l2 = ((s_1[hcount[10:1] - lx2][vcount[9:0] - ly2]));
        end
        else if (E4_thousand == 2) begin
                write_okay_l2 = ((s_2[hcount[10:1] - lx2][vcount[9:0] - ly2]));
        end
        else if (E4_thousand == 3) begin
                write_okay_l2 = ((s_3[hcount[10:1] - lx2][vcount[9:0] - ly2]));
        end
        else if (E4_thousand == 4) begin
                write_okay_l2 = ((s_4[hcount[10:1] - lx2][vcount[9:0] - ly2]));
        end
        else if (E4_thousand == 5) begin
                write_okay_l2 = ((s_5[hcount[10:1] - lx2][vcount[9:0] - ly2]));
        end
        else if (E4_thousand == 6) begin
                write_okay_l2 = ((s_6[hcount[10:1] - lx2][vcount[9:0] - ly2]));
        end
        else if (E4_thousand == 7) begin
                write_okay_l2 = ((s_7[hcount[10:1] - lx2][vcount[9:0] - ly2]));
        end
```

```
        else if (E4_thousand == 8) begin
                write_okay_l2 = ((s_8[hcount[10:1] - lx2][vcount[9:0] - ly2]));
        end
        else if (E4_thousand == 9) begin
                write_okay_l2 = ((s_9[hcount[10:1] - lx2][vcount[9:0] - ly2]));
        end
        else begin
                write_okay_l2 = ((s_blank[hcount[10:1] - lx2][vcount[9:0] - ly2]));
        end
end

always_comb begin
        if (E4_hundred == 0) begin
                write_okay_l3 = ((s_O[hcount[10:1] - lx3][vcount[9:0] - ly3]));
        end
        else if (E4_hundred == 1) begin
                write_okay_l3 = ((s_1[hcount[10:1] - lx3][vcount[9:0] - ly3]));
        end
        else if (E4_hundred == 2) begin
                write_okay_l3 = ((s_2[hcount[10:1] - lx3][vcount[9:0] - ly3]));
        end
        else if (E4_hundred == 3) begin
                write_okay_l3 = ((s_3[hcount[10:1] - lx3][vcount[9:0] - ly3]));
        end
        else if (E4_hundred == 4) begin
                write_okay_l3 = ((s_4[hcount[10:1] - lx3][vcount[9:0] - ly3]));
        end
        else if (E4_hundred == 5) begin
                write_okay_l3 = ((s_5[hcount[10:1] - lx3][vcount[9:0] - ly3]));
        end
        else if (E4_hundred == 6) begin
                write_okay_l3 = ((s_6[hcount[10:1] - lx3][vcount[9:0] - ly3]));
        end
        else if (E4_hundred == 7) begin
                write_okay_l3 = ((s_7[hcount[10:1] - lx3][vcount[9:0] - ly3]));
        end
        else if (E4_hundred == 8) begin
                write_okay_l3 = ((s_8[hcount[10:1] - lx3][vcount[9:0] - ly3]));
        end
        else if (E4_hundred == 9) begin
                write_okay_l3 = ((s_9[hcount[10:1] - lx3][vcount[9:0] - ly3]));
        end
        else begin
```

```
                write_okay_l3 = ((s_blank[hcount[10:1] - lx3][vcount[9:0] - ly3]));
        end
end
//
//
//
//For prices under graph
logic write_okay_sp1;
logic write_okay_sp2;
logic write_okay_sp3;
logic write_okay_sp4;
logic write_okay_sp5;
logic write_okay_sp6;
logic write_okay_sp7;
logic write_okay_sp8;
logic write_okay_sp9;
logic write_okay_sp10;
logic write_okay_sp11;
logic write_okay_sp12;
logic write_okay_sp13;

logic st_pos1;
logic st_pos2;
logic st_pos3;
logic st_pos4;
logic st_pos5;
logic st_pos6;
logic st_pos7;
logic st_pos8;
logic st_pos9;
logic st_pos10;
logic st_pos11;
logic st_pos12;
logic st_pos13;
//
logic [9:0] sx1;
logic [9:0] sy1;
logic [9:0] sx2;
logic [9:0] sy2;
logic [9:0] sx3;
logic [9:0] sy3;
logic [9:0] sx4;
logic [9:0] sy4;
```

```
logic [9:0] sx5;
logic [9:0] sy5;
logic [9:0] sx6;
logic [9:0] sy6;
logic [9:0] sx7;
logic [9:0] sy7;
logic [9:0] sx8;
logic [9:0] sy8;
logic [9:0] sx9;
logic [9:0] sy9;
logic [9:0] sx10;
logic [9:0] sy10;
logic [9:0] sx11;
logic [9:0] sy11;
logic [9:0] sx12;
logic [9:0] sy12;
logic [9:0] sx13;
logic [9:0] sy13;

assign sx1 = 60;
assign sy1 = 400;
assign sx2 = 80;
assign sy2 = 400;
assign sx3 = 100;
assign sy3 = 400;
assign sx4 = 120;
assign sy4 = 400;
assign sx5 = 140;
assign sy5 = 400;
assign sx6 = 160;
assign sy6 = 400;
assign sx7 = 180;
assign sy7 = 400;
assign sx8 = 200;
assign sy8 = 400;
assign sx9 = 220;
assign sy9 = 400;
assign sx10 = 240;
assign sy10 = 400;
assign sx11 = 260;
assign sy11 = 400;
assign sx12 = 280;
assign sy12 = 400;
```

```
        assign sx13= 300;
        assign sy13 = 400;


        //
        assign st_pos1 = ((hcount[10:1] <= 75)&&(hcount[10:1] >= 60)&&(vcount[9:0] >=
400)&&(vcount[9:0] < 420));
        assign st_pos2 = ((hcount[10:1] <= 95)&&(hcount[10:1] >= 80)&&(vcount[9:0]
>=400)&&(vcount[9:0] < 420));
        assign st_pos3 = ((hcount[10:1] <= 115)&&(hcount[10:1] >= 100)&&(vcount[9:0] >=
400)&&(vcount[9:0] < 420));
        assign st_pos4 = ((hcount[10:1] <= 135)&&(hcount[10:1] >= 120)&&(vcount[9:0] >=
400)&&(vcount[9:0] < 420));
        assign st_pos5 = ((hcount[10:1] <= 155)&&(hcount[10:1] >= 140)&&(vcount[9:0] >=
400)&&(vcount[9:0] < 420));
        assign st_pos6 = ((hcount[10:1] <= 175)&&(hcount[10:1] >= 160)&&(vcount[9:0] >=
400)&&(vcount[9:0] < 420));
        assign st_pos7 = ((hcount[10:1] <= 195)&&(hcount[10:1] >= 180)&&(vcount[9:0] >=
400)&&(vcount[9:0] < 420));
        assign st_pos8 = ((hcount[10:1] <= 215)&&(hcount[10:1] >= 200)&&(vcount[9:0] >=
400)&&(vcount[9:0] < 420));
        assign st_pos9 = ((hcount[10:1] <= 235)&&(hcount[10:1] >= 220)&&(vcount[9:0] >=
400)&&(vcount[9:0] < 420));
        assign st_pos10 = ((hcount[10:1] <= 255)&&(hcount[10:1] >= 240)&&(vcount[9:0] >=
400)&&(vcount[9:0] < 420));
        assign st_pos11 = ((hcount[10:1] <= 275)&&(hcount[10:1] >= 260)&&(vcount[9:0] >=
400)&&(vcount[9:0] < 420));
        assign st_pos12 = ((hcount[10:1] <= 295)&&(hcount[10:1] >= 280)&&(vcount[9:0] >=
400)&&(vcount[9:0] < 420));
        assign st_pos13 = ((hcount[10:1] <= 315)&&(hcount[10:1] >= 300)&&(vcount[9:0] >=
400)&&(vcount[9:0] < 420));


        //
        //
        assign write_okay_sp1 = ((s_S[hcount[10:1] - sx1][vcount[9:0] - sy1]));
        assign write_okay_sp2 = ((s_T[hcount[10:1] - sx2][vcount[9:0] - sy2]));
        assign write_okay_sp3 = ((s_O[hcount[10:1] - sx3][vcount[9:0] - sy3]));
        assign write_okay_sp4 = ((s_C[hcount[10:1] - sx4][vcount[9:0] - sy4]));
        assign write_okay_sp5 = ((s_K[hcount[10:1] - sx5][vcount[9:0] - sy5]));
        assign write_okay_sp6 = ((s_colon[hcount[10:1] - sx6][vcount[9:0] - sy6]));

        assign write_okay_sp11 = ((s_dot[hcount[10:1] - sx11][vcount[9:0] - sy11]));
```

```
always_comb begin
	if (S_hundredthousand == 0) begin
		write_okay_sp7 = ((s_O[hcount[10:1] - sx7][vcount[9:0] - sy7]));
	end
	else if (S_hundredthousand == 1) begin
		write_okay_sp7 = ((s_1[hcount[10:1] - sx7][vcount[9:0] - sy7]));
	end
	else if (S_hundredthousand == 2) begin
		write_okay_sp7 = ((s_2[hcount[10:1] - sx7][vcount[9:0] - sy7]));
	end
	else if (S_hundredthousand == 3) begin
		write_okay_sp7 = ((s_3[hcount[10:1] - sx7][vcount[9:0] - sy7]));
	end
	else if (S_hundredthousand == 4) begin
		write_okay_sp7 = ((s_4[hcount[10:1] - sx7][vcount[9:0] - sy7]));
	end
	else if (S_hundredthousand == 5) begin
		write_okay_sp7 = ((s_5[hcount[10:1] - sx7][vcount[9:0] - sy7]));
	end
	else if (S_hundredthousand == 6) begin
		write_okay_sp7 = ((s_6[hcount[10:1] - sx7][vcount[9:0] - sy7]));
	end
	else if (S_hundredthousand == 7) begin
		write_okay_sp7 = ((s_7[hcount[10:1] - sx7][vcount[9:0] - sy7]));
	end
	else if (S_hundredthousand == 8) begin
		write_okay_sp7 = ((s_8[hcount[10:1] - sx7][vcount[9:0] - sy7]));
	end
	else if (S_hundredthousand == 9) begin
		write_okay_sp7 = ((s_9[hcount[10:1] - sx7][vcount[9:0] - sy7]));
	end
	else begin
		write_okay_sp7 = ((s_blank[hcount[10:1] - sx7][vcount[9:0] - sy7]));
	end
end

always_comb begin
	if (S_tenthousand == 0) begin
		write_okay_sp8 = ((s_O[hcount[10:1] - sx8][vcount[9:0] - sy8]));
```

```
        end
        else if (S_tenthousand == 1) begin
                write_okay_sp8 = ((s_1[hcount[10:1] - sx8][vcount[9:0] - sy8]));
        end
        else if (S_tenthousand == 2) begin
                write_okay_sp8 = ((s_2[hcount[10:1] - sx8][vcount[9:0] - sy8]));
        end
        else if (S_tenthousand == 3) begin
                write_okay_sp8 = ((s_3[hcount[10:1] - sx8][vcount[9:0] - sy8]));
        end
        else if (S_tenthousand == 4) begin
                write_okay_sp8 = ((s_4[hcount[10:1] - sx8][vcount[9:0] - sy8]));
        end
        else if (S_tenthousand == 5) begin
                write_okay_sp8 = ((s_5[hcount[10:1] - sx8][vcount[9:0] - sy8]));
        end
        else if (S_tenthousand == 6) begin
                write_okay_sp8 = ((s_6[hcount[10:1] - sx8][vcount[9:0] - sy8]));
        end
        else if (S_tenthousand == 7) begin
                write_okay_sp8 = ((s_7[hcount[10:1] - sx8][vcount[9:0] - sy8]));
        end
        else if (S_tenthousand == 8) begin
                write_okay_sp8 = ((s_8[hcount[10:1] - sx8][vcount[9:0] - sy8]));
        end
        else if (S_tenthousand == 9) begin
                write_okay_sp8 = ((s_9[hcount[10:1] - sx8][vcount[9:0] - sy8]));
        end
        else begin
                write_okay_sp8 = ((s_blank[hcount[10:1] - sx8][vcount[9:0] - sy8]));
        end
end

always_comb begin
        if (S_thousand == 0) begin
                write_okay_sp9 = ((s_O[hcount[10:1] - sx9][vcount[9:0] - sy9]));
        end
        else if (S_thousand == 1) begin
                write_okay_sp9 = ((s_1[hcount[10:1] - sx9][vcount[9:0] - sy9]));
        end
        else if (S_thousand == 2) begin
                write_okay_sp9 = ((s_2[hcount[10:1] - sx9][vcount[9:0] - sy9]));
        end
```

```
        else if (S_thousand == 3) begin
                write_okay_sp9 = ((s_3[hcount[10:1] - sx9][vcount[9:0] - sy9]));
        end
        else if (S_thousand == 4) begin
                write_okay_sp9 = ((s_4[hcount[10:1] - sx9][vcount[9:0] - sy9]));
        end
        else if (S_thousand == 5) begin
                write_okay_sp9 = ((s_5[hcount[10:1] - sx9][vcount[9:0] - sy9]));
        end
        else if (S_thousand == 6) begin
                write_okay_sp9 = ((s_6[hcount[10:1] - sx9][vcount[9:0] - sy9]));
        end
        else if (S_thousand == 7) begin
                write_okay_sp9 = ((s_7[hcount[10:1] - sx9][vcount[9:0] - sy9]));
        end
        else if (S_thousand == 8) begin
                write_okay_sp9 = ((s_8[hcount[10:1] - sx9][vcount[9:0] - sy9]));
        end
        else if (S_thousand == 9) begin
                write_okay_sp9 = ((s_9[hcount[10:1] - sx9][vcount[9:0] - sy9]));
        end
        else begin
                write_okay_sp9 = ((s_blank[hcount[10:1] - sx9][vcount[9:0] - sy9]));
        end
end

always_comb begin
        if (S_hundred == 0) begin
                write_okay_sp10 = ((s_O[hcount[10:1] - sx10][vcount[9:0] - sy10]));
        end
        else if (S_hundred == 1) begin
                write_okay_sp10 = ((s_1[hcount[10:1] - sx10][vcount[9:0] - sy10]));
        end
        else if (S_hundred == 2) begin
                write_okay_sp10 = ((s_2[hcount[10:1] - sx10][vcount[9:0] - sy10]));
        end
        else if (S_hundred == 3) begin
                write_okay_sp10 = ((s_3[hcount[10:1] - sx10][vcount[9:0] - sy10]));
        end
        else if (S_hundred == 4) begin
                write_okay_sp10 = ((s_4[hcount[10:1] - sx10][vcount[9:0] - sy10]));
        end
        else if (S_hundred == 5) begin
```

```
                write_okay_sp10 = ((s_5[hcount[10:1] - sx10][vcount[9:0] - sy10]));
            end
            else if (S_hundred == 6) begin
                write_okay_sp10 = ((s_6[hcount[10:1] - sx10][vcount[9:0] - sy10]));
            end
            else if (S_hundred == 7) begin
                write_okay_sp10 = ((s_7[hcount[10:1] - sx10][vcount[9:0] - sy10]));
            end
            else if (S_hundred == 8) begin
                write_okay_sp10 = ((s_8[hcount[10:1] - sx10][vcount[9:0] - sy10]));
            end
            else if (S_hundred == 9) begin
                write_okay_sp10 = ((s_9[hcount[10:1] - sx10][vcount[9:0] - sy10]));
            end
            else begin
                write_okay_sp10 = ((s_blank[hcount[10:1] - sx10][vcount[9:0] - sy10]));
            end
        end

always_comb begin
            if (S_ten == 0) begin
                write_okay_sp12 = ((s_O[hcount[10:1] - sx12][vcount[9:0] - sy12]));
            end
            else if (S_ten == 1) begin
                write_okay_sp12 = ((s_1[hcount[10:1] - sx12][vcount[9:0] - sy12]));
            end
            else if (S_ten == 2) begin
                write_okay_sp12 = ((s_2[hcount[10:1] - sx12][vcount[9:0] - sy12]));
            end
            else if (S_ten == 3) begin
                write_okay_sp12 = ((s_3[hcount[10:1] - sx12][vcount[9:0] - sy12]));
            end
            else if (S_ten == 4) begin
                write_okay_sp12 = ((s_4[hcount[10:1] - sx12][vcount[9:0] - sy12]));
            end
            else if (S_ten == 5) begin
                write_okay_sp12 = ((s_5[hcount[10:1] - sx12][vcount[9:0] - sy12]));
            end
            else if (S_ten == 6) begin
                write_okay_sp12 = ((s_6[hcount[10:1] - sx12][vcount[9:0] - sy12]));
            end
            else if (S_ten == 7) begin
                write_okay_sp12 = ((s_7[hcount[10:1] - sx12][vcount[9:0] - sy12]));
```

```
                end
                else if (S_ten == 8) begin
                        write_okay_sp12 = ((s_8[hcount[10:1] - sx12][vcount[9:0] - sy12]));
                end
                else if (S_ten == 9) begin
                        write_okay_sp12 = ((s_9[hcount[10:1] - sx12][vcount[9:0] - sy12]));
                end
                else begin
                        write_okay_sp12 = ((s_blank[hcount[10:1] - sx12][vcount[9:0] - sy12]));
                end
        end

always_comb begin
                if (S_one == 0) begin
                        write_okay_sp13 = ((s_O[hcount[10:1] - sx13][vcount[9:0] - sy13]));
                end
                else if (S_one == 1) begin
                        write_okay_sp13 = ((s_1[hcount[10:1] - sx13][vcount[9:0] - sy13]));
                end
                else if (S_one == 2) begin
                        write_okay_sp13 = ((s_2[hcount[10:1] - sx13][vcount[9:0] - sy13]));
                end
                else if (S_one == 3) begin
                        write_okay_sp13 = ((s_3[hcount[10:1] - sx13][vcount[9:0] - sy13]));
                end
                else if (S_one == 4) begin
                        write_okay_sp13 = ((s_4[hcount[10:1] - sx13][vcount[9:0] - sy13]));
                end
                else if (S_one == 5) begin
                        write_okay_sp13 = ((s_5[hcount[10:1] - sx13][vcount[9:0] - sy13]));
                end
                else if (S_one == 6) begin
                        write_okay_sp13 = ((s_6[hcount[10:1] - sx13][vcount[9:0] - sy13]));
                end
                else if (S_one == 7) begin
                        write_okay_sp13 = ((s_7[hcount[10:1] - sx13][vcount[9:0] - sy13]));
                end
                else if (S_one == 8) begin
                        write_okay_sp13 = ((s_8[hcount[10:1] - sx13][vcount[9:0] - sy13]));
                end
                else if (S_one == 9) begin
                        write_okay_sp13 = ((s_9[hcount[10:1] - sx13][vcount[9:0] - sy13]));
                end
```

```
                else begin
                        write_okay_sp13 = ((s_blank[hcount[10:1] - sx13][vcount[9:0] - sy13]));
                end
        end

        //drawing plot box
        assign write_okay_xaxis1 = ((37 < vcount[9:0])&&(vcount[9:0] < 43)&&(37 <
hcount[10:1])&&(hcount[10:1] < 449));
        assign write_okay_xaxis2 = ((337 < vcount[9:0])&&(vcount[9:0] < 343)&&(37 <
hcount[10:1])&&(hcount[10:1] < 449));

        assign write_okay_yaxis1 = ((37 < vcount[9:0])&&(vcount[9:0] < 343)&&(37 <
hcount[10:1])&&(hcount[10:1] < 43));
        assign write_okay_yaxis2 = ((37 < vcount[9:0])&&(vcount[9:0] < 343)&&(443 <
hcount[10:1])&&(hcount[10:1] < 449));


        //write okay for drawing stock
          //assign temp_address = hcount[10:1];
          assign write_okay_r = (vcount[9:0] - 10 == disp_stock_x[hcount[10:1] - 42]);
          assign write_okay_g = (vcount[9:0] - 10 == disp_stock_a1[hcount[10:1] - 42]);
          assign write_okay_b = (vcount[9:0] - 10 == disp_stock_a2[hcount[10:1] - 42]);
          assign write_okay_o = (vcount[9:0] - 10 == disp_stock_a3[hcount[10:1] - 42]);
          assign write_stock_inbounds = ((hcount[10:1] < 443)&&(hcount[10:1] >
43)&&(vcount[9:0] > 40)&&(vcount[9:0] < 340));

        //Grey box
        assign write_trafficbox = ((hcount[10:1] <= 630)&&(hcount[10:1] >= 530)&&(vcount[9:0]
>= 150)&&(vcount[9:0] <= 450));

        //Draw ball for light
         parameter R = 1600; // radius of ball squared
         logic [9:0] cx_1,cx_2,cx_3;
   logic [9:0] cy_1,cy_2,cy_3;
         logic [9:0] R_dark_1,R_dark_2,R_dark_3;
         logic [9:0] Dark_light1,Dark_light2,Dark_light3;

   logic G_light; // condition whether (hc,vc) is inside ball
         parameter cx_g = 580;
         parameter cy_g = 200;
   assign G_light = (( (cx_g - hcount[10:1])*(cx_g - hcount[10:1]) + (cy_g - vcount)*(cy_g -
vcount) ) < R);
```

```
        logic Y_light; // condition whether (hc,vc) is inside ball

        parameter cx_y = 580;
        parameter cy_y = 300;
   assign Y_light = (( (cx_y - hcount[10:1])*(cx_y - hcount[10:1]) + (cy_y - vcount)*(cy_y -
vcount) ) < R);

        logic R_light; // condition whether (hc,vc) is inside ball

        parameter cx_r = 580;
        parameter cy_r = 400;
   assign R_light = (( (cx_r - hcount[10:1])*(cx_r - hcount[10:1]) + (cy_r - vcount)*(cy_r -
vcount) ) < R);



        ////////////////////////
        // show Sell Hold Buy //
        ////////////////////////


        logic write_okay_os;
        logic [9:0] x_os_letter1;
        logic [9:0] y_os_letter1;
        assign x_os_letter1 = 500;
        assign y_os_letter1 = 200;

        logic [9:0] x_os_letter2;
        logic [9:0] y_os_letter2;
        assign x_os_letter2 = 500;
        assign y_os_letter2 = 300;

        logic [9:0] x_os_letter3;
        logic [9:0] y_os_letter3;
        assign x_os_letter3 = 500;
        assign y_os_letter3 = 400;

        logic os_pos1;
        logic os_pos2;
        logic os_pos3;
        logic select_pos;
        //logic sn_pos4;
```

```
        // Write in location only ones
        assign os_pos1 = ((hcount[10:1] < 525)&&(hcount[10:1] > 500)&&(vcount[9:0] >
200)&&(vcount[9:0] < 240));
        assign os_pos2 = ((hcount[10:1] < 525)&&(hcount[10:1] > 500)&&(vcount[9:0] >
300)&&(vcount[9:0] < 340));
        assign os_pos3 = ((hcount[10:1] < 525)&&(hcount[10:1] > 500)&&(vcount[9:0] >
400)&&(vcount[9:0] < 440));
        //assign sn_pos4 = ((hcount[10:1] < 600)&&(hcount[10:1] > 575)&&(vcount[9:0] >
100)&&(vcount[9:0] < 140));


        //
        //
        //
        //Write letters for buy hold sell
        always_comb begin

                if (bhs == 2'b00) begin // Buy
                        write_okay_os = ((S_B[hcount[10:1] - x_os_letter1][vcount[9:0] -
y_os_letter1]));
                        select_pos = os_pos1;

                end
                else if (bhs == 2'b01) begin // Hold
                        write_okay_os = ((S_H[hcount[10:1] - x_os_letter2][vcount[9:0] -
y_os_letter2]));
        select_pos = os_pos2;

                end
                else begin // Sell
                        write_okay_os = ((S_S[hcount[10:1] - x_os_letter3][vcount[9:0] -
y_os_letter3]));
        select_pos = os_pos3;
                end

        end

 //
 //
 //Write okay conditions and color
        always_comb begin
        {VGA_R, VGA_G, VGA_B} = {8'h0, 8'h0, 8'h0}; // Black
```

```
        if (write_okay_r && write_stock_inbounds) begin
                {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00}; // Red, stock data
        end
        if (write_okay_b && write_stock_inbounds) begin
                {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h00, 8'hff}; // Blue, EMA32
        end
        if (write_okay_g && write_stock_inbounds) begin
                {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hff, 8'h00}; // Green, EMA4
        end
        if (write_okay_o && write_stock_inbounds) begin
                {VGA_R, VGA_G, VGA_B} = {8'haa, 8'h00, 8'haa}; // Purple, EMA128
        end
        //
        //Numbers for stock price
        if ((write_okay_sp1 && st_pos1) || (write_okay_sp2 && st_pos2) || (write_okay_sp3 &&
st_pos3)) begin
                {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff}; // white
        end
        if ((write_okay_sp2 && st_pos2) || (write_okay_sp3 && st_pos3) || (write_okay_sp4 &&
st_pos4)) begin
                {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff}; // white
        end
        if ((write_okay_sp5 && st_pos5) || (write_okay_sp6 && st_pos6) || (write_okay_sp7 &&
st_pos7)) begin
                {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff}; // white
        end
        if ((write_okay_sp8 && st_pos8) || (write_okay_sp9 && st_pos9) || (write_okay_sp10
&& st_pos10)) begin
                {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff}; // white
        end
        if ((write_okay_sp11 && st_pos11) || (write_okay_sp12 && st_pos12) ||
(write_okay_sp13 && st_pos13)) begin
                {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff}; // white
        end
        //
        //Numbers for high price on y-axis
        if ((write_okay_h1 && left_h1)||(write_okay_h2 && left_h2)||(write_okay_h3 &&
left_h3)) begin
                {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff}; // white
        end
        //Numbers for low price on y-axis
        if ((write_okay_l1 && left_l1)||(write_okay_l2 && left_l2)||(write_okay_l3 && left_l3))
begin
```

```
                {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff}; // white
        end
        //
        if ((write_okay_sn1 && sn_pos1)|| (write_okay_sn2 && sn_pos2) || (write_okay_sn3
&& sn_pos3) || (write_okay_sn4 && sn_pos4) ) begin
                {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff}; // white
        end
        if (write_okay_xaxis1 || write_okay_xaxis2 || write_okay_yaxis1 || write_okay_yaxis2)
begin
                {VGA_R, VGA_G, VGA_B} = {8'hbb, 8'hbb, 8'hbb}; // grey
        end
        //Traffic Light conditions
        if (write_trafficbox) begin
                {VGA_R, VGA_G, VGA_B} = {8'haa, 8'haa, 8'haa}; // grey
        end
        if ((G_light) && (bhs == 2'b00)) begin
                {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hff, 8'h00}; // Green
        end
        if ((G_light) && (bhs != 2'b00)) begin
                {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h00, 8'h00}; // Black
        end
        if ((Y_light) && (bhs == 2'b01)) begin
                {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'h00}; // Yellow
        end
        if ((Y_light) && (bhs != 2'b01)) begin
                {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h00, 8'h00}; // Black
        end
        if ((R_light) && (bhs == 2'b10)) begin
                {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00}; // Red
        end
        if ((R_light) && (bhs != 2'b10)) begin
                {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h00, 8'h00}; // Black
        end
        if ((write_okay_os && select_pos) ) begin
                {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff}; // Write decision letters
        end

        end

        //
        //Array of registers for drawing letters
        logic[39:0] S_A[24:0];
        logic[39:0] S_B[24:0];
```

```
logic[39:0] S_C[24:0];
logic[39:0] S_D[24:0];
logic[39:0] S_E[24:0];
logic[39:0] S_F[24:0];
logic[39:0] S_G[24:0];
logic[39:0] S_H[24:0];
logic[39:0] S_I[24:0];
logic[39:0] S_J[24:0];
logic[39:0] S_K[24:0];
logic[39:0] S_L[24:0];
logic[39:0] S_M[24:0];
logic[39:0] S_N[24:0];
logic[39:0] S_O[24:0];
logic[39:0] S_P[24:0];
logic[39:0] S_Q[24:0];
logic[39:0] S_R[24:0];
logic[39:0] S_S[24:0];
logic[39:0] S_T[24:0];
logic[39:0] S_U[24:0];
logic[39:0] S_V[24:0];
logic[39:0] S_W[24:0];
logic[39:0] S_X[24:0];
logic[39:0] S_Y[24:0];
logic[39:0] S_Z[24:0];
logic[39:0] S_colon[24:0];
logic[39:0] S_1[24:0];
logic[39:0] S_2[24:0];
logic[39:0] S_3[24:0];
logic[39:0] S_4[24:0];
logic[39:0] S_5[24:0];
logic[39:0] S_6[24:0];
logic[39:0] S_7[24:0];
logic[39:0] S_8[24:0];
logic[39:0] S_9[24:0];
logic[39:0] S_blank[24:0];


assign S_blank [0] = 40'b0000000000000000000000000000000000000000;
assign S_blank [1] = 40'b0000000000000000000000000000000000000000;
assign S_blank [2] = 40'b0000000000000000000000000000000000000000;
assign S_blank [3] = 40'b0000000000000000000000000000000000000000;
assign S_blank [4] = 40'b0000000000000000000000000000000000000000;
assign S_blank [5] = 40'b0000000000000000000000000000000000000000;
```

```
assign S_blank [6] = 40'b0000000000000000000000000000000000000000;
assign S_blank [7] = 40'b0000000000000000000000000000000000000000;
assign S_blank [8] = 40'b0000000000000000000000000000000000000000;
assign S_blank [9] = 40'b0000000000000000000000000000000000000000;
assign S_blank [10] = 40'b0000000000000000000000000000000000000000;
assign S_blank [11] = 40'b0000000000000000000000000000000000000000;
assign S_blank [12] = 40'b0000000000000000000000000000000000000000;
assign S_blank [13] = 40'b0000000000000000000000000000000000000000;
assign S_blank [14] = 40'b0000000000000000000000000000000000000000;
assign S_blank [15] = 40'b0000000000000000000000000000000000000000;
assign S_blank [16] = 40'b0000000000000000000000000000000000000000;
assign S_blank [17] = 40'b0000000000000000000000000000000000000000;
assign S_blank [18] = 40'b0000000000000000000000000000000000000000;
assign S_blank [19] = 40'b0000000000000000000000000000000000000000;
assign S_blank [20] = 40'b0000000000000000000000000000000000000000;
assign S_blank [21] = 40'b0000000000000000000000000000000000000000;
assign S_blank [22] = 40'b0000000000000000000000000000000000000000;
assign S_blank [23] = 40'b0000000000000000000000000000000000000000;
assign S_blank [24] = 40'b0000000000000000000000000000000000000000;
assign S_A[0] = 40'b0000000001100000000000000000000000000000;
assign S_A[1] = 40'b0000000011111000000000000000000000000000;
assign S_A[2] = 40'b0000000011111111000000000000000000000000;
assign S_A[3] = 40'b0000000000011111111000000000000000000000;
assign S_A[4] = 40'b0000000000000011111111000000000000000000;
assign S_A[5] = 40'b0000000000000001111111110000000000000000;
assign S_A[6] = 40'b0000000000000001110111111100000000000000;
assign S_A[7] = 40'b0000000000000001110001111111100000000000;
assign S_A[8] = 40'b0000000000000001110000001111111000000000;
assign S_A[9] = 40'b0000000000000001110000000001111100000000;
assign S_A[10] = 40'b0000000000000001110000001111111000000000;
assign S_A[11] = 40'b0000000000000001110001111111100000000000;
assign S_A[12] = 40'b0000000000000001110111111100000000000000;
assign S_A[13] = 40'b0000000000000001111111110000000000000000;
assign S_A[14] = 40'b0000000000000011111111000000000000000000;
assign S_A[15] = 40'b0000000000011111111000000000000000000000;
assign S_A[16] = 40'b0000000001111111000000000000000000000000;
assign S_A[17] = 40'b0000000011111000000000000000000000000000;
assign S_A[18] = 40'b0000000001100000000000000000000000000000;
assign S_A[19] = 40'b0000000000000000000000000000000000000000;
assign S_A[20] = 40'b0000000000000000000000000000000000000000;
assign S_A[21] = 40'b0000000000000000000000000000000000000000;
assign S_A[22] = 40'b0000000000000000000000000000000000000000;
assign S_A[23] = 40'b0000000000000000000000000000000000000000;
```

```
assign S_A[24] = 40'b0000000000000000000000000000000000000000;
assign S_B[0] = 40'b0000000000000000000000000000000000000000;
assign S_B[1] = 40'b0000000000000000000000000000000000000000;
assign S_B[2] = 40'b0000000000000000000000000000000000000000;
assign S_B[3] = 40'b0000000001111111111111111111111000000000;
assign S_B[4] = 40'b0000000011111111111111111111111100000000;
assign S_B[5] = 40'b0000000011111111111111111111111100000000;
assign S_B[6] = 40'b0000000011100000000111000000011100000000;
assign S_B[7] = 40'b0000000011100000000111000000011100000000;
assign S_B[8] = 40'b0000000011100000000111000000011100000000;
assign S_B[9] = 40'b0000000011100000000111000000011100000000;
assign S_B[10] = 40'b0000000011100000000111000000011100000000;
assign S_B[11] = 40'b0000000011100000000111000000011100000000;
assign S_B[12] = 40'b0000000011100000000111000000011100000000;
assign S_B[13] = 40'b0000000011100000000111000000011100000000;
assign S_B[14] = 40'b0000000011100000000111000000011100000000;
assign S_B[15] = 40'b0000000001110000001111100000111000000000;
assign S_B[16] = 40'b0000000001111000011111110001111000000000;
assign S_B[17] = 40'b0000000001111111111111111111111000000000;
assign S_B[18] = 40'b0000000000111111111101111111110000000000;
assign S_B[19] = 40'b0000000000000011110000000111000000000000;
assign S_B[20] = 40'b0000000000000000000000000000000000000000;
assign S_B[21] = 40'b0000000000000000000000000000000000000000;
assign S_B[22] = 40'b0000000000000000000000000000000000000000;
assign S_B[23] = 40'b0000000000000000000000000000000000000000;
assign S_B[24] = 40'b0000000000000000000000000000000000000000;
assign S_C[0] = 40'b0000000000000000000000000000000000000000;
assign S_C[1] = 40'b0000000000000000000000000000000000000000;
assign S_C[2] = 40'b0000000000000000111111110000000000000000;
assign S_C[3] = 40'b0000000000000011111111111100000000000000;
assign S_C[4] = 40'b0000000000011111111111111111100000000000;
assign S_C[5] = 40'b0000000000111111000000001111110000000000;
assign S_C[6] = 40'b0000000001111100000000000011111000000000;
assign S_C[7] = 40'b0000000001111000000000000001111000000000;
assign S_C[8] = 40'b0000000001110000000000000000111000000000;
assign S_C[9] = 40'b0000000011100000000000000000011100000000;
assign S_C[10] = 40'b0000000011100000000000000000011100000000;
assign S_C[11] = 40'b0000000011100000000000000000011100000000;
assign S_C[12] = 40'b0000000011100000000000000000011100000000;
assign S_C[13] = 40'b0000000011100000000000000000011100000000;
assign S_C[14] = 40'b0000000001110000000000000000111000000000;
assign S_C[15] = 40'b0000000001111000000000000001111000000000;
assign S_C[16] = 40'b0000000001111100000000000011111000000000;
```

```
assign S_C[17] = 40'b0000000000111110000000000111110000000000;
assign S_C[18] = 40'b0000000000011111000000001111100000000000;
assign S_C[19] = 40'b0000000000000010000000000100000000000000;
assign S_C[20] = 40'b0000000000000000000000000000000000000000;
assign S_C[21] = 40'b0000000000000000000000000000000000000000;
assign S_C[22] = 40'b0000000000000000000000000000000000000000;
assign S_C[23] = 40'b0000000000000000000000000000000000000000;
assign S_C[24] = 40'b0000000000000000000000000000000000000000;
assign S_D[0] = 40'b0000000000000000000000000000000000000000;
assign S_D[1] = 40'b0000000000000000000000000000000000000000;
assign S_D[2] = 40'b0000000000000000000000000000000000000000;
assign S_D[3] = 40'b0000000001111111111111111111111000000000;
assign S_D[4] = 40'b0000000011111111111111111111111100000000;
assign S_D[5] = 40'b0000000011111111111111111111111100000000;
assign S_D[6] = 40'b0000000011100000000000000000011100000000;
assign S_D[7] = 40'b0000000011100000000000000000011100000000;
assign S_D[8] = 40'b0000000011100000000000000000011100000000;
assign S_D[9] = 40'b0000000011100000000000000000011100000000;
assign S_D[10] = 40'b0000000011100000000000000000011100000000;
assign S_D[11] = 40'b0000000011100000000000000000011100000000;
assign S_D[12] = 40'b0000000011100000000000000000011100000000;
assign S_D[13] = 40'b0000000001110000000000000000111000000000;
assign S_D[14] = 40'b0000000001111000000000000001111000000000;
assign S_D[15] = 40'b0000000001111100000000000011111000000000;
assign S_D[16] = 40'b0000000000111111000000001111110000000000;
assign S_D[17] = 40'b0000000000011111111111111111100000000000;
assign S_D[18] = 40'b0000000000000011111111111100000000000000;
assign S_D[19] = 40'b0000000000000000111111110000000000000000;
assign S_D[20] = 40'b0000000000000000000000000000000000000000;
assign S_D[21] = 40'b0000000000000000000000000000000000000000;
assign S_D[22] = 40'b0000000000000000000000000000000000000000;
assign S_D[23] = 40'b0000000000000000000000000000000000000000;
assign S_D[24] = 40'b0000000000000000000000000000000000000000;
assign S_E[0] = 40'b0000000000000000000000000000000000000000;
assign S_E[1] = 40'b0000000000000000000000000000000000000000;
assign S_E[2] = 40'b0000000000000000000000000000000000000000;
assign S_E[3] = 40'b0000000001111111111111111111111000000000;
assign S_E[4] = 40'b0000000011111111111111111111111100000000;
assign S_E[5] = 40'b0000000011111111111111111111111100000000;
assign S_E[6] = 40'b0000000011100000000111000000011100000000;
assign S_E[7] = 40'b0000000011100000000111000000011100000000;
assign S_E[8] = 40'b0000000011100000000111000000011100000000;
assign S_E[9] = 40'b0000000011100000000111000000011100000000;
```

```
assign S_E[10] = 40'b0000000011100000000111000000011100000000;
assign S_E[11] = 40'b0000000011100000000111000000011100000000;
assign S_E[12] = 40'b0000000011100000000111000000011100000000;
assign S_E[13] = 40'b0000000011100000000010000000011100000000;
assign S_E[14] = 40'b0000000011100000000000000000011100000000;
assign S_E[15] = 40'b0000000011100000000000000000011100000000;
assign S_E[16] = 40'b0000000011100000000000000000011100000000;
assign S_E[17] = 40'b0000000011100000000000000000011100000000;
assign S_E[18] = 40'b0000000001000000000000000000001000000000;
assign S_E[19] = 40'b0000000000000000000000000000000000000000;
assign S_E[20] = 40'b0000000000000000000000000000000000000000;
assign S_E[21] = 40'b0000000000000000000000000000000000000000;
assign S_E[22] = 40'b0000000000000000000000000000000000000000;
assign S_E[23] = 40'b0000000000000000000000000000000000000000;
assign S_E[24] = 40'b0000000000000000000000000000000000000000;
assign S_F[0] = 40'b0000000000000000000000000000000000000000;
assign S_F[1] = 40'b0000000000000000000000000000000000000000;
assign S_F[2] = 40'b0000000000000000000000000000000000000000;
assign S_F[3] = 40'b0000000001111111111111111111111000000000;
assign S_F[4] = 40'b0000000011111111111111111111111100000000;
assign S_F[5] = 40'b0000000011111111111111111111111100000000;
assign S_F[6] = 40'b0000000000000000000111000000011100000000;
assign S_F[7] = 40'b0000000000000000000111000000011100000000;
assign S_F[8] = 40'b0000000000000000000111000000011100000000;
assign S_F[9] = 40'b0000000000000000000111000000011100000000;
assign S_F[10] = 40'b0000000000000000000111000000011100000000;
assign S_F[11] = 40'b0000000000000000000111000000011100000000;
assign S_F[12] = 40'b0000000000000000000111000000011100000000;
assign S_F[13] = 40'b0000000000000000000010000000011100000000;
assign S_F[14] = 40'b0000000000000000000000000000011100000000;
assign S_F[15] = 40'b0000000000000000000000000000011100000000;
assign S_F[16] = 40'b0000000000000000000000000000011100000000;
assign S_F[17] = 40'b0000000000000000000000000000011100000000;
assign S_F[18] = 40'b0000000000000000000000000000001000000000;
assign S_F[19] = 40'b0000000000000000000000000000000000000000;
assign S_F[20] = 40'b0000000000000000000000000000000000000000;
assign S_F[21] = 40'b0000000000000000000000000000000000000000;
assign S_F[22] = 40'b0000000000000000000000000000000000000000;
assign S_F[23] = 40'b0000000000000000000000000000000000000000;
assign S_F[24] = 40'b0000000000000000000000000000000000000000;
assign S_G[0] = 40'b0000000000000000000000000000000000000000;
assign S_G[1] = 40'b0000000000000000000000000000000000000000;
assign S_G[2] = 40'b0000000000000000111111110000000000000000;
```

```
assign S_G[3] = 40'b0000000000000011111111111100000000000000;
assign S_G[4] = 40'b0000000000011111111111111111100000000000;
assign S_G[5] = 40'b0000000000111111000000001111110000000000;
assign S_G[6] = 40'b0000000001111100000000000011111000000000;
assign S_G[7] = 40'b0000000001111000000000000001111000000000;
assign S_G[8] = 40'b0000000001110000000000000000111000000000;
assign S_G[9] = 40'b0000000011100000000000000000011100000000;
assign S_G[10] = 40'b0000000011100000000000000000011100000000;
assign S_G[11] = 40'b0000000011100000000000000000011100000000;
assign S_G[12] = 40'b0000000011100000100000000000011100000000;
assign S_G[13] = 40'b0000000011100000111000000000011100000000;
assign S_G[14] = 40'b0000000001110000111000000000111000000000;
assign S_G[15] = 40'b0000000001111000111000000001111000000000;
assign S_G[16] = 40'b0000000001111100111000000011111000000000;
assign S_G[17] = 40'b0000000000011111111000000111110000000000;
assign S_G[18] = 40'b0000000000011111111000001111100000000000;
assign S_G[19] = 40'b0000000000000011110000000100000000000000;
assign S_G[20] = 40'b0000000000000000000000000000000000000000;
assign S_G[21] = 40'b0000000000000000000000000000000000000000;
assign S_G[22] = 40'b0000000000000000000000000000000000000000;
assign S_G[23] = 40'b0000000000000000000000000000000000000000;
assign S_G[24] = 40'b0000000000000000000000000000000000000000;
assign S_H[0] = 40'b0000000000000000000000000000000000000000;
assign S_H[1] = 40'b0000000000000000000000000000000000000000;
assign S_H[2] = 40'b0000000000000000000000000000000000000000;
assign S_H[3] = 40'b0000000001111111111111111111111000000000;
assign S_H[4] = 40'b0000000011111111111111111111111100000000;
assign S_H[5] = 40'b0000000001111111111111111111111000000000;
assign S_H[6] = 40'b0000000000000000000111000000000000000000;
assign S_H[7] = 40'b0000000000000000000111000000000000000000;
assign S_H[8] = 40'b0000000000000000000111000000000000000000;
assign S_H[9] = 40'b0000000000000000000111000000000000000000;
assign S_H[10] = 40'b0000000000000000000111000000000000000000;
assign S_H[11] = 40'b0000000000000000000111000000000000000000;
assign S_H[12] = 40'b0000000000000000000111000000000000000000;
assign S_H[13] = 40'b0000000000000000000111000000000000000000;
assign S_H[14] = 40'b0000000000000000000111000000000000000000;
assign S_H[15] = 40'b0000000000000000000111000000000000000000;
assign S_H[16] = 40'b0000000000000000000111000000000000000000;
assign S_H[17] = 40'b0000000001111111111111111111111000000000;
assign S_H[18] = 40'b0000000011111111111111111111111100000000;
assign S_H[19] = 40'b0000000001111111111111111111111000000000;
assign S_H[20] = 40'b0000000000000000000000000000000000000000;
```

```
assign S_H[21] = 40'b0000000000000000000000000000000000000000;
assign S_H[22] = 40'b0000000000000000000000000000000000000000;
assign S_H[23] = 40'b0000000000000000000000000000000000000000;
assign S_H[24] = 40'b0000000000000000000000000000000000000000;
assign S_I[0] = 40'b0000000000000000000000000000000000000000;
assign S_I[1] = 40'b0000000000000000000000000000000000000000;
assign S_I[2] = 40'b0000000000000000000000000000000000000000;
assign S_I[3] = 40'b0000000001111111111111111111111000000000;
assign S_I[4] = 40'b0000000011111111111111111111111100000000;
assign S_I[5] = 40'b0000000001111111111111111111111000000000;
assign S_I[6] = 40'b0000000000000000000000000000000000000000;
assign S_I[7] = 40'b0000000000000000000000000000000000000000;
assign S_I[8] = 40'b0000000000000000000000000000000000000000;
assign S_I[9] = 40'b0000000000000000000000000000000000000000;
assign S_I[10] = 40'b0000000000000000000000000000000000000000;
assign S_I[11] = 40'b0000000000000000000000000000000000000000;
assign S_I[12] = 40'b0000000000000000000000000000000000000000;
assign S_I[13] = 40'b0000000000000000000000000000000000000000;
assign S_I[14] = 40'b0000000000000000000000000000000000000000;
assign S_I[15] = 40'b0000000000000000000000000000000000000000;
assign S_I[16] = 40'b0000000000000000000000000000000000000000;
assign S_I[17] = 40'b0000000000000000000000000000000000000000;
assign S_I[18] = 40'b0000000000000000000000000000000000000000;
assign S_I[19] = 40'b0000000000000000000000000000000000000000;
assign S_I[20] = 40'b0000000000000000000000000000000000000000;
assign S_I[21] = 40'b0000000000000000000000000000000000000000;
assign S_I[22] = 40'b0000000000000000000000000000000000000000;
assign S_I[23] = 40'b0000000000000000000000000000000000000000;
assign S_I[24] = 40'b0000000000000000000000000000000000000000;
assign S_J[0] = 40'b0000000000000000000000000000000000000000;
assign S_J[1] = 40'b0000000000000111100000000000000000000000;
assign S_J[2] = 40'b0000000000111111110000000000000000000000;
assign S_J[3] = 40'b0000000001111111110000000000000000000000;
assign S_J[4] = 40'b0000000001111000000000000000000000000000;
assign S_J[5] = 40'b0000000001110000000000000000000000000000;
assign S_J[6] = 40'b0000000011100000000000000000000000000000;
assign S_J[7] = 40'b0000000011100000000000000000000000000000;
assign S_J[8] = 40'b0000000011100000000000000000000000000000;
assign S_J[9] = 40'b0000000001110000000000000000000000000000;
assign S_J[10] = 40'b0000000001111000000000000000000000000000;
assign S_J[11] = 40'b0000000001111111111111111111111000000000;
assign S_J[12] = 40'b0000000000111111111111111111111100000000;
assign S_J[13] = 40'b0000000000000111111111111111111000000000;
```

```
assign S_J[14] = 40'b0000000000000000000000000000000000000000;
assign S_J[15] = 40'b0000000000000000000000000000000000000000;
assign S_J[16] = 40'b0000000000000000000000000000000000000000;
assign S_J[17] = 40'b0000000000000000000000000000000000000000;
assign S_J[18] = 40'b0000000000000000000000000000000000000000;
assign S_J[19] = 40'b0000000000000000000000000000000000000000;
assign S_J[20] = 40'b0000000000000000000000000000000000000000;
assign S_J[21] = 40'b0000000000000000000000000000000000000000;
assign S_J[22] = 40'b0000000000000000000000000000000000000000;
assign S_J[23] = 40'b0000000000000000000000000000000000000000;
assign S_J[24] = 40'b0000000000000000000000000000000000000000;
assign S_K[0] = 40'b0000000000000000000000000000000000000000;
assign S_K[1] = 40'b0000000000000000000000000000000000000000;
assign S_K[2] = 40'b0000000000000000000000000000000000000000;
assign S_K[3] = 40'b0000000001111111111111111111111111000000000;
assign S_K[4] = 40'b0000000011111111111111111111111111100000000;
assign S_K[5] = 40'b0000000001111111111111111111111111000000000;
assign S_K[6] = 40'b0000000000000000111110000000000000000000000;
assign S_K[7] = 40'b0000000000000000011111000000000000000000000;
assign S_K[8] = 40'b0000000000000000011111000000000000000000000;
assign S_K[9] = 40'b0000000000000000001111100000000000000000000;
assign S_K[10] = 40'b0000000000000000011111111100000000000000000;
assign S_K[11] = 40'b0000000000000000111111111100000000000000000;
assign S_K[12] = 40'b0000000000000001111100111110000000000000000;
assign S_K[13] = 40'b0000000000000111111000011111000000000000000;
assign S_K[14] = 40'b0000000000001111110000011111000000000000000;
assign S_K[15] = 40'b0000000000011111000000000111110000000000000;
assign S_K[16] = 40'b0000000001111110000000000011111000000000000;
assign S_K[17] = 40'b0000000011111100000000000001111100000000000;
assign S_K[18] = 40'b0000000011110000000000000000111100000000000;
assign S_K[19] = 40'b0000000001100000000000000000011000000000000;
assign S_K[20] = 40'b0000000000000000000000000000000000000000000;
assign S_K[21] = 40'b0000000000000000000000000000000000000000000;
assign S_K[22] = 40'b0000000000000000000000000000000000000000000;
assign S_K[23] = 40'b0000000000000000000000000000000000000000000;
assign S_K[24] = 40'b0000000000000000000000000000000000000000000;
assign S_L[0] = 40'b0000000000000000000000000000000000000000;
assign S_L[1] = 40'b0000000000000000000000000000000000000000;
assign S_L[2] = 40'b0000000000000000000000000000000000000000;
assign S_L[3] = 40'b0000000001111111111111111111111111000000000;
assign S_L[4] = 40'b0000000011111111111111111111111111100000000;
assign S_L[5] = 40'b0000000011111111111111111111111111000000000;
assign S_L[6] = 40'b0000000011100000000000000000000000000000000;
```

```
assign S_L[7] = 40'b0000000011100000000000000000000000000000;
assign S_L[8] = 40'b0000000011100000000000000000000000000000;
assign S_L[9] = 40'b0000000011100000000000000000000000000000;
assign S_L[10] = 40'b0000000011100000000000000000000000000000;
assign S_L[11] = 40'b0000000011100000000000000000000000000000;
assign S_L[12] = 40'b0000000011100000000000000000000000000000;
assign S_L[13] = 40'b0000000011100000000000000000000000000000;
assign S_L[14] = 40'b0000000011100000000000000000000000000000;
assign S_L[15] = 40'b0000000011100000000000000000000000000000;
assign S_L[16] = 40'b0000000011100000000000000000000000000000;
assign S_L[17] = 40'b0000000001000000000000000000000000000000;
assign S_L[18] = 40'b0000000000000000000000000000000000000000;
assign S_L[19] = 40'b0000000000000000000000000000000000000000;
assign S_L[20] = 40'b0000000000000000000000000000000000000000;
assign S_L[21] = 40'b0000000000000000000000000000000000000000;
assign S_L[22] = 40'b0000000000000000000000000000000000000000;
assign S_L[23] = 40'b0000000000000000000000000000000000000000;
assign S_L[24] = 40'b0000000000000000000000000000000000000000;
assign S_M[0] = 40'b0000000000000000000000000000000000000000;
assign S_M[1] = 40'b0000000000000000000000000000000000000000;
assign S_M[2] = 40'b0000000000000000000000000000000000000000;
assign S_M[3] = 40'b0000000001111111111111111111111000000000;
assign S_M[4] = 40'b0000000011111111111111111111111100000000;
assign S_M[5] = 40'b0000000001111111111111111111111000000000;
assign S_M[6] = 40'b0000000000000000000011111111100000000000;
assign S_M[7] = 40'b0000000000000000000111111100000000000000;
assign S_M[8] = 40'b0000000000000000111111110000000000000000;
assign S_M[9] = 40'b0000000000000011111111000000000000000000;
assign S_M[10] = 40'b0000000000011111111000000000000000000000;
assign S_M[11] = 40'b0000000001111111000000000000000000000000;
assign S_M[12] = 40'b0000000011111000000000000000000000000000;
assign S_M[13] = 40'b0000000001111111000000000000000000000000;
assign S_M[14] = 40'b0000000000011111111000000000000000000000;
assign S_M[15] = 40'b0000000000000011111111000000000000000000;
assign S_M[16] = 40'b0000000000000000111111110000000000000000;
assign S_M[17] = 40'b0000000000000000000111111100000000000000;
assign S_M[18] = 40'b0000000000000000000011111111100000000000;
assign S_M[19] = 40'b0000000001111111111111111111111000000000;
assign S_M[20] = 40'b0000000011111111111111111111111100000000;
assign S_M[21] = 40'b0000000001111111111111111111111000000000;
assign S_M[22] = 40'b0000000000000000000000000000000000000000;
assign S_M[23] = 40'b0000000000000000000000000000000000000000;
assign S_M[24] = 40'b0000000000000000000000000000000000000000;
```

```
assign S_N[0] = 40'b0000000000000000000000000000000000000000;
assign S_N[1] = 40'b0000000000000000000000000000000000000000;
assign S_N[2] = 40'b0000000000000000000000000000000000000000;
assign S_N[3] = 40'b0000000001111111111111111111111000000000;
assign S_N[4] = 40'b0000000011111111111111111111111100000000;
assign S_N[5] = 40'b0000000001111111111111111111111100000000;
assign S_N[6] = 40'b0000000000000000000000000111110000000000;
assign S_N[7] = 40'b0000000000000000000000001111110000000000;
assign S_N[8] = 40'b0000000000000000000000111110000000000000;
assign S_N[9] = 40'b0000000000000000000001111110000000000000;
assign S_N[10] = 40'b0000000000000000000111110000000000000000;
assign S_N[11] = 40'b0000000000000000001111110000000000000000;
assign S_N[12] = 40'b0000000000000000111111000000000000000000;
assign S_N[13] = 40'b0000000000000001111110000000000000000000;
assign S_N[14] = 40'b0000000000000111111000000000000000000000;
assign S_N[15] = 40'b0000000000011111100000000000000000000000;
assign S_N[16] = 40'b0000000000111111000000000000000000000000;
assign S_N[17] = 40'b0000000011111111111111111111111000000000;
assign S_N[18] = 40'b0000000011111111111111111111111100000000;
assign S_N[19] = 40'b0000000001111111111111111111111000000000;
assign S_N[20] = 40'b0000000000000000000000000000000000000000;
assign S_N[21] = 40'b0000000000000000000000000000000000000000;
assign S_N[22] = 40'b0000000000000000000000000000000000000000;
assign S_N[23] = 40'b0000000000000000000000000000000000000000;
assign S_N[24] = 40'b0000000000000000000000000000000000000000;
assign S_O[0] = 40'b0000000000000000000000000000000000000000;
assign S_O[1] = 40'b0000000000000000000000000000000000000000;
assign S_O[2] = 40'b0000000000000000111111110000000000000000;
assign S_O[3] = 40'b0000000000000011111111111100000000000000;
assign S_O[4] = 40'b0000000000011111111111111111100000000000;
assign S_O[5] = 40'b0000000000111111000000001111110000000000;
assign S_O[6] = 40'b0000000001111100000000000011111000000000;
assign S_O[7] = 40'b0000000001111000000000000001111000000000;
assign S_O[8] = 40'b0000000001110000000000000000111000000000;
assign S_O[9] = 40'b0000000011100000000000000000011100000000;
assign S_O[10] = 40'b0000000011100000000000000000011100000000;
assign S_O[11] = 40'b0000000011100000000000000000011100000000;
assign S_O[12] = 40'b0000000011100000000000000000011100000000;
assign S_O[13] = 40'b0000000011100000000000000000011100000000;
assign S_O[14] = 40'b0000000001110000000000000000111000000000;
assign S_O[15] = 40'b0000000001111000000000000001111000000000;
assign S_O[16] = 40'b0000000001111100000000000011111000000000;
assign S_O[17] = 40'b0000000000111111000000001111110000000000;
```

```
assign S_O[18] = 40'b0000000000011111111111111111100000000000;
assign S_O[19] = 40'b0000000000000011111111111100000000000000;
assign S_O[20] = 40'b0000000000000000111111110000000000000000;
assign S_O[21] = 40'b0000000000000000000000000000000000000000;
assign S_O[22] = 40'b0000000000000000000000000000000000000000;
assign S_O[23] = 40'b0000000000000000000000000000000000000000;
assign S_O[24] = 40'b0000000000000000000000000000000000000000;
assign S_P[0] = 40'b0000000000000000000000000000000000000000;
assign S_P[1] = 40'b0000000000000000000000000000000000000000;
assign S_P[2] = 40'b0000000000000000000000000000000000000000;
assign S_P[3] = 40'b0000000001111111111111111111111000000000;
assign S_P[4] = 40'b0000000011111111111111111111111100000000;
assign S_P[5] = 40'b0000000001111111111111111111111100000000;
assign S_P[6] = 40'b0000000000000000001110000000011100000000;
assign S_P[7] = 40'b0000000000000000001110000000011100000000;
assign S_P[8] = 40'b0000000000000000001110000000011100000000;
assign S_P[9] = 40'b0000000000000000001110000000011100000000;
assign S_P[10] = 40'b0000000000000000001110000000011100000000;
assign S_P[11] = 40'b0000000000000000001110000000011100000000;
assign S_P[12] = 40'b0000000000000000001110000000011100000000;
assign S_P[13] = 40'b0000000000000000001110000000011100000000;
assign S_P[14] = 40'b0000000000000000001110000000011100000000;
assign S_P[15] = 40'b0000000000000000001110000001111000000000;
assign S_P[16] = 40'b0000000000000000001111000011111000000000;
assign S_P[17] = 40'b0000000000000000001111111111111000000000;
assign S_P[18] = 40'b0000000000000000000111111111110000000000;
assign S_P[19] = 40'b0000000000000000000000011110000000000000;
assign S_P[20] = 40'b0000000000000000000000000000000000000000;
assign S_P[21] = 40'b0000000000000000000000000000000000000000;
assign S_P[22] = 40'b0000000000000000000000000000000000000000;
assign S_P[23] = 40'b0000000000000000000000000000000000000000;
assign S_P[24] = 40'b0000000000000000000000000000000000000000;
assign S_Q[0] = 40'b0000000000000000000000000000000000000000;
assign S_Q[1] = 40'b0000000000000000000000000000000000000000;
assign S_Q[2] = 40'b0000000000000000111111110000000000000000;
assign S_Q[3] = 40'b0000000000000011111111111110000000000000;
assign S_Q[4] = 40'b0000000000011111111111111111100000000000;
assign S_Q[5] = 40'b0000000000111111000000001111110000000000;
assign S_Q[6] = 40'b0000000001111100000000000011111000000000;
assign S_Q[7] = 40'b0000000001111000000000000001111000000000;
assign S_Q[8] = 40'b0000000001110000000000000000111000000000;
assign S_Q[9] = 40'b0000000011100000000000000000011100000000;
assign S_Q[10] = 40'b0000000011100000000000000000011100000000;
```

```
assign S_Q[11] = 40'b0000000011101100000000000000011100000000;
assign S_Q[12] = 40'b0000000011111110000000000000011100000000;
assign S_Q[13] = 40'b0000000011111110000000000000011100000000;
assign S_Q[14] = 40'b0000000001111100000000000000111000000000;
assign S_Q[15] = 40'b0000000011111000000000000001111000000000;
assign S_Q[16] = 40'b0000000111111100000000000011111000000000;
assign S_Q[17] = 40'b0000001111111111000000001111110000000000;
assign S_Q[18] = 40'b0000001111011111111111111111100000000000;
assign S_Q[19] = 40'b0000000110000011111111111100000000000000;
assign S_Q[20] = 40'b0000000000000000111111110000000000000000;
assign S_Q[21] = 40'b0000000000000000000000000000000000000000;
assign S_Q[22] = 40'b0000000000000000000000000000000000000000;
assign S_Q[23] = 40'b0000000000000000000000000000000000000000;
assign S_Q[24] = 40'b0000000000000000000000000000000000000000;
assign S_R[0] = 40'b0000000000000000000000000000000000000000;
assign S_R[1] = 40'b0000000000000000000000000000000000000000;
assign S_R[2] = 40'b0000000000000000000000000000000000000000;
assign S_R[3] = 40'b0000000001111111111111111111111000000000;
assign S_R[4] = 40'b0000000011111111111111111111111100000000;
assign S_R[5] = 40'b0000000001111111111111111111111100000000;
assign S_R[6] = 40'b0000000000000000000111000000011100000000;
assign S_R[7] = 40'b0000000000000000000111000000011100000000;
assign S_R[8] = 40'b0000000000000000000111000000011100000000;
assign S_R[9] = 40'b0000000000000000000111000000011100000000;
assign S_R[10] = 40'b0000000000000000000111000000011100000000;
assign S_R[11] = 40'b0000000000000000011111000000011100000000;
assign S_R[12] = 40'b0000000000000000111111000000011100000000;
assign S_R[13] = 40'b0000000000000001111111100000011100000000;
assign S_R[14] = 40'b0000000000000011111111100000011100000000;
assign S_R[15] = 40'b0000000000011111100011100000111000000000;
assign S_R[16] = 40'b0000000000111111000011110001111000000000;
assign S_R[17] = 40'b0000000011111110000011111111111000000000;
assign S_R[18] = 40'b0000000011111000000001111111110000000000;
assign S_R[19] = 40'b0000000000110000000000000111000000000000;
assign S_R[20] = 40'b0000000000000000000000000000000000000000;
assign S_R[21] = 40'b0000000000000000000000000000000000000000;
assign S_R[22] = 40'b0000000000000000000000000000000000000000;
assign S_R[23] = 40'b0000000000000000000000000000000000000000;
assign S_R[24] = 40'b0000000000000000000000000000000000000000;
assign S_S[0] = 40'b0000000000000000000000000000000000000000;
assign S_S[1] = 40'b0000000000000000000000000000000000000000;
assign S_S[2] = 40'b0000000000001100000000000011110000000000;
assign S_S[3] = 40'b0000000000111100000000111111110000000000;
```

```
assign S_S[4] = 40'b0000000001111100000001111111111000000000;
assign S_S[5] = 40'b0000000001111000000001111001111000000000;
assign S_S[6] = 40'b0000000001110000000001110000111000000000;
assign S_S[7] = 40'b0000000011100000000011100000011100000000;
assign S_S[8] = 40'b0000000011100000000011100000011100000000;
assign S_S[9] = 40'b0000000011100000000011100000011100000000;
assign S_S[10] = 40'b0000000011100000000011100000011100000000;
assign S_S[11] = 40'b0000000011100000000011100000011100000000;
assign S_S[12] = 40'b0000000011100000001110000000011100000000;
assign S_S[13] = 40'b0000000011100000001110000000011100000000;
assign S_S[14] = 40'b0000000001110000011100000000111000000000;
assign S_S[15] = 40'b0000000001111000111100000001111000000000;
assign S_S[16] = 40'b0000000001111111111100000011111000000000;
assign S_S[17] = 40'b0000000000111111111000000011110000000000;
assign S_S[18] = 40'b0000000000001111100000000001100000000000;
assign S_S[19] = 40'b0000000000000000000000000000000000000000;
assign S_S[20] = 40'b0000000000000000000000000000000000000000;
assign S_S[21] = 40'b0000000000000000000000000000000000000000;
assign S_S[22] = 40'b0000000000000000000000000000000000000000;
assign S_S[23] = 40'b0000000000000000000000000000000000000000;
assign S_S[24] = 40'b0000000000000000000000000000000000000000;
assign S_T[0] = 40'b0000000000000000000000000000001000000000;
assign S_T[1] = 40'b0000000000000000000000000000011100000000;
assign S_T[2] = 40'b0000000000000000000000000000011100000000;
assign S_T[3] = 40'b0000000000000000000000000000011100000000;
assign S_T[4] = 40'b0000000000000000000000000000011100000000;
assign S_T[5] = 40'b0000000000000000000000000000011100000000;
assign S_T[6] = 40'b0000000000000000000000000000011100000000;
assign S_T[7] = 40'b0000000001111111111111111111111100000000;
assign S_T[8] = 40'b0000000011111111111111111111111100000000;
assign S_T[9] = 40'b0000000001111111111111111111111100000000;
assign S_T[10] = 40'b0000000000000000000000000000011100000000;
assign S_T[11] = 40'b0000000000000000000000000000011100000000;
assign S_T[12] = 40'b0000000000000000000000000000011100000000;
assign S_T[13] = 40'b0000000000000000000000000000011100000000;
assign S_T[14] = 40'b0000000000000000000000000000011100000000;
assign S_T[15] = 40'b0000000000000000000000000000011100000000;
assign S_T[16] = 40'b0000000000000000000000000000001000000000;
assign S_T[17] = 40'b0000000000000000000000000000000000000000;
assign S_T[18] = 40'b0000000000000000000000000000000000000000;
assign S_T[19] = 40'b0000000000000000000000000000000000000000;
assign S_T[20] = 40'b0000000000000000000000000000000000000000;
assign S_T[21] = 40'b0000000000000000000000000000000000000000;
```

```
assign S_T[22] = 40'b0000000000000000000000000000000000000000;
assign S_T[23] = 40'b0000000000000000000000000000000000000000;
assign S_T[24] = 40'b0000000000000000000000000000000000000000;
assign S_U[0] = 40'b0000000000000000000000000000000000000000;
assign S_U[1] = 40'b0000000000000000000000000000000000000000;
assign S_U[2] = 40'b0000000000000000000000000000000000000000;
assign S_U[3] = 40'b0000000000000011111111111111111111000000000;
assign S_U[4] = 40'b0000000000011111111111111111111100000000;
assign S_U[5] = 40'b0000000000111111111111111111111000000000;
assign S_U[6] = 40'b0000000001111100000000000000000000000000;
assign S_U[7] = 40'b0000000001111000000000000000000000000000;
assign S_U[8] = 40'b0000000001110000000000000000000000000000;
assign S_U[9] = 40'b0000000011100000000000000000000000000000;
assign S_U[10] = 40'b0000000011100000000000000000000000000000;
assign S_U[11] = 40'b0000000011100000000000000000000000000000;
assign S_U[12] = 40'b0000000011100000000000000000000000000000;
assign S_U[13] = 40'b0000000011100000000000000000000000000000;
assign S_U[14] = 40'b0000000001110000000000000000000000000000;
assign S_U[15] = 40'b0000000001111000000000000000000000000000;
assign S_U[16] = 40'b0000000001111100000000000000000000000000;
assign S_U[17] = 40'b0000000000111111111111111111111000000000;
assign S_U[18] = 40'b0000000000011111111111111111111100000000;
assign S_U[19] = 40'b0000000000000011111111111111111000000000;
assign S_U[20] = 40'b0000000000000000000000000000000000000000;
assign S_U[21] = 40'b0000000000000000000000000000000000000000;
assign S_U[22] = 40'b0000000000000000000000000000000000000000;
assign S_U[23] = 40'b0000000000000000000000000000000000000000;
assign S_U[24] = 40'b0000000000000000000000000000000000000000;
assign S_V[0] = 40'b0000000000000000000000000011000000000;
assign S_V[1] = 40'b0000000000000000000000001111100000000;
assign S_V[2] = 40'b0000000000000000000000111111000000000;
assign S_V[3] = 40'b0000000000000000000011111110000000000;
assign S_V[4] = 40'b0000000000000000001111111000000000000;
assign S_V[5] = 40'b0000000000000000111111100000000000000;
assign S_V[6] = 40'b0000000000000011111110000000000000000;
assign S_V[7] = 40'b0000000000011111111000000000000000000;
assign S_V[8] = 40'b0000000001111111000000000000000000000;
assign S_V[9] = 40'b0000000011111000000000000000000000000;
assign S_V[10] = 40'b0000000001111111000000000000000000000;
assign S_V[11] = 40'b0000000000011111111000000000000000000;
assign S_V[12] = 40'b0000000000000011111111000000000000000;
assign S_V[13] = 40'b0000000000000000111111110000000000000;
assign S_V[14] = 40'b0000000000000000001111111000000000000;
```

```
assign S_V[15] = 40'b0000000000000000000001111111100000000000;
assign S_V[16] = 40'b0000000000000000000000011111111000000000;
assign S_V[17] = 40'b0000000000000000000000000011111100000000;
assign S_V[18] = 40'b0000000000000000000000000000011000000000;
assign S_V[19] = 40'b0000000000000000000000000000000000000000;
assign S_V[20] = 40'b0000000000000000000000000000000000000000;
assign S_V[21] = 40'b0000000000000000000000000000000000000000;
assign S_V[22] = 40'b0000000000000000000000000000000000000000;
assign S_V[23] = 40'b0000000000000000000000000000000000000000;
assign S_V[24] = 40'b0000000000000000000000000000000000000000;
assign S_W[0] = 40'b0000000000000000000000000000000000000000;
assign S_W[1] = 40'b0000000000000000000000000000011000000000;
assign S_W[2] = 40'b0000000000000000000000001111111100000000;
assign S_W[3] = 40'b0000000000000000000011111111111000000000;
assign S_W[4] = 40'b0000000000000000111111111111110000000000;
assign S_W[5] = 40'b0000000000001111111111111000000000000000;
assign S_W[6] = 40'b0000000001111111111100000000000000000000;
assign S_W[7] = 40'b0000000011111111000000000000000000000000;
assign S_W[8] = 40'b0000000001111111111000000000000000000000;
assign S_W[9] = 40'b0000000000001111111111110000000000000000;
assign S_W[10] = 40'b0000000000000000111111111111000000000000;
assign S_W[11] = 40'b0000000000000000000111111111111000000000;
assign S_W[12] = 40'b0000000000000000000000001111111100000000;
assign S_W[13] = 40'b0000000000000000000011111111111000000000;
assign S_W[14] = 40'b0000000000000000111111111111000000000000;
assign S_W[15] = 40'b0000000000001111111111110000000000000000;
assign S_W[16] = 40'b0000000001111111111100000000000000000000;
assign S_W[17] = 40'b0000000011111111000000000000000000000000;
assign S_W[18] = 40'b0000000001111111111100000000000000000000;
assign S_W[19] = 40'b0000000000001111111111110000000000000000;
assign S_W[20] = 40'b0000000000000000111111111111000000000000;
assign S_W[21] = 40'b0000000000000000000111111111111000000000;
assign S_W[22] = 40'b0000000000000000000000001111111100000000;
assign S_W[23] = 40'b0000000000000000000000000000011000000000;
assign S_W[24] = 40'b0000000000000000000000000000000000000000;
assign S_X[0] = 40'b0000000000000000000000000000000000000000;
assign S_X[1] = 40'b0000000000000000000000000000000000000000;
assign S_X[2] = 40'b0000000001100000000000000000011000000000;
assign S_X[3] = 40'b0000000011110000000000000000111100000000;
assign S_X[4] = 40'b0000000011111100000000000011111100000000;
assign S_X[5] = 40'b0000000000111111000000000111111000000000;
assign S_X[6] = 40'b0000000000001111110000011111110000000000;
assign S_X[7] = 40'b0000000000000111111000011111100000000000;
```

```
assign S_X[8] = 40'b0000000000000011111111111100000000000000;
assign S_X[9] = 40'b0000000000000001111111100000000000000000;
assign S_X[10] = 40'b0000000000000000111111000000000000000000;
assign S_X[11] = 40'b0000000000000001111111100000000000000000;
assign S_X[12] = 40'b0000000000000011111111111100000000000000;
assign S_X[13] = 40'b0000000000000111110000111110000000000000;
assign S_X[14] = 40'b0000000000011111100000011111100000000000;
assign S_X[15] = 40'b0000000000111111000000000111110000000000;
assign S_X[16] = 40'b0000000011111100000000000011111100000000;
assign S_X[17] = 40'b0000000011110000000000000000111100000000;
assign S_X[18] = 40'b0000000001100000000000000000011000000000;
assign S_X[19] = 40'b0000000000000000000000000000000000000000;
assign S_X[20] = 40'b0000000000000000000000000000000000000000;
assign S_X[21] = 40'b0000000000000000000000000000000000000000;
assign S_X[22] = 40'b0000000000000000000000000000000000000000;
assign S_X[23] = 40'b0000000000000000000000000000000000000000;
assign S_X[24] = 40'b0000000000000000000000000000000000000000;
assign S_Y[0] = 40'b0000000000000000000000000000011000000000;
assign S_Y[1] = 40'b0000000000000000000000000000111100000000;
assign S_Y[2] = 40'b0000000000000000000000000001111100000000;
assign S_Y[3] = 40'b0000000000000000000000000011111000000000;
assign S_Y[4] = 40'b0000000000000000000000000111110000000000;
assign S_Y[5] = 40'b0000000000000000000000001111100000000000;
assign S_Y[6] = 40'b0000000000000000000000011111000000000000;
assign S_Y[7] = 40'b0000000000000000000001111110000000000000;
assign S_Y[8] = 40'b0000000001111111111111110000000000000000;
assign S_Y[9] = 40'b0000000011111111111111100000000000000000;
assign S_Y[10] = 40'b0000000001111111111111110000000000000000;
assign S_Y[11] = 40'b0000000000000000000011111100000000000000;
assign S_Y[12] = 40'b0000000000000000000001111100000000000000;
assign S_Y[13] = 40'b0000000000000000000000111110000000000000;
assign S_Y[14] = 40'b0000000000000000000000011111000000000000;
assign S_Y[15] = 40'b0000000000000000000000001111100000000000;
assign S_Y[16] = 40'b0000000000000000000000000111110000000000;
assign S_Y[17] = 40'b0000000000000000000000000011110000000000;
assign S_Y[18] = 40'b0000000000000000000000000001100000000000;
assign S_Y[19] = 40'b0000000000000000000000000000000000000000;
assign S_Y[20] = 40'b0000000000000000000000000000000000000000;
assign S_Y[21] = 40'b0000000000000000000000000000000000000000;
assign S_Y[22] = 40'b0000000000000000000000000000000000000000;
assign S_Y[23] = 40'b0000000000000000000000000000000000000000;
assign S_Y[24] = 40'b0000000000000000000000000000000000000000;
assign S_Z[0] = 40'b0000000000000000000000000000000000000000;
```

```
assign S_Z[1] = 40'b0000000000000000000000000000000000000000;
assign S_Z[2] = 40'b0000000001100000000000000000001000000000;
assign S_Z[3] = 40'b0000000011110000000000000000011100000000;
assign S_Z[4] = 40'b0000000011111100000000000000011100000000;
assign S_Z[5] = 40'b0000000011111110000000000000011100000000;
assign S_Z[6] = 40'b0000000011111111100000000000011100000000;
assign S_Z[7] = 40'b0000000011100111110000000000011100000000;
assign S_Z[8] = 40'b0000000011100011111110000000011100000000;
assign S_Z[9] = 40'b0000000011100000111111000000011100000000;
assign S_Z[10] = 40'b0000000011100000011111100000011100000000;
assign S_Z[11] = 40'b0000000011100000000111111000011100000000;
assign S_Z[12] = 40'b0000000011100000000011111100011100000000;
assign S_Z[13] = 40'b0000000011100000000000111111011100000000;
assign S_Z[14] = 40'b0000000011100000000000011111111100000000;
assign S_Z[15] = 40'b0000000011100000000000000111111100000000;
assign S_Z[16] = 40'b0000000011100000000000000011111100000000;
assign S_Z[17] = 40'b0000000011100000000000000000111100000000;
assign S_Z[18] = 40'b0000000001000000000000000000011000000000;
assign S_Z[19] = 40'b0000000000000000000000000000000000000000;
assign S_Z[20] = 40'b0000000000000000000000000000000000000000;
assign S_Z[21] = 40'b0000000000000000000000000000000000000000;
assign S_Z[22] = 40'b0000000000000000000000000000000000000000;
assign S_Z[23] = 40'b0000000000000000000000000000000000000000;
assign S_Z[24] = 40'b0000000000000000000000000000000000000000;
assign S_colon[0] = 40'b0000000000000000000000000000000000000000;
assign S_colon[1] = 40'b0000000000000000000000000000000000000000;
assign S_colon[2] = 40'b0000000000000000000000000000000000000000;
assign S_colon[3] = 40'b0000000001110000000001110000000000000000;
assign S_colon[4] = 40'b0000000011111000000011111000000000000000;
assign S_colon[5] = 40'b0000000011111000000011111000000000000000;
assign S_colon[6] = 40'b0000000011111000000011111000000000000000;
assign S_colon[7] = 40'b0000000001110000000001110000000000000000;
assign S_colon[8] = 40'b0000000000000000000000000000000000000000;
assign S_colon[9] = 40'b0000000000000000000000000000000000000000;
assign S_colon[10] = 40'b0000000000000000000000000000000000000000;
assign S_colon[11] = 40'b0000000000000000000000000000000000000000;
assign S_colon[12] = 40'b0000000000000000000000000000000000000000;
assign S_colon[13] = 40'b0000000000000000000000000000000000000000;
assign S_colon[14] = 40'b0000000000000000000000000000000000000000;
assign S_colon[15] = 40'b0000000000000000000000000000000000000000;
assign S_colon[16] = 40'b0000000000000000000000000000000000000000;
assign S_colon[17] = 40'b0000000000000000000000000000000000000000;
assign S_colon[18] = 40'b0000000000000000000000000000000000000000;
```

```
assign S_colon[19] = 40'b0000000000000000000000000000000000000000;
assign S_colon[20] = 40'b0000000000000000000000000000000000000000;
assign S_colon[21] = 40'b0000000000000000000000000000000000000000;
assign S_colon[22] = 40'b0000000000000000000000000000000000000000;
assign S_colon[23] = 40'b0000000000000000000000000000000000000000;
assign S_colon[24] = 40'b0000000000000000000000000000000000000000;
assign S_1[0] = 40'b0000000000000000000000000000000000000000;
assign S_1[1] = 40'b0000000000000000000000000000000000000000;
assign S_1[2] = 40'b0000000000000000000000000000000000000000;
assign S_1[3] = 40'b0000000000000000000000000000000000000000;
assign S_1[4] = 40'b0000000000000000000000000000000000000000;
assign S_1[5] = 40'b0000000000000000000000000010000000000000;
assign S_1[6] = 40'b0000000000000000000000000111000000000000;
assign S_1[7] = 40'b0000000000000000000000000111100000000000;
assign S_1[8] = 40'b0000000000000000000000000111110000000000;
assign S_1[9] = 40'b0000000000000000000000000111111000000000;
assign S_1[10] = 40'b0000000001111111111111111111111100000000;
assign S_1[11] = 40'b0000000011111111111111111111111100000000;
assign S_1[12] = 40'b0000000001111111111111111111111000000000;
assign S_1[13] = 40'b0000000000000000000000000000000000000000;
assign S_1[14] = 40'b0000000000000000000000000000000000000000;
assign S_1[15] = 40'b0000000000000000000000000000000000000000;
assign S_1[16] = 40'b0000000000000000000000000000000000000000;
assign S_1[17] = 40'b0000000000000000000000000000000000000000;
assign S_1[18] = 40'b0000000000000000000000000000000000000000;
assign S_1[19] = 40'b0000000000000000000000000000000000000000;
assign S_1[20] = 40'b0000000000000000000000000000000000000000;
assign S_1[21] = 40'b0000000000000000000000000000000000000000;
assign S_1[22] = 40'b0000000000000000000000000000000000000000;
assign S_1[23] = 40'b0000000000000000000000000000000000000000;
assign S_1[24] = 40'b0000000000000000000000000000000000000000;
assign S_2[0] = 40'b0000000000000000000000000000000000000000;
assign S_2[1] = 40'b0000000000000000000000000000000000000000;
assign S_2[2] = 40'b0000000001100000000000000000000000000000;
assign S_2[3] = 40'b0000000011110000000000000110000000000000;
assign S_2[4] = 40'b0000000011111000000000000111111000000000;
assign S_2[5] = 40'b0000000011111100000000000111111000000000;
assign S_2[6] = 40'b0000000011111110000000000001111000000000;
assign S_2[7] = 40'b0000000011111111000000000000111000000000;
assign S_2[8] = 40'b0000000011101111100000000000011100000000;
assign S_2[9] = 40'b0000000011100111110000000000011100000000;
assign S_2[10] = 40'b0000000011100011111000000000011100000000;
assign S_2[11] = 40'b0000000011100001111100000000011100000000;
```

```
assign S_2[12] = 40'b0000000011100000111110000000011100000000;
assign S_2[13] = 40'b0000000011100000011111000000111000000000;
assign S_2[14] = 40'b0000000011100000001111110001111000000000;
assign S_2[15] = 40'b0000000011100000000011111111111000000000;
assign S_2[16] = 40'b0000000011100000000001111111110000000000;
assign S_2[17] = 40'b0000000011100000000000001110000000000000;
assign S_2[18] = 40'b0000000001000000000000000000000000000000;
assign S_2[19] = 40'b0000000000000000000000000000000000000000;
assign S_2[20] = 40'b0000000000000000000000000000000000000000;
assign S_2[21] = 40'b0000000000000000000000000000000000000000;
assign S_2[22] = 40'b0000000000000000000000000000000000000000;
assign S_2[23] = 40'b0000000000000000000000000000000000000000;
assign S_2[24] = 40'b0000000000000000000000000000000000000000;
assign S_3[0] = 40'b0000000000000000000000000000000000000000;
assign S_3[1] = 40'b0000000000000000000000000000000000000000;
assign S_3[2] = 40'b0000000000000100000000000000000000000000;
assign S_3[3] = 40'b0000000000111110000000000000000000000000;
assign S_3[4] = 40'b0000000001111100000000000000001000000000;
assign S_3[5] = 40'b0000000001111000000000000000011100000000;
assign S_3[6] = 40'b0000000001110000000000000000011100000000;
assign S_3[7] = 40'b0000000011100000000000000000011100000000;
assign S_3[8] = 40'b0000000011100000000000000000011100000000;
assign S_3[9] = 40'b0000000011100000000000110000011100000000;
assign S_3[10] = 40'b0000000011100000000001111000011100000000;
assign S_3[11] = 40'b0000000011100000000001111110011100000000;
assign S_3[12] = 40'b0000000011100000000001111111011100000000;
assign S_3[13] = 40'b0000000001110000000001111111111100000000;
assign S_3[14] = 40'b0000000001111000000011100111111100000000;
assign S_3[15] = 40'b0000000001111100000111100011111100000000;
assign S_3[16] = 40'b0000000000111111111111000000111100000000;
assign S_3[17] = 40'b0000000000011111111111000000011000000000;
assign S_3[18] = 40'b0000000000000001111100000000000000000000;
assign S_3[19] = 40'b0000000000000000000000000000000000000000;
assign S_3[20] = 40'b0000000000000000000000000000000000000000;
assign S_3[21] = 40'b0000000000000000000000000000000000000000;
assign S_3[22] = 40'b0000000000000000000000000000000000000000;
assign S_3[23] = 40'b0000000000000000000000000000000000000000;
assign S_3[24] = 40'b0000000000000000000000000000000000000000;
assign S_4[0] = 40'b0000000000000000000000000000000000000000;
assign S_4[1] = 40'b0000000000000000000000000000000000000000;
assign S_4[2] = 40'b0000000000000000110000000000000000000000;
assign S_4[3] = 40'b0000000000000001111000000000000000000000;
assign S_4[4] = 40'b0000000000000011111100000000000000000000;
```

```
assign S_4[5] = 40'b0000000000000001111111000000000000000000;
assign S_4[6] = 40'b0000000000000001111111110000000000000000;
assign S_4[7] = 40'b0000000000000001110111111000000000000000;
assign S_4[8] = 40'b0000000000000001110001111100000000000000;
assign S_4[9] = 40'b0000000000000001110000111111000000000000;
assign S_4[10] = 40'b0000000000000001110000011111100000000000;
assign S_4[11] = 40'b0000000000000001110000000111111000000000;
assign S_4[12] = 40'b0000000001111111111111111111111100000000;
assign S_4[13] = 40'b0000000011111111111111111111111100000000;
assign S_4[14] = 40'b0000000001111111111111111111111000000000;
assign S_4[15] = 40'b0000000000000001110000000000000000000000;
assign S_4[16] = 40'b0000000000000001110000000000000000000000;
assign S_4[17] = 40'b0000000000000001110000000000000000000000;
assign S_4[18] = 40'b0000000000000001110000000000000000000000;
assign S_4[19] = 40'b0000000000000000100000000000000000000000;
assign S_4[20] = 40'b0000000000000000000000000000000000000000;
assign S_4[21] = 40'b0000000000000000000000000000000000000000;
assign S_4[22] = 40'b0000000000000000000000000000000000000000;
assign S_4[23] = 40'b0000000000000000000000000000000000000000;
assign S_4[24] = 40'b0000000000000000000000000000000000000000;
assign S_5[0] = 40'b0000000000000000000000000000000000000000;
assign S_5[1] = 40'b0000000000000000000000000000000000000000;
assign S_5[2] = 40'b0000000000000100000000000000000000000000;
assign S_5[3] = 40'b0000000001111100000011111000000000000000;
assign S_5[4] = 40'b0000000001111100000011111111111000000000;
assign S_5[5] = 40'b0000000001111000000011111111111100000000;
assign S_5[6] = 40'b0000000001110000000001110011111100000000;
assign S_5[7] = 40'b0000000011100000000000111000011100000000;
assign S_5[8] = 40'b0000000011100000000000111000011100000000;
assign S_5[9] = 40'b0000000011100000000000111000011100000000;
assign S_5[10] = 40'b0000000011100000000000111000011100000000;
assign S_5[11] = 40'b0000000011100000000000111000011100000000;
assign S_5[12] = 40'b0000000011100000000000111000011100000000;
assign S_5[13] = 40'b0000000001110000000001110000011100000000;
assign S_5[14] = 40'b0000000001111000000011110000011100000000;
assign S_5[15] = 40'b0000000001111100000111110000011100000000;
assign S_5[16] = 40'b0000000000111111111111000000001000000000;
assign S_5[17] = 40'b0000000000011111111110000000000000000000;
assign S_5[18] = 40'b0000000000000001111100000000000000000000;
assign S_5[19] = 40'b0000000000000000000000000000000000000000;
assign S_5[20] = 40'b0000000000000000000000000000000000000000;
assign S_5[21] = 40'b0000000000000000000000000000000000000000;
assign S_5[22] = 40'b0000000000000000000000000000000000000000;
```

```
assign S_5[23] = 40'b0000000000000000000000000000000000000000;
assign S_5[24] = 40'b0000000000000000000000000000000000000000;
assign S_6[0] = 40'b0000000000000000000000000000000000000000;
assign S_6[1] = 40'b0000000000000000000000000000000000000000;
assign S_6[2] = 40'b0000000000000000000000000000000000000000;
assign S_6[3] = 40'b0000000000000001111111110000000000000000;
assign S_6[4] = 40'b0000000000011111111111111111000000000000;
assign S_6[5] = 40'b0000000000111111111111111111100000000000;
assign S_6[6] = 40'b0000000001111100001111101111111000000000;
assign S_6[7] = 40'b0000000001111000000111100001111000000000;
assign S_6[8] = 40'b0000000001110000000011100000111000000000;
assign S_6[9] = 40'b0000000011100000000001110000011100000000;
assign S_6[10] = 40'b0000000011100000000001110000011100000000;
assign S_6[11] = 40'b0000000011100000000001110000011100000000;
assign S_6[12] = 40'b0000000011100000000001110000011100000000;
assign S_6[13] = 40'b0000000001110000000011100000011100000000;
assign S_6[14] = 40'b0000000001111000000111100000111000000000;
assign S_6[15] = 40'b0000000001111100001111100001111000000000;
assign S_6[16] = 40'b0000000000111111111111000011110000000000;
assign S_6[17] = 40'b0000000000011111111110000001000000000000;
assign S_6[18] = 40'b0000000000000001111000000000000000000000;
assign S_6[19] = 40'b0000000000000000000000000000000000000000;
assign S_6[20] = 40'b0000000000000000000000000000000000000000;
assign S_6[21] = 40'b0000000000000000000000000000000000000000;
assign S_6[22] = 40'b0000000000000000000000000000000000000000;
assign S_6[23] = 40'b0000000000000000000000000000000000000000;
assign S_6[24] = 40'b0000000000000000000000000000000000000000;
assign S_7[0] = 40'b0000000000000000000000000000000000000000;
assign S_7[1] = 40'b0000000000000000000000000000000000000000;
assign S_7[2] = 40'b0000000000000000000000000000001000000000;
assign S_7[3] = 40'b0000000000000000000000000000011100000000;
assign S_7[4] = 40'b0000000000000000000000000000011100000000;
assign S_7[5] = 40'b0000000000000000000000000000011100000000;
assign S_7[6] = 40'b0000000001000000000000000000011100000000;
assign S_7[7] = 40'b0000000011110000000000000000011100000000;
assign S_7[8] = 40'b0000000001111110000000000000011100000000;
assign S_7[9] = 40'b0000000000011111100000000000011100000000;
assign S_7[10] = 40'b0000000000001111110000000000011100000000;
assign S_7[11] = 40'b0000000000000011111100000000011100000000;
assign S_7[12] = 40'b0000000000000000111111000000011100000000;
assign S_7[13] = 40'b0000000000000000001111110000011100000000;
assign S_7[14] = 40'b0000000000000000000011111110011100000000;
assign S_7[15] = 40'b0000000000000000000000111111111100000000;
```

```
assign S_7[16] = 40'b0000000000000000000000000111111100000000;
assign S_7[17] = 40'b0000000000000000000000000000111100000000;
assign S_7[18] = 40'b0000000000000000000000000000001000000000;
assign S_7[19] = 40'b0000000000000000000000000000000000000000;
assign S_7[20] = 40'b0000000000000000000000000000000000000000;
assign S_7[21] = 40'b0000000000000000000000000000000000000000;
assign S_7[22] = 40'b0000000000000000000000000000000000000000;
assign S_7[23] = 40'b0000000000000000000000000000000000000000;
assign S_7[24] = 40'b0000000000000000000000000000000000000000;
assign S_8[0] = 40'b0000000000000000000000000000000000000000;
assign S_8[1] = 40'b0000000000000000000000000000000000000000;
assign S_8[2] = 40'b0000000000000111100000000000000000000000;
assign S_8[3] = 40'b0000000000111111111100000111000000000000;
assign S_8[4] = 40'b0000000001111111111110011111110000000000;
assign S_8[5] = 40'b0000000001111000011111111111111000000000;
assign S_8[6] = 40'b0000000001110000011111100001110000000000;
assign S_8[7] = 40'b0000000011100000000111110000011100000000;
assign S_8[8] = 40'b0000000011100000000011110000011100000000;
assign S_8[9] = 40'b0000000011100000000011100000011100000000;
assign S_8[10] = 40'b0000000011100000000011100000011100000000;
assign S_8[11] = 40'b0000000011100000000011100000011100000000;
assign S_8[12] = 40'b0000000011100000000011110000011100000000;
assign S_8[13] = 40'b0000000011100000000111110000011100000000;
assign S_8[14] = 40'b0000000001110000011111110000111000000000;
assign S_8[15] = 40'b0000000001111000111111111111111000000000;
assign S_8[16] = 40'b0000000001111111111110011111110000000000;
assign S_8[17] = 40'b0000000000111111111100000111000000000000;
assign S_8[18] = 40'b0000000000000011110000000000000000000000;
assign S_8[19] = 40'b0000000000000000000000000000000000000000;
assign S_8[20] = 40'b0000000000000000000000000000000000000000;
assign S_8[21] = 40'b0000000000000000000000000000000000000000;
assign S_8[22] = 40'b0000000000000000000000000000000000000000;
assign S_8[23] = 40'b0000000000000000000000000000000000000000;
assign S_8[24] = 40'b0000000000000000000000000000000000000000;
assign S_9[0] = 40'b0000000000000000000000000000000000000000;
assign S_9[1] = 40'b0000000000000000000000000000000000000000;
assign S_9[2] = 40'b0000000000000000000001111000000000000000;
assign S_9[3] = 40'b0000000000001000000111111111100000000000;
assign S_9[4] = 40'b0000000000111100001111111111110000000000;
assign S_9[5] = 40'b0000000001111000011111000011111000000000;
assign S_9[6] = 40'b0000000001110000011111000000111000000000;
assign S_9[7] = 40'b0000000011100000011100000000111000000000;
assign S_9[8] = 40'b0000000011100000111000000000011100000000;
```

```
        assign S_9[9] = 40'b0000000011100000111000000000011100000000;
        assign S_9[10] = 40'b0000000011100000111000000000011100000000;
        assign S_9[11] = 40'b0000000011100000111000000000011100000000;
        assign S_9[12] = 40'b0000000001110000011100000000111000000000;
        assign S_9[13] = 40'b0000000001111000011110000001111000000000;
        assign S_9[14] = 40'b0000000001111111011111000011111000000000;
        assign S_9[15] = 40'b0000000000011111111111111111110000000000;
        assign S_9[16] = 40'b0000000000001111111111111111100000000000;
        assign S_9[17] = 40'b0000000000000000111111111000000000000000;
        assign S_9[18] = 40'b0000000000000000000000000000000000000000;
        assign S_9[19] = 40'b0000000000000000000000000000000000000000;
        assign S_9[20] = 40'b0000000000000000000000000000000000000000;
        assign S_9[21] = 40'b0000000000000000000000000000000000000000;
        assign S_9[22] = 40'b0000000000000000000000000000000000000000;
        assign S_9[23] = 40'b0000000000000000000000000000000000000000;
        assign S_9[24] = 40'b0000000000000000000000000000000000000000;

        logic[19:0] s_A[14:0];
logic[19:0] s_B[14:0];
logic[19:0] s_C[14:0];
logic[19:0] s_D[14:0];
logic[19:0] s_E[14:0];
logic[19:0] s_F[14:0];
logic[19:0] s_G[14:0];
logic[19:0] s_H[14:0];
logic[19:0] s_I[14:0];
logic[19:0] s_J[14:0];
logic[19:0] s_K[14:0];
logic[19:0] s_L[14:0];
logic[19:0] s_M[14:0];
logic[19:0] s_N[14:0];
logic[19:0] s_O[14:0];
logic[19:0] s_P[14:0];
logic[19:0] s_Q[14:0];
logic[19:0] s_R[14:0];
logic[19:0] s_S[14:0];
logic[19:0] s_T[14:0];
logic[19:0] s_U[14:0];
logic[19:0] s_V[14:0];
logic[19:0] s_W[14:0];
logic[19:0] s_X[14:0];
logic[19:0] s_Y[14:0];
logic[19:0] s_Z[14:0];
```

```
logic[19:0] s_dot[14:0];
logic[19:0] s_colon[14:0];
logic[19:0] s_blank [14:0];
logic[19:0] s_1[14:0];
logic[19:0] s_2[14:0];
logic[19:0] s_3[14:0];
logic[19:0] s_4[14:0];
logic[19:0] s_5[14:0];
logic[19:0] s_6[14:0];
logic[19:0] s_7[14:0];
logic[19:0] s_8[14:0];
logic[19:0] s_9[14:0];
assign s_A[0] = 20'b00000000000000000000;
assign s_A[1] = 20'b00001110000000000000;
assign s_A[2] = 20'b00011111110000000000;
assign s_A[3] = 20'b00011111111100000000;
assign s_A[4] = 20'b00000011111111100000;
assign s_A[5] = 20'b00000001111111111100;
assign s_A[6] = 20'b00000011110011111110;
assign s_A[7] = 20'b00000011110111111100;
assign s_A[8] = 20'b00000011111111111000;
assign s_A[9] = 20'b00000011111111000000;
assign s_A[10] = 20'b00011111111100000000;
assign s_A[11] = 20'b00011111100000000000;
assign s_A[12] = 20'b00000000000000000000;
assign s_A[13] = 20'b00000000000000000000;
assign s_A[14] = 20'b00000000000000000000;
assign s_B[0] = 20'b00000000000000000000;
assign s_B[1] = 20'b00000000000000000000;
assign s_B[2] = 20'b00011111111111111100;
assign s_B[3] = 20'b00011111111111111110;
assign s_B[4] = 20'b00011111111111111110;
assign s_B[5] = 20'b00011100001110001110;
assign s_B[6] = 20'b00011100001110001110;
assign s_B[7] = 20'b00011100001110001110;
assign s_B[8] = 20'b00011100001110001110;
assign s_B[9] = 20'b00011100011110011110;
assign s_B[10] = 20'b00011110011111011100;
assign s_B[11] = 20'b00001111111111111100;
assign s_B[12] = 20'b00000000000000000000;
assign s_B[13] = 20'b00000000000000000000;
assign s_B[14] = 20'b00000000000000000000;
assign s_C[0] = 20'b00000000000000000000;
```

```
assign s_C[1] = 20'b00000000000000000000;
assign s_C[2] = 20'b00000001111111100000;
assign s_C[3] = 20'b00000111111111111000;
assign s_C[4] = 20'b00001111111111111100;
assign s_C[5] = 20'b00001111000000111100;
assign s_C[6] = 20'b00011110000000011110;
assign s_C[7] = 20'b00011100000000001110;
assign s_C[8] = 20'b00011100000000001110;
assign s_C[9] = 20'b00011110000000011110;
assign s_C[10] = 20'b00001111000000111100;
assign s_C[11] = 20'b00001111000001111100;
assign s_C[12] = 20'b00000000000000000000;
assign s_C[13] = 20'b00000000000000000000;
assign s_C[14] = 20'b00000000000000000000;
assign s_D[0] = 20'b00000000000000000000;
assign s_D[1] = 20'b00000000000000000000;
assign s_D[2] = 20'b00001111111111111100;
assign s_D[3] = 20'b00011111111111111110;
assign s_D[4] = 20'b00011111111111111110;
assign s_D[5] = 20'b00011100000000001110;
assign s_D[6] = 20'b00011100000000001110;
assign s_D[7] = 20'b00011100000000001110;
assign s_D[8] = 20'b00011100000000011110;
assign s_D[9] = 20'b00011110000000011100;
assign s_D[10] = 20'b00001111000001111100;
assign s_D[11] = 20'b00000111111111111000;
assign s_D[12] = 20'b00000000000000000000;
assign s_D[13] = 20'b00000000000000000000;
assign s_D[14] = 20'b00000000000000000000;
assign s_E[0] = 20'b00000000000000000000;
assign s_E[1] = 20'b00000000000000000000;
assign s_E[2] = 20'b00001111111111111100;
assign s_E[3] = 20'b00011111111111111110;
assign s_E[4] = 20'b00011111111111111110;
assign s_E[5] = 20'b00011100001110001110;
assign s_E[6] = 20'b00011100001110001110;
assign s_E[7] = 20'b00011100001110001110;
assign s_E[8] = 20'b00011100001110001110;
assign s_E[9] = 20'b00011100000100001110;
assign s_E[10] = 20'b00011100000000001110;
assign s_E[11] = 20'b00011100000000001110;
assign s_E[12] = 20'b00000000000000000000;
assign s_E[13] = 20'b00000000000000000000;
```

```
assign s_E[14] = 20'b00000000000000000000;
assign s_F[0] = 20'b00000000000000000000;
assign s_F[1] = 20'b00000000000000000000;
assign s_F[2] = 20'b00001111111111111100;
assign s_F[3] = 20'b00011111111111111110;
assign s_F[4] = 20'b00001111111111111110;
assign s_F[5] = 20'b00000000001110001110;
assign s_F[6] = 20'b00000000001110001110;
assign s_F[7] = 20'b00000000001110001110;
assign s_F[8] = 20'b00000000001110001110;
assign s_F[9] = 20'b00000000000100001110;
assign s_F[10] = 20'b00000000000000001110;
assign s_F[11] = 20'b00000000000000001110;
assign s_F[12] = 20'b00000000000000000000;
assign s_F[13] = 20'b00000000000000000000;
assign s_F[14] = 20'b00000000000000000000;
assign s_G[0] = 20'b00000000000000000000;
assign s_G[1] = 20'b00000000000000000000;
assign s_G[2] = 20'b00000001111111100000;
assign s_G[3] = 20'b00000111111111111000;
assign s_G[4] = 20'b00001111111111111100;
assign s_G[5] = 20'b00001111000000111100;
assign s_G[6] = 20'b00011110000000011110;
assign s_G[7] = 20'b00011100000000001110;
assign s_G[8] = 20'b00011100010000001110;
assign s_G[9] = 20'b00011110111000011110;
assign s_G[10] = 20'b00001111111000111100;
assign s_G[11] = 20'b00001111111001111100;
assign s_G[12] = 20'b00000000000000000000;
assign s_G[13] = 20'b00000000000000000000;
assign s_G[14] = 20'b00000000000000000000;
assign s_H[0] = 20'b00000000000000000000;
assign s_H[1] = 20'b00000000000000000000;
assign s_H[2] = 20'b00001111111111111100;
assign s_H[3] = 20'b00011111111111111110;
assign s_H[4] = 20'b00001111111111111100;
assign s_H[5] = 20'b00000000001110000000;
assign s_H[6] = 20'b00000000001110000000;
assign s_H[7] = 20'b00000000001110000000;
assign s_H[8] = 20'b00000000001110000000;
assign s_H[9] = 20'b00000000001110000000;
assign s_H[10] = 20'b00000000001110000000;
assign s_H[11] = 20'b00001111111111111100;
```

```
assign s_H[12] = 20'b00000000000000000000;
assign s_H[13] = 20'b00000000000000000000;
assign s_H[14] = 20'b00000000000000000000;
assign s_I[0] = 20'b00000000000000000000;
assign s_I[1] = 20'b00000000000000000000;
assign s_I[2] = 20'b00001111111111111100;
assign s_I[3] = 20'b00011111111111111110;
assign s_I[4] = 20'b00001111111111111100;
assign s_I[5] = 20'b00000000000000000000;
assign s_I[6] = 20'b00000000000000000000;
assign s_I[7] = 20'b00000000000000000000;
assign s_I[8] = 20'b00000000000000000000;
assign s_I[9] = 20'b00000000000000000000;
assign s_I[10] = 20'b00000000000000000000;
assign s_I[11] = 20'b00000000000000000000;
assign s_I[12] = 20'b00000000000000000000;
assign s_I[13] = 20'b00000000000000000000;
assign s_I[14] = 20'b00000000000000000000;
assign s_J[0] = 20'b00000000000000000000;
assign s_J[1] = 20'b00000001100000000000;
assign s_J[2] = 20'b00001111110000000000;
assign s_J[3] = 20'b00001111100000000000;
assign s_J[4] = 20'b00011110000000000000;
assign s_J[5] = 20'b00011100000000000000;
assign s_J[6] = 20'b00011100000000000000;
assign s_J[7] = 20'b00001111111111111100;
assign s_J[8] = 20'b00011111111111111110;
assign s_J[9] = 20'b00001111111111111100;
assign s_J[10] = 20'b00000000000000000000;
assign s_J[11] = 20'b00000000000000000000;
assign s_J[12] = 20'b00000000000000000000;
assign s_J[13] = 20'b00000000000000000000;
assign s_J[14] = 20'b00000000000000000000;
assign s_K[0] = 20'b00000000000000000000;
assign s_K[1] = 20'b00000000000000000000;
assign s_K[2] = 20'b00001111111111111100;
assign s_K[3] = 20'b00011111111111111110;
assign s_K[4] = 20'b00001111111111111100;
assign s_K[5] = 20'b00000000111100000000;
assign s_K[6] = 20'b00000000011110000000;
assign s_K[7] = 20'b00000000011111000000;
assign s_K[8] = 20'b00000001111111100000;
assign s_K[9] = 20'b00000011111011110000;
```

```
assign s_K[10] = 20'b00001111110001111000;
assign s_K[11] = 20'b00011111000000111100;
assign s_K[12] = 20'b00000000000000000000;
assign s_K[13] = 20'b00000000000000000000;
assign s_K[14] = 20'b00000000000000000000;
assign s_L[0] = 20'b00000000000000000000;
assign s_L[1] = 20'b00000000000000000000;
assign s_L[2] = 20'b00001111111111111100;
assign s_L[3] = 20'b00011111111111111110;
assign s_L[4] = 20'b00011111111111111100;
assign s_L[5] = 20'b00011100000000000000;
assign s_L[6] = 20'b00011100000000000000;
assign s_L[7] = 20'b00011100000000000000;
assign s_L[8] = 20'b00011100000000000000;
assign s_L[9] = 20'b00011100000000000000;
assign s_L[10] = 20'b00011100000000000000;
assign s_L[11] = 20'b00011100000000000000;
assign s_L[12] = 20'b00000000000000000000;
assign s_L[13] = 20'b00000000000000000000;
assign s_L[14] = 20'b00000000000000000000;
assign s_M[0] = 20'b00000000000000000000;
assign s_M[1] = 20'b00000000000000000000;
assign s_M[2] = 20'b00001111111111111100;
assign s_M[3] = 20'b00011111111111111110;
assign s_M[4] = 20'b00011111111111111100;
assign s_M[5] = 20'b00000000011111111000;
assign s_M[6] = 20'b00000011111111000000;
assign s_M[7] = 20'b00001111111100000000;
assign s_M[8] = 20'b00011111100000000000;
assign s_M[9] = 20'b00001111111000000000;
assign s_M[10] = 20'b00000111111110000000;
assign s_M[11] = 20'b00000001111111110000;
assign s_M[12] = 20'b00000000000000000000;
assign s_M[13] = 20'b00000000000000000000;
assign s_M[14] = 20'b00000000000000000000;
assign s_N[0] = 20'b00000000000000000000;
assign s_N[1] = 20'b00000000000000000000;
assign s_N[2] = 20'b00001111111111111100;
assign s_N[3] = 20'b00011111111111111110;
assign s_N[4] = 20'b00001111111111111100;
assign s_N[5] = 20'b00000000000011111000;
assign s_N[6] = 20'b00000000001111110000;
assign s_N[7] = 20'b00000000011111000000;
```

```
assign s_N[8] = 20'b00000001111110000000;
assign s_N[9] = 20'b00000011111000000000;
assign s_N[10] = 20'b00001111110000000000;
assign s_N[11] = 20'b00011111111111111100;
assign s_N[12] = 20'b00000000000000000000;
assign s_N[13] = 20'b00000000000000000000;
assign s_N[14] = 20'b00000000000000000000;
assign s_O[0] = 20'b00000000000000000000;
assign s_O[1] = 20'b00000000000000000000;
assign s_O[2] = 20'b00000001111111100000;
assign s_O[3] = 20'b00000111111111111000;
assign s_O[4] = 20'b00001111111111111100;
assign s_O[5] = 20'b00001111000000111100;
assign s_O[6] = 20'b00011110000000011110;
assign s_O[7] = 20'b00011100000000001110;
assign s_O[8] = 20'b00011100000000001110;
assign s_O[9] = 20'b00011110000000011110;
assign s_O[10] = 20'b00001111000000111100;
assign s_O[11] = 20'b00001111111111111100;
assign s_O[12] = 20'b00000011111111100000;
assign s_O[13] = 20'b00000000000000000000;
assign s_O[14] = 20'b00000000000000000000;
assign s_P[0] = 20'b00000000000000000000;
assign s_P[1] = 20'b00000000000000000000;
assign s_P[2] = 20'b00001111111111111100;
assign s_P[3] = 20'b00011111111111111110;
assign s_P[4] = 20'b00011111111111111110;
assign s_P[5] = 20'b00000000011100001110;
assign s_P[6] = 20'b00000000011100001110;
assign s_P[7] = 20'b00000000011100001110;
assign s_P[8] = 20'b00000000011100001110;
assign s_P[9] = 20'b00000000011100011110;
assign s_P[10] = 20'b00000000011110011100;
assign s_P[11] = 20'b00000000001111111100;
assign s_P[12] = 20'b00000000000000000000;
assign s_P[13] = 20'b00000000000000000000;
assign s_P[14] = 20'b00000000000000000000;
assign s_Q[0] = 20'b00000000000000000000;
assign s_Q[1] = 20'b00000000000000000000;
assign s_Q[2] = 20'b00000001111111100000;
assign s_Q[3] = 20'b00000111111111111000;
assign s_Q[4] = 20'b00001111111111111100;
assign s_Q[5] = 20'b00001111000000111100;
```

```
assign s_Q[6] = 20'b00011110000000011110;
assign s_Q[7] = 20'b00011110000000001110;
assign s_Q[8] = 20'b00011111000000001110;
assign s_Q[9] = 20'b00011111000000011110;
assign s_Q[10] = 20'b00011111000000111100;
assign s_Q[11] = 20'b00111111111111111100;
assign s_Q[12] = 20'b00000000000000000000;
assign s_Q[13] = 20'b00000000000000000000;
assign s_Q[14] = 20'b00000000000000000000;
assign s_R[0] = 20'b00000000000000000000;
assign s_R[1] = 20'b00000000000000000000;
assign s_R[2] = 20'b00001111111111111100;
assign s_R[3] = 20'b00011111111111111110;
assign s_R[4] = 20'b00001111111111111110;
assign s_R[5] = 20'b00000000001110001110;
assign s_R[6] = 20'b00000000001110001110;
assign s_R[7] = 20'b00000000011110001110;
assign s_R[8] = 20'b00000000111110001110;
assign s_R[9] = 20'b00000011111110011110;
assign s_R[10] = 20'b00001111111111011100;
assign s_R[11] = 20'b00011111101111111100;
assign s_R[12] = 20'b00000000000000000000;
assign s_R[13] = 20'b00000000000000000000;
assign s_R[14] = 20'b00000000000000000000;
assign s_S[0] = 20'b00000000000000000000;
assign s_S[1] = 20'b00000000000000000000;
assign s_S[2] = 20'b00000110000011111000;
assign s_S[3] = 20'b00001111000111111100;
assign s_S[4] = 20'b00001111000111111100;
assign s_S[5] = 20'b00011110001111011110;
assign s_S[6] = 20'b00011100001110001110;
assign s_S[7] = 20'b00011100001110001110;
assign s_S[8] = 20'b00011100011100001110;
assign s_S[9] = 20'b00011110011100011110;
assign s_S[10] = 20'b00011111111000111100;
assign s_S[11] = 20'b00001111111000111100;
assign s_S[12] = 20'b00000011110000000000;
assign s_S[13] = 20'b00000000000000000000;
assign s_S[14] = 20'b00000000000000000000;
assign s_T[0] = 20'b00000000000000000000;
assign s_T[1] = 20'b00000000000000000100;
assign s_T[2] = 20'b00000000000000001110;
assign s_T[3] = 20'b00000000000000001110;
```

```
assign s_T[4] = 20'b00000000000000001110;
assign s_T[5] = 20'b00001111111111111110;
assign s_T[6] = 20'b00011111111111111110;
assign s_T[7] = 20'b00001111111111111110;
assign s_T[8] = 20'b00000000000000001110;
assign s_T[9] = 20'b00000000000000001110;
assign s_T[10] = 20'b00000000000000001110;
assign s_T[11] = 20'b00000000000000000100;
assign s_T[12] = 20'b00000000000000000000;
assign s_T[13] = 20'b00000000000000000000;
assign s_T[14] = 20'b00000000000000000000;
assign s_U[0] = 20'b00000000000000000000;
assign s_U[1] = 20'b00000000000000000000;
assign s_U[2] = 20'b00000001111111111100;
assign s_U[3] = 20'b00000111111111111110;
assign s_U[4] = 20'b00001111111111111100;
assign s_U[5] = 20'b00001111000000000000;
assign s_U[6] = 20'b00011110000000000000;
assign s_U[7] = 20'b00011100000000000000;
assign s_U[8] = 20'b00011100000000000000;
assign s_U[9] = 20'b00011110000000000000;
assign s_U[10] = 20'b00011111000000000000;
assign s_U[11] = 20'b00001111111111111100;
assign s_U[12] = 20'b00000000000000000000;
assign s_U[13] = 20'b00000000000000000000;
assign s_U[14] = 20'b00000000000000000000;
assign s_V[0] = 20'b00000000000000000000;
assign s_V[1] = 20'b00000000000000111100;
assign s_V[2] = 20'b00000000000011111110;
assign s_V[3] = 20'b00000000011111111100;
assign s_V[4] = 20'b00000011111111100000;
assign s_V[5] = 20'b00001111111100000000;
assign s_V[6] = 20'b00011111110000000000;
assign s_V[7] = 20'b00001111110000000000;
assign s_V[8] = 20'b00001111111110000000;
assign s_V[9] = 20'b00000001111111100000;
assign s_V[10] = 20'b00000000001111111100;
assign s_V[11] = 20'b00000000000011111110;
assign s_V[12] = 20'b00000000000000000000;
assign s_V[13] = 20'b00000000000000000000;
assign s_V[14] = 20'b00000000000000000000;
assign s_W[0] = 20'b00000000000000000000;
assign s_W[1] = 20'b00000000000000011100;
```

```
assign s_W[2] = 20'b00000000000111111110;
assign s_W[3] = 20'b00000001111111111100;
assign s_W[4] = 20'b00001111111111110000;
assign s_W[5] = 20'b00011111111100000000;
assign s_W[6] = 20'b00001111111111000000;
assign s_W[7] = 20'b00000111111111111100;
assign s_W[8] = 20'b00000000011111111110;
assign s_W[9] = 20'b00000001111111111100;
assign s_W[10] = 20'b00001111111111110000;
assign s_W[11] = 20'b00011111111100000000;
assign s_W[12] = 20'b00000000000000000000;
assign s_W[13] = 20'b00000000000000000000;
assign s_W[14] = 20'b00000000000000000000;
assign s_X[0] = 20'b00000000000000000000;
assign s_X[1] = 20'b00000000000000000000;
assign s_X[2] = 20'b00001100000000001100;
assign s_X[3] = 20'b00011111000000111110;
assign s_X[4] = 20'b00011111100001111100;
assign s_X[5] = 20'b00000111111111110000;
assign s_X[6] = 20'b00000011111111100000;
assign s_X[7] = 20'b00000000111110000000;
assign s_X[8] = 20'b00000011111111100000;
assign s_X[9] = 20'b00000111111111110000;
assign s_X[10] = 20'b00011111100001111100;
assign s_X[11] = 20'b00011111000000111110;
assign s_X[12] = 20'b00000000000000000000;
assign s_X[13] = 20'b00000000000000000000;
assign s_X[14] = 20'b00000000000000000000;
assign s_Y[0] = 20'b00000000000000000000;
assign s_Y[1] = 20'b00000000000000001100;
assign s_Y[2] = 20'b00000000000000111110;
assign s_Y[3] = 20'b00000000000001111100;
assign s_Y[4] = 20'b00000000000011110000;
assign s_Y[5] = 20'b00001111111111100000;
assign s_Y[6] = 20'b00011111111111000000;
assign s_Y[7] = 20'b00001111111111000000;
assign s_Y[8] = 20'b00000000000111100000;
assign s_Y[9] = 20'b00000000000011110000;
assign s_Y[10] = 20'b00000000000001111100;
assign s_Y[11] = 20'b00000000000000011110;
assign s_Y[12] = 20'b00000000000000000000;
assign s_Y[13] = 20'b00000000000000000000;
assign s_Y[14] = 20'b00000000000000000000;
```

```
assign s_Z[0] = 20'b00000000000000000000;
assign s_Z[1] = 20'b00000000000000000000;
assign s_Z[2] = 20'b00001100000000000100;
assign s_Z[3] = 20'b00011111000000001110;
assign s_Z[4] = 20'b00011111100000001110;
assign s_Z[5] = 20'b00011111111000001110;
assign s_Z[6] = 20'b00011111111100001110;
assign s_Z[7] = 20'b00011100111110001110;
assign s_Z[8] = 20'b00011100011111101110;
assign s_Z[9] = 20'b00011100000111111110;
assign s_Z[10] = 20'b00011100000001111110;
assign s_Z[11] = 20'b00011100000000111110;
assign s_Z[12] = 20'b00000000000000000000;
assign s_Z[13] = 20'b00000000000000000000;
assign s_Z[14] = 20'b00000000000000000000;
assign s_dot[0] = 20'b00000000000000000000;
assign s_dot[1] = 20'b00000000000000000000;
assign s_dot[2] = 20'b00000100000000000000;
assign s_dot[3] = 20'b00011110000000000000;
assign s_dot[4] = 20'b00011110000000000000;
assign s_dot[5] = 20'b00011110000000000000;
assign s_dot[6] = 20'b00000100000000000000;
assign s_dot[7] = 20'b00000000000000000000;
assign s_dot[8] = 20'b00000000000000000000;
assign s_dot[9] = 20'b00000000000000000000;
assign s_dot[10] = 20'b00000000000000000000;
assign s_dot[11] = 20'b00000000000000000000;
assign s_dot[12] = 20'b00000000000000000000;
assign s_dot[13] = 20'b00000000000000000000;
assign s_dot[14] = 20'b00000000000000000000;
assign s_colon[0] = 20'b00000000000000000000;
assign s_colon[1] = 20'b00000000000000000000;
assign s_colon[2] = 20'b00000100000010000000;
assign s_colon[3] = 20'b00011110001111000000;
assign s_colon[4] = 20'b00011110001111000000;
assign s_colon[5] = 20'b00011110001111000000;
assign s_colon[6] = 20'b00000100000010000000;
assign s_colon[7] = 20'b00000000000000000000;
assign s_colon[8] = 20'b00000000000000000000;
assign s_colon[9] = 20'b00000000000000000000;
assign s_colon[10] = 20'b00000000000000000000;
assign s_colon[11] = 20'b00000000000000000000;
assign s_colon[12] = 20'b00000000000000000000;
```

```
assign s_colon[13] = 20'b00000000000000000000;
assign s_colon[14] = 20'b00000000000000000000;
assign s_blank [0] = 20'b00000000000000000000;
assign s_blank [1] = 20'b00000000000000000000;
assign s_blank [2] = 20'b00000000000000000000;
assign s_blank [3] = 20'b00000000000000000000;
assign s_blank [4] = 20'b00000000000000000000;
assign s_blank [5] = 20'b00000000000000000000;
assign s_blank [6] = 20'b00000000000000000000;
assign s_blank [7] = 20'b00000000000000000000;
assign s_blank [8] = 20'b00000000000000000000;
assign s_blank [9] = 20'b00000000000000000000;
assign s_blank [10] = 20'b00000000000000000000;
assign s_blank [11] = 20'b00000000000000000000;
assign s_blank [12] = 20'b00000000000000000000;
assign s_blank [13] = 20'b00000000000000000000;
assign s_blank [14] = 20'b00000000000000000000;
assign s_1[0] = 20'b00000000000000000000;
assign s_1[1] = 20'b00000000000000000000;
assign s_1[2] = 20'b00000000000000000000;
assign s_1[3] = 20'b00000000000000000000;
assign s_1[4] = 20'b00000000000000110000;
assign s_1[5] = 20'b00000000000001111000;
assign s_1[6] = 20'b00000000000000111000;
assign s_1[7] = 20'b00001111111111111100;
assign s_1[8] = 20'b00011111111111111110;
assign s_1[9] = 20'b00001111111111111100;
assign s_1[10] = 20'b00000000000000000000;
assign s_1[11] = 20'b00000000000000000000;
assign s_1[12] = 20'b00000000000000000000;
assign s_1[13] = 20'b00000000000000000000;
assign s_1[14] = 20'b00000000000000000000;
assign s_2[0] = 20'b00000000000000000000;
assign s_2[1] = 20'b00000000000000000000;
assign s_2[2] = 20'b00001100000000100000;
assign s_2[3] = 20'b00011110000001111000;
assign s_2[4] = 20'b00011111000000111100;
assign s_2[5] = 20'b00011111100000011100;
assign s_2[6] = 20'b00011111110000001110;
assign s_2[7] = 20'b00011111111000001110;
assign s_2[8] = 20'b00011101111100001110;
assign s_2[9] = 20'b00011100111110011100;
assign s_2[10] = 20'b00011100011111111100;
```

```
assign s_2[11] = 20'b00011100000111111000;
assign s_2[12] = 20'b00000000000000000000;
assign s_2[13] = 20'b00000000000000000000;
assign s_2[14] = 20'b00000000000000000000;
assign s_3[0] = 20'b00000000000000000000;
assign s_3[1] = 20'b00000000000000000000;
assign s_3[2] = 20'b00000110000000000000;
assign s_3[3] = 20'b00001111000000000100;
assign s_3[4] = 20'b00001111000000001110;
assign s_3[5] = 20'b00011100000000001110;
assign s_3[6] = 20'b00011100000010001110;
assign s_3[7] = 20'b00011100000111001110;
assign s_3[8] = 20'b00011100000111111110;
assign s_3[9] = 20'b00011110001111111110;
assign s_3[10] = 20'b00001111111110111110;
assign s_3[11] = 20'b00000111111110011110;
assign s_3[12] = 20'b00000000000000000000;
assign s_3[13] = 20'b00000000000000000000;
assign s_3[14] = 20'b00000000000000000000;
assign s_4[0] = 20'b00000000000000000000;
assign s_4[1] = 20'b00000000000000000000;
assign s_4[2] = 20'b00000000110000000000;
assign s_4[3] = 20'b00000001111100000000;
assign s_4[4] = 20'b00000001111110000000;
assign s_4[5] = 20'b00000001111111000000;
assign s_4[6] = 20'b00000001110111110000;
assign s_4[7] = 20'b00000001110011111000;
assign s_4[8] = 20'b00001111111111111100;
assign s_4[9] = 20'b00011111111111111110;
assign s_4[10] = 20'b00001111111111111100;
assign s_4[11] = 20'b00000001110000000000;
assign s_4[12] = 20'b00000000000000000000;
assign s_4[13] = 20'b00000000000000000000;
assign s_4[14] = 20'b00000000000000000000;
assign s_5[0] = 20'b00000000000000000000;
assign s_5[1] = 20'b00000000000000000000;
assign s_5[2] = 20'b00000110000110000000;
assign s_5[3] = 20'b00001111001111111100;
assign s_5[4] = 20'b00001111001111111110;
assign s_5[5] = 20'b00011100000111111110;
assign s_5[6] = 20'b00011100000111001110;
assign s_5[7] = 20'b00011100000111001110;
assign s_5[8] = 20'b00011100000111001110;
```

```
assign s_5[9] = 20'b00011110000111001110;
assign s_5[10] = 20'b00001111111111001110;
assign s_5[11] = 20'b00000111111110000100;
assign s_5[12] = 20'b00000000000000000000;
assign s_5[13] = 20'b00000000000000000000;
assign s_5[14] = 20'b00000000000000000000;
assign s_6[0] = 20'b00000000000000000000;
assign s_6[1] = 20'b00000000000000000000;
assign s_6[2] = 20'b00000000111100000000;
assign s_6[3] = 20'b00000111111111110000;
assign s_6[4] = 20'b00001111111111111100;
assign s_6[5] = 20'b00001111011111111100;
assign s_6[6] = 20'b00011110001110011100;
assign s_6[7] = 20'b00011100000111011110;
assign s_6[8] = 20'b00011100000111001110;
assign s_6[9] = 20'b00011110001110011110;
assign s_6[10] = 20'b00001111111110111100;
assign s_6[11] = 20'b00000111111100111000;
assign s_6[12] = 20'b00000000000000000000;
assign s_6[13] = 20'b00000000000000000000;
assign s_6[14] = 20'b00000000000000000000;
assign s_7[0] = 20'b00000000000000000000;
assign s_7[1] = 20'b00000000000000000000;
assign s_7[2] = 20'b00000000000000000100;
assign s_7[3] = 20'b00000000000000001110;
assign s_7[4] = 20'b00001000000000001110;
assign s_7[5] = 20'b00011110000000001110;
assign s_7[6] = 20'b00001111110000001110;
assign s_7[7] = 20'b00000111111100001110;
assign s_7[8] = 20'b00000001111111001110;
assign s_7[9] = 20'b00000000011111111110;
assign s_7[10] = 20'b00000000000111111110;
assign s_7[11] = 20'b00000000000001111110;
assign s_7[12] = 20'b00000000000000000000;
assign s_7[13] = 20'b00000000000000000000;
assign s_7[14] = 20'b00000000000000000000;
assign s_8[0] = 20'b00000000000000000000;
assign s_8[1] = 20'b00000000000000000000;
assign s_8[2] = 20'b00000111110000110000;
assign s_8[3] = 20'b00001111111011111000;
assign s_8[4] = 20'b00001111111111111100;
assign s_8[5] = 20'b00011100001111011110;
assign s_8[6] = 20'b00011100001111001110;
```

```
assign s_8[7] = 20'b00011100001110001110;
assign s_8[8] = 20'b00011100001111001110;
assign s_8[9] = 20'b00011100001111011110;
assign s_8[10] = 20'b00011111111111111100;
assign s_8[11] = 20'b00001111111011111000;
assign s_8[12] = 20'b00000000000000000000;
assign s_8[13] = 20'b00000000000000000000;
assign s_8[14] = 20'b00000000000000000000;
assign s_9[0] = 20'b00000000000000000000;
assign s_9[1] = 20'b00000000000000000000;
assign s_9[2] = 20'b00000010001111110000;
assign s_9[3] = 20'b00001111001111111000;
assign s_9[4] = 20'b00001110111111111100;
assign s_9[5] = 20'b00011100111000011100;
assign s_9[6] = 20'b00011100111000011110;
assign s_9[7] = 20'b00011100111000001110;
assign s_9[8] = 20'b00011110111000011100;
assign s_9[9] = 20'b00001111111111111100;
assign s_9[10] = 20'b00001111111111111100;
assign s_9[11] = 20'b00000011111111111000;
assign s_9[12] = 20'b00000000000000000000;
assign s_9[13] = 20'b00000000000000000000;
assign s_9[14] = 20'b00000000000000000000;


endmodule // VGA_LED_Emulator
```

Vga_LED.sv:

```
module VGA_LED(input logic        clk,
               input logic            reset,
               input logic [31:0]  writedata,
               input logic            write,
               input                  chipselect,
               input logic [2:0]  address,

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

```
logic [9:0] x_coord, y_coord_in, y_coord_out, y_coord;
        logic [1:0] x_code, y_code;
        logic [9:0] radius;
        logic [31:0] display;
        logic [19:0] new_stock, stored_stock;
        logic [4:0] stock_address;
        logic [24:0] sum32;
        logic [5:0] sum_state;
        logic change_check;
        logic display_choice;
        //----------------------------------------
        logic [9:0] disp_stock_x [399:0];
        logic [9:0] disp_stock_a1 [399:0];
        logic [9:0] disp_stock_a2 [399:0];
        logic [9:0] disp_stock_a3 [399:0];
        logic send_to_vga;
        logic [6:0] stock_name;
        //----------------------------------------

        logic [16:0] ema4_out;
        logic [16:0] ema32_out;
        logic [16:0] ema128_out;
        logic [16:0] disp_stock [19:0]; //19 for the number of stock modules...
        logic [16:0] disp_avg4 [19:0];
logic [16:0] disp_avg32 [19:0];
logic [16:0] disp_avg128 [19:0];

        logic [16:0] final_dispS [399:0];
        logic [16:0] final_dispA4 [399:0];
        logic [16:0] final_dispA32 [399:0];
        logic [16:0] final_dispA128 [399:0];
        logic [16:0] fd1;
        logic [16:0] fd2;
        logic [16:0] fd3;
        logic [16:0] fd4;
        logic [8:0] in_ind;

        logic [8:0] out_ind;
        logic [9:0] sd1;
        logic [9:0] sd2;
```

```
logic [9:0] sd3;
logic [9:0] sd4;
logic [9:0] scale_dispS [399:0];
logic [9:0] scale_dispA4 [399:0];
logic [9:0] scale_dispA32 [399:0];
logic [9:0] scale_dispA128 [399:0];

logic [16:0] fake1;
logic [16:0] fake2;
logic [16:0] fake3;
logic [16:0] fake4;

logic update_done[19:0];
logic update_begin[19:0];
logic last_update_done[19:0];
logic last_update_begin[19:0];
logic begin_ranger;

logic start_flag;
logic end_flag;
logic [8:0] output_index;

logic [15:0] sum64_out;

logic [16:0] data_send [19:0]; //number of stock modules

logic [9:0] counter_top;
logic [9:0] counter_delay;
logic [19:0] disp_delay;
logic change_flag [19:0]; //number of stock modules
logic request_disp [19:0];
logic new_disp [19:0];
logic [8:0] output_id;
logic [8:0] debug1;
logic [16:0] debug2;
logic [16:0] min;
logic [16:0] divide_min;
logic [16:0] max;
logic done_range;

logic scale_output;
logic [16:0] range;
logic [16:0] hard_range;
```

```
        logic hard_range_flag;

        //
        logic [1:0] bhs;
        //---------------------
        logic read_soft;
        logic write_soft;
        // flags for hang
        logic hang_start;
        logic [1:0] hang_count;
        //
        // have conversions to decimal
        logic [3:0] S_hundredthousand;
        logic [3:0] S_tenthousand;
        logic [3:0] S_thousand;
        logic [3:0] S_hundred;
        logic [3:0] S_ten;
        logic [3:0] S_one;

        logic [3:0] E4_hundredthousand;
        logic [3:0] E4_tenthousand;
        logic [3:0] E4_thousand;
        logic [3:0] E4_hundred;
        logic [3:0] E4_ten;
        logic [3:0] E4_one;

        logic [3:0] E32_hundredthousand;
        logic [3:0] E32_tenthousand;
        logic [3:0] E32_thousand;
        logic [3:0] E32_hundred;
        logic [3:0] E32_ten;
        logic [3:0] E32_one;

        logic [3:0] E128_hundredthousand;
        logic [3:0] E128_tenthousand;
        logic [3:0] E128_thousand;
        logic [3:0] E128_hundred;
        logic [3:0] E128_ten;
        logic [3:0] E128_one;
        logic start_dec;
//-----------------------------------------
        logic [16:0] low_disp;
        logic [16:0] high_disp;
```

```
	logic [16:0] adj_range;



//Hook up modules
//
//Divider for scaling to pixels
	divider dv_instance(
		.clk(clk),
		.range(range),
		.min(divide_min),
		.input_val1(fd1),
		.input_val2(fd2),
		.input_val3(fd3),
		.input_val4(fd4),
		.input_index(in_ind),
		.output_val1(sd1),
		.output_val2(sd2),
		.output_val3(sd3),
		.output_val4(sd4),
		.adj_range(adj_range),
		.output_index(out_ind)
	);
//For finding min and max and range
	nothing_master nm_instance(
		.clk(clk),
		.debug1(debug1),
		.begin_ranger(begin_ranger),
		.debug2(debug2),
		.disp1(fake1),
		.disp2(fake2),
		.disp3(fake3),
		.disp4(fake4),
		.done_range(done_range),
		.max(max),
		.min(min)
	);

//Stock module for stock 0
	stock_master sm_instance0(
			.clk(clk),
```

```
                .stock_in(data_send[0]),
                .update_begin(update_begin[0]),
                .update_done(update_done[0]),
                .request_disp(request_disp[0]),
                .new_disp(new_disp[0]),
                .disp_stock(disp_stock[0]),
                .disp_avg4(disp_avg4[0]),
                .disp_avg32(disp_avg32[0]),
                .disp_avg128(disp_avg128[0]),
                .change_flag(change_flag[0])
        );

//Stock module for stock 1
        stock_master sm_instance1(
                .clk(clk),
                .stock_in(data_send[1]),
                .update_begin(update_begin[1]),
                .update_done(update_done[1]),
                .request_disp(request_disp[1]),
                .new_disp(new_disp[1]),
                .disp_stock(disp_stock[1]),
                .disp_avg4(disp_avg4[1]),
                .disp_avg32(disp_avg32[1]),
                .disp_avg128(disp_avg128[1]),
                .change_flag(change_flag[1])
        );
//Stock module for stock 2
        stock_master sm_instance2(
                .clk(clk),
                .stock_in(data_send[2]),
                .update_begin(update_begin[2]),
                .update_done(update_done[2]),
                .request_disp(request_disp[2]),
                .new_disp(new_disp[2]),
                .disp_stock(disp_stock[2]),
                .disp_avg4(disp_avg4[2]),
                .disp_avg32(disp_avg32[2]),
                .disp_avg128(disp_avg128[2]),
                .change_flag(change_flag[2])
        );

//Stock module for stock 3
        stock_master sm_instance3(
```

```
                .clk(clk),
                .stock_in(data_send[3]),
                .update_begin(update_begin[3]),
                .update_done(update_done[3]),
                .request_disp(request_disp[3]),
                .new_disp(new_disp[3]),
                .disp_stock(disp_stock[3]),
                .disp_avg4(disp_avg4[3]),
                .disp_avg32(disp_avg32[3]),
                .disp_avg128(disp_avg128[3]),
                .change_flag(change_flag[3])
        );
//Stock module for stock 4
        stock_master sm_instance4(
                .clk(clk),
                .stock_in(data_send[4]),
                .update_begin(update_begin[4]),
                .update_done(update_done[4]),
                .request_disp(request_disp[4]),
                .new_disp(new_disp[4]),
                .disp_stock(disp_stock[4]),
                .disp_avg4(disp_avg4[4]),
                .disp_avg32(disp_avg32[4]),
                .disp_avg128(disp_avg128[4]),
                .change_flag(change_flag[4])
        );
//Stock module for stock 5
        stock_master sm_instance5(
                .clk(clk),
                .stock_in(data_send[5]),
                .update_begin(update_begin[5]),
                .update_done(update_done[5]),
                .request_disp(request_disp[5]),
                .new_disp(new_disp[5]),
                .disp_stock(disp_stock[5]),
                .disp_avg4(disp_avg4[5]),
                .disp_avg32(disp_avg32[5]),
                .disp_avg128(disp_avg128[5]),
                .change_flag(change_flag[5])
        );
//Stock module for stock 6
        stock_master sm_instance6(
                .clk(clk),
```

```
                        .stock_in(data_send[6]),
                        .update_begin(update_begin[6]),
                        .update_done(update_done[6]),
                        .request_disp(request_disp[6]),
                        .new_disp(new_disp[6]),
                        .disp_stock(disp_stock[6]),
                        .disp_avg4(disp_avg4[6]),
                        .disp_avg32(disp_avg32[6]),
                        .disp_avg128(disp_avg128[6]),
                        .change_flag(change_flag[6])
        );

//Converting to decimal from binary
                to_decimal td_S(
                .binary(final_dispS[399]),
                .clk(clk),
                .start(start_dec),
                .hundred_thousands(S_hundredthousand),
                .ten_thousands(S_tenthousand),
                .thousands(S_thousand),
                .hundreds(S_hundred),
                .tens(S_ten),
                .ones(S_one)
        );
//Converting to decimal from binary
                to_decimal td_E4(
                .binary(low_disp),
                .clk(clk),
                .start(start_dec),
                .hundred_thousands(E4_hundredthousand),
                .ten_thousands(E4_tenthousand),
                .thousands(E4_thousand),
                .hundreds(E4_hundred),
                .tens(E4_ten),
                .ones(E4_one)
        );
//Converting to decimal from binary
                to_decimal td_E32(
                .binary(high_disp),
                .clk(clk),
                .start(start_dec),
                .hundred_thousands(E32_hundredthousand),
                .ten_thousands(E32_tenthousand),
```

```
                .thousands(E32_thousand),
                .hundreds(E32_hundred),
                .tens(E32_ten),
                .ones(E32_one)
        );
//Converting to decimal from binary
                to_decimal td_E128(
                .binary(disp_stock_a3[399]),
                .clk(clk),
                .start(start_dec),
                .hundred_thousands(E128_hundredthousand),
                .ten_thousands(E128_tenthousand),
                .thousands(E128_thousand),
                .hundreds(E128_hundred),
                .tens(E128_ten),
                .ones(E128_one)
        );




  VGA_LED_Emulator led_emulator(.clk50(clk), .*);


  always_ff @(posedge clk) begin

//
//
//Top Module Control Center
//
                if (chipselect && write) begin
                        if (writedata[31] == 1'b0) begin
                                read_soft <= 1'b1;
                        end
                        if (writedata[31] == 1'b1) begin
                                write_soft <= 1'b1;
                                stock_name <= writedata[23:17];
                                if (writedata[16] == 1'b1) begin
                                        hard_range[16:1] <= writedata[15:0];
                                        hard_range[0] <= 1'b0;
                                        hard_range_flag <= 1'b1;
```

```
                                        //set the range from pass data
                                end
                                if (writedata[16] == 1'b0) begin
                                        hard_range_flag <= 1'b0;
                                end

                        end
                end

                if ((final_dispS[399] > final_dispA4[399]) && (final_dispA32[399] >
final_dispA128[399])) begin
                        bhs <= 2'b00; //buy
                end
                if ((final_dispS[399] > final_dispA4[399]) && (final_dispA32[399] <
final_dispA128[399])) begin
                        bhs <= 2'b01; //hold
                end
                if ((final_dispS[399] < final_dispA4[399]) && (final_dispA32[399] <
final_dispA128[399])) begin
                        bhs <= 2'b10; //sell
                end
                if ((final_dispS[399] < final_dispA4[399]) && (final_dispA32[399] >
final_dispA128[399])) begin
                        bhs <= 2'b01; //hold
                end

//Reading stock data from software
                        if (read_soft == 1'b1) begin
                                debug1 <= writedata[23:17];
                                debug2 <= writedata[16:0];
                                change_flag[writedata[23:17]] <= change_flag[writedata[23:17]]
+ 1'b1;
                                data_send[writedata[23:17]] <= writedata[16:0];
                                read_soft <= 1'b0;
                        end
//Sending display request to hardware
                        if (write_soft == 1'b1) begin
                                debug1 <= writedata[23:17];
                                debug2 <= writedata[16:0];
                                if (disp_delay == 0) begin
                                        request_disp[writedata[23:17]] <= 1'b1;
                                        output_id <= writedata[23:17];
                                        disp_delay <= disp_delay + 1'b1;
```

```
                        end
                        if (disp_delay > 0 && disp_delay < 420) begin //could change this
to make a longer wait in between
                                request_disp[writedata[23:17]] <= 1'b0;
                                disp_delay <= disp_delay + 1'b1;
                        end
                        if (disp_delay >= 420) begin //also have to change this to make
longer.

                                disp_delay <= 0;
                                write_soft <= 1'b0;
                        end
                        //wait 100 clock cycles
                end
        fake2 <= disp_avg4[output_id];
        fake3 <= disp_avg32[output_id];
        fake4 <= disp_avg128[output_id];
        fake1 <= disp_stock[output_id];

        last_update_begin[output_id] <= update_begin[output_id];
        last_update_done[output_id] <= update_done[output_id];

        if (last_update_begin[output_id] != update_begin[output_id]) begin
                start_flag <= 1'b1;
                output_index <= 1'b1;
                begin_ranger <= 1'b1;
                final_dispS[0] <= disp_stock[output_id];
                final_dispA4[0] <= disp_avg4[output_id];
                final_dispA32[0] <= disp_avg32[output_id];
                final_dispA128[0] <= disp_avg128[output_id];
        end

        if (last_update_done[output_id] != update_done[output_id]) begin
                start_flag <= 1'b0;
        end

        if (start_flag == 1'b1) begin
                begin_ranger <= 1'b0;
                final_dispS[output_index] <= disp_stock[output_id];
                final_dispA4[output_index] <= disp_avg4[output_id];
                final_dispA32[output_index] <= disp_avg32[output_id];
                final_dispA128[output_index] <= disp_avg128[output_id];
                output_index <= output_index + 1'b1;
```

```
		end

		if (done_range == 1'b1) begin
			scale_output <= 1'b1;
			in_ind <= 1'b0;
			if (hard_range_flag == 1'b0) begin
				range <= max - min;
			end
			if (hard_range_flag == 1'b1) begin
				range <= hard_range;
			end
			divide_min <= min;
		end
	//For changing from stock data to pixels
		if(scale_output == 1'b1) begin
			fd1 <= final_dispS[in_ind];
			fd2 <= final_dispA4[in_ind];
			fd3 <= final_dispA32[in_ind];
			fd4 <= final_dispA128[in_ind];
			in_ind <= in_ind + 1'b1;
			disp_stock_x[out_ind] <= sd1;
			disp_stock_a1[out_ind] <= sd2;
			disp_stock_a2[out_ind] <= sd3;
			disp_stock_a3[out_ind] <= sd4;


			if (out_ind == 399) begin
				scale_output <= 1'b0; //could use this as flag for Hang (with
slight tweak.
				start_dec <= 1'b1;
				hang_start <= 1'b1;
				hang_count <= 2'b0;

				low_disp <= min;
				high_disp <= min + adj_range;
			end
		end

		if (start_dec == 1'b1) begin
			start_dec <= 1'b0;
		end

		if (hang_start == 1'b1) begin
```

```
                        if (hang_count < 2'b11) begin
                                hang_count <= hang_count + 1'b1;
                        end
                        if (hang_count == 2'b11) begin
                                hang_count <= 2'b0;
                                hang_start <= 1'b0;
                        end
                end
        end
endmodule
//---------------------
//----------------------

module to_decimal(
        input logic clk,
        input logic start,
        input logic [16:0] binary,
        output logic [3:0] hundred_thousands,
        output logic [3:0] ten_thousands,
        output logic [3:0] thousands,
        output logic [3:0] hundreds,
        output logic [3:0] tens,
        output logic [3:0] ones
        );

        logic [4:0] i;
        logic [3:0] d_100000;
        logic [3:0] d_10000;
        logic [3:0] d_1000;
        logic [3:0] d_100;
        logic [3:0] d_10;
        logic [3:0] d_1;
        logic [16:0] in_val;
        logic sep_add;
        logic running;

        always_ff @(posedge clk) begin

                if (start == 1'b1) begin
                        i <= 17;
                        d_100000 <= 1'b0;
                        d_10000 <= 1'b0;
                        d_1000 <= 1'b0;
```

```
                d_100 <= 1'b0;
                d_10 <= 1'b0;
                d_1 <= 1'b0;
                in_val <= binary;
                running <= 1'b1;
            end
            if (running == 1'b1) begin
                if (i > 1'b0) begin
                    if ((d_1 >= 5 || d_10 >= 5 || d_100 >= 5 || d_1000 >= 5 || d_10000
>= 5 || d_100000 >= 5)&&sep_add == 1'b0)begin
                        if (d_100000 >= 5) begin
                            d_100000 <= d_100000 + 3;
                        end
                        if (d_10000 >= 5) begin
                            d_10000 <= d_10000 + 3;
                        end
                        if (d_1000 >= 5) begin
                            d_1000 <= d_1000 + 3;
                        end
                        if (d_100 >= 5) begin
                            d_100 <= d_100 + 3;
                        end
                        if (d_10 >= 5) begin
                            d_10 <= d_10 + 3;
                        end
                        if (d_1 >= 5) begin
                            d_1 <= d_1 + 3;
                        end
                        sep_add <= 1'b1;
                    end
                    if (d_1 < 5 && d_10 < 5 && d_100 < 5 && d_1000 < 5 &&
d_10000 < 5 && d_100000 < 5 && sep_add == 1'b0) begin
                        d_100000 <= d_100000 << 1;
                        d_100000[0] <= d_10000[3];
                        d_10000 <= d_10000 << 1;
                        d_10000[0] <= d_1000[3];
                        d_1000 <= d_1000 << 1;
                        d_1000[0] <= d_100[3];
                        d_100 <= d_100 << 1;
                        d_100[0] <= d_10[3];
                        d_10 <= d_10 << 1;
                        d_10[0] <= d_1[3];
                        d_1 <= d_1 << 1;
```

```
                                d_1[0] <= in_val[16];
                                in_val <= in_val << 1;
                                i <= i - 1;
                        end
                        if (sep_add == 1'b1) begin
                                d_100000 <= d_100000 << 1;
                                d_100000[0] <= d_10000[3];
                                d_10000 <= d_10000 << 1;
                                d_10000[0] <= d_1000[3];
                                d_1000 <= d_1000 << 1;
                                d_1000[0] <= d_100[3];
                                d_100 <= d_100 << 1;
                                d_100[0] <= d_10[3];
                                d_10 <= d_10 << 1;
                                d_10[0] <= d_1[3];
                                d_1 <= d_1 << 1;
                                d_1[0] <= in_val[16];
                                in_val <= in_val << 1;
                                i <= i - 1;
                                sep_add <= 1'b0;
                        end
                end
                if (i == 0) begin
                        running <= 1'b0;
                        hundred_thousands <= d_100000;
                        ten_thousands <= d_10000;
                        thousands <= d_1000;
                        hundreds <= d_100;
                        tens <= d_10;
                        ones <= d_1;
                end
        end
    end
endmodule


module divider(
        input logic clk,
        input logic [16:0] range,
        input logic [16:0] min,
        input logic [16:0] input_val1,
        input logic [16:0] input_val2,
        input logic [16:0] input_val3,
```

```
input logic [16:0] input_val4,
input logic [8:0] input_index,
output logic [9:0] output_val1,
output logic [9:0] output_val2,
output logic [9:0] output_val3,
output logic [9:0] output_val4,
output logic [16:0] adj_range,
output logic [8:0] output_index


);

logic [16:0] temp_in;
logic [16:0] temp_range;
logic [25:0] temp1;
logic [25:0] temp2;
logic [25:0] temp3;
logic [25:0] temp4;
logic [9:0] mid1;
logic [9:0] mid2;
logic [9:0] mid3;
logic [9:0] mid4;
logic [8:0] temp_out;
logic [8:0] temp_ind;
logic [8:0] mid_ind;
logic round1;
logic round2;
logic round3;
logic round4;
always_ff @(posedge clk) begin
        // First stage
        temp1 <= 300*(input_val1 - min);
        temp2 <= 300*(input_val2 - min);
        temp3 <= 300*(input_val3 - min);
        temp4 <= 300*(input_val4 - min);
        temp_ind <= input_index;

        //Second stage
        if (range[16:15] == 2'b01 || range[16:14] == 3'b001) begin
                adj_range <= 17'b10000000000000000;
                mid1[9:0] <= temp1[25:16];
                mid2[9:0] <= temp2[25:16];
                mid3[9:0] <= temp3[25:16];
```

```
        mid4[9:0] <= temp4[25:16];
        if (temp1[15] == 1'b1) begin
        round1 <= 1'b1;
        end
        if (temp1[15] == 1'b0) begin
        round1 <= 1'b0;
        end
        if (temp2[15] == 1'b1) begin
        round2 <= 1'b1;
        end
        if (temp2[15] == 1'b0) begin
        round2 <= 1'b0;
        end
        if (temp3[15] == 1'b1) begin
        round3 <= 1'b1;
        end
        if (temp3[15] == 1'b0) begin
        round3 <= 1'b0;
        end
        if (temp4[15] == 1'b1) begin
        round4 <= 1'b1;
        end
        if (temp4[15] == 1'b0) begin
        round4 <= 1'b0;
        end
end
if (range[16:13] == 4'b0001 || range[16:12] == 5'b00001) begin
        adj_range <= 17'b00100000000000000;
        mid1[9:0] <= temp1[23:14];
        mid2[9:0] <= temp2[23:14];
        mid3[9:0] <= temp3[23:14];
        mid4[9:0] <= temp4[23:14];
        if (temp1[13] == 1'b1) begin
        round1 <= 1'b1;
        end
        if (temp1[13] == 1'b0) begin
        round1 <= 1'b0;
        end
        if (temp2[13] == 1'b1) begin
        round2 <= 1'b1;
        end
        if (temp2[13] == 1'b0) begin
        round2 <= 1'b0;
```

```
        end
        if (temp3[13] == 1'b1) begin
        round3 <= 1'b1;
        end
        if (temp3[13] == 1'b0) begin
        round3 <= 1'b0;
        end
        if (temp4[13] == 1'b1) begin
        round4 <= 1'b1;
        end
        if (temp4[13] == 1'b0) begin
        round4 <= 1'b0;
        end
end
if (range[16:11] == 6'b000001 || range[16:10] == 7'b0000001) begin
        adj_range <= 17'b00001000000000000;
        mid1[9:0] <= temp1[21:12];
        mid2[9:0] <= temp2[21:12];
        mid3[9:0] <= temp3[21:12];
        mid4[9:0] <= temp4[21:12];
        if (temp1[11] == 1'b1) begin
        round1 <= 1'b1;
        end
        if (temp1[11] == 1'b0) begin
        round1 <= 1'b0;
        end
        if (temp2[11] == 1'b1) begin
        round2 <= 1'b1;
        end
        if (temp2[11] == 1'b0) begin
        round2 <= 1'b0;
        end
        if (temp3[11] == 1'b1) begin
        round3 <= 1'b1;
        end
        if (temp3[11] == 1'b0) begin
        round3 <= 1'b0;
        end
        if (temp4[11] == 1'b1) begin
        round4 <= 1'b1;
        end
        if (temp4[11] == 1'b0) begin
        round4 <= 1'b0;
```

```
        end
end
if (range[16:9] == 8'b00000001 || range[16:8] == 9'b000000001) begin
        adj_range <= 17'b00000010000000000;
        mid1[9:0] <= temp1[19:10];
        mid2[9:0] <= temp2[19:10];
        mid3[9:0] <= temp3[19:10];
        mid4[9:0] <= temp4[19:10];
        if (temp1[9] == 1'b1) begin
        round1 <= 1'b1;
        end
        if (temp1[9] == 1'b0) begin
        round1 <= 1'b0;
        end
        if (temp2[9] == 1'b1) begin
        round2 <= 1'b1;
        end
        if (temp2[9] == 1'b0) begin
        round2 <= 1'b0;
        end
        if (temp3[9] == 1'b1) begin
        round3 <= 1'b1;
        end
        if (temp3[9] == 1'b0) begin
        round3 <= 1'b0;
        end
        if (temp4[9] == 1'b1) begin
        round4 <= 1'b1;
        end
        if (temp4[9] == 1'b0) begin
        round4 <= 1'b0;
        end
end
if (range[16:7] == 10'b0000000001 || range[16:6] == 11'b00000000001) begin
        adj_range <= 17'b00000000100000000;
        mid1[9:0] <= temp1[17:8];
        mid2[9:0] <= temp2[17:8];
        mid3[9:0] <= temp3[17:8];
        mid4[9:0] <= temp4[17:8];
        if (temp1[7] == 1'b1) begin
        round1 <= 1'b1;
        end
        if (temp1[7] == 1'b0) begin
```

```
                round1 <= 1'b0;
                end
                if (temp2[7] == 1'b1) begin
                round2 <= 1'b1;
                end
                if (temp2[7] == 1'b0) begin
                round2 <= 1'b0;
                end
                if (temp3[7] == 1'b1) begin
                round3 <= 1'b1;
                end
                if (temp3[7] == 1'b0) begin
                round3 <= 1'b0;
                end
                if (temp4[7] == 1'b1) begin
                round4 <= 1'b1;
                end
                if (temp4[7] == 1'b0) begin
                round4 <= 1'b0;
                end
        end
        if (range[16:5] == 12'b000000000001 || range[16:4] == 13'b0000000000001)
begin
                adj_range <= 17'b00000000001000000;
                mid1[9:0] <= temp1[15:6];
                mid2[9:0] <= temp2[15:6];
                mid3[9:0] <= temp3[15:6];
                mid4[9:0] <= temp4[15:6];
                if (temp1[5] == 1'b1) begin
                round1 <= 1'b1;
                end
                if (temp1[5] == 1'b0) begin
                round1 <= 1'b0;
                end
                if (temp2[5] == 1'b1) begin
                round2 <= 1'b1;
                end
                if (temp2[5] == 1'b0) begin
                round2 <= 1'b0;
                end
                if (temp3[5] == 1'b1) begin
                round3 <= 1'b1;
                end
```

```
                    if (temp3[5] == 1'b0) begin
                    round3 <= 1'b0;
                    end
                    if (temp4[5] == 1'b1) begin
                    round4 <= 1'b1;
                    end
                    if (temp4[5] == 1'b0) begin
                    round4 <= 1'b0;
                    end
                end
                if (range[16:3] == 14'b00000000000001 || range[16:2] ==
15'b000000000000001) begin
                    adj_range <= 17'b00000000000010000;
                    mid1[9:0] <= temp1[13:4];
                    mid2[9:0] <= temp2[13:4];
                    mid3[9:0] <= temp3[13:4];
                    mid4[9:0] <= temp4[13:4];
                    if (temp1[3] == 1'b1) begin
                    round1 <= 1'b1;
                    end
                    if (temp1[3] == 1'b0) begin
                    round1 <= 1'b0;
                    end
                    if (temp2[3] == 1'b1) begin
                    round2 <= 1'b1;
                    end
                    if (temp2[3] == 1'b0) begin
                    round2 <= 1'b0;
                    end
                    if (temp3[3] == 1'b1) begin
                    round3 <= 1'b1;
                    end
                    if (temp3[3] == 1'b0) begin
                    round3 <= 1'b0;
                    end
                    if (temp4[3] == 1'b1) begin
                    round4 <= 1'b1;
                    end
                    if (temp4[3] == 1'b0) begin
                    round4 <= 1'b0;
                    end
                end
```

```
                if (range[16:1] == 16'b0000000000000001 || range[16:0] ==
17'b00000000000000000) begin
                        adj_range <= 17'b00000000000000100;
                        mid1[9:0] <= temp1[11:2];
                        mid2[9:0] <= temp2[11:2];
                        mid3[9:0] <= temp3[11:2];
                        mid4[9:0] <= temp4[11:2];
                        if (temp1[1] == 1'b1) begin
                        round1 <= 1'b1;
                        end
                        if (temp1[1] == 1'b0) begin
                        round1 <= 1'b0;
                        end
                        if (temp2[1] == 1'b1) begin
                        round2 <= 1'b1;
                        end
                        if (temp2[1] == 1'b0) begin
                        round2 <= 1'b0;
                        end
                        if (temp3[1] == 1'b1) begin
                        round3 <= 1'b1;
                        end
                        if (temp3[1] == 1'b0) begin
                        round3 <= 1'b0;
                        end
                        if (temp4[1] == 1'b1) begin
                        round4 <= 1'b1;
                        end
                        if (temp4[1] == 1'b0) begin
                        round4 <= 1'b0;
                        end
                end
                mid_ind <= temp_ind;

                //Third stage
                output_val1 <= 320 - (mid1 + round1);
                output_val2 <= 320 - (mid2 + round2);
                output_val3 <= 320 - (mid3 + round3);
                output_val4 <= 320 - (mid4 + round4);
                output_index <= mid_ind;
        end

endmodule
```

```
//For finding min and max
module nothing_master(
                        input logic clk,
                        input logic [8:0] debug1,
                        input logic [16:0] debug2,
                        input logic begin_ranger,
                        input logic [16:0] disp1,
                        input logic [16:0] disp2,
                        input logic [16:0] disp3,
                        input logic [16:0] disp4,

                        output logic done_range,
                        output logic [16:0] max,
                        output logic [16:0] min
                        );


                        logic find_range;
                        logic ending;
                        logic [8:0] counter;
                        logic [16:0] min1;
                        logic [16:0] max1;
                        logic [16:0] min2;
                        logic [16:0] max2;
                        logic [16:0] min3;
                        logic [16:0] max3;
                        logic [16:0] min4;
                        logic [16:0] max4;
                        logic [16:0] range;

                        always_ff @(posedge clk) begin
                                if (ending == 1'b1) begin
                                        done_range <= 1'b0;
                                end

                                if (begin_ranger == 1'b1) begin
                                        find_range <= 1'b1;
                                        min1 <= 17'b11111111111111111;
                                        max1 <=17'b0;
                                        min2 <= 17'b11111111111111111;
                                        max2 <=17'b0;
                                        min3 <= 17'b11111111111111111;
```

```
                max3 <=17'b0;
                min4 <= 17'b11111111111111111;
                max4 <=17'b0;
                counter <= 1'b0;
                max <= 1'b0;
                min <= 1'b0;
            end

            if (find_range == 1'b1) begin
                counter <= counter + 1'b1;
                if (counter == 400) begin
                    find_range <= 1'b0;
                    done_range <= 1'b1;
                    ending <= 1'b1;

                    if (min1 < min2 && min1 < min3 && min1 < min4)
begin
                        min <= min1;
                    end
                    if (min2 < min1 && min2 < min3 && min2 < min4)
begin
                        min <= min2;
                    end
                    if (min3 < min1 && min3 < min2 && min3 < min4)
begin
                        min <= min3;
                    end
                    if (min4 < min1 && min4 < min2 && min4 < min3)
begin
                        min <= min4;
                    end

                    if (max1 > max2 && max1 > max3 && max1 >
max4) begin
                        max <= max1;
                    end
                    if (max2 > max1 && max2 > max3 && max2 >
max4) begin
                        max <= max2;
                    end
                    if (max3 > max1 && max3 > max2 && max3 >
max4) begin
                        max <= max3;
```

```
                                        end
                                        if (max4 > max1 && max4 > max2 && max4 >
max3) begin
                                                max <= max4;
                                        end

                                end

                                if (disp1 < min1) begin
                                        min1 <= disp1;
                                end
                                if (disp2 < min2) begin
                                        min2 <= disp2;
                                end
                                if (disp3 < min3) begin
                                        min3 <= disp3;
                                end
                                if (disp4 < min4) begin
                                        min4 <= disp4;
                                end
                                if (disp1 > max1) begin
                                        max1 <= disp1;
                                end
                                if (disp2 > max2) begin
                                        max2 <= disp2;
                                end
                                if (disp3 > max3) begin
                                        max3 <= disp3;
                                end
                                if (disp4 > max4) begin
                                        max4 <= disp4;
                                end
                        end


                end

endmodule

module stock_master(
                input logic clk,
                input logic [16:0] stock_in,
```

```
input logic change_flag,
input logic request_disp,
output logic new_disp,

output logic [16:0] disp_stock,
output logic [16:0] disp_avg4,
output logic [16:0] disp_avg32,
output logic [16:0] disp_avg128,
output logic update_begin,
output logic update_done
);

//Declaring memory modules for the four curves
memory_chunk mem_4(
                        .clk(clk),
                        .din(stock_mem_in4),
                        .a(stock_mem_addr4),
                        .we(we_4),
                        .dout(stock_mem_out4)
);
memory_chunk mem_32(
                        .clk(clk),
                        .din(stock_mem_in32),
                        .a(stock_mem_addr32),
                        .we(we_32),
                        .dout(stock_mem_out32)
);
memory_chunk mem_128(
                        .clk(clk),
                        .din(stock_mem_in128),
                        .a(stock_mem_addr128),
                        .we(we_128),
                        .dout(stock_mem_out128)
);
memory_chunk mem_S(
                        .clk(clk),
                        .din(stock_mem_inS),
                        .a(stock_mem_addrS),
                        .we(we_S),
                        .dout(stock_mem_outS)
);

logic [16:0] stock_mem_out4;
```

```
logic [16:0] stock_mem_in4;
logic [13:0] stock_mem_addr4;
logic [16:0] stock_mem_out32;
logic [16:0] stock_mem_in32;
logic [13:0] stock_mem_addr32;
logic [16:0] stock_mem_out128;
logic [16:0] stock_mem_in128;
logic [13:0] stock_mem_addr128;
logic [16:0] stock_mem_outS;
logic [16:0] stock_mem_inS;
logic [13:0] stock_mem_addrS;

logic we_S;
logic we_4;
logic we_32;
logic we_128;

logic [16:0] temp_disp1;
logic [16:0] temp_disp2;
logic [16:0] temp_disp3;
logic [16:0] temp_disp4;

logic [16:0] ema_4;
logic [16:0] ema_32;
logic [16:0] ema_128;

logic [23:0] past_ema128;
logic [22:0] past_ema32;
logic [22:0] past_ema4;

logic [19:0] new_4;

logic [23:0] sum_128;
logic [22:0] sum_32;
logic [22:0] sum_4;

logic round_4;
logic round_32;
logic round_128;

logic [8:0] last_stock_counter;
logic [8:0] draw_counter;
```

```
logic last_change_flag;
logic sum_together;
logic divide_sum;
logic write_mem;
logic wait_1;
logic update_draw;
logic drawing_done;
logic zero_we;
logic do_nothing;

//Flip flop for stock
always_ff @(posedge clk) begin
	last_change_flag <= change_flag;
	if (change_flag != last_change_flag) begin
		past_ema128 <= 127*ema_128;
		past_ema32 <= 63*ema_32;
		past_ema4 <= 61*ema_4;
		new_4 <= 3*stock_in;
		sum_together <= 1'b1;
	end
	if (sum_together == 1'b1) begin
		sum_128 <= past_ema128 + stock_in;
		sum_32 <= past_ema32 + stock_in;
		sum_4 <= past_ema4 + new_4;
		divide_sum <= 1'b1;
		sum_together <= 1'b0;
	end
	if (divide_sum == 1'b1) begin
		ema_4[16:0] <= sum_4[22:6];
		if (sum_4[5] == 1'b1) begin
			round_4 <= 1'b1;
		end
		ema_32[16:0] <= sum_32[22:6];
		if (sum_32[5] == 1'b1) begin
			round_32 <= 1'b1;
		end
		ema_128[16:0] <= sum_128[23:7];
		if (sum_128[6] == 1'b1) begin
			round_128 <= 1'b1;
		end
		divide_sum <= 1'b0;
		wait_1 <= 1'b1;
	end
```

```
                        if (round_4 == 1'b1) begin
                                ema_4 <= ema_4 + 1'b1;
                                round_4 <= 1'b0;
                        end
                        if (round_32 == 1'b1) begin
                                ema_32 <= ema_32 + 1'b1;
                                round_32 <= 1'b0;
                        end
                        if (round_128 == 1'b1) begin
                                ema_128 <= ema_128 + 1'b1;
                                round_128 <= 1'b0;
                        end

                        if (wait_1 == 1'b1) begin //need this in case rounding occurs
(otherwise stock in values take non rounded value)
                                wait_1 <= 1'b0;
                                write_mem <= 1'b1;
                        end

                        //write to memory now, have flag of which reading from, set that
up too!!!!!!! only read or write at one time.
                        if (write_mem == 1'b1) begin

                                stock_mem_addr4 <= last_stock_counter;
                                stock_mem_addr32 <= last_stock_counter;
                                stock_mem_addr128 <= last_stock_counter;
                                stock_mem_addrS <= last_stock_counter;

                                stock_mem_in4 <= ema_4;
                                stock_mem_in32 <= ema_32;
                                stock_mem_in128 <= ema_128;
                                stock_mem_inS <= stock_in;

                                we_S <= 1'b1;
                                we_4 <= 1'b1;
                                we_32 <= 1'b1;
                                we_128 <= 1'b1;

                                if (last_stock_counter >= 399) begin
                                        last_stock_counter <= 0;
                                end
                                if (last_stock_counter < 399) begin
```

```
                        last_stock_counter <= last_stock_counter + 1'b1;
                end
                zero_we <= 1'b1;
                write_mem <= 1'b0;
        end

        if (zero_we) begin
                we_S <= 1'b0;
                we_4 <= 1'b0;
                we_32 <= 1'b0;
                we_128 <= 1'b0;
                zero_we <= 1'b0;
        end

        if (request_disp == 1'b1) begin
                stock_mem_addr4 <= last_stock_counter;
                stock_mem_addr32 <= last_stock_counter;
                stock_mem_addr128 <= last_stock_counter;
                stock_mem_addrS <= last_stock_counter;
                update_draw <= 1'b1;
        end

        if (update_draw == 1'b1) begin
                //make put the output of the mem in the write block syntax
wise!!!!
                update_begin <= update_begin + 1'b1;
                disp_stock <= stock_mem_outS;
                disp_avg4 <= stock_mem_out4;
                disp_avg32 <= stock_mem_out32;
                disp_avg128 <= stock_mem_out128;

                new_disp <= new_disp + 1'b1;

                if (draw_counter < 399) begin
                        if (stock_mem_addrS < 399) begin
                                stock_mem_addr4 <= stock_mem_addr4 +
1'b1;
                                stock_mem_addr32 <=
stock_mem_addr32 + 1'b1;
                                stock_mem_addr128 <=
stock_mem_addr128 + 1'b1;
                                stock_mem_addrS <= stock_mem_addrS
+ 1'b1;
```

```
                                        end
                                        if (stock_mem_addrS >= 399) begin
                                                stock_mem_addrS <= 1'b0;
                                                stock_mem_addr4 <= 1'b0;
                                                stock_mem_addr32 <= 1'b0;
                                                stock_mem_addr128 <= 1'b0;
                                        end

                                        draw_counter <= draw_counter + 1'b1;
                                end
                                if (draw_counter >= 399) begin
                                        update_draw <= 1'b0;
                                        update_done <= update_done + 1'b1;
                                        draw_counter <= 0;
                                end
                        end
                end //for always_ff




endmodule

module memory_chunk(input logic        clk,
                input logic [13:0]  a,
                input logic [16:0]  din,
                input logic             we,
                output logic [16:0] dout);

  logic [16:0]                  mem [399:0];

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

endmodule
```

VGA_LED.sv with Bresenham's Line Drawing

VGA_LED_Emulator.sv with framebuffer