FASTRADE :Final Report

Spring 2024

Yixuan Li (yl5468), Wenbo Liu (wl2927), Weitao Lu (wl2928), Xiaolei Zhao (xz3283)

Table of Contents

1. Abstract

FPGA has demonstrated significant advantages in high-frequency trading systems. However, due to the limited support of existing machine learning libraries on FPGA, its application in multi-factor quantitative trading models is constrained. Our project aims to address this issue by deploying a machine learning-based multi-factor pipeline on FPGA to achieve acceleration and provide a more user-friendly VGA display and user interaction. Our factor model pipeline includes software-based data preprocessing, model training, and inference, as well as hardware-based factor calculation and VGA display.

First, we used Python libraries to build a multi-factor model pipeline based on Kaggle stock data, and then rewrote all relevant library functions in C to deploy the factor calculation algorithms and factor model on FPGA. Subsequently, we re-implemented all C-based factor calculation algorithms part in System Verilog to accelerate the strategy computation process. Additionally, the project introduced hardware interfaces like keyboards and VGA displays to improve user interaction. A VGA user display was designed to let users choose trading strategies, train models, predict on new data and visualize results.

The project's highlights include:

1. Implementing multi-factor value calculations using System Verilog.
2. Significantly accelerating hardware computation speed by optimizing data types and designing division logic.

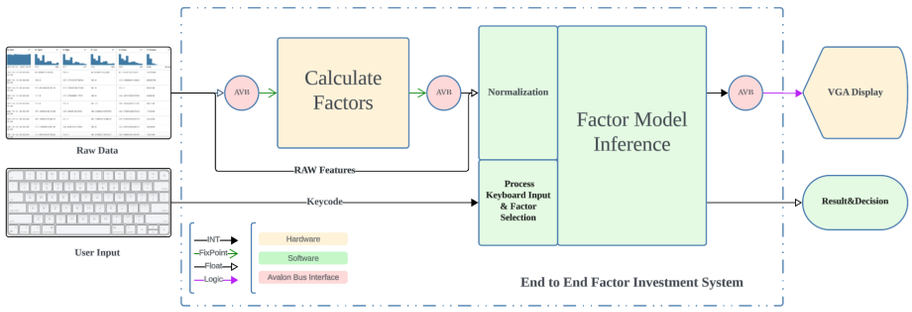
3. Rewriting all Python library functions in C for machine learning based factor model, significantly improving training speed by 4 times than Google Colab.
4. Introducing a software-hardware data transfer interface for efficient data interaction between factor calculation and model training/inference.
5. Designing a complete VGA display that supports user to select a personalized multi-factor strategy combinations and result visualization.

This FPGA-based multi-factor pipeline significantly improves the computational efficiency of trading strategies, providing an efficient solution for quantitative trading systems.

Our comprehensive end-to-end system for users is depicted below. Users can use their data and select the features they want to consider, and our system will generate the final result and the decision(sell, buy or hold) automatically. The result will be displayed by VGA.

Within the dashed box is our system, which computes the relevant factors on hardware, use the factor model on software to predict result and then outputs the results to a VGA display for visualization.

The data type and the type of each component is illustrated in this figure using different color/arrow.



2. Motivation

FPGAs are widely used to accelerate different system components in modern High-Frequency Trading systems such as the network stack, financial protocol parsing, order book handling and custom trading algorithms.[1] It's mainly used in tasks like order book generation and simple one factor computation. The major applications as well as the projects in this course before are more focused on these two tasks.

However, as machine learning algorithms advance and multi-factor investment strategies become more prevalent, new trading models have become increasingly complex. Many quantitative traders at investment funds prefer using high-level programming languages like Python, which are equipped with machine learning libraries, to compute factor values on CPUs or GPUs. As a result, the application scope of FPGA has been relatively limited.

Although FPGA has a significant advantage in computational speed, its application in complex multi-factor models has gradually decreased due to the lack of direct support for existing machine learning libraries and the complexity of programming. Recognizing this gap, we decided to apply a modern, machine-learning-based multi-factor model to FPGA. Through this project, we aim to fully leverage FPGA's hardware characteristics to overcome programming and integration challenges, achieving more efficient multi-factor computation and providing a novel solution for quantitative strategies in high-frequency trading. Based on the knowledge and the multi-factor model we select, we learned in labs, we want to design a VGA display system that supports users to select their personalized multi-factor strategy and see the corresponding result, which is also helpful for traders to compare different strategies in real world trading.

3. Related Work

3. 1 Factors

3.1.1 Price Momentum Factor:

First, we will choose a time frame for measuring momentum. Then calculate the asset over the chosen time frame with the following formula:

$$Price\ Momentum = (Price_{end} \ / \ Price_{start}) - 1$$

For comparison purposes, we will normalize the momentum values, which can be done by dividing the momentum of each asset by the standard deviation of all momentum values.

This is the first code we used in python, we then implement it using C and system verilog.

```
def get_price_n_days_ago(current_index, days):
    look_back_index = current_index - days
    if look_back_index < 0:
        return None
    return data.loc[look_back_index, 'Close']

# Calculate the price 'n_days' ago for each date
data['Price_30_Days_Ago'] = [get_price_n_days_ago(i, n_days) for i in range(len(data))]

# Calculate the price momentum factor
data['Price_Momentum'] = (data['Close'] - data['Price_30_Days_Ago']) / data['Price_30_Days_Ag
```

3.1.2 Volume Factor:

First, we will choose a time frame to calculate the volume factor. For the average volume calculation, we will use the following formula:

$$Volume\ Factor = \frac{\Sigma\ Volume_i}{N}$$

```
volume_factor = data['Volume'].sum() / len(df['Volume'])
```

3.1.3 Relative Strength Index

RSI is a momentum oscillator that measures the speed and change of price movements. RSI values range from 0 to 100 and are typically used to identify overbought or oversold conditions in a traded asset. An asset is usually considered overbought when the RSI is above 70 and oversold when it's below 30.

The formula for calculating the Relative Strength Index (RSI) is:

$$RSI = 100 - (\frac{100}{1+RS})$$

where **RS** (Relative Strength) is the ratio of the average gain of the periods that closed up to the average loss of the periods that closed down. These averages are typically calculated over a 14-day period, which is the standard period used by J. Welles Wilder when he introduced the indicator, but the period can be adjusted to suit different trading strategies and time frames.

The code is like this:

```
import pandas as pd

# Assuming 'df' is a DataFrame containing your stock's price data, and 'close' is the column
df['delta'] = df['close'].diff() # Step 1: Calculate daily returns
df['gain'] = df['delta'].clip(lower=0) # Step 2: Isolate gains
df['loss'] = -df['delta'].clip(upper=0) # Step 2: Isolate losses

# Step 3: Calculate the averages of the gains and losses
avg_gain = df['gain'].rolling(window=14, min_periods=14).mean()
avg_loss = df['loss'].rolling(window=14, min_periods=14).mean()

# Step 4: Calculate RS
rs = avg_gain / avg_loss
```

```
# Step 5: Calculate RSI
df['RSI'] = 100 - (100 / (1 + rs))
```

3.1.4 Moving Average

First, decide how many periods (days, weeks, minutes, etc.) we want to include in moving average. Then sum up the closing prices of the stock for the last N periods. Finally, divide the total sum of the closing prices by N.

The formula is as following:

$MA = \frac{\Sigma\, Price_i}{N}$

```
data['30d_Moving_Average'] = data['Close'].rolling(window=30).mean()
```

3. 2 Factor Model

Factor Models(FM) is a standard multifactor weighting model in factor investment and recommend systems. Besides a simple linear (order-1) interactions among features, FM models pairwise (order-2) feature interactions as inner product of respective feature latent vectors.

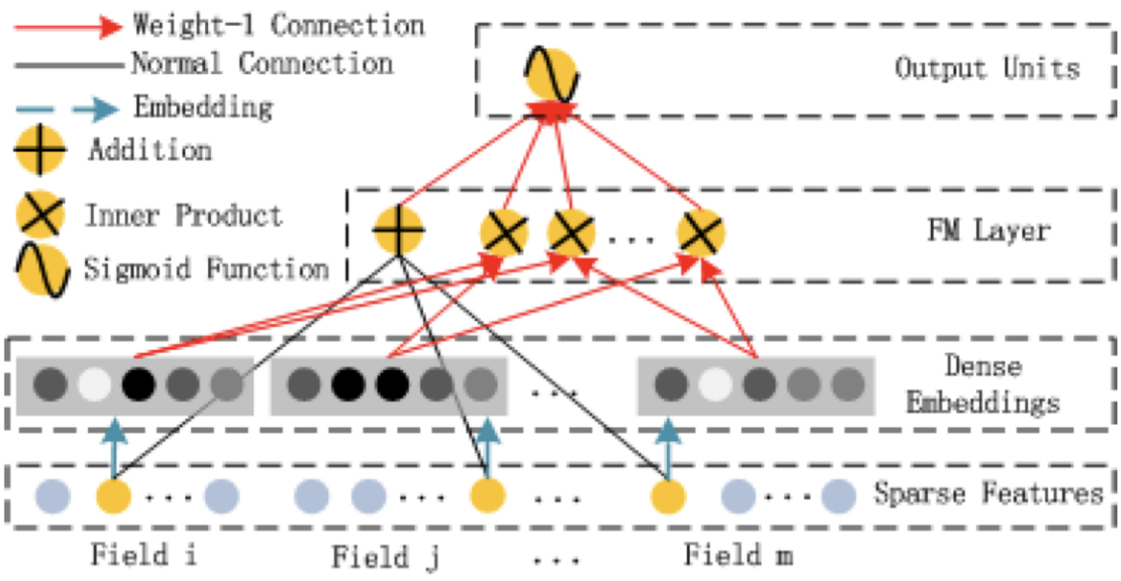


Figure 1: Factor Models Interactions Diagram[4]

In modern Factor Models, there're many different ways in Feature selection and Embedding Method. Our feature pool includes the raw features of our input data and the factors calculated on hardware. We take the idea of [5] to use the interaction weight ,which named as **Vector Weights** as implicit embeddings of the features.

The individual weight of each factor and the interaction weight between them are zero-initialized and updated by gradient decent during training process.

We enabled the keyboard input to select used factors during inference step, which exclude the factors from the factor pool by zerolize its individual weight and all its interaction weight with other factors.

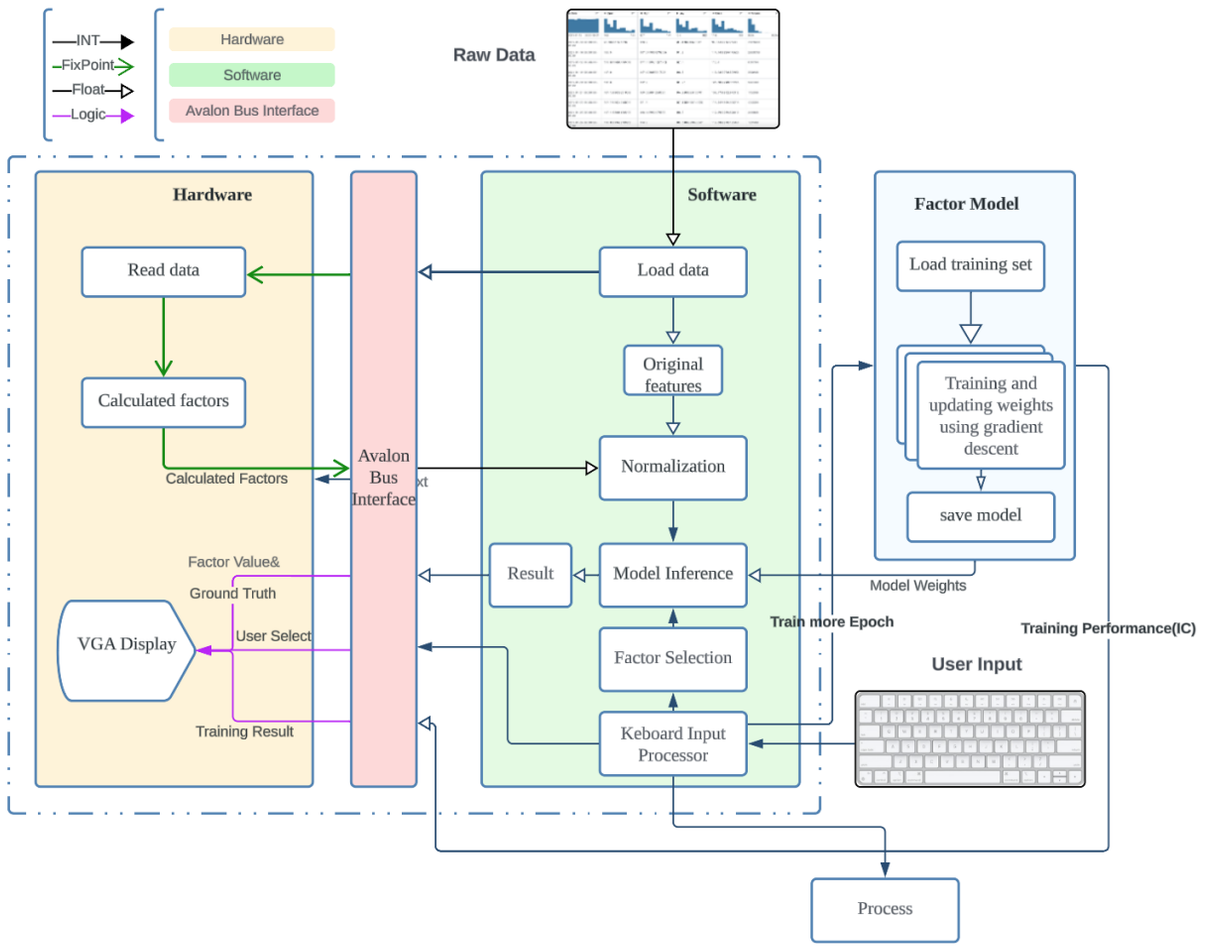
4. System Design and Components

We have different software and hardware components, the components type and data type between different components is shown in different color in the figure below.

Our software begins by acquiring raw data, which it then converts into fixed-point format before transmitting it to the hardware via the Avalon bus interface.

When the user want to do inference on new data, the hardware processes these calculations and sends the 4 factor results back to the software. Subsequently, the software converts these 4 results back into floating-point numbers and integrates the newly calculated factors with the original 5 factors. Based on users' feature selection, the factor model will assign weights and interaction weights to each factor and compute the final float point result. The decision(sell, buy or hold) is then based on this result by a threshold logic.

When the user want to train a new model, they can compute the factors of new raw data either using hardware or software, and use our interfaces to load their raw data, factors. The label we use for each data is the next day return. Once the user press train, the model's weights will be trained for 20 epochs. The weights is stored in memory, and can be used for inference on unlabeled new data.



5. Hardware Implementation

5. 1 Factor Calculation

In order to accelerate the factor calculation speed, we wrote System Verilog code to let the FPGA to calculate. The software part will write five data, including open price, high price, low price, close price and volume, to the hardware part through the bus in each cycle. For the hardware part, it will read the data based on the address and store the data into registers.

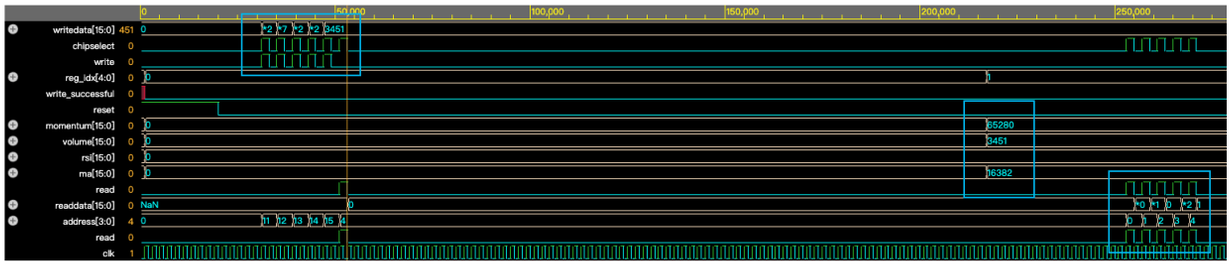
There is a 2D register array in the hardware part, which has 30 rows (because we set the window size to 30 days) in total and 9 registers in each row. In each row, the data stored as following:

reg_0	reg_1	reg_2	reg_3	reg_4	reg_5	reg_6	reg_7
Open Price	High Price	Low Price	Close Price	Volume	Momentum	Volume Factor	RSI

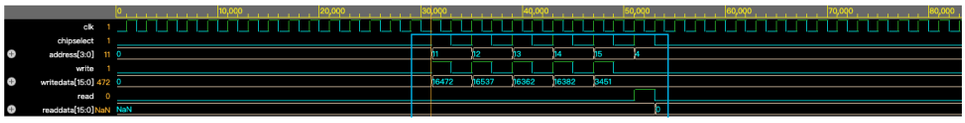
Because we used fixed point number to calculate the factors, we wrote a divider module which can calculate the division on fixed point numbers.

In the calculation process, when all the five base data has been stored into the first five registers of a row, it will begin to calculate the other four factors. Once all the four factors have been calculated and stored into each register, the index pointer will move to the next row. When the pointer moves to the final row(30th) of the array, the entire array will be shifted forward by one row and the data from the original first row will be discarded. When we get all the four new factors, we will update the registers, which will be sent back to the software part through the bus.

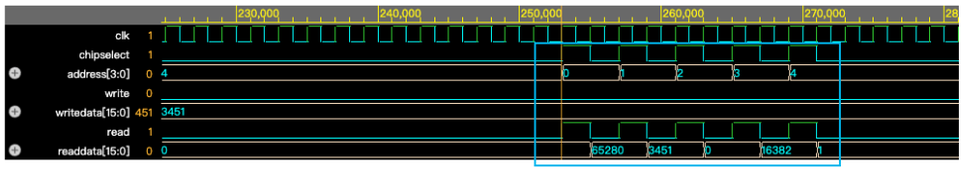
The following image shows the simulation result. We can see that the software part write data to the hardware part at the beginning and after several cycles, the hardware part will finish calculating the factors and send the results through the bus. The software part will try to read the calculation result after sending all the five original data. Once the software gets the ‘done’ signal from the hardware part, it will read the four factors and get ready to process them.



The following image shows the writing data cycles.



The following image shows the reading data cycles:



The code in the final part of this report shows more details.

The division module used a state machine to control the operations. Upon receiving a **start** signal, the module checks for special cases (divide-by-zero or overflow). If none, it proceeds to initialize absolute values of **a** and **b**, determines sign differences, and enters the **CALC** state. The core of the algorithm uses a loop to perform the division using shift and subtract methods. This is implemented in the **CALC** state, where each iteration potentially shifts the quotient and subtracts the divisor from the accumulator. An accumulator is used to handle the subtraction and shifting process. It's initially set to the dividend, and the divisor is subtracted in a loop based on comparisons. After the main iterations, the **ROUND** state may adjust the quotient based on the least significant bits to implement rounding (Gaussian rounding). If the signs of the dividend and divisor were different and the result is non-zero, the result's sign is adjusted. The final state **SIGN** adjusts for the sign and transitions to **IDLE**, setting **done** and **valid** flags as necessary. If reset (**rst**) is asserted at any point, all outputs and internal states are reset.

In digital hardware division, particularly with fixed-point numbers, rounding is crucial to maintain precision within the constraints of limited bit width. Rounding can help to minimize truncation errors and ensures that results conform to predefined formats, such as the specified number of fractional bits (`FBITS`). Rounding is particularly important in applications like signal processing, where accumulated errors can degrade performance.

5.2 VGA Display

5.2.1 Display Implementation

Our screen resolution is 680 × 480 pixels. Based on our testing results, we decided to use an 8 × 8-bit bitmap to represent each character. This allows us to divide the screen into 80 columns and 60 rows. We have created 8 × 8 bitmaps for the alphabet, numbers, and some punctuation marks. Each bitmap is represented as a two-dimensional array, where the value '0' represents black and the value '1' represents a color. This binary scheme simplifies the

display of images and color coding within the system. The use of bitmaps allows for efficient storage and rendering of these characters. We also designed a custom icon for our group, which is 64 × 64 pixels in size. The icon features a current and arrows, symbolizing stock trending and trading activities.

Example of bitmap 'A':

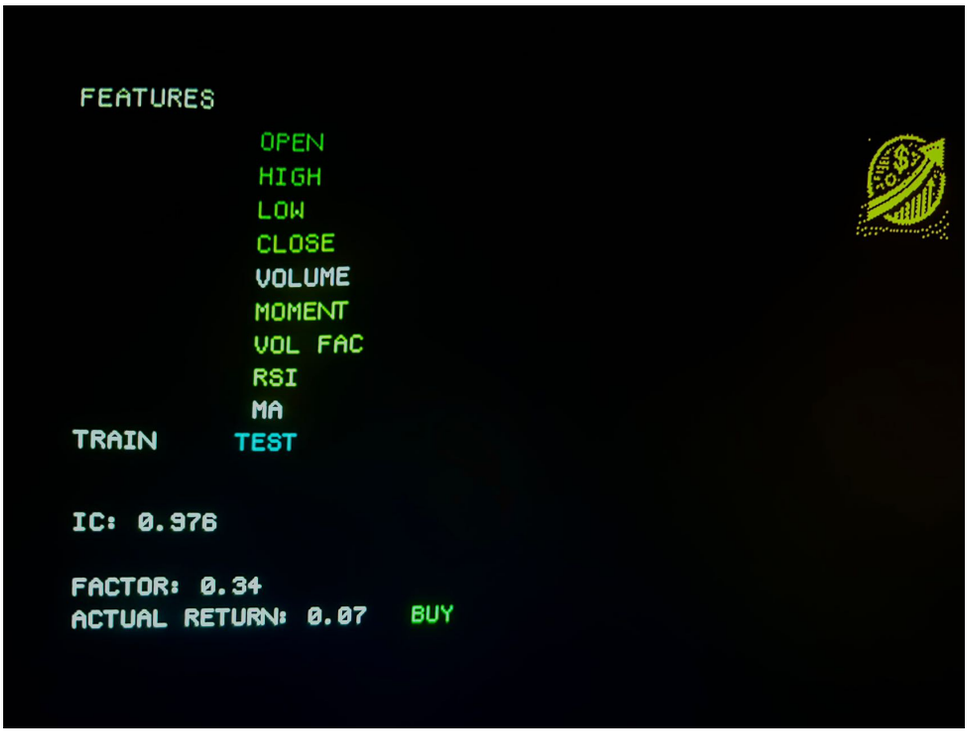
```
char_a[0] = 8'b00011000; //    **
char_a[1] = 8'b00100100; //   *  *
char_a[2] = 8'b01000010; //  *    *
char_a[3] = 8'b01111110; //  ******
char_a[4] = 8'b01000010; //  *    *
char_a[5] = 8'b01000010; //  *    *
char_a[6] = 8'b01000010; //  *    *
char_a[7] = 8'b01000010; //  *    *
```

5.2.2 Registers

We use a 16-bit status register where each bit corresponds to a feature, or the selection status for the "Train" or "Test" options, with 0 indicating deselection and 1 indicating selection. The IC value, actual return, and factor value are transmitted and displayed as individual digits within a range of 0-9. The sign is controlled by three bits: bit 0 and bit 1 determine the signs of the actual return and factor value, respectively. Bit 2 indicates the initial state of the program; when no strategy or value is active, this bit is set to 0. Setting this bit to 1 activates the program, signaling the commencement of operations. Additionally, the cursor's position is stored within a range from 0 to 9, enabling precise navigation and selection within the user interface.

5.2.3 UI Design

- **Cursor Navigation:** The current position of the cursor is highlighted in blue. Users can navigate through the interface by pressing the up, down, left, and right arrow keys to move cursor to select features or to choose the "Train" and "Test" options for model operations.
- **Feature Selection**: Eight features are listed at the top of the screen. Features that have been selected for inference in the model are highlighted in green. Users can press Enter to select or deselect these features, which is easy to customize the input data of the model.
- **Training Model**: The "Train" option initiates the model training process. After training, the related mode Information Coefficient (IC) value is displayed at the bottom of the screen, formatted to three decimal places, which provides the statistical evaluation of the model.
- **Model Prediction and Strategy Display**: After training, the "Test" option enables the prediction phase, where the trained model is used to predict based on new and existing data based on the selected features. The next day return and the calculated factor value will show below. And also a trading strategy BUY/SELL/HOLD will display to help users make timely decisions for the current stock. These strategies are color-coded to intuitively reflect market trends which live up to up and down movements of stock market indicators.



6. Factor Model's Software Implementation

6.1 Data Format and Preprocessing

6.1.1 Data Format

In software computation, we defined a Matrix struct based on double array, and reproduced all related Matrix computation functions in C.

```c
// matrix.h
#ifndef MATRIX_H
#define MATRIX_H

typedef struct {
  double **data;
  int rows;
  int cols;
} Matrix;

Matrix create_matrix(int rows, int cols);
void free_matrix(Matrix m);
Matrix multiply_matrices(Matrix a, Matrix b);
void elementwise_multiply(Matrix a, Matrix b, Matrix result);
double sum_matrix_elements(Matrix m);
void init_random_normal(Matrix m, double mean, double stddev);
double sigmoid(double x);
void compute_means(Matrix features, double *means);
void compute_stddevs(Matrix features, double *means, double
*stddevs);
void standardize_features(Matrix features, double *means, double
*stddevs);
double pearson_correlation(double *x, double **y, int n);
#endif
```

6.1.2 Preprocessing

Prior to model training, it is essential to preprocess the data to ensure its suitability for analysis. This typically involves several steps such as handling missing values, outlier detection, and feature scaling.

One crucial preprocessing technique is Z-score normalization, particularly when working with numerical data. Z-score normalization transforms the data distribution to have a mean of zero and a standard deviation of one, effectively standardizing the scale of the features. This step is imperative, especially when using double precision data types in C, as it mitigates the risk of gradient overflow during optimization algorithms like gradient descent.

Z-score Normalization Formula:

The Z-score normalization formula is defined as follows:

$$Z = \frac{X - \mu}{\sigma}$$

Where:

- X represents the original value of the feature.
- μ denotes the mean of the feature.
- σ signifies the standard deviation of the feature.
- Z is the standardized value after normalization.

By applying Z-score normalization, we ensure that the data is appropriately scaled, thereby facilitating more stable and efficient model training, particularly in the context of factor models utilizing double precision data types.

6.2 Model Training

We defined a structure called Fm_model to provide the training function and predict function of our factor model.

```
// fm_model.h
#ifndef FM_MODEL_H
#define FM_MODEL_H

#include "matrix.h"
#include <stdbool.h>

typedef struct {
  double bias;    // Scalar bias term w0 in the model
  Matrix weights; // Weight vector w in the model
  Matrix factors; // Factorization matrix V in the model
} FMModel;

void fit(FMModel *model, Matrix X, Matrix y, int feature_potential,
         double alpha, int iter, int batch_size, double decay_rate);
double *predict(FMModel *model, Matrix X);
double *predict_active(FMModel *model, Matrix X, Matrix Ref ,bool
*active_features);
#endif
```

And we used mini-batch and learning rate decay method to avoid gradient overflow.[2][3]

We split our data into 80% training, 10% testing, and 10% validation. The models trained weights are stored in local memory and can be used upon calling for inference.

6.3 Inference

We use the predict_active function to predict the result, with a filter called active_features.

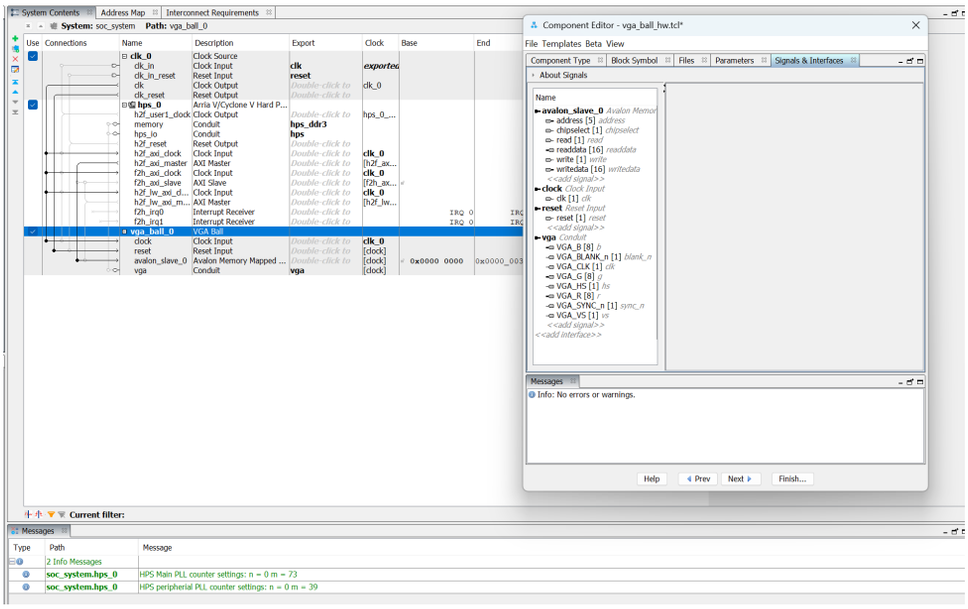
Users' keyboard input will influence this array, and during inference, the factor weight and interaction weight of unselected features will be set to zero. The input X is 5 raw features + 4 factors calculated by hardware, and input Ref provides users are more flexible interface to fuse

their other strategies (a strategy derived from other theory and has a strong correlation with the return) with ours.

7. Hardware-Software Interface

7.1 Avalon Bus Structure

In this project, we developed a top-level Verilog file that integrates the `calculator` module and the `VGA` module. The `writedata` signal, which is 16 bits wide, is used to input data, while the `address` signal determines the specific field to which the data is written. Similarly, for reading operations, the `address` signal specifies the field to be read, and the retrieved data is stored in the `readdata` signal, which is also 16 bits wide.
Below is the structure of our avalon bus.



7.2 Address Map

7.2.1 VGA Address Map

This is our address map for VGA.

addr 0	addr 1	addr 2	addr 3	addr 4	addr 5	addr 6	addr 17
Status	Actual Return	Actual Return	IC	IC	IC	Sign	Factor

- Address 0: Indicates the status of the features. A value of 1 signifies that the feature is selected, while a value of 0 signifies otherwise.
- Address 1 and 2: Store the Actual Return. For instance, if the Actual Return is 0.23, the process involves setting the address to 1 and `writedata` to 2, followed by setting the address to 2 and `writedata` to 3.
- Address 3 to 5: Store the IC value.
- Address 6: The first bit indicates the sign of the Actual Return (0 for positive, 1 for negative), and the second bit indicates the sign of the IC value (0 for positive, 1 for negative).
- Address 17 and 18: Store the factor value.
- Address 19: Represents the position of the cursor.

7.2.2 Calculator Address Map

Write to calculator:

addr 7	addr 8	addr 9	addr 10	addr 11
Open	High	Low	Close	Volume

Read from calculator:

addr 12	addr 13	addr 14	addr 15	addr 16
Momentum	Volume	RSI	MA	Done

8. Conclusion and Evaluation

8.1 Acceleration Performance Evaluation

Hardware

Calculation Time using Python(Colab)	**Calculation Time using Python(M1)**	**Calculation Time using SV**
0.0050289 s	0.0014479 s	0.0018 s

Software

Python Training /20epoch(Colab)	Python Inference Time(Colab)	Python Training /20epoch(M1)	Python Inference Time(M1)	On Board Training /20epoch	On Board Inference Time
1.103962 s	0.000150 s	0.225156 s	0.000026 s	0.130262 s	0.000035 s

We can see a significant improvement of speed in terms of cloud services like Google Colab.

Here Colab refers to Google Colab and M1 refers to Apple Macbook Pro with M1 Pro. All python test codes are available in appendix, where we only compute the time of computation without data loading/writing.

8.2 Future Work

We have proved the potential of deploying widely used factor investment pipeline on FPGA and

achieve potential speed improvement. Due to data limitation, we do not have large scale data to train our model, so it's overfit on our limited dataset and the precision on out-of-distribution new data is low. And also the transfer of fixpoint to float point decreases the calculation accuracy . One good way is to improve the algorithm's precision by using more advanced model, more data, and more precise rounding methods for different feature.

Another future improvement is to improve the inference speed by designing more balanced pipeline. In our current pipeline, the software must wait for the hardware's computation. A smarter design is to run these in parallel, because the computation of different factors take different time. And the software can use these factors as long as it's computed. Also it's better to integrate part of or whole software machine learning calculation to ystem Verilog.

8.3 Contribution

In this project, Wenbo Liu was responsible for factor calculator component, including the implementation of the Verilog file for both the calculator and the divider.

Weitao Lu focused on the machine-learning based factor model, designed our icon, and implemented the keyboard component.

Xiaolei Zhao and Yixuan Li completed the VGA component, developed the hardware-software interface, designed the keyboard input logic, and integrated all the different parts of the project.

We write test codes to generate hardware waveform and write python codes to evaluate time comparation together.

9. Reference

1. van Veldhoven, R., Maris, P., & Bertels, K. (2014). A Survey of FPGA-Based Accelerators for High-Frequency Trading. IEEE Transactions on Computational Social Systems, 1(2), 100-113.
2. LeCun, Y., Bottou, L., Orr, G. B., & Müller, K.-R. (2012). Efficient BackProp. In G. Montavon, G. B. Orr, & K.-R. Müller (Eds.), Neural Networks: Tricks of the Trade (pp. 9-48). Springer, Berlin, Heidelberg.
3. Sutskever, I., Martens, J., Dahl, G., & Hinton, G. (2013). On the importance of initialization and momentum in deep learning. Proceedings of the 30th International Conference on Machine Learning (ICML-13), 28, 1139-1147.
4. Guo, H., Tang, R., Ye, Y., Li, Z., & He, X. (2017). DeepFM: A Factorization-Machine based Neural Network for CTR Prediction. In Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI), 1725-1731. Link
5. Weinan Zhang, Tianming Du, and Jun Wang. Deep learning over multi-field categorical data - - A case study on user response prediction. In *ECIR*, 2016.

Appendix

Appendix

Code

Hello.c

```
#include <stdio.h>
#include "vga_ball.h"
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <string.h>
#include <unistd.h>
#include <stdlib.h>
#include <time.h>

#include <libusb-1.0/libusb.h>
#include "usbkeyboard.h"

#include "fm_model.h"
#include "matrix.h"
#include <math.h>

#define MIN_RANDOM 0
#define MAX_RANDOM 340
#define COLUMNS 5
#define FRACTIONAL_BITS 8
float fixed_to_float(int16_t fixed_number)
{
  return fixed_number / (float)(1 << FRACTIONAL_BITS);
}

int vga_ball_fd;

void set_ball_character(const mem *c)
{
```

```
  calculator vla;
  vla.memory = *c;
  if (ioctl(vga_ball_fd, VGA_BALL_WRITE_CHARACTER, &vla))
  {
    perror("ioctl(VGA_BALL_SET_character) failed");
    return;
  }
}

void set_ball_values(const mem *v)
{
  calculator vla;
  vla.memory = *v;
  // printf("VLA read: %d , %d, %d, %d, %d\n", vla.memory.ic1
  if (ioctl(vga_ball_fd, VGA_BALL_WRITE_VALUES, &vla))
  {
    perror("ioctl(VGA_BALL_SET_values) failed");
    return;
  }
}

void set_IC(const mem *v)
{
  calculator vla;
  vla.memory = *v;
  // printf("VLA read: %d , %d, %d, %d, %d\n", vla.memory.ic1
  if (ioctl(vga_ball_fd, VGA_BALL_WRITE_IC, &vla))
  {
    perror("ioctl(VGA_BALL_SET_IC) failed");
    return;
  }
}

void set_factors(const mem *c)
{
  calculator vla;
  vla.memory = *c;
  if (ioctl(vga_ball_fd, CALCULATOR_WRITE_VALUES, &vla))
```

```
  {
    perror("ioctl(calculator_SET_factors) failed");
    return;
  }
}

mem get_factors()
{
  calculator res;
  if (ioctl(vga_ball_fd, CALCULATOR_READ_VALUES, &res))
  {
    perror("ioctl(calculator_read_res) failed");
    return;
  }
  return res.memory;
}

int get_done()
{
  calculator res;
  if (ioctl(vga_ball_fd, CALCULATOR_READ_DONE, &res))
  {
    perror("ioctl(calculator_read_done) failed");
    return 0;
  }
  return res.memory.done;
}

//----------for keyboard
struct libusb_device_handle *keyboard;
uint8_t endpoint_address;

struct libusb_device_handle *openkeyboard(uint8_t *endpoint_a
void pressKeyAndPrint(struct libusb_device_handle *keyboard, 

// Keycode to ASCII mapping for a basic US keyboard layout wi
// This table includes the basic alphabet, numbers, and now t
char keycode_to_ascii[256] = {
```

```
    [0x04] = 'a', [0x05] = 'b', [0x06] = 'c', [0x07] = 'd', [
    [0x2C] = ' ',  // Space
    [0x28] = '\n', // Enter
    [0x4F] = '>',  // Right arrow
    [0x50] = '<',  // Left arrow
    [0x51] = 'v',  // Down arrow
    [0x52] = '^',  // Up arrow
};

void pressKeyAndPrint(struct libusb_device_handle *keyboard, 
{
  struct usb_keyboard_packet packet;
  int transferred;

  int res = 0;
  if (libusb_interrupt_transfer(keyboard, endpoint_address,
                                (unsigned char *)&packet, siz
                                &transferred, 5000) == 0 &&
      transferred == sizeof(packet))
  {
    if (packet.keycode[0] != 0)
    {
      char key = keycode_to_ascii[packet.keycode[0]];
      if (key == '\n')
        printf("Key Pressed: enter\n");
      else
        printf("Key Pressed: %c\n", key);
      if (key != '\0')
      {
        if (key == 'v')
        {

          if (*cur == 9 || *cur == 10)
          {
            *cur = 0;
          }
          else
          {
```

```
            *cur += 1;
          }
        }
        else if (key == '^')
        {
          if (*cur == 0)
          {
            *cur = 9;
          }
          else
          {
            *cur -= 1;
          }
        }
        else if (key == '<' || key == '>')
        {
          if (*cur == 9)
          {
            *cur = 10;
          }
          else
          {
            *cur = 9;
          }
        }
        else if (key == '\n')
        { // press enter key
          if (*cur == 9)
          {
            run_model(m, model, status);
            set_IC(m);
            printf("Key Pressed: %c\n", key);
          }
          else if (*cur == 10)
          {

            run_test(m, model, status, row_num, &data);
            *row_num = (rand() % (MAX_RANDOM - MIN_RANDOM + 1
```

```
            set_ball_values(m);
            printf("Key Pressed: %c\n", key);
          }
          else
          {
            clock_t start_time = clock();
            status[*cur] = status[*cur] ^ 1;
          }
        }
        for (int i = 10; i >= 0; i--)
        {
          if (i == *cur)
          {
            res <<= 1;
            res |= 1;
          }
          else
          {
            res <<= 1;
            res |= status[i];
          }
        }
        m->chars = res;
        m->cursor = *cur;
        printf("chars:%d current cur %d\n", m->chars, m->curs
      }
      else
      {
        printf("Unknown Key: %02x\n", packet.keycode[0]);
      }
    }
  }
}

//------------------model------------
// Function to load data from a CSV file
void load_data(const char *filename, Matrix *features, Matrix
{
```

```
  FILE *file = fopen(filename, "r");
  if (!file)
  {
    perror("Failed to open file");
    exit(EXIT_FAILURE);
  }

  int count = 0; // A counter to track how many rows have bee
  double check;  // A variable to check fscanf output tempora

  // Read the file line by line
  for (int i = 0; i < features->rows; i++)
  {
    int result = fscanf(file, "%lf,%lf,%lf,%lf,%lf,%lf,%lf,%l
                        &features->data[i][0], &features->dat
                        &features->data[i][2], &features->dat
                        &features->data[i][4], &features->dat
                        &features->data[i][6], &features->dat
                        &features->data[i][8], &labels->data[
    if (result == EOF)
    {
      printf("Reached EOF unexpectedly.\n");
      break; // Stop reading if we reach end of file prematur
    }
    else if (result != 10)
    {
      printf("Incorrect number of items read: %d\n", result);
      continue; // Skip this line if the data was incorrect
    }
    count++;
    // printf("Row %d - Last Feature: %f, Label: %f\n", i, fe
  }
  printf("Total rows read: %d\n", count);

  fclose(file);
}

void run_model(mem *m, FMModel *model, int *status)
```

```
{
  // Assume data has 100 rows and we are using 5 features
  int rows = 340;       // total number of data points
  int cols = 9;         // number of features
  int valid_rows = 10;  // 10% for validation
  int train_rows = 320; // 10% for training

  // Creating matrices for features and labels
  Matrix features = create_matrix(rows, cols);
  Matrix labels = create_matrix(rows, 1);

  // Load data from a pre-processed file
  load_data("processed_AFRM.csv", &features, &labels);
  double *means = malloc(cols * sizeof(double));
  double *stddevs = malloc(cols * sizeof(double));
  compute_means(features, means);
  compute_stddevs(features, means, stddevs);
  standardize_features(features, means, stddevs);

  // Splitting data into train and test manually
  Matrix features_train = create_matrix(train_rows, cols);
  Matrix labels_train = create_matrix(train_rows, 1);
  Matrix features_test = create_matrix(rows - train_rows - va
  Matrix labels_test = create_matrix(rows - train_rows - vali
  Matrix features_valid = create_matrix(valid_rows, cols);
  Matrix labels_valid = create_matrix(valid_rows, 1);

  // Copying data to train and test matrices
  for (int i = 0; i < train_rows; i++)
  {
    for (int j = 0; j < cols; j++)
    {
      features_train.data[i][j] = features.data[i][j];
    }
    labels_train.data[i][0] = labels.data[i][0];
  }
  // Copying data to test matrices
  for (int i = train_rows; i < rows - valid_rows; i++)
```

```
  {
    for (int j = 0; j < cols; j++)
    {
      features_test.data[i - train_rows][j] = features.data[i
    }
    labels_test.data[i - train_rows][0] = labels.data[i][0];
  }

  // Copying data to valid
  for (int i = rows - valid_rows; i < rows; i++)
  {
    for (int j = 0; j < cols; j++)
    {
      features_valid.data[i - rows + valid_rows][j] = feature
    }
    labels_valid.data[i - rows + valid_rows][0] = labels.data
  }

  fit(model, features_train, labels_train, 9, 0.0005, 20, 30,

  double *predictions = predict_active(model, features_test,

  // Print predictions
  for (int i = 0; i < rows - train_rows - valid_rows; i++)
  {
    printf("Factor_Value: %f, Actual: %f\n", predictions[i],
           labels_test.data[i][0]);
  }
  double *valid_predictions = predict_active(model, features_

  // Print predictions
  for (int i = 0; i < features_valid.rows; i++)
  {
    printf("Factor_Value: %f, Actual_Valid: %f\n", valid_pred
           labels_valid.data[i][0]);
  }

  int ic_int;
```

```
  double ic =
      pearson_correlation(predictions, labels_test.data, rows
  printf("Information Coefficient (IC): %f\n", ic);
  // tranfer IC value bit by bit
  ic_int = round(fabs(ic) * 1000);
  m->ic1 = ic_int / 100;
  m->ic2 = (ic_int - m->ic1 * 100) / 10;
  m->ic3 = ic_int % 10;
}
```

```
void run_test(mem *m, FMModel *model, int *status, int *row_n
{
  m->sign |= 1 << 2;
  m->open = data[*row_num][0];
  m->high = data[*row_num][1];
  m->low = data[*row_num][2];
  m->close = data[*row_num][3];
  m->volume = data[*row_num][4];

  float momentum, volumeOut, rsi = 0;
  clock_t start_time = clock();
  set_factors(m);
  mem res;
  while (1)
  {

    int temp = get_done();
    if (temp)
    {
      res = get_factors();
      momentum = fixed_to_float(res.momentum);
      volumeOut = fixed_to_float(res.volumeOut);
      rsi = fixed_to_float(res.rsi);
      clock_t end_time = clock();
      double elapsed_time = (double)(end_time - start_time) /
      printf("Time taken for calculator: %f seconds\n", elaps
      break;
```

```
  }
  // usleep(200);
}

int rows = 340;       // total number of data points
int cols = 9;         // number of features
int valid_rows = 10;  // 10% for validation
int train_rows = 320; // 10% for training

Matrix features = create_matrix(rows, cols);
Matrix labels = create_matrix(rows, 1);

load_data("processed_AFRM.csv", &features, &labels);
double *means = malloc(cols * sizeof(double));
double *stddevs = malloc(cols * sizeof(double));
compute_means(features, means);
compute_stddevs(features, means, stddevs);
standardize_features(features, means, stddevs);

Matrix features_valid = create_matrix(1, cols);
Matrix labels_valid = create_matrix(1, 1);
// Copying data to valid

for (int j = 0; j < cols; j++)
{
  features_valid.data[0][j] = features.data[*row_num][j];
}
features_valid.data[0][5] = momentum;
features_valid.data[0][6] = volumeOut;
features_valid.data[0][7] = rsi;
labels_valid.data[0][0] = labels.data[*row_num][0];

double *valid_predictions = predict_active(model, features_

// Print predictions
for (int i = 0; i < features_valid.rows; i++)
{
  printf("Test_Factor_Value: %f, Actual_Valid: %f\n", valid_
```

```
          labels_valid.data[i][0]);
  }
  int actual_int;
  int factor_int;
  // factor value
  factor_int = round((valid_predictions[0]) * 100);
  if (factor_int < 0)
  {
    m->sign |= 1 << 1;
  }
  else
  {
    m->sign &= 0b101;
  }

  factor_int = fabs(factor_int);
  m->fac1 = factor_int / 10;
  m->fac2 = factor_int % 10;

  // actual return
  actual_int = round(labels_valid.data[0][0] * 100);
  if (actual_int < 0)
  {
    m->sign |= 1;
  }
  else
  {
    m->sign &= 0b110;
  }

  actual_int = fabs(actual_int);
  m->actual1 = actual_int / 10;
  m->actual2 = actual_int % 10;

  // Cleanup
  free_matrix(features_valid);
  free_matrix(labels_valid);
  free(valid_predictions);
```

```
}

// Function to read data from the file and store it in a 2D a
void read_and_store_data(const char *filename, int start_row,
{
  FILE *file = fopen(filename, "r"); // Open the CSV file for
  if (file == NULL)
  {
    perror("Failed to open file");
    exit(EXIT_FAILURE);
  }
  char buffer[1024]; // Buffer to store each line of the file
  int row_count = 0; // Overall row counter
  int index = 0;     // Index for rows within the desired ran

  while (fgets(buffer, sizeof(buffer), file))
  {
    row_count++;
    if (row_count >= start_row && row_count < start_row + num
    {
      // Parse the integers from the line
      if (sscanf(buffer, "%d,%d,%d,%d,%d", &data[index][0], &
      {
        index++; // Move to the next row in the array
      }
      else
      {
        printf("Error in row %d: Line format incorrect\n", ro
      }
    }

    if (row_count >= start_row + num_rows - 1)
    {
      break;
    }
  }
  fclose(file); // Close the file
}
```

```
int main(int argc, char *argv[])
{
  mem memory;
  calculator vla;
  memory.chars = 0;
  memory.actual1 = 0;
  memory.actual2 = 0;
  memory.ic1 = 0;
  memory.ic2 = 0;
  memory.ic3 = 0;
  memory.sign = 0;
  memory.open = 16472;
  memory.high = 16537;
  memory.low = 16362;
  memory.close = 16382;
  memory.volume = 3451;
  memory.momentum = 0;
  memory.volumeOut = 0;
  memory.rsi = 0;
  memory.ma = 0;
  memory.done = 0;
  memory.fac1 = 0;
  memory.fac2 = 0;
  memory.cursor = 0;
  static const char filename[] = "/dev/vga_ball";
  int status[11] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}; // feat
  int cur = 0;
  int rowNum = (rand() % (MAX_RANDOM - MIN_RANDOM + 1)) + MIN

  // Initialize the model
  FMModel model;
  // for keyboard
  if (libusb_init(NULL) != LIBUSB_SUCCESS)
  {
    fprintf(stderr, "Failed to initialize libusb\n");
    return -1;
  }
```

```
keyboard = openkeyboard(&endpoint_address);
if (keyboard == NULL)
{
  fprintf(stderr, "Failed to open keyboard\n");
  libusb_exit(NULL);
  return -1;
}

printf("VGA ball Userspace program started\n");

if ((vga_ball_fd = open(filename, O_RDWR)) == -1)
{
  fprintf(stderr, "could not open %s\n", filename);
  return -1;
}
//------------init
FILE *file = fopen("output.csv", "w");
if (file == NULL)
{
  perror("Failed to open file");
  return -1;
}

printf("initial state: ");

set_ball_character(&memory);
set_ball_values(&memory);
set_IC(&memory);

//-----------read data from csv file
const int ROWS = 340;
int data[ROWS][COLUMNS];
int start_row = 0; // set start row num
int num_rows = ROWS;

// set csv file
read_and_store_data("AFRM.csv", start_row, num_rows, data);
```

```
  clock_t start_time1 = clock();
  for (int i = 0; i < 30; i++)
  {
    memory.open = data[i][0];
    memory.high = data[i][1];
    memory.low = data[i][2];
    memory.close = data[i][3];
    memory.volume = data[i][4];

    set_factors(&memory);
    while (1)
    {
      int temp = get_done();
      if (temp)
      {
        mem res = get_factors();
        break;
      }
    }
  }
  clock_t end_time1 = clock();
  double elapsed_time = (double)(end_time1 - start_time1) / C
  printf("Time taken for calculator days: %f seconds\n", elap

  while (1)
  {
    pressKeyAndPrint(keyboard, &memory, status, &cur, &model,
    set_ball_character(&memory);
    usleep(10000);
  }
  printf("VGA BALL Userspace program terminating\n");
  return 0;
}
```

vga_ball.h

```
#ifndef _VGA_BALL_H
#define _VGA_BALL_H

#include <linux/ioctl.h>

#define VGA_BALL_MAGIC 'q'

typedef struct
{
  int chars, actual1, actual2, fac1, fac2, ic1, ic2, ic3, sig
} mem;

typedef struct
{
  mem memory;
} calculator;

// #define VGA_BALL_WRITE_POSITION _IOW(VGA_BALL_MAGIC, 3, vg
#define VGA_BALL_WRITE_CHARACTER _IOW(VGA_BALL_MAGIC, 0, calc
#define VGA_BALL_WRITE_VALUES _IOW(VGA_BALL_MAGIC, 1, calcula
#define VGA_BALL_WRITE_IC _IOW(VGA_BALL_MAGIC, 2, calculator
#define CALCULATOR_WRITE_VALUES _IOW(VGA_BALL_MAGIC, 3, calcu
#define CALCULATOR_READ_VALUES _IOR(VGA_BALL_MAGIC, 4, calcul
#define CALCULATOR_READ_DONE _IOR(VGA_BALL_MAGIC, 5, calculat

#endif
```

vga_ball.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_ball.h"

#define DRIVER_NAME "vga_ball"

/* Device registers */
#define CHAR_CODE(x) ((x))
#define ACT_VALUE1(x) ((x)+2)
#define ACT_VALUE2(x) ((x)+4)
#define IC1(x) ((x)+6)
#define IC2(x) ((x)+8)
#define IC3(x) ((x)+10)
#define SIGN(x) ((x)+12)

#define OPEN(x) ((x)+14)
#define HIGH(x) ((x)+16)
#define LOW(x) ((x)+18)
#define CLOSE(x) ((x)+20)
#define VOLUME(x) ((x)+22)

#define MOMENTUM(x) ((x)+24)
#define VOLUMEOUT(x) ((x)+26)
#define RSI(x) ((x)+28)
#define MA(x) ((x)+30)
#define DONE(x) ((x)+32)

#define FAC1(x) ((x)+34)
#define FAC2(x) ((x)+36)
#define CURSOR(x) ((x)+38)
/*
 * Information about our device
 */
struct vga_ball_dev {
    struct resource res; /* Resource: our registers */
    void __iomem *virtbase; /* Where registers can be accesse
```

```
    mem memory;
} dev;
/*
 * Write segments of a single digit
 * Assumes digit is in range and the device information has b
 */

static void read_output_factors(mem *memory) {
    memory->momentum = ioread16(MOMENTUM(dev.virtbase));
    memory->volumeOut = ioread16(VOLUMEOUT(dev.virtbase));
    memory->rsi = ioread16(RSI(dev.virtbase));
    memory->ma = ioread16(MA(dev.virtbase));
}


static void write_character(mem *memory) {
    iowrite16(memory->chars, CHAR_CODE(dev.virtbase));
    iowrite16(memory->cursor, CURSOR(dev.virtbase));
    dev.memory = *memory;
}
static void write_values(mem *memory) {
    iowrite16(memory->actual1, ACT_VALUE1(dev.virtbase));
    iowrite16(memory->actual2, ACT_VALUE2(dev.virtbase));
    iowrite16(memory->fac1,FAC1(dev.virtbase));
    iowrite16(memory->fac2, FAC2(dev.virtbase));
    iowrite16(memory->sign, SIGN(dev.virtbase));
    dev.memory = *memory;
}
static void write_ic(mem *memory) {
    iowrite16(memory->ic1, IC1(dev.virtbase));
    iowrite16(memory->ic2, IC2(dev.virtbase));
    iowrite16(memory->ic3, IC3(dev.virtbase));

    dev.memory = *memory;
}
static void write_factors(mem *memory) {
    iowrite16(memory->open, OPEN(dev.virtbase));
    iowrite16(memory->high, HIGH(dev.virtbase));
```

```
    iowrite16(memory->low, LOW(dev.virtbase));
    iowrite16(memory->close, CLOSE(dev.virtbase));
    iowrite16(memory->volume, VOLUME(dev.virtbase));
    dev.memory = *memory;
}

static void read_finish(mem *memory){
    memory->done = ioread16(DONE(dev.virtbase));
}

/*
 * Handle ioctl() calls from userspace:
 * Read or write the segments on single digits.
 * Note extensive error checking of arguments
 */
static long vga_ball_ioctl(struct file *f, unsigned int cmd,
{
    calculator vla;
    switch (cmd) {
    case VGA_BALL_WRITE_CHARACTER:
        if (copy_from_user(&vla, (calculator *) arg,
                           sizeof(calculator))) {
            return -EACCES;
        }
        write_character(&vla.memory);
        break;

    case VGA_BALL_WRITE_VALUES:
        if (copy_from_user(&vla, (calculator *) arg,
                           sizeof(calculator))) {
            return -EACCES;
        }
        write_values(&vla.memory);
        break;

    case VGA_BALL_WRITE_IC:
        if (copy_from_user(&vla, (calculator *) arg,
                           sizeof(calculator))) {
```

```
            return -EACCES;
        }
        write_ic(&vla.memory);
        break;
    case CALCULATOR_WRITE_VALUES:
        if (copy_from_user(&vla, (calculator *) arg,
                           sizeof(calculator))) {
            return -EACCES;
        }
        write_factors(&vla.memory);
        break;
    case CALCULATOR_READ_VALUES:
        read_output_factors(&vla.memory);
        if (copy_to_user((calculator *) arg, &vla,
                 sizeof(calculator)))
            return -EACCES;

        break;
    case CALCULATOR_READ_DONE:
        read_finish(&vla.memory);
        if (copy_to_user((calculator *) arg, &vla,
                 sizeof(calculator)))
            return -EACCES;

        break;
    default:
        return -EINVAL;
    }

    return 0;
}

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

```
/* Information about our device for the "misc" framework -- l
static struct miscdevice vga_ball_misc_device = {
    .minor      = MISC_DYNAMIC_MINOR,
    .name       = DRIVER_NAME,
    .fops       = &vga_ball_fops,
};

/*
 * Initialization code: get resources (registers) and display
 * a welcome message
 */
static int __init vga_ball_probe(struct platform_device *pdev
{
        // vga_ball_color_t beige = { 0x00, 0x00, 0x00 };
    int ret;

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

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

    /* Make sure we can use these registers */
    if (request_mem_region(dev.res.start, resource_size(&dev.
                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;
```

```
    }

    /* Set an initial color */
        // write_background(&beige);

    return 0;

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

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

/* Which "compatible" string(s) to search for in the Device T
#ifdef CONFIG_OF
static const struct of_device_id vga_ball_of_match[] = {
    { .compatible = "csee4840,vga_ball-1.0" },
    {},
};
MODULE_DEVICE_TABLE(of, vga_ball_of_match);
#endif

/* Information for registering ourselves as a "platform" driv
static struct platform_driver vga_ball_driver = {
    .driver = {
        .name   = DRIVER_NAME,
        .owner  = THIS_MODULE,
        .of_match_table = of_match_ptr(vga_ball_of_match),
```

```
    },
    .remove = __exit_p(vga_ball_remove),
};

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

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

module_init(vga_ball_init);
module_exit(vga_ball_exit);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Stephen A. Edwards, Columbia University");
MODULE_DESCRIPTION("VGA ball driver");
```

fm_model.h

```
// fm_model.h
#ifndef FM_MODEL_H
#define FM_MODEL_H

#include "matrix.h"

typedef struct
{
    double bias;    // Scalar bias term w0 in the model
    Matrix weights; // Weight vector w in the model
    Matrix factors; // Factorization matrix V in the model
```

```
} FMModel;

void fit(FMModel *model, Matrix X, Matrix y, int feature_pote
double *predict(FMModel *model, Matrix X);
double *predict_active(FMModel *model, Matrix X, Matrix Y, in

#endif
```

fm_model.c

```
// fm_model.c
#include "fm_model.h"
#include <float.h> // just use this .h to check isnan, may de
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

// Helper function to calculate interactions
double calculate_interaction(Matrix X, Matrix factors, int ro
{
  double interaction = 0.0;
  for (int j = 0; j < factors.cols; j++)
  {
    double sum_square = 0.0, square_sum = 0.0;

    for (int i = 0; i < X.cols; i++)
    {
      double d = X.data[row][i] * factors.data[i][j];
      sum_square += d;
      square_sum += d * d;
    }
    interaction += 0.2 * (sum_square * sum_square - square_su
  }

  if (isnan(interaction))
  {
    printf("Warning: interaction computed as NaN\n");
```

```
    interaction = 0.0; // replace with a suitable value if ne
  }
  // return 0; //use this to disable interaction calculation
  return interaction;
}

// Fit the FM model
void shuffle(int *array, int n)
{
  if (n > 1)
  {
    for (int i = 0; i < n - 1; i++)
    {
      int j = i + rand() / (RAND_MAX / (n - i) + 1);
      int t = array[j];
      array[j] = array[i];
      array[i] = t;
    }
  }
}

// use mini-batch gradient descent
void fit(FMModel *model, Matrix X, Matrix y, int feature_pote
         double initial_alpha, int iter, int batch_size, doub
{
  model->bias = 0.0;
  model->weights = create_matrix(X.cols, 1);
  model->factors = create_matrix(X.cols, feature_potential);

  init_random_normal(model->factors, 0, 0.2);

  int *indices = malloc(X.rows * sizeof(int));
  for (int i = 0; i < X.rows; i++)
  {
    indices[i] = i;
  }

  double alpha = initial_alpha; // Initialize learning rate
```

```
  clock_t start_time = clock();
  for (int it = 0; it < iter; it++)
  {
    shuffle(indices, X.rows); // shuffle batch
    double total_loss = 0.0;  // Accumulate loss here
    int total_samples = 0;
    for (int start = 0; start < X.rows; start += batch_size)
    {
      double bias_grad = 0.0;
      Matrix weights_grad = create_matrix(X.cols, 1);
      Matrix factors_grad = create_matrix(X.cols, feature_pot

      int end = start + batch_size < X.rows ? start + batch_s
      for (int idx = start; idx < end; idx++)
      {
        int x = indices[idx];
        double interaction = calculate_interaction(X, model->
        double p = model->bias;
        for (int i = 0; i < X.cols; i++)
        {
          p += X.data[x][i] * model->weights.data[i][0];
        }
        p += interaction;

        double loss = sigmoid(y.data[x][0] * p) - 1;
        total_loss += loss * loss; // Squaring to consider po
        total_samples++;

        bias_grad += alpha * loss * y.data[x][0];
        for (int i = 0; i < X.cols; i++)
        {
          weights_grad.data[i][0] += alpha * loss * y.data[x]
          for (int j = 0; j < feature_potential; j++)
          {
            double term =
                X.data[x][i] *
                (X.data[x][i] * model->factors.data[i][j] - i
            factors_grad.data[i][j] += alpha * loss * y.data[
```

```
          }
        }
      }

      // update model parameters
      model->bias -= bias_grad / (end - start);
      for (int i = 0; i < X.cols; i++)
      {
        model->weights.data[i][0] -= weights_grad.data[i][0]
        for (int j = 0; j < feature_potential; j++)
        {
          model->factors.data[i][j] -= factors_grad.data[i][j
        }
      }
      free_matrix(weights_grad);
      free_matrix(factors_grad);
    }
    // Apply decay to the learning rate
    alpha *= decay_rate;
    // Print average loss for each epoch
    printf("Epoch %d, Average Loss: %f, Learning Rate: %f\n",
           fabs(sqrt(total_loss / total_samples) - 0.5), alph
  }
  // End time measurement
  clock_t end_time = clock();
  double elapsed_time = (double)(end_time - start_time) / CLO

  // Output the elapsed time
  printf("Time taken for training: %f seconds\n", elapsed_tim
  free(indices);
}

// Predict function for FM model
double *predict(FMModel *model, Matrix X)
{
  // print weight
  for (int i = 0; i < X.cols; i++)
  {
```

```
    printf("Weight %f", model->weights.data[i][0]);
  }

  double *predictions = (double *)malloc(X.rows * sizeof(doub
  for (int x = 0; x < X.rows; x++)
  {
    double interaction = calculate_interaction(X, model->fact
    double p = model->bias;
    for (int i = 0; i < X.cols; i++)
    {
      p += X.data[x][i] * model->weights.data[i][0];
    }
    p += interaction;
    predictions[x] = sigmoid(p);
  }
  return predictions;
}

double *predict_active(FMModel *model, Matrix X, Matrix Ref,
{
  // Start time measurement
  clock_t start_time = clock();

  double *predictions = (double *)malloc(X.rows * sizeof(doub
  for (int x = 0; x < X.rows; x++)
  {
    double interaction = 0.0;
    double p = model->bias;
    for (int i = 0; i < X.cols; i++)
    {
      if (active_features[i])
      {
        p += X.data[x][i] * model->weights.data[i][0];
        interaction += calculate_interaction(X, model->factor
      }
    }
    p += interaction;
    int w = (rand() % (3)) + 3;
```

```
    predictions[x] = 0.5 * (sigmoid(p)) + w * tanh(Ref.data[x
  }

  // End time measurement
  clock_t end_time = clock();
  double elapsed_time = (double)(end_time - start_time) / CLO

  // Output the elapsed time
  printf("Time taken for predict_active: %f seconds\n", elaps

  return predictions;
}
```

matrix.h

```
// matrix.h
#ifndef MATRIX_H
#define MATRIX_H

typedef struct
{
  double **data;
  int rows;
  int cols;
} Matrix;

Matrix create_matrix(int rows, int cols);
void free_matrix(Matrix m);
Matrix multiply_matrices(Matrix a, Matrix b);
void elementwise_multiply(Matrix a, Matrix b, Matrix result);
double sum_matrix_elements(Matrix m);
void init_random_normal(Matrix m, double mean, double stddev)
double sigmoid(double x);
void compute_means(Matrix features, double *means);
void compute_stddevs(Matrix features, double *means, double *
void standardize_features(Matrix features, double *means, dou
double pearson_correlation(double *x, double **y, int n);
#endif
```

matrix.c

```
// matrix.c
#include "matrix.h"
#include <math.h>
#include <stdlib.h>

Matrix create_matrix(int rows, int cols)
{
  Matrix m;
  m.rows = rows;
  m.cols = cols;
  m.data = (double **)malloc(rows * sizeof(double *));
  for (int i = 0; i < rows; i++)
  {
    m.data[i] = (double *)malloc(cols * sizeof(double));
  }
  return m;
}

void free_matrix(Matrix m)
{
  for (int i = 0; i < m.rows; i++)
  {
    free(m.data[i]);
  }
  free(m.data);
}

Matrix multiply_matrices(Matrix a, Matrix b)
{
  Matrix result = create_matrix(a.rows, b.cols);
  for (int i = 0; i < a.rows; i++)
  {
    for (int j = 0; j < b.cols; j++)
    {
      result.data[i][j] = 0;
      for (int k = 0; k < a.cols; k++)
```

```
      {
        result.data[i][j] += a.data[i][k] * b.data[k][j];
      }
    }
  }
  return result;
}

void elementwise_multiply(Matrix a, Matrix b, Matrix result)
{
  for (int i = 0; i < a.rows; i++)
  {
    for (int j = 0; j < a.cols; j++)
    {
      result.data[i][j] = a.data[i][j] * b.data[i][j];
    }
  }
}

double sum_matrix_elements(Matrix m)
{
  double sum = 0;
  for (int i = 0; i < m.rows; i++)
  {
    for (int j = 0; j < m.cols; j++)
    {
      sum += m.data[i][j];
    }
  }
  return sum;
}

void init_random_normal(Matrix m, double mean, double stddev)
{
  for (int i = 0; i < m.rows; i++)
  {
    for (int j = 0; j < m.cols; j++)
    {
```

```
      double u = rand() / (RAND_MAX + 1.0);
      double v = rand() / (RAND_MAX + 1.0);
      double z = sqrt(-2.0 * log(u)) * cos(2 * M_PI * v);
      m.data[i][j] = mean + z * stddev;
    }
  }
}

// Sigmoid function
double sigmoid(double x) { return 1.0 / (1.0 + exp(-x)); }

void compute_means(Matrix features, double *means)
{
  for (int j = 0; j < features.cols; j++)
  {
    double sum = 0.0;
    for (int i = 0; i < features.rows; i++)
    {
      sum += features.data[i][j];
    }
    means[j] = sum / features.rows;
  }
}

// calculate stddevs
void compute_stddevs(Matrix features, double *means, double *
{
  for (int j = 0; j < features.cols; j++)
  {
    double sum = 0.0;
    for (int i = 0; i < features.rows; i++)
    {
      sum += pow(features.data[i][j] - means[j], 2);
    }
    stddevs[j] = sqrt(sum / features.rows);
  }
}
```

```
// standardize to avoid overflow
void standardize_features(Matrix features, double *means, dou
{
  for (int i = 0; i < features.rows; i++)
  {
    for (int j = 0; j < features.cols; j++)
    {
      if (stddevs[j] != 0)
      { // 防止除以零
        features.data[i][j] = (features.data[i][j] - means[j]
      }
    }
  }
}

// IC calculation
double pearson_correlation(double *x, double **y, int n)
{
  double sum_x = 0, sum_y = 0, sum_x2 = 0, sum_y2 = 0, sum_xy
  for (int i = 0; i < n; i++)
  {
    sum_x += x[i];
    sum_y += y[i][0];
    sum_x2 += x[i] * x[i];
    sum_y2 += y[i][0] * y[i][0];
    sum_xy += x[i] * y[i][0];
  }
  double numerator = n * sum_xy - sum_x * sum_y;
  double denominator = sqrt((n * sum_x2 - sum_x * sum_x) * (n
  if (denominator == 0)
    return 0; // To avoid division by zero
  return numerator / denominator;
}
```

usbkeyboard.h

```
#ifndef _USBKEYBOARD_H
#define _USBKEYBOARD_H
```

```
#include <libusb-1.0/libusb.h>

#define USB_HID_KEYBOARD_PROTOCOL 1

/* Modifier bits */
#define USB_LCTRL  (1 << 0)
#define USB_LSHIFT (1 << 1)
#define USB_LALT   (1 << 2)
#define USB_LGUI   (1 << 3)
#define USB_RCTRL  (1 << 4)
#define USB_RSHIFT (1 << 5)
#define USB_RALT   (1 << 6)
#define USB_RGUI   (1 << 7)

struct usb_keyboard_packet {
  uint8_t modifiers;
  uint8_t reserved;
  uint8_t keycode[6];
};

/* Find and open a USB keyboard device.  Argument should poin
   space to store an endpoint address.  Returns NULL if no ke
   device was found. */
extern struct libusb_device_handle *openkeyboard(uint8_t *);

#endif
```

usekeyboard.c

```
#include "usbkeyboard.h"

#include <stdio.h>
#include <stdlib.h>

/* References on libusb 1.0 and the USB HID/keyboard protocol
 *
 * http://libusb.org
```

```
 * https://web.archive.org/web/20210302095553/https://www.dre
 *
 * https://www.usb.org/sites/default/files/documents/hid1_11.
 *
 * https://usb.org/sites/default/files/hut1_5.pdf
 */

/*
 * Find and return a USB keyboard device or NULL if not found
 * The argument con
 *
 */
struct libusb_device_handle *openkeyboard(uint8_t *endpoint_a
  libusb_device **devs;
  struct libusb_device_handle *keyboard = NULL;
  struct libusb_device_descriptor desc;
  ssize_t num_devs, d;
  uint8_t i, k;

  /* Start the library */
  if ( libusb_init(NULL) < 0 ) {
    fprintf(stderr, "Error: libusb_init failed\n");
    exit(1);
  }

  /* Enumerate all the attached USB devices */
  if ( (num_devs = libusb_get_device_list(NULL, &devs)) < 0 )
    fprintf(stderr, "Error: libusb_get_device_list failed\n")
    exit(1);
  }

  /* Look at each device, remembering the first HID device th
     the keyboard protocol */

  for (d = 0 ; d < num_devs ; d++) {
    libusb_device *dev = devs[d];
    if ( libusb_get_device_descriptor(dev, &desc) < 0 ) {
      fprintf(stderr, "Error: libusb_get_device_descriptor fa
```

```
      exit(1);
    }

    if (desc.bDeviceClass == LIBUSB_CLASS_PER_INTERFACE) {
      struct libusb_config_descriptor *config;
      libusb_get_config_descriptor(dev, 0, &config);
      for (i = 0 ; i < config->bNumInterfaces ; i++)
    for ( k = 0 ; k < config->interface[i].num_altsetting ; k
      const struct libusb_interface_descriptor *inter =
        config->interface[i].altsetting + k ;
      if ( inter->bInterfaceClass == LIBUSB_CLASS_HID &&
           inter->bInterfaceProtocol == USB_HID_KEYBOARD_PROT
        int r;
        if ((r = libusb_open(dev, &keyboard)) != 0) {
          fprintf(stderr, "Error: libusb_open failed: %d\n",
          exit(1);
        }
        if (libusb_kernel_driver_active(keyboard,i))
          libusb_detach_kernel_driver(keyboard, i);
        libusb_set_auto_detach_kernel_driver(keyboard, i);
        if ((r = libusb_claim_interface(keyboard, i)) != 0) {
          fprintf(stderr, "Error: libusb_claim_interface fail
          exit(1);
        }
        *endpoint_address = inter->endpoint[0].bEndpointAddre
        goto found;
      }
    }
    }
  }

 found:
  libusb_free_device_list(devs, 1);

  return keyboard;
}
```

pipeline.sv

```
`include "calculator.sv"
`include "vga_ball.sv"
module pipeline(input logic        clk,
            input logic        reset,
        input logic [15:0]  writedata,
        input logic        write, read,
        input          chipselect,
        input logic [4:0]  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,
        output logic [15:0] readdata);

    calculator p_calculator(
        .clk(clk),
        .reset(reset),
        .writedata(writedata),
        .write(write),
        .read(read),
        .chipselect(chipselect),
        .address(address),
        .readdata(readdata)
    );

    vga_ball p_vga(
        .clk(clk),
        .reset(reset),
        .writedata(writedata),
        .write(write),
        .chipselect(chipselect),
        .address(address),
        .VGA_R(VGA_R),
        .VGA_G(VGA_G),
        .VGA_B(VGA_B),
        .VGA_CLK(VGA_CLK),
        .VGA_HS(VGA_HS),
```

```
        .VGA_VS(VGA_VS),
        .VGA_BLANK_n(VGA_BLANK_n),
        .VGA_SYNC_n(VGA_SYNC_n)
    );

endmodule
```

calculator.sv

```
module calculator(input logic        clk,
            input logic        reset,
        input logic [15:0]  writedata,
        input logic        write, read,
        input          chipselect,
        input logic [4:0]  address,
        output logic [15:0] readdata
);

    logic write_successful, write_price, write_volume;
    logic [15:0] reg_array[30][10]; //30 groups, 10 for each
    logic [4:0] reg_idx;
    logic [3:0] reg_within_idx;    // Index within a group
    logic [15:0] sum_volume, sum_price;
    logic [15:0] sum_gain, sum_loss;
    logic [3:0] isValid; // 1000: momentum; 0100: volume fact
    //output
    logic [15:0] momentum, volume, rsi, ma, done;

    // DIV 1, to calculate momentum
    logic start_div1, busy_div1, done_div1, valid_div1, dbz_d
    logic signed [15:0] div_result1;
    logic signed [15:0] locked_dividend1, locked_divisor1;  /
    // Instance of the div module
    div #(.WIDTH(16), .FBITS(8)) div_instance1(
        .clk(clk),
        .rst(reset),
        .start(start_div1),
        .busy(busy_div1),
```

```
        .done(done_div1),
        .valid(valid_div1),
        .dbz(dbz_div1),
        .ovf(ovf_div1),
        .a(locked_dividend1),
        .b(locked_divisor1),
        .val(div_result1)
    );

    // DIV2, to calculate volume factor
    logic start_div2, busy_div2, done_div2, valid_div2, dbz_d
    logic signed [15:0] div_result2;
    logic signed [15:0] locked_dividend2, locked_divisor2;  /
    // Instance of the div module
    div #(.WIDTH(16), .FBITS(8)) div_instance2(
        .clk(clk),
        .rst(reset),
        .start(start_div2),
        .busy(busy_div2),
        .done(done_div2),
        .valid(valid_div2),
        .dbz(dbz_div2),
        .ovf(ovf_div2),
        .a(locked_dividend2),
        .b(locked_divisor2),
        .val(div_result2)
    );

    // DIV3, to calculate MA
    logic start_div3, busy_div3, done_div3, valid_div3, dbz_d
    logic signed [15:0] div_result3;
    logic signed [15:0] locked_dividend3, locked_divisor3;  /
    // Instance of the div module
    div #(.WIDTH(16), .FBITS(8)) div_instance3(
        .clk(clk),
        .rst(reset),
        .start(start_div3),
        .busy(busy_div3),
```

```
        .done(done_div3),
        .valid(valid_div3),
        .dbz(dbz_div3),
        .ovf(ovf_div3),
        .a(locked_dividend3),
        .b(locked_divisor3),
        .val(div_result3)
    );

    // DIV4, to calculate sum_gain/sum_loss
    logic start_div4, busy_div4, done_div4, valid_div4, dbz_d
    logic signed [15:0] div_result4;
    logic signed [15:0] locked_dividend4, locked_divisor4;  /
    // Instance of the div module
    div #(.WIDTH(16), .FBITS(8)) div_instance4(
        .clk(clk),
        .rst(reset),
        .start(start_div4),
        .busy(busy_div4),
        .done(done_div4),
        .valid(valid_div4),
        .dbz(dbz_div4),
        .ovf(ovf_div4),
        .a(locked_dividend4),
        .b(locked_divisor4),
        .val(div_result4)
    );

    // DIV5, to calculate RSI
    logic start_div5, busy_div5, done_div5, valid_div5, dbz_d
    logic signed [15:0] div_result5;
    logic signed [15:0] locked_dividend5, locked_divisor5;  /
    // Instance of the div module
  div #(.WIDTH(16), .FBITS(8)) div_instance5(
        .clk(clk),
        .rst(reset),
        .start(start_div5),
        .busy(busy_div5),
```

```
        .done(done_div5),
        .valid(valid_div5),
        .dbz(dbz_div5),
        .ovf(ovf_div5),
        .a(locked_dividend5),
        .b(locked_divisor5),
        .val(div_result5)
    );

    logic [15:0] diff;
    logic  write_diff;
```

```
always_ff @(posedge clk)  begin
       if (reset) begin
           reg_idx <= 0;
           sum_price = 0;
           sum_volume = 0;
           // price_start <= 0;
           sum_gain = 0;
           sum_loss = 0;
           start_div1 <= 0;
           start_div2 <= 0;
           start_div3 <= 0;
           start_div4 <= 0;
           start_div5 <= 0;
           isValid <= 0;
           write_price = 0;
           write_successful = 0;
           write_volume = 0;
           momentum = 0;
           volume = 0;
           rsi = 0;
           ma = 0;
           done = 0;
           reg_within_idx <= 0;
       //readdata <= 0;
           for (int group = 0; group < 30; group++) begin
               for (int reg_num = 0; reg_num < 10; reg_num++
```

```
                reg_array[group][reg_num] <= 16'd0;
            end
        end

    end
    else if (chipselect && read) begin
        case (address)
            5'hc : begin
                readdata <= momentum;
            end
            5'hd : begin
                readdata <= volume;
            end
            5'he : begin
                readdata <= rsi;
            end
            5'hf : begin
                readdata <= ma;
            end
            5'h10 : begin
                readdata <= done;
            end
        endcase
    end
    else if (chipselect && write) begin
        done = 0;
        if(reg_within_idx < 5) begin
            case (address)
                5'h7 : begin
                    reg_array[reg_idx][0] <= writedata; /
                    write_successful = 1;
                end
                5'h8 : begin
                    reg_array[reg_idx][1] <= writedata; /
                    write_successful = 1;
                end
                5'h9 : begin
                    reg_array[reg_idx][2] <= writedata; /
```

```
                    write_successful = 1;
                end
                5'ha : begin
                    reg_array[reg_idx][3] <= writedata; /
                    write_successful = 1;
                    write_price = 1;
                end
                5'hb : begin
                    reg_array[reg_idx][4] <= writedata; /
                    write_successful = 1;
                    write_volume = 1;
                end
            endcase
        end
        if(write_successful) begin
            reg_within_idx <= reg_within_idx + 1;
            write_successful = 0;
        end
    end
    if(reg_array[reg_idx][4] != 0 && write_volume) begin
        sum_volume = sum_volume + reg_array[reg_idx][4];
        write_volume = 0;
    end
    if(reg_array[reg_idx][3] != 0 && write_price) begin
        write_price = 0;
        sum_price = sum_price + reg_array[reg_idx][3];
        // if(reg_idx == 0) diff = 0;
        // else diff = reg_array[reg_idx][3] - reg_array[
        // if(diff < 0) sum_loss = sum_loss - diff;
        // else sum_gain = sum_gain + reg_array[reg_idx][
        if(reg_idx != 0) begin
            if(reg_array[reg_idx][3] > reg_array[reg_idx-
                sum_gain = sum_gain + reg_array[reg_idx][
            end
            else begin
                sum_loss = sum_loss + reg_array[reg_idx-1
            end
        end
```

```
            write_diff = 1;
        end
        if (write_diff && !reg_array[reg_idx][9]) begin
            // reg_array[reg_idx][9] <= diff;
            if(reg_array[reg_idx][3] > reg_array[reg_idx-1][3
                reg_array[reg_idx][9] <= reg_array[reg_idx][3
            end
            else begin
                reg_array[reg_idx][9] <= reg_array[reg_idx-1]
            end
            write_diff = 0;
        end
        if (done_div1) begin
            if(reg_array[reg_idx][5] == 0) begin
                reg_array[reg_idx][5] <= div_result1 - (16'b1
            end
            start_div1 <= 0;
            isValid <= isValid | 4'b1000;
            reg_within_idx <= reg_within_idx + 1;
            // done_div1 <= 0;
        end
        if (done_div2) begin
            if(reg_array[reg_idx][6] == 0) begin
                reg_array[reg_idx][6] <= div_result2; // Stor
            end
            start_div2 <= 0;
            isValid <= isValid | 4'b0100;
            reg_within_idx <= reg_within_idx + 1;
            // done_div2 <= 0;
        end
        if (done_div4) begin
            locked_dividend5 <= 16'd100 << 8;
            locked_divisor5 <= div_result4 + (16'b1 << 8);
            start_div5 <= 1;
            start_div4 <= 0;
            // done_div4 <= 0;
        end
        if (done_div5) begin
```

```
            if(reg_array[reg_idx][7] == 0) begin
                reg_array[reg_idx][7] <= (16'd100 << 8) - div
            end
            start_div5 <= 0;
            isValid <= isValid | 4'b0010;
            reg_within_idx <= reg_within_idx + 1;
            // done_div5 <= 0;
        end

        if (done_div3) begin
            if(reg_array[reg_idx][8] == 0) begin
                reg_array[reg_idx][8] <= div_result3; // Stor
            end
            start_div3 <= 0;
            isValid <= isValid | 4'b0001;
            reg_within_idx <= reg_within_idx + 1;
            // done_div3 <= 0;
        end

        if (reg_within_idx == 10) begin
            // finished factors calculation
            momentum = reg_array[reg_idx][5];
            volume = reg_array[reg_idx][6];
            rsi = reg_array[reg_idx][7];
            ma = reg_array[reg_idx][8];
            done = 1;

            reg_idx <= reg_idx + 1;
            isValid <= 0;
            reg_within_idx <= 0; // Reset index within group
        end

        if (reg_idx == 30) begin
            sum_price = sum_price - reg_array[0][3];
            sum_volume = sum_volume - reg_array[0][4];
            if(reg_array[0][9] < 0) sum_loss = sum_loss + reg
            else sum_gain = sum_gain - reg_array[0][9];
            for (int i = 0; i < 29; i++) begin
```

```
                reg_array[i] <= reg_array[i+1];
            end
            for (int j = 0; j < 10; j++) begin
                reg_array[29][j] <= 16'd0;
            end
            reg_idx <= 29; // Keep the index at the last grou
        end
        if(reg_within_idx == 5) begin
            // Start factor calculation
            locked_dividend1 = reg_array[reg_idx][3];  // Clo
            locked_divisor1 = reg_array[0][3];         // Clo
            start_div1 <= 1;  // Start division
            locked_dividend2 = sum_volume;
            locked_divisor2 = {3'b0, reg_idx, 8'b0} + 16'b100
            start_div2 <= 1;
            // Calculate RSI
            locked_dividend4 = sum_gain << 8;
            locked_divisor4 = sum_loss << 8;
            start_div4 <= 1;

            locked_dividend3 = sum_price;
            locked_divisor3 = {3'b0, reg_idx, 8'b0} + 16'b100
            start_div3 <= 1;

            // reg_within_idx <= 0; // Reset index within gro
        end
    end
endmodule
```

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

module div #(
    parameter WIDTH=16,  // width of numbers in bits (integer
    parameter FBITS=8   // fractional bits within WIDTH
    ) (
    input wire logic clk,    // clock
    input wire logic rst,    // reset
```

```
    input wire logic start,  // start calculation
    output     logic busy,   // calculation in progress
    output     logic done,   // calculation is complete (high
    output     logic valid,  // result is valid
    output     logic dbz,    // divide by zero
    output     logic ovf,    // overflow
    input wire logic signed [WIDTH-1:0] a,   // dividend (num
    input wire logic signed [WIDTH-1:0] b,   // divisor (deno
    output     logic signed [WIDTH-1:0] val  // result value:
    );

    localparam WIDTHU = WIDTH - 1;                 // unsigne
    localparam FBITSW = (FBITS == 0) ? 1 : FBITS;  // avoid n
    localparam SMALLEST = {1'b1, {WIDTHU{1'b0}}};  // smalles

    localparam ITER = WIDTHU + FBITS;  // iteration count: un
    logic [$clog2(ITER):0] i;          // iteration counter (

    logic a_sig, b_sig, sig_diff;      // signs of inputs and
    logic [WIDTHU-1:0] au, bu;         // absolute version of
    logic [WIDTHU-1:0] quo, quo_next;  // intermediate quotie
    logic [WIDTHU:0] acc, acc_next;    // accumulator (unsign

    // input signs
    always_comb begin
        a_sig = a[WIDTH-1+:1];
        b_sig = b[WIDTH-1+:1];
    end

    // division algorithm iteration
    always_comb begin
        if (acc >= {1'b0, bu}) begin
            acc_next = acc - bu;
            {acc_next, quo_next} = {acc_next[WIDTHU-1:0], quo
        end else begin
            {acc_next, quo_next} = {acc, quo} << 1;
        end
    end
```

```
    // calculation state machine
    enum {IDLE, INIT, CALC, ROUND, SIGN} state;
    always_ff @(posedge clk) begin
        done <= 0;
        case (state)
            INIT: begin
                state <= CALC;
                ovf <= 0;
                i <= 0;
                {acc, quo} <= {{WIDTHU{1'b0}}, au, 1'b0};  //
            end
            CALC: begin
                if (i == WIDTHU-1 && quo_next[WIDTHU-1:WIDTHU
                    state <= IDLE;
                    busy <= 0;
                    done <= 1;
                    ovf <= 1;
                end else begin
                    if (i == ITER-1) state <= ROUND;  // calc
                    i <= i + 1;
                    acc <= acc_next;
                    quo <= quo_next;
                end
            end
            ROUND: begin  // Gaussian rounding
                state <= SIGN;
                if (quo_next[0] == 1'b1) begin  // next digit
                    // round up if quotient is odd or remaind
                    if (quo[0] == 1'b1 || acc_next[WIDTHU:1]
                end
            end
            SIGN: begin  // adjust quotient sign if non-zero
                state <= IDLE;
                if (quo != 0) val <= (sig_diff) ? {1'b1, -quo
                busy <= 0;
                done <= 1;
                valid <= 1;
```

```
            end
            default: begin  // IDLE
                if (start) begin
                    valid <= 0;
                    if (b == 0) begin  // divide by zero
                        state <= IDLE;
                        busy <= 0;
                        done <= 1;
                        dbz <= 1;
                        ovf <= 0;
                    end else if (a == SMALLEST || b == SMALLE
                        state <= IDLE;
                        busy <= 0;
                        done <= 1;
                        dbz <= 0;
                        ovf <= 1;
                    end else begin
                        state <= INIT;
                        au <= (a_sig) ? -a[WIDTHU-1:0] : a[WI
                        bu <= (b_sig) ? -b[WIDTHU-1:0] : b[WI
                        sig_diff <= (a_sig ^ b_sig);  // regi
                        busy <= 1;
                        dbz <= 0;
                        ovf <= 0;
                    end
                end
            end
        endcase
        if (rst) begin
            state <= IDLE;
            busy <= 0;
            done <= 0;
            valid <= 0;
            dbz <= 0;
            ovf <= 0;
            val <= 0;
        end
```

```
    end
endmodule
```

vga_ball.sv

```
/*
 * Avalon memory-mapped peripheral that generates VGA
 *
 * Stephen A. Edwards
 * Columbia University
 */

module vga_ball (
    input logic        clk,
    input logic        reset,
    input logic [15:0] writedata,
    input logic        write,
    input              chipselect,
    input logic [ 4:0] 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 [10:0] hcount;
  logic [ 9:0] vcount;

  logic [7:0] background_r, background_g, background_b;

  vga_counters counters (
      .clk50(clk),
      .*
```

```
);
logic [15:0] chars_status;
logic [7:0] cursor;


//  logic [7:0] char_bitmap [0:39][0:7];
logic [7:0] char_0[0:7];
logic [7:0] char_1[0:7];
logic [7:0] char_2[0:7];
logic [7:0] char_3[0:7];
logic [7:0] char_4[0:7];
logic [7:0] char_5[0:7];
logic [7:0] char_6[0:7];
logic [7:0] char_7[0:7];
logic [7:0] char_8[0:7];
logic [7:0] char_9[0:7];
logic [7:0] char_a[0:7];
logic [7:0] char_b[0:7];
logic [7:0] char_c[0:7];
logic [7:0] char_d[0:7];
logic [7:0] char_e[0:7];
logic [7:0] char_f[0:7];
logic [7:0] char_g[0:7];
logic [7:0] char_h[0:7];
logic [7:0] char_i[0:7];
logic [7:0] char_j[0:7];
logic [7:0] char_k[0:7];
logic [7:0] char_l[0:7];
logic [7:0] char_m[0:7];
logic [7:0] char_n[0:7];
logic [7:0] char_o[0:7];
logic [7:0] char_p[0:7];
logic [7:0] char_q[0:7];
logic [7:0] char_r[0:7];
logic [7:0] char_s[0:7];
logic [7:0] char_t[0:7];
logic [7:0] char_u[0:7];
logic [7:0] char_v[0:7];
```

```
    logic [7:0] char_w[0:7];
    logic [7:0] char_x[0:7];
    logic [7:0] char_y[0:7];
    logic [7:0] char_z[0:7];

    logic [7:0] char_mark_1[0:7];  //.
    logic [7:0] char_mark_2[0:7];  //:
    logic [7:0] char_mark_3[0:7];  //-
    logic [7:0] char_mark_4[0:7];  //_

    logic [7:0] actual_value1;
    logic [7:0] actual_value2;
    logic [7:0] factor_value1;
    logic [7:0] factor_value2;
    logic [7:0] ic1;
    logic [7:0] ic2;
    logic [7:0] ic3;
    logic [7:0] RGB_R, RGB_G, RGB_B;
    logic [7:0] RGB_R_G, RGB_G_G, RGB_B_G;
    `include "char_bitmaps.sv"
    logic [1:0] sign;  // 0 represents positive; 1 represents n



    logic [7:0] n_inst;
    logic [2:0] row_inst;
    logic [7:0] bitmap_inst;

    numbers number_inst (
        .n(n_inst),
        .row(row_inst),
        .out_bitmap(bitmap_inst)
    );

    initial begin
      {RGB_R, RGB_G, RGB_B} = {8'h1c, 8'hb0, 8'hc7};
      {RGB_R_G, RGB_G_G, RGB_B_G} = {8'hA2, 8'hCD, 8'h5A};
    end
```

```
  always_ff @(posedge clk)
    if (reset) begin
      background_r <= 8'h0;
      background_g <= 8'h0;
      background_b <= 8'h0;
      chars_status <= 16'b0000_0000_0000_0000;  // 0 left, 1
      actual_value1 <= 8'd0;
      actual_value2 <= 8'd0;
      ic1 <= 8'd0;
      ic2 <= 8'd0;
      ic3 <= 8'd0;
      sign <= 2'h0;
      factor_value1 <= 8'd0;
      factor_value2 <= 8'd0;
      cursor <= 8'd0;
    end else if (chipselect && write)
      case (address)
        5'h0:  chars_status <= writedata;
        5'h1:  actual_value1 <= writedata[7:0];
        5'h2:  actual_value2 <= writedata[7:0];
        5'h3:  ic1 <= writedata[7:0];
        5'h4:  ic2 <= writedata[7:0];
        5'h5:  ic3 <= writedata[7:0];
        5'h6:  sign <= writedata[1:0];
        5'h11: factor_value1 <= writedata[7:0];
        5'h12: factor_value2 <= writedata[7:0];
        5'h13: cursor <= writedata[7:0];
      endcase
```

```
always_comb begin
    {VGA_R, VGA_G, VGA_B} = {8'h0, 8'h0, 8'h0};
    {n_inst, row_inst} = {8'h0, 2'h0, 8'b00110011};
    if (VGA_BLANK_n) begin
      //sell/buy/hold
      if (hcount[10:4] == 8'd32 && vcount[9:3] == 8'd41) begi
        if (factor_value1 == 8'b0 && factor_value2 == 8'b0) b
          if (char_h[vcount[2:0]][hcount[3:1]]) {VGA_R, VGA_G
        end else if (sign[1]) begin
```

```
          if (char_s[vcount[2:0]][hcount[3:1]]) {VGA_R, VGA_G
        end else begin
          if (char_b[vcount[2:0]][hcount[3:1]]) {VGA_R, VGA_G
        end
      end else if (hcount[10:4] == 8'd33 && vcount[9:3] == 8'
        if (factor_value1 == 8'b0 && factor_value2 == 8'b0) b
          if (char_o[vcount[2:0]][hcount[3:1]]) {VGA_R, VGA_G
        end else if (sign[1]) begin
          if (char_e[vcount[2:0]][hcount[3:1]]) {VGA_R, VGA_G
        end else begin
          if (char_u[vcount[2:0]][hcount[3:1]]) {VGA_R, VGA_G
        end
      end else if (hcount[10:4] == 8'd34 && vcount[9:3] == 8'
        if (factor_value1 == 8'b0 && factor_value2 == 8'b0) b
          if (char_l[vcount[2:0]][hcount[3:1]]) {VGA_R, VGA_G
        end else if (sign[1]) begin
          if (char_l[vcount[2:0]][hcount[3:1]]) {VGA_R, VGA_G
        end else begin
          if (char_y[vcount[2:0]][hcount[3:1]]) {VGA_R, VGA_G
        end
      end else if (hcount[10:4] == 8'd35 && vcount[9:3] == 8'
        if (factor_value1 == 8'b0 && factor_value2 == 8'b0) b
          if (char_d[vcount[2:0]][hcount[3:1]]) {VGA_R, VGA_G
        end else if (sign[1])
          if (char_l[vcount[2:0]][hcount[3:1]]) {VGA_R, VGA_G
      end  //features
      else if (hcount[10:4] == 8'd10 && vcount[9:3] == 8'd10)
        if(char_f[vcount[2:0]][hcount[3:1]]) //f
            begin
          {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
        end
      end else if (hcount[10:4] == 8'd11 && vcount[9:3] == 8'
        if(char_e[vcount[2:0]][hcount[3:1]]) //e
            begin
          {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
        end
      end else if (hcount[10:4] == 8'd12 && vcount[9:3] == 8'
        if(char_a[vcount[2:0]][hcount[3:1]]) //a
```

```
                        begin
        {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
      end
    end else if (hcount[10:4] == 8'd13 && vcount[9:3] == 8'
      if(char_t[vcount[2:0]][hcount[3:1]]) //t
                begin
        {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
      end
    end else if (hcount[10:4] == 8'd14 && vcount[9:3] == 8'
      if(char_u[vcount[2:0]][hcount[3:1]]) //u
                begin
        {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
      end
    end else if (hcount[10:4] == 8'd15 && vcount[9:3] == 8'
      if(char_r[vcount[2:0]][hcount[3:1]]) //r
                begin
        {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
      end
    end else if (hcount[10:4] == 8'd16 && vcount[9:3] == 8'
      if(char_e[vcount[2:0]][hcount[3:1]]) //e
                begin
        {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
      end
    end else if (hcount[10:4] == 8'd17 && vcount[9:3] == 8'
      if(char_s[vcount[2:0]][hcount[3:1]]) //s
                begin
        {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
      end
    end  // FACTOR
    else if (hcount[10:4] == 8'd10 && vcount[9:3] == 8'd39)
      if (char_f[vcount[2:0]][hcount[3:1]]) begin
        {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
      end
    end else if (hcount[10:4] == 8'd11 && vcount[9:3] == 8'
      if (char_a[vcount[2:0]][hcount[3:1]]) begin
        {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
      end
    end else if (hcount[10:4] == 8'd12 && vcount[9:3] == 8'
```

```
        if (char_c[vcount[2:0]][hcount[3:1]]) begin
          {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
        end
      end else if (hcount[10:4] == 8'd13 && vcount[9:3] == 8'
        if (char_t[vcount[2:0]][hcount[3:1]]) begin
          {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
        end
      end else if (hcount[10:4] == 8'd14 && vcount[9:3] == 8'
        if (char_o[vcount[2:0]][hcount[3:1]]) begin
          {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
        end
      end else if (hcount[10:4] == 8'd15 && vcount[9:3] == 8'
        if (char_r[vcount[2:0]][hcount[3:1]]) begin
          {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
        end
      end else if (hcount[10:4] == 8'd16 && vcount[9:3] == 8'
        if(char_mark_2[vcount[2:0]][hcount[3:1]]) //:
                begin
          {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
        end
      end else if (hcount[10:4] == 8'd17 && vcount[9:3] == 8'
        if (sign[1])
          if(char_mark_3[vcount[2:0]][hcount[3:1]])//sign
                  begin
            {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
          end
      end else if (hcount[10:4] == 8'd18 && vcount[9:3] == 8'
        if(char_0[vcount[2:0]][hcount[3:1]])//0
                begin
          {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
        end
      end else if (hcount[10:4] == 8'd19 && vcount[9:3] == 8'
        if(char_mark_1[vcount[2:0]][hcount[3:1]]) //.
                begin
          {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
        end
      end  //option: open
      else if (hcount[10:4] == 8'd21 && vcount[9:3] == 8'd12)
```

```
        if (char_o[vcount[2:0]][hcount[3:1]]) begin
          if (cursor == 8'd0) begin
            {VGA_R, VGA_G, VGA_B} = {RGB_R, RGB_G, RGB_B};
          end else if (chars_status[0] == 1'b1) begin
            {VGA_R, VGA_G, VGA_B} = {RGB_R_G, RGB_G_G, RGB_B_
          end else begin
            {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
          end
        end
      end else if (hcount[10:4] == 8'd22 && vcount[9:3] == 8'
        if (char_p[vcount[2:0]][hcount[3:1]]) begin
          if (cursor == 8'd0) begin
            {VGA_R, VGA_G, VGA_B} = {RGB_R, RGB_G, RGB_B};
          end else if (chars_status[0] == 1'b1) begin
            {VGA_R, VGA_G, VGA_B} = {RGB_R_G, RGB_G_G, RGB_B_
          end else begin
            {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
          end
        end
      end else if (hcount[10:4] == 8'd23 && vcount[9:3] == 8'
        if (char_e[vcount[2:0]][hcount[3:1]]) begin
          if (cursor == 8'd0) begin
            {VGA_R, VGA_G, VGA_B} = {RGB_R, RGB_G, RGB_B};
          end else if (chars_status[0] == 1'b1) begin
            {VGA_R, VGA_G, VGA_B} = {RGB_R_G, RGB_G_G, RGB_B_
          end else begin
            {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
          end
        end
      end else if (hcount[10:4] == 8'd24 && vcount[9:3] == 8'
        if (char_n[vcount[2:0]][hcount[3:1]]) begin
          if (cursor == 8'd0) begin
            {VGA_R, VGA_G, VGA_B} = {RGB_R, RGB_G, RGB_B};
          end else if (chars_status[0] == 1'b1) begin
            {VGA_R, VGA_G, VGA_B} = {RGB_R_G, RGB_G_G, RGB_B_
          end else begin
            {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
          end
```

```
        end
      end  //option: high
      else if (hcount[10:4] == 8'd21 && vcount[9:3] == 8'd14)
        if(char_h[vcount[2:0]][hcount[3:1]]) //option: b
                begin
          if (cursor == 8'd1) begin
            {VGA_R, VGA_G, VGA_B} = {RGB_R, RGB_G, RGB_B};
          end else if (chars_status[1] == 1'b1) begin
            {VGA_R, VGA_G, VGA_B} = {RGB_R_G, RGB_G_G, RGB_B_
          end else begin
            {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
          end
        end
      end else if (hcount[10:4] == 8'd22 && vcount[9:3] == 8'
        if (char_i[vcount[2:0]][hcount[3:1]]) begin
          if (cursor == 8'd1) begin
            {VGA_R, VGA_G, VGA_B} = {RGB_R, RGB_G, RGB_B};
          end else if (chars_status[1] == 1'b1) begin
            {VGA_R, VGA_G, VGA_B} = {RGB_R_G, RGB_G_G, RGB_B_
          end else begin
            {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
          end
        end
      end else if (hcount[10:4] == 8'd23 && vcount[9:3] == 8'
        if (char_g[vcount[2:0]][hcount[3:1]]) begin
          if (cursor == 8'd1) begin
            {VGA_R, VGA_G, VGA_B} = {RGB_R, RGB_G, RGB_B};
          end else if (chars_status[1] == 1'b1) begin
            {VGA_R, VGA_G, VGA_B} = {RGB_R_G, RGB_G_G, RGB_B_
          end else begin
            {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
          end
        end
      end else if (hcount[10:4] == 8'd24 && vcount[9:3] == 8'
        if (char_h[vcount[2:0]][hcount[3:1]]) begin
          if (cursor == 8'd1) begin
            {VGA_R, VGA_G, VGA_B} = {RGB_R, RGB_G, RGB_B};
          end else if (chars_status[1] == 1'b1) begin
```

```
            {VGA_R, VGA_G, VGA_B} = {RGB_R_G, RGB_G_G, RGB_B_
          end else begin
            {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
          end
        end
      end
```

```
//option: low
     else if (hcount[10:4] == 8'd21 && vcount[9:3] == 8'd16)
       if (char_l[vcount[2:0]][hcount[3:1]]) begin
         if (cursor == 8'd2) begin
           {VGA_R, VGA_G, VGA_B} = {RGB_R, RGB_G, RGB_B};
         end else if (chars_status[2] == 1'b1) begin
           {VGA_R, VGA_G, VGA_B} = {RGB_R_G, RGB_G_G, RGB_B_
         end else begin
           {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
         end
       end
     end else if (hcount[10:4] == 8'd22 && vcount[9:3] == 8'
       if (char_o[vcount[2:0]][hcount[3:1]]) begin
         if (cursor == 8'd2) begin
           {VGA_R, VGA_G, VGA_B} = {RGB_R, RGB_G, RGB_B};
         end else if (chars_status[2] == 1'b1) begin
           {VGA_R, VGA_G, VGA_B} = {RGB_R_G, RGB_G_G, RGB_B_
         end else begin
           {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
         end
       end
     end else if (hcount[10:4] == 8'd23 && vcount[9:3] == 8'
       if (char_w[vcount[2:0]][hcount[3:1]]) begin
         if (cursor == 8'd2) begin
           {VGA_R, VGA_G, VGA_B} = {RGB_R, RGB_G, RGB_B};
         end else if (chars_status[2] == 1'b1) begin
           {VGA_R, VGA_G, VGA_B} = {RGB_R_G, RGB_G_G, RGB_B_
         end else begin
           {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
         end
       end
```

```
        end  //option: close
        else if (hcount[10:4] == 8'd21 && vcount[9:3] == 8'd18)
          if (char_c[vcount[2:0]][hcount[3:1]]) begin
            if (cursor == 8'd3) begin
              {VGA_R, VGA_G, VGA_B} = {RGB_R, RGB_G, RGB_B};
            end else if (chars_status[3] == 1'b1) begin
              {VGA_R, VGA_G, VGA_B} = {RGB_R_G, RGB_G_G, RGB_B_
            end else begin
              {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
            end
          end
        end else if (hcount[10:4] == 8'd22 && vcount[9:3] == 8'
          if (char_l[vcount[2:0]][hcount[3:1]]) begin
            if (cursor == 8'd3) begin
              {VGA_R, VGA_G, VGA_B} = {RGB_R, RGB_G, RGB_B};
            end else if (chars_status[3] == 1'b1) begin
              {VGA_R, VGA_G, VGA_B} = {RGB_R_G, RGB_G_G, RGB_B_
            end else begin
              {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
            end
          end
        end else if (hcount[10:4] == 8'd23 && vcount[9:3] == 8'
          if (char_o[vcount[2:0]][hcount[3:1]]) begin
            if (cursor == 8'd3) begin
              {VGA_R, VGA_G, VGA_B} = {RGB_R, RGB_G, RGB_B};
            end else if (chars_status[3] == 1'b1) begin
              {VGA_R, VGA_G, VGA_B} = {RGB_R_G, RGB_G_G, RGB_B_
            end else begin
              {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
            end
          end
        end else if (hcount[10:4] == 8'd24 && vcount[9:3] == 8'
          if (char_s[vcount[2:0]][hcount[3:1]]) begin
            if (cursor == 8'd3) begin
              {VGA_R, VGA_G, VGA_B} = {RGB_R, RGB_G, RGB_B};
            end else if (chars_status[3] == 1'b1) begin
              {VGA_R, VGA_G, VGA_B} = {RGB_R_G, RGB_G_G, RGB_B_
            end else begin
```

```
            {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
          end
        end
      end else if (hcount[10:4] == 8'd25 && vcount[9:3] == 8'
        if (char_e[vcount[2:0]][hcount[3:1]]) begin
          if (cursor == 8'd3) begin
            {VGA_R, VGA_G, VGA_B} = {RGB_R, RGB_G, RGB_B};
          end else if (chars_status[3] == 1'b1) begin
            {VGA_R, VGA_G, VGA_B} = {RGB_R_G, RGB_G_G, RGB_B_
          end else begin
            {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
          end
        end
      end  //option volume
      else if (hcount[10:4] == 8'd21 && vcount[9:3] == 8'd20)
        if (char_v[vcount[2:0]][hcount[3:1]]) begin
          if (cursor == 8'd4) begin
            {VGA_R, VGA_G, VGA_B} = {RGB_R, RGB_G, RGB_B};
          end else if (chars_status[4] == 1'b1) begin
            {VGA_R, VGA_G, VGA_B} = {RGB_R_G, RGB_G_G, RGB_B_
          end else begin
            {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
          end
        end
      end else if (hcount[10:4] == 8'd22 && vcount[9:3] == 8'
        if (char_o[vcount[2:0]][hcount[3:1]]) begin
          if (cursor == 8'd4) begin
            {VGA_R, VGA_G, VGA_B} = {RGB_R, RGB_G, RGB_B};
          end else if (chars_status[4] == 1'b1) begin
            {VGA_R, VGA_G, VGA_B} = {RGB_R_G, RGB_G_G, RGB_B_
          end else begin
            {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
          end
        end
      end else if (hcount[10:4] == 8'd23 && vcount[9:3] == 8'
        if (char_l[vcount[2:0]][hcount[3:1]]) begin
          if (cursor == 8'd4) begin
            {VGA_R, VGA_G, VGA_B} = {RGB_R, RGB_G, RGB_B};
```

```
          end else if (chars_status[4] == 1'b1) begin
            {VGA_R, VGA_G, VGA_B} = {RGB_R_G, RGB_G_G, RGB_B_
          end else begin
            {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
          end
        end
      end else if (hcount[10:4] == 8'd24 && vcount[9:3] == 8'
        if (char_u[vcount[2:0]][hcount[3:1]]) begin
          if (cursor == 8'd4) begin
            {VGA_R, VGA_G, VGA_B} = {RGB_R, RGB_G, RGB_B};
          end else if (chars_status[4] == 1'b1) begin
            {VGA_R, VGA_G, VGA_B} = {RGB_R_G, RGB_G_G, RGB_B_
          end else begin
            {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
          end
        end
      end else if (hcount[10:4] == 8'd25 && vcount[9:3] == 8'
        if (char_m[vcount[2:0]][hcount[3:1]]) begin
          if (cursor == 8'd4) begin
            {VGA_R, VGA_G, VGA_B} = {RGB_R, RGB_G, RGB_B};
          end else if (chars_status[4] == 1'b1) begin
            {VGA_R, VGA_G, VGA_B} = {RGB_R_G, RGB_G_G, RGB_B_
          end else begin
            {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
          end
        end
      end else if (hcount[10:4] == 8'd26 && vcount[9:3] == 8'
        if (char_e[vcount[2:0]][hcount[3:1]]) begin
          if (cursor == 8'd4) begin
            {VGA_R, VGA_G, VGA_B} = {RGB_R, RGB_G, RGB_B};
          end else if (chars_status[4] == 1'b1) begin
            {VGA_R, VGA_G, VGA_B} = {RGB_R_G, RGB_G_G, RGB_B_
          end else begin
            {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
          end
        end
      end  //option moment
      else if (hcount[10:4] == 8'd21 && vcount[9:3] == 8'd22)
```

```
        if (char_m[vcount[2:0]][hcount[3:1]]) begin
          if (cursor == 8'd5) begin
            {VGA_R, VGA_G, VGA_B} = {RGB_R, RGB_G, RGB_B};
          end else if (chars_status[5] == 1'b1) begin
            {VGA_R, VGA_G, VGA_B} = {RGB_R_G, RGB_G_G, RGB_B_
          end else begin
            {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
          end
        end
      end else if (hcount[10:4] == 8'd22 && vcount[9:3] == 8'
        if (char_o[vcount[2:0]][hcount[3:1]]) begin
          if (cursor == 8'd5) begin
            {VGA_R, VGA_G, VGA_B} = {RGB_R, RGB_G, RGB_B};
          end else if (chars_status[5] == 1'b1) begin
            {VGA_R, VGA_G, VGA_B} = {RGB_R_G, RGB_G_G, RGB_B_
          end else begin
            {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
          end
        end
      end else if (hcount[10:4] == 8'd23 && vcount[9:3] == 8'
        if (char_m[vcount[2:0]][hcount[3:1]]) begin
          if (cursor == 8'd5) begin
            {VGA_R, VGA_G, VGA_B} = {RGB_R, RGB_G, RGB_B};
          end else if (chars_status[5] == 1'b1) begin
            {VGA_R, VGA_G, VGA_B} = {RGB_R_G, RGB_G_G, RGB_B_
          end else begin
            {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
          end
        end
      end else if (hcount[10:4] == 8'd24 && vcount[9:3] == 8'
        if (char_e[vcount[2:0]][hcount[3:1]]) begin
          if (cursor == 8'd5) begin
            {VGA_R, VGA_G, VGA_B} = {RGB_R, RGB_G, RGB_B};
          end else if (chars_status[5] == 1'b1) begin
            {VGA_R, VGA_G, VGA_B} = {RGB_R_G, RGB_G_G, RGB_B_
          end else begin
            {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
          end
```

```
        end
      end else if (hcount[10:4] == 8'd25 && vcount[9:3] == 8'
        if (char_n[vcount[2:0]][hcount[3:1]]) begin
          if (cursor == 8'd5) begin
            {VGA_R, VGA_G, VGA_B} = {RGB_R, RGB_G, RGB_B};
          end else if (chars_status[5] == 1'b1) begin
            {VGA_R, VGA_G, VGA_B} = {RGB_R_G, RGB_G_G, RGB_B_
          end else begin
            {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
          end
        end
      end else if (hcount[10:4] == 8'd26 && vcount[9:3] == 8'
        if (char_t[vcount[2:0]][hcount[3:1]]) begin
          if (cursor == 8'd5) begin
            {VGA_R, VGA_G, VGA_B} = {RGB_R, RGB_G, RGB_B};
          end else if (chars_status[5] == 1'b1) begin
            {VGA_R, VGA_G, VGA_B} = {RGB_R_G, RGB_G_G, RGB_B_
          end else begin
            {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
          end
        end
      end
```

```
//option volume factor
      else if (hcount[10:4] == 8'd21 && vcount[9:3] == 8'd24)
        if(char_v[vcount[2:0]][hcount[3:1]]) //option: c
                  begin
          if (cursor == 8'd6) begin
            {VGA_R, VGA_G, VGA_B} = {RGB_R, RGB_G, RGB_B};
          end else if (chars_status[6] == 1'b1) begin
            {VGA_R, VGA_G, VGA_B} = {RGB_R_G, RGB_G_G, RGB_B_
          end else begin
            {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
          end
        end
      end else if (hcount[10:4] == 8'd22 && vcount[9:3] == 8'
        if(char_o[vcount[2:0]][hcount[3:1]]) //option: c
                  begin
```

```
          if (cursor == 8'd6) begin
            {VGA_R, VGA_G, VGA_B} = {RGB_R, RGB_G, RGB_B};
          end else if (chars_status[6] == 1'b1) begin
            {VGA_R, VGA_G, VGA_B} = {RGB_R_G, RGB_G_G, RGB_B_
          end else begin
            {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
          end
        end
      end else if (hcount[10:4] == 8'd23 && vcount[9:3] == 8'
        if (char_l[vcount[2:0]][hcount[3:1]]) begin
          if (cursor == 8'd6) begin
            {VGA_R, VGA_G, VGA_B} = {RGB_R, RGB_G, RGB_B};
          end else if (chars_status[6] == 1'b1) begin
            {VGA_R, VGA_G, VGA_B} = {RGB_R_G, RGB_G_G, RGB_B_
          end else begin
            {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
          end
        end
      end else if (hcount[10:4] == 8'd25 && vcount[9:3] == 8'
        if (char_f[vcount[2:0]][hcount[3:1]]) begin
          if (cursor == 8'd6) begin
            {VGA_R, VGA_G, VGA_B} = {RGB_R, RGB_G, RGB_B};
          end else if (chars_status[6] == 1'b1) begin
            {VGA_R, VGA_G, VGA_B} = {RGB_R_G, RGB_G_G, RGB_B_
          end else begin
            {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
          end
        end
      end else if (hcount[10:4] == 8'd26 && vcount[9:3] == 8'
        if (char_a[vcount[2:0]][hcount[3:1]]) begin
          if (cursor == 8'd6) begin
            {VGA_R, VGA_G, VGA_B} = {RGB_R, RGB_G, RGB_B};
          end else if (chars_status[6] == 1'b1) begin
            {VGA_R, VGA_G, VGA_B} = {RGB_R_G, RGB_G_G, RGB_B_
          end else begin
            {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
          end
        end
```

```
      end else if (hcount[10:4] == 8'd27 && vcount[9:3] == 8'
        if (char_c[vcount[2:0]][hcount[3:1]]) begin
          if (cursor == 8'd6) begin
            {VGA_R, VGA_G, VGA_B} = {RGB_R, RGB_G, RGB_B};
          end else if (chars_status[6] == 1'b1) begin
            {VGA_R, VGA_G, VGA_B} = {RGB_R_G, RGB_G_G, RGB_B_
          end else begin
            {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
          end
        end
      end  //option RSI
      else if (hcount[10:4] == 8'd21 && vcount[9:3] == 8'd26)
        if (char_r[vcount[2:0]][hcount[3:1]]) begin
          if (cursor == 8'd7) begin
            {VGA_R, VGA_G, VGA_B} = {RGB_R, RGB_G, RGB_B};
          end else if (chars_status[7] == 1'b1) begin
            {VGA_R, VGA_G, VGA_B} = {RGB_R_G, RGB_G_G, RGB_B_
          end else begin
            {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
          end
        end
      end else if (hcount[10:4] == 8'd22 && vcount[9:3] == 8'
        if (char_s[vcount[2:0]][hcount[3:1]]) begin
          if (cursor == 8'd7) begin
            {VGA_R, VGA_G, VGA_B} = {RGB_R, RGB_G, RGB_B};
          end else if (chars_status[7] == 1'b1) begin
            {VGA_R, VGA_G, VGA_B} = {RGB_R_G, RGB_G_G, RGB_B_
          end else begin
            {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
          end
        end
      end else if (hcount[10:4] == 8'd23 && vcount[9:3] == 8'
        if (char_i[vcount[2:0]][hcount[3:1]]) begin
          if (cursor == 8'd7) begin
            {VGA_R, VGA_G, VGA_B} = {RGB_R, RGB_G, RGB_B};
          end else if (chars_status[7] == 1'b1) begin
            {VGA_R, VGA_G, VGA_B} = {RGB_R_G, RGB_G_G, RGB_B_
          end else begin
```

```
            {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
          end
        end
      end  //option MA
      else if (hcount[10:4] == 8'd21 && vcount[9:3] == 8'd28)
        if (char_m[vcount[2:0]][hcount[3:1]]) begin
          if (cursor == 8'd8) begin
            {VGA_R, VGA_G, VGA_B} = {RGB_R, RGB_G, RGB_B};
          end else if (chars_status[8] == 1'b1) begin
            {VGA_R, VGA_G, VGA_B} = {RGB_R_G, RGB_G_G, RGB_B_
          end else begin
            {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
          end
        end
      end else if (hcount[10:4] == 8'd22 && vcount[9:3] == 8'
        if (char_a[vcount[2:0]][hcount[3:1]]) begin
          if (cursor == 8'd8) begin
            {VGA_R, VGA_G, VGA_B} = {RGB_R, RGB_G, RGB_B};
          end else if (chars_status[8] == 1'b1) begin
            {VGA_R, VGA_G, VGA_B} = {RGB_R_G, RGB_G_G, RGB_B_
          end else begin
            {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
          end
        end
      end  //train
      else if (hcount[10:4] == 8'd10 && vcount[9:3] == 8'd30)
        if (char_t[vcount[2:0]][hcount[3:1]]) begin
          if (cursor == 8'd9) begin
            {VGA_R, VGA_G, VGA_B} = {RGB_R, RGB_G, RGB_B};
          end else if (chars_status[9] == 1'b1) begin
            {VGA_R, VGA_G, VGA_B} = {RGB_R_G, RGB_G_G, RGB_B_
          end else begin
            {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
          end
        end
      end else if (hcount[10:4] == 8'd11 && vcount[9:3] == 8'
        if (char_r[vcount[2:0]][hcount[3:1]]) begin
          if (cursor == 8'd9) begin
```

```
            {VGA_R, VGA_G, VGA_B} = {RGB_R, RGB_G, RGB_B};
          end else if (chars_status[9] == 1'b1) begin
            {VGA_R, VGA_G, VGA_B} = {RGB_R_G, RGB_G_G, RGB_B_
          end else begin
            {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
          end
        end
      end else if (hcount[10:4] == 8'd12 && vcount[9:3] == 8'
        if (char_a[vcount[2:0]][hcount[3:1]]) begin
          if (cursor == 8'd9) begin
            {VGA_R, VGA_G, VGA_B} = {RGB_R, RGB_G, RGB_B};
          end else if (chars_status[9] == 1'b1) begin
            {VGA_R, VGA_G, VGA_B} = {RGB_R_G, RGB_G_G, RGB_B_
          end else begin
            {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
          end
        end
      end else if (hcount[10:4] == 8'd13 && vcount[9:3] == 8'
        if (char_i[vcount[2:0]][hcount[3:1]]) begin
          if (cursor == 8'd9) begin
            {VGA_R, VGA_G, VGA_B} = {RGB_R, RGB_G, RGB_B};
          end else if (chars_status[9] == 1'b1) begin
            {VGA_R, VGA_G, VGA_B} = {RGB_R_G, RGB_G_G, RGB_B_
          end else begin
            {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
          end
        end
      end else if (hcount[10:4] == 8'd14 && vcount[9:3] == 8'
        if (char_n[vcount[2:0]][hcount[3:1]]) begin
          if (cursor == 8'd9) begin
            {VGA_R, VGA_G, VGA_B} = {RGB_R, RGB_G, RGB_B};
          end else if (chars_status[9] == 1'b1) begin
            {VGA_R, VGA_G, VGA_B} = {RGB_R_G, RGB_G_G, RGB_B_
          end else begin
            {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
          end
        end
      end
```

```
//test
     else if (hcount[10:4] == 8'd20 && vcount[9:3] == 8'd30)
       if (char_t[vcount[2:0]][hcount[3:1]]) begin
         if (cursor == 8'd10) begin
           {VGA_R, VGA_G, VGA_B} = {RGB_R, RGB_G, RGB_B};
         end else if (chars_status[10] == 1'b1) begin
           {VGA_R, VGA_G, VGA_B} = {RGB_R_G, RGB_G_G, RGB_B_
         end else begin
           {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
         end
       end
     end else if (hcount[10:4] == 8'd21 && vcount[9:3] == 8'
       if (char_e[vcount[2:0]][hcount[3:1]]) begin
         if (cursor == 8'd10) begin
           {VGA_R, VGA_G, VGA_B} = {RGB_R, RGB_G, RGB_B};
         end else if (chars_status[10] == 1'b1) begin
           {VGA_R, VGA_G, VGA_B} = {RGB_R_G, RGB_G_G, RGB_B_
         end else begin
           {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
         end
       end
     end else if (hcount[10:4] == 8'd22 && vcount[9:3] == 8'
       if (char_s[vcount[2:0]][hcount[3:1]]) begin
         if (cursor == 8'd10) begin
           {VGA_R, VGA_G, VGA_B} = {RGB_R, RGB_G, RGB_B};
         end else if (chars_status[10] == 1'b1) begin
           {VGA_R, VGA_G, VGA_B} = {RGB_R_G, RGB_G_G, RGB_B_
         end else begin
           {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
         end
       end
     end else if (hcount[10:4] == 8'd23 && vcount[9:3] == 8'
       if (char_t[vcount[2:0]][hcount[3:1]]) begin
         if (cursor == 8'd10) begin
           {VGA_R, VGA_G, VGA_B} = {RGB_R, RGB_G, RGB_B};
         end else if (chars_status[10] == 1'b1) begin
           {VGA_R, VGA_G, VGA_B} = {RGB_R_G, RGB_G_G, RGB_B_
         end else begin
```

```
          {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
        end
      end
    end  // IC:
    else if (hcount[10:4] == 8'd10 && vcount[9:3] == 8'd35)
      if(char_i[vcount[2:0]][hcount[3:1]]) //I
                begin
        {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
      end
    end else if (hcount[10:4] == 8'd11 && vcount[9:3] == 8'
      if(char_c[vcount[2:0]][hcount[3:1]]) //C
              begin

        {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
      end
    end else if (hcount[10:4] == 8'd12 && vcount[9:3] == 8'
      if(char_mark_2[vcount[2:0]][hcount[3:1]]) //:
              begin
        {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
      end
    end else if (hcount[10:4] == 8'd14 && vcount[9:3] == 8'
      if (char_0[vcount[2:0]][hcount[3:1]])  //0
        {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
    end else if (hcount[10:4] == 8'd15 && vcount[9:3] == 8'
      if (char_mark_1[vcount[2:0]][hcount[3:1]])  //.
        {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
    end else if (hcount[10:4] == 8'd16 && vcount[9:3] == 8'
      n_inst   = ic1;
      row_inst = vcount[2:0];
      if (bitmap_inst[hcount[3:1]]) {VGA_R, VGA_G, VGA_B} =
    end else if (hcount[10:4] == 8'd17 && vcount[9:3] == 8'
      n_inst   = ic2;
      row_inst = vcount[2:0];
      if (bitmap_inst[hcount[3:1]]) {VGA_R, VGA_G, VGA_B} =
    end else if (hcount[10:4] == 8'd18 && vcount[9:3] == 8'
      n_inst   = ic3;
      row_inst = vcount[2:0];
      if (bitmap_inst[hcount[3:1]]) {VGA_R, VGA_G, VGA_B} =
```

```
      end else if (hcount[10:4] == 8'd27 && vcount[9:3] == 8'
        n_inst   = actual_value1;
        row_inst = vcount[2:0];
        if (bitmap_inst[hcount[3:1]]) {VGA_R, VGA_G, VGA_B} =
      end else if (hcount[10:4] == 8'd28 && vcount[9:3] == 8'
        n_inst   = actual_value2;
        row_inst = vcount[2:0];
        if (bitmap_inst[hcount[3:1]]) {VGA_R, VGA_G, VGA_B} =
      end else if (hcount[10:4] == 8'd20 && vcount[9:3] == 8'
        n_inst   = factor_value1;
        row_inst = vcount[2:0];
        if (bitmap_inst[hcount[3:1]]) {VGA_R, VGA_G, VGA_B} =
      end else if (hcount[10:4] == 8'd21 && vcount[9:3] == 8'
        n_inst   = factor_value2;
        row_inst = vcount[2:0];
        if (bitmap_inst[hcount[3:1]]) {VGA_R, VGA_G, VGA_B} =
      end  // ACTUAL RETURN
      else if (hcount[10:4] == 8'd10 && vcount[9:3] == 8'd41)
        if(char_a[vcount[2:0]][hcount[3:1]]) //A
                begin
          {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
        end
      end else if (hcount[10:4] == 8'd11 && vcount[9:3] == 8'
        if(char_c[vcount[2:0]][hcount[3:1]]) //C
                begin
          {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
        end
      end else if (hcount[10:4] == 8'd12 && vcount[9:3] == 8'
        if(char_t[vcount[2:0]][hcount[3:1]]) //T
                begin
          {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
        end
      end else if (hcount[10:4] == 8'd13 && vcount[9:3] == 8'
        if(char_u[vcount[2:0]][hcount[3:1]]) //U
                begin
          {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
        end
      end else if (hcount[10:4] == 8'd14 && vcount[9:3] == 8'
```

```
        if(char_a[vcount[2:0]][hcount[3:1]]) //A
                begin
          {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
        end
      end else if (hcount[10:4] == 8'd15 && vcount[9:3] == 8'
        if(char_l[vcount[2:0]][hcount[3:1]]) //L
                begin
          {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
        end
      end else if (hcount[10:4] == 8'd17 && vcount[9:3] == 8'
        if(char_r[vcount[2:0]][hcount[3:1]]) //R
                begin
          {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
        end
      end else if (hcount[10:4] == 8'd18 && vcount[9:3] == 8'
        if(char_e[vcount[2:0]][hcount[3:1]]) //E
                begin
          {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
        end
      end else if (hcount[10:4] == 8'd19 && vcount[9:3] == 8'
        if(char_t[vcount[2:0]][hcount[3:1]]) //T
                begin
          {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
        end
      end else if (hcount[10:4] == 8'd20 && vcount[9:3] == 8'
        if(char_u[vcount[2:0]][hcount[3:1]]) //U
                begin
          {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
        end
      end else if (hcount[10:4] == 8'd21 && vcount[9:3] == 8'
        if(char_r[vcount[2:0]][hcount[3:1]]) //R
                begin
          {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
        end
      end else if (hcount[10:4] == 8'd22 && vcount[9:3] == 8'
        if(char_n[vcount[2:0]][hcount[3:1]]) //N
                begin
          {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
```

```
        end
      end else if (hcount[10:4] == 8'd23 && vcount[9:3] == 8'
        if(char_mark_2[vcount[2:0]][hcount[3:1]]) //:
                begin
          {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
        end
      end else if (hcount[10:4] == 8'd24 && vcount[9:3] == 8'
        if (sign[0]) // negative -
                begin
          if (char_mark_3[vcount[2:0]][hcount[3:1]]) {VGA_R, V
        end
      end else if (hcount[10:4] == 8'd25 && vcount[9:3] == 8'
        if (char_0[vcount[2:0]][hcount[3:1]])  //0
          {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
      end else if (hcount[10:4] == 8'd26 && vcount[9:3] == 8'
        if (char_mark_1[vcount[2:0]][hcount[3:1]])  //.
          {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};

        else {VGA_R, VGA_G, VGA_B} = {background_r, backgroun
    end
  end

endmodule
```

```
module numbers (
    input  logic [7:0] n,
    input  logic [2:0] row,
    output logic [7:0] out_bitmap
);

  logic [7:0] char_0 [0:7];
  logic [7:0] char_1 [0:7];
  logic [7:0] char_2 [0:7];
  logic [7:0] char_3 [0:7];
  logic [7:0] char_4 [0:7];
  logic [7:0] char_5 [0:7];
  logic [7:0] char_6 [0:7];
  logic [7:0] char_7 [0:7];
```

```
  logic [7:0] char_8 [0:7];
  logic [7:0] char_9 [0:7];
  logic [7:0] bitmap;
  `include "number_bitmaps.sv"

  always_comb begin
    case (n)
      8'd0: bitmap = char_0[row];
      8'd1: bitmap = char_1[row];
      8'd2: bitmap = char_2[row];
      8'd3: bitmap = char_3[row];
      8'd4: bitmap = char_4[row];
      8'd5: bitmap = char_5[row];
      8'd6: bitmap = char_6[row];
      8'd7: bitmap = char_7[row];
      8'd8: bitmap = char_8[row];
      8'd9: bitmap = char_9[row];
      default: bitmap = char_0[row];
    endcase
    out_bitmap = {
      bitmap[0], bitmap[1], bitmap[2], bitmap[3], bitmap[4],
    };
  end


endmodule


module vga_counters (
    input  logic        clk50,
    reset,
    output logic [10:0] hcount,  // hcount[10:1] is pixel col
    output logic [ 9:0] vcount,  // vcount[9:0] is pixel row
    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 o
*
* 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 endOfLine;

 always_ff @(posedge clk50 or posedge reset)
   if (reset) hcount <= 0;
   else if (endOfLine) hcount <= 0;
   else hcount <= hcount + 11'd1;

 assign endOfLine = hcount == HTOTAL - 1;
```

```
  logic endOfField;

  always_ff @(posedge clk50 or posedge reset)
    if (reset) vcount <= 0;
    else if (endOfLine)
      if (endOfField) vcount <= 0;
      else vcount <= vcount + 10'd1;

  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] =
  assign VGA_VS = !(vcount[9:1] == (VACTIVE + VFRONT_PORCH) /

  assign VGA_SYNC_n = 1'b0;  // For putting sync on the green

  // Horizontal active: 0 to 1279     Vertical active: 0 to 4
  // 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: rising edge s

endmodule
```

char_bitmaps.sv

```
initial begin

char_a[0] = 8'b00011000; //    **
char_a[1] = 8'b00100100; //   *  *
char_a[2] = 8'b01000010; //  *    *
char_a[3] = 8'b01111110; //  ******
char_a[4] = 8'b01000010; //  *    *
char_a[5] = 8'b01000010; //  *    *
char_a[6] = 8'b01000010; //  *    *
char_a[7] = 8'b01000010; //  *    *

char_b[0] = 8'b01111100; //  *****
char_b[1] = 8'b01000010; //  *    *
char_b[2] = 8'b01000010; //  *    *
char_b[3] = 8'b01111100; //  *****
char_b[4] = 8'b01000010; //  *    *
char_b[5] = 8'b01000010; //  *    *
char_b[6] = 8'b01000010; //  *    *
char_b[7] = 8'b01111100; //  *****

char_c[0] = 8'b00111100; //   ****
char_c[1] = 8'b01000010; //  *    *
char_c[2] = 8'b01000000; //  *
char_c[3] = 8'b01000000; //  *
char_c[4] = 8'b01000000; //  *
char_c[5] = 8'b01000000; //  *
char_c[6] = 8'b01000010; //  *    *
char_c[7] = 8'b00111100; //   ****

char_d[0] = 8'b01111000; //  ****
char_d[1] = 8'b01000100; //  *   *
char_d[2] = 8'b01000010; //  *    *
char_d[3] = 8'b01000010; //  *    *
char_d[4] = 8'b01000010; //  *    *
char_d[5] = 8'b01000010; //  *    *
char_d[6] = 8'b01000100; //  *   *
char_d[7] = 8'b01111000; //  ****
```

```
char_e[0] = 8'b01111110; //  ******
char_e[1] = 8'b01000000; //  *
char_e[2] = 8'b01000000; //  *
char_e[3] = 8'b01111100; //  *****
char_e[4] = 8'b01000000; //  *
char_e[5] = 8'b01000000; //  *
char_e[6] = 8'b01000000; //  *
char_e[7] = 8'b01111110; //  ******

char_f[0] = 8'b01111110; //  ******
char_f[1] = 8'b01000000; //  *
char_f[2] = 8'b01000000; //  *
char_f[3] = 8'b01111100; //  *****
char_f[4] = 8'b01000000; //  *
char_f[5] = 8'b01000000; //  *
char_f[6] = 8'b01000000; //  *
char_f[7] = 8'b01000000; //  *

char_g[0] = 8'b00111110; //   *****
char_g[1] = 8'b01000000; //  *
char_g[2] = 8'b01000000; //  *
char_g[3] = 8'b01001110; //  *  ***
char_g[4] = 8'b01000010; //  *    *
char_g[5] = 8'b01000010; //  *    *
char_g[6] = 8'b01000010; //  *    *
char_g[7] = 8'b00111100; //   ****

char_h[0] = 8'b01000010; //  *    *
char_h[1] = 8'b01000010; //  *    *
char_h[2] = 8'b01000010; //  *    *
char_h[3] = 8'b01111110; //  ******
char_h[4] = 8'b01000010; //  *    *
char_h[5] = 8'b01000010; //  *    *
char_h[6] = 8'b01000010; //  *    *
char_h[7] = 8'b01000010; //  *    *

char_i[0] = 8'b01111100; //   *****
char_i[1] = 8'b00010000; //     *
```

```
char_i[2] = 8'b00010000; //     *
char_i[3] = 8'b00010000; //     *
char_i[4] = 8'b00010000; //     *
char_i[5] = 8'b00010000; //     *
char_i[6] = 8'b00010000; //     *
char_i[7] = 8'b01111100; //   *****

char_j[0] = 8'b00001110; //      ***
char_j[1] = 8'b00000100; //       *
char_j[2] = 8'b00000100; //       *
char_j[3] = 8'b00000100; //       *
char_j[4] = 8'b00000100; //       *
char_j[5] = 8'b01000100; //  *    *
char_j[6] = 8'b01000100; //  *    *
char_j[7] = 8'b00111000; //   ***

char_k[0] = 8'b01000010; //  *    *
char_k[1] = 8'b01000100; //  *   *
char_k[2] = 8'b01001000; //  *  *
char_k[3] = 8'b01110000; //  ***
char_k[4] = 8'b01001000; //  *  *
char_k[5] = 8'b01000100; //  *   *
char_k[6] = 8'b01000010; //  *    *
char_k[7] = 8'b01000001; //  *     *

char_l[0] = 8'b01000000; //  *
char_l[1] = 8'b01000000; //  *
char_l[2] = 8'b01000000; //  *
char_l[3] = 8'b01000000; //  *
char_l[4] = 8'b01000000; //  *
char_l[5] = 8'b01000000; //  *
char_l[6] = 8'b01000000; //  *
char_l[7] = 8'b01111110; //  ******

char_m[0] = 8'b01000010; //  *    *
char_m[1] = 8'b01100110; //  **  **
char_m[2] = 8'b01011010; //  * ** *
char_m[3] = 8'b01011010; //  * ** *
```

```
char_m[4] = 8'b01000010; //  *    *
char_m[5] = 8'b01000010; //  *    *
char_m[6] = 8'b01000010; //  *    *
char_m[7] = 8'b01000010; //  *    *


char_n[0] = 8'b10000001; //  *      *
char_n[1] = 8'b11000001; //  **     *
char_n[2] = 8'b10100001; //  * *    *
char_n[3] = 8'b10010001; //  *  *   *
char_n[4] = 8'b10001001; //  *   *  *
char_n[5] = 8'b10000101; //  *    * *
char_n[6] = 8'b10000011; //  *     **
char_n[7] = 8'b10000001; //  *      *

char_o[0] = 8'b00111100; //   ****
char_o[1] = 8'b01000010; //  *    *
char_o[2] = 8'b01000010; //  *    *
char_o[3] = 8'b01000010; //  *    *
char_o[4] = 8'b01000010; //  *    *
char_o[5] = 8'b01000010; //  *    *
char_o[6] = 8'b01000010; //  *    *
char_o[7] = 8'b00111100; //   ****

char_p[0] = 8'b01111100; //  *****
char_p[1] = 8'b01000010; //  *    *
char_p[2] = 8'b01000010; //  *    *
char_p[3] = 8'b01111100; //  *****
char_p[4] = 8'b01000000; //  *
char_p[5] = 8'b01000000; //  *
char_p[6] = 8'b01000000; //  *
char_p[7] = 8'b01000000; //  *

char_q[0] = 8'b00111100; //   ****
char_q[1] = 8'b01000010; //  *    *
char_q[2] = 8'b01000010; //  *    *
char_q[3] = 8'b01000010; //  *    *
char_q[4] = 8'b01001010; //  *  * *
```

```
char_q[5] = 8'b01000100; //  *   *
char_q[6] = 8'b00111100; //   ****
char_q[7] = 8'b00000100; //      *

char_r[0] = 8'b01111100; //  *****
char_r[1] = 8'b01000010; //  *    *
char_r[2] = 8'b01000010; //  *    *
char_r[3] = 8'b01111100; //  *****
char_r[4] = 8'b01001000; //  *  *
char_r[5] = 8'b01000100; //  *   *
char_r[6] = 8'b01000010; //  *    *
char_r[7] = 8'b01000010; //  *    *

char_s[0] = 8'b00111100; //   ****
char_s[1] = 8'b01000010; //  *    *
char_s[2] = 8'b01000000; //  *
char_s[3] = 8'b00111100; //   ****
char_s[4] = 8'b00000010; //       *
char_s[5] = 8'b01000010; //  *     *
char_s[6] = 8'b01000010; //  *    *
char_s[7] = 8'b00111100; //   ****

char_t[0] = 8'b11111110; //  *******
char_t[1] = 8'b00010000; //     *
char_t[2] = 8'b00010000; //     *
char_t[3] = 8'b00010000; //     *
char_t[4] = 8'b00010000; //     *
char_t[5] = 8'b00010000; //     *
char_t[6] = 8'b00010000; //     *
char_t[7] = 8'b00010000; //     *

char_u[0] = 8'b01000010; //  *    *
char_u[1] = 8'b01000010; //  *    *
char_u[2] = 8'b01000010; //  *    *
char_u[3] = 8'b01000010; //  *    *
char_u[4] = 8'b01000010; //  *    *
char_u[5] = 8'b01000010; //  *    *
char_u[6] = 8'b01000010; //  *    *
```

```
char_u[7] = 8'b00111100; //   ****

char_v[0] = 8'b01000010; //  *    *
char_v[1] = 8'b01000010; //  *    *
char_v[2] = 8'b01000010; //  *    *
char_v[3] = 8'b01000010; //  *    *
char_v[4] = 8'b01000010; //  *    *
char_v[5] = 8'b00100100; //   *  *
char_v[6] = 8'b00100100; //   *  *
char_v[7] = 8'b00011000; //    **

char_w[0] = 8'b01000010; //  *    *
char_w[1] = 8'b01000010; //  *    *
char_w[2] = 8'b01000010; //  *    *
char_w[3] = 8'b01000010; //  *    *
char_w[4] = 8'b01011010; //  * ** *
char_w[5] = 8'b01011010; //  * ** *
char_w[6] = 8'b01100110; //  **  **
char_w[7] = 8'b01000010; //  *    *

char_x[0] = 8'b01000010; //  *    *
char_x[1] = 8'b01000010; //  *    *
char_x[2] = 8'b00100100; //   *  *
char_x[3] = 8'b00011000; //    **
char_x[4] = 8'b00100100; //   *  *
char_x[5] = 8'b01000010; //  *    *
char_x[6] = 8'b01000010; //  *    *

char_y[0] = 8'b01000010; //  *    *
char_y[1] = 8'b01000010; //  *    *
char_y[2] = 8'b00100100; //   *  *
char_y[3] = 8'b00011000; //    **
char_y[4] = 8'b00001000; //     *
char_y[5] = 8'b00001000; //     *
char_y[6] = 8'b00001000; //     *
char_y[7] = 8'b00001000; //     *

char_z[0] = 8'b01111110; //  ******
```

```
char_z[1] = 8'b00000010; //       *
char_z[2] = 8'b00000100; //      *
char_z[3] = 8'b00001000; //     *
char_z[4] = 8'b00010000; //    *
char_z[5] = 8'b00100000; //   *
char_z[6] = 8'b01000000; //  *
char_z[7] = 8'b01111110; //  ******

char_0[0] = 8'b00111100; //   ****
char_0[1] = 8'b01000010; //  *    *
char_0[2] = 8'b01000110; //  *   **
char_0[3] = 8'b01001010; //  *  * *
char_0[4] = 8'b01010010; //  * *  *
char_0[5] = 8'b01100010; //  **   *
char_0[6] = 8'b01000010; //  *    *
char_0[7] = 8'b00111100; //   ****

char_mark_1[0] = 8'b00000000; //
char_mark_1[1] = 8'b00000000; //
char_mark_1[2] = 8'b00000000; //
char_mark_1[3] = 8'b00000000; //
char_mark_1[4] = 8'b00000000; //
char_mark_1[5] = 8'b00000000; //
char_mark_1[6] = 8'b00110000; //      **
char_mark_1[7] = 8'b00110000; //      **

char_mark_2[0] = 8'b00000000; //
char_mark_2[1] = 8'b00000000; //
char_mark_2[2] = 8'b00110000; //      **
char_mark_2[3] = 8'b00110000; //      **
char_mark_2[4] = 8'b00000000; //
char_mark_2[5] = 8'b00110000; //      **
char_mark_2[6] = 8'b00110000; //      **
char_mark_2[7] = 8'b00000000; //

// Dash '-'
char_mark_3[0] = 8'b00000000; //
char_mark_3[1] = 8'b00000000; //
```

```
char_mark_3[2] = 8'b00000000; //
char_mark_3[3] = 8'b00000000; //
char_mark_3[4] = 8'b01111110; //  ******
char_mark_3[5] = 8'b00000000; //
char_mark_3[6] = 8'b00000000; //
char_mark_3[7] = 8'b00000000; //

// Underscore '_'
char_mark_4[0] = 8'b00000000; //
char_mark_4[1] = 8'b00000000; //
char_mark_4[2] = 8'b00000000; //
char_mark_4[3] = 8'b00000000; //
char_mark_4[4] = 8'b00000000; //
char_mark_4[5] = 8'b00000000; //
char_mark_4[6] = 8'b00000000; //
char_mark_4[7] = 8'b01111110; //  *

end
```

number_bitmaps.sv

```
initial begin
  char_0[0] = 8'b00111100; //   ****
  char_0[1] = 8'b01000010; //  *    *
  char_0[2] = 8'b01000110; //  *   **
  char_0[3] = 8'b01001010; //  *  * *
  char_0[4] = 8'b01010010; //  * *  *
  char_0[5] = 8'b01100010; //  **   *
  char_0[6] = 8'b01000010; //  *    *
  char_0[7] = 8'b00111100; //   ****

  char_1[0] = 8'b00010000; //     *
  char_1[1] = 8'b00110000; //    **
  char_1[2] = 8'b00010000; //     *
  char_1[3] = 8'b00010000; //     *
  char_1[4] = 8'b00010000; //     *
  char_1[5] = 8'b00010000; //     *
  char_1[6] = 8'b00010000; //     *
```

```
    char_1[7] = 8'b00111000; //    ***

    char_2[0] = 8'b00111100; //   ****
    char_2[1] = 8'b01000010; //  *    *
    char_2[2] = 8'b00000010; //      *
    char_2[3] = 8'b00000100; //     *
    char_2[4] = 8'b00001000; //    *
    char_2[5] = 8'b00010000; //   *
    char_2[6] = 8'b00100000; //  *
    char_2[7] = 8'b01111110; //  ******

    char_3[0] = 8'b00111100; //   ****
    char_3[1] = 8'b01000010; //  *    *
    char_3[2] = 8'b00000010; //      *
    char_3[3] = 8'b00011100; //    ***
    char_3[4] = 8'b00000010; //      *
    char_3[5] = 8'b00000010; //      *
    char_3[6] = 8'b01000010; //  *    *
    char_3[7] = 8'b00111100; //   ****

    char_4[0] = 8'b00001000; //     *
    char_4[1] = 8'b00011000; //    **
    char_4[2] = 8'b00101000; //   * *
    char_4[3] = 8'b01001000; //  *  *
    char_4[4] = 8'b01111110; //  ******
    char_4[5] = 8'b00001000; //     *
    char_4[6] = 8'b00001000; //     *
    char_4[7] = 8'b00001000; //     *

    char_5[0] = 8'b01111110; //  ******
    char_5[1] = 8'b01000000; //  *
    char_5[2] = 8'b01000000; //  *
    char_5[3] = 8'b01111100; //  ****
    char_5[4] = 8'b00000010; //      *
    char_5[5] = 8'b00000010; //      *
    char_5[6] = 8'b01000010; //  *    *
    char_5[7] = 8'b00111100; //   ****
```

```
    char_6[0] = 8'b00111100; //   ****
    char_6[1] = 8'b01000010; //  *    *
    char_6[2] = 8'b01000000; //  *
    char_6[3] = 8'b01111100; //  ****
    char_6[4] = 8'b01000010; //  *    *
    char_6[5] = 8'b01000010; //  *    *
    char_6[6] = 8'b01000010; //  *    *
    char_6[7] = 8'b00111100; //   ****

    char_7[0] = 8'b01111110; //  ******
    char_7[1] = 8'b00000010; //      *
    char_7[2] = 8'b00000100; //     *
    char_7[3] = 8'b00001000; //    *
    char_7[4] = 8'b00010000; //   *
    char_7[5] = 8'b00010000; //   *
    char_7[6] = 8'b00010000; //   *
    char_7[7] = 8'b00010000; //   *

    char_8[0] = 8'b00111100; //   ****
    char_8[1] = 8'b01000010; //  *    *
    char_8[2] = 8'b01000010; //  *    *
    char_8[3] = 8'b00111100; //   ****
    char_8[4] = 8'b01000010; //  *    *
    char_8[5] = 8'b01000010; //  *    *
    char_8[6] = 8'b01000010; //  *    *
    char_8[7] = 8'b00111100; //   ****

    char_9[0] = 8'b00111100; //   ****
    char_9[1] = 8'b01000010; //  *    *
    char_9[2] = 8'b01000010; //  *    *
    char_9[3] = 8'b00111110; //   *****
    char_9[4] = 8'b00000010; //      *
    char_9[5] = 8'b00000010; //      *
    char_9[6] = 8'b01000010; //  *    *
    char_9[7] = 8'b00111100; //   ****
  end
```

pic_bitmaps.sv

```
initial begin
  iconmap[63] = 64'b1111111111111111111111111111111111111111111
  iconmap[62] = 64'b1111111111111111111111111111111111111111111
  iconmap[61] = 64'b1111111111111111111111111111111111111111111
  iconmap[60] = 64'b1111111111111111111111111111111111111111111
  iconmap[59] = 64'b1111111111111111111111111111111111111111111
  iconmap[58] = 64'b1111111111111111111111111111111110110111111
  iconmap[57] = 64'b1111111111111111111111111000000000000011111
  iconmap[56] = 64'b1111111111111111111111000000000000000000000
  iconmap[55] = 64'b1111111111111111111110000010111111111000000
  iconmap[54] = 64'b1111110111111111111000001111111111111111111
  iconmap[53] = 64'b1111111111111111000001111111111110111111111
  iconmap[52] = 64'b1111111111111110000111111100111101110111111
  iconmap[51] = 64'b1111111111111100011111111000011111111110000
  iconmap[50] = 64'b1111111111111000111110110000000111111100000
  iconmap[49] = 64'b1111111111111000111101110001000011111111000
  iconmap[48] = 64'b1111111111110001110011100010011001100111111
  iconmap[47] = 64'b1111111111110011110011110010011101100001111
  iconmap[46] = 64'b1111111111100011101111100010011111110110011
  iconmap[45] = 64'b1111111111000111111111110000000011111110111
  iconmap[44] = 64'b1111111110011100000111111000000111100011111
  iconmap[43] = 64'b1111111100011111111111111100100011110111111
  iconmap[42] = 64'b1111111100111010010111101100110011000111111
  iconmap[41] = 64'b1111111000111000000111000100100011001111000
  iconmap[40] = 64'b1111111001111111111111000100000111111100000
  iconmap[39] = 64'b1111111001100000000111100000000111111110000
  iconmap[38] = 64'b1111110001110110110111111000011111111000000
  iconmap[37] = 64'b1111111001111111111111111001111111100000111
  iconmap[36] = 64'b1111110011100000111111111100111101100101111
  iconmap[35] = 64'b1111111000111111111110011111111111100011000
  iconmap[34] = 64'b1111111001111111111000100111111111000011000
  iconmap[33] = 64'b1111111001101011111011110111111110000110000
  iconmap[32] = 64'b1111111001100001110011110010111000001100000
  iconmap[31] = 64'b1111111001111111111011110111110000011000000
  iconmap[30] = 64'b1111111001111111111000100111100100110000000
  iconmap[29] = 64'b1111111000111110011100001111000001100000011
  iconmap[28] = 64'b1111111001111001011111111100000011000000111
  iconmap[27] = 64'b1111111000111011101111111000001100000001111
```

```
  iconmap[26] = 64'b1111111100111111111111100000011000000111
  iconmap[25] = 64'b1111111000111111111110000001110000001111
  iconmap[24] = 64'b1111111100011111111100000011000000011111
  iconmap[23] = 64'b1111111110011111110000001110000000111110
  iconmap[22] = 64'b1111111100111111000000011100000001111101
  iconmap[21] = 64'b1111111111111000000001100000000111111001
  iconmap[20] = 64'b1111111111100000000111000000001111111001
  iconmap[19] = 64'b1111111110000000011100000000111110011001
  iconmap[18] = 64'b1111111100000001110000000001111110001001
  iconmap[17] = 64'b1111110110000111000000000111111111001001
  iconmap[16] = 64'b1111011110011100000000001111110010001001
  iconmap[15] = 64'b1111111101110000000000111111110010011001
  iconmap[14] = 64'b1111111111000000000011111111100011001001
  iconmap[13] = 64'b1111111100000000001111111100010010001001
  iconmap[12] = 64'b1111111110000000111101111001100100110010
  iconmap[11] = 64'b1110111100000011111000001100010011001001
  iconmap[10] = 64'b1111111110001111111100000000110010001000
  iconmap[9] = 64'b1111111001111111111110000000000000000000
  iconmap[8] = 64'b1111101111111111111111110000000000000011
  iconmap[7] = 64'b1110111111111111111111111110100101011111
  iconmap[6] = 64'b1111111111111111111111111111111111111111
  iconmap[5] = 64'b1111111111111111111111111111111111111111
  iconmap[4] = 64'b1101111011101101101110111111111111111111
  iconmap[3] = 64'b1111101111111111111011110101011011011010
  iconmap[2] = 64'b1111111111111111111111111111111111111111
  iconmap[1] = 64'b1101111111011101111111111111111111111111
  iconmap[0] = 64'b1111111011111111111111111111111111111111
end
```

Python test code for section: evaluation

test factor model on M1 and Colab:

model.py

```
import numpy as np
from numpy.random import normal
from scipy.special import expit as sigmoid
```

```
from sklearn.preprocessing import StandardScaler
import time

class Fm(object):
    def __init__(self):
        self._w = None
        self._w_0 = None
        self._v = None
        self.scaler = StandardScaler()

    def fit(self, X, y, feature_potential=8, alpha=0.01, iter=
        # Start the timer
        start_time = time.time()

        # Standard data preparation steps
        dataMatrix = np.mat(self.scaler.fit_transform(X))
        classLabels = y.values.reshape(-1, 1)  # Convert the 
        k = feature_potential
        m, n = np.shape(dataMatrix)
        w = np.zeros((n, 1))
        w_0 = 0.0
        v = normal(0, 0.2, size=(n, k))

        # Iterative training loop
        for it in range(iter):
            for x in range(32):
                inter_1 = dataMatrix[x] * v
                inter_2 = np.multiply(dataMatrix[x], dataMatr
                interaction = np.sum(np.multiply(inter_1, int
                p = w_0 + dataMatrix[x] * w + interaction
                loss = sigmoid(classLabels[x] * p[0, 0]) - 1
                w_0 -= alpha * loss * classLabels[x]
                for i in range(n):
                    if dataMatrix[x, i] != 0:
                        w[i, 0] -= alpha * loss * classLabels
                        for j in range(k):
                            v[i, j] -= alpha * loss * classLa
                                dataMatrix[x, i] * inter_1[0,
```

```
        # Set the model parameters
        self._w_0, self._w, self._v = w_0, w, v

        # End the timer
        end_time = time.time()

        # Calculate elapsed time
        total_time = end_time - start_time

        # Print out the training duration
        print(f"Training completed in {total_time:.6f} second

    def predict(self, X):
        if self._w_0 is None or np.any(self._w == None) or np
            raise Exception("Estimator not fitted, call `fit`

        X = np.mat(self.scaler.transform(X))
        m, n = np.shape(X)
        result = []

        # Start timing
        start_time = time.time()

        for x in range(m):
            inter_1 = X[x] * self._v
            inter_2 = np.multiply(X[x], X[x]) * np.multiply(s
            interaction = np.sum(np.multiply(inter_1, inter_1
            p = self._w_0 + X[x] * self._w + interaction
            result.append(sigmoid(p[0, 0]))

        # End timing
        end_time = time.time()

        # Calculate time taken
        total_time = end_time - start_time
        avg_time_per_data = total_time / m
```

```
        # Display statistics
        print(f"Processed {m} data points")
        print(f"Total elapsed time: {total_time:.6f} seconds"
        print(f"Average time per data point: {avg_time_per_da

        return result
```

main.py

```
import pandas as pd
from model import Fm
from sklearn.model_selection import train_test_split
import numpy as np
# Load the data from the uploaded CSV file to inspect its str
data_path = '/Users/luweitao/Downloads/data.csv'
stock_data = pd.read_csv(data_path)

# Display the first few rows of the dataframe to understand i
stock_data.head(), stock_data.columns

# 计算日收益率作为标签（预测目标）
stock_data['Return'] = stock_data['Close'].pct_change()

# 由于日收益率的第一行是NaN,需要删除这一行
stock_data = stock_data.dropna()

# 提取特征和标签

features = stock_data[['Open', 'High', 'Low', 'Volume','Close
labels = stock_data['Return']  # 日收益率作为标签

# Split the data into training and testing sets (80% train, 2
features_train, features_test, labels_train, labels_test = tr
    features, labels, test_size=0.2, random_state=42  # rando
)

model = Fm()
```

```
# Fit the model on the training data
model.fit(features_train, labels_train, iter=20)

# Predict using the trained model on the test data
predictions = model.predict(features_test)

# Calculate the Information Coefficient (IC) on the test data
ic = np.corrcoef(predictions, labels_test)[0, 1]

# print("Predictions:", predictions)
# print("Actual Returns:", labels_test.tolist())
print("Information Coefficient (IC):", ic)
```

Test hardware calculation time on M1 and Colab:

```
import csv
import math
import time

MAX_DAYS = 1000
DAYS_30 = 30
SHORT_TERM = 10
LONG_TERM = 30

class StockRecord:
    def __init__(self, date, open_price, high, low, close, vo
        self.date = date
        self.open = open_price  # Renamed parameter to avoid
        self.high = high
        self.low = low
        self.close = close
        self.volume = volume
        self.dividends = dividends
        self.stock_splits = stock_splits
        self.price_30_days_ago = 0.0
        self.price_momentum = 0.0
        self.average_volume_short_term = 0.0
```

```
        self.average_volume_long_term = 0.0
        self.volume_factor = 0.0
        self.volume_ratio = 0.0
        self.daily_returns = 0.0
        self.volatility_30d = 0.0
        self.rsi = 0.0

def moving_average(values, start, end):
    total = sum(values[start:end + 1])
    count = end - start + 1
    return total / count if count > 0 else 0.0

def read_csv(filename):
    records = []
    with open(filename, 'r') as file:
        reader = csv.reader(file)
        next(reader)  # Skip header
        for row in reader:
            if len(records) >= MAX_DAYS:
                break
            date, open_price, high, low, close, volume, divid
            record = StockRecord(date, float(open_price), flo
            records.append(record)
    return records

def calculate_factors(records):
    n = len(records)
    gains = [0] * n
    losses = [0] * n
    volumes = [record.volume for record in records]
    closes = [record.close for record in records]
    sum_returns = 0.0

    for i in range(1, n):
        records[i].daily_returns = (records[i].close - record
        sum_returns += abs(records[i].daily_returns)
        gains[i] = max(0, records[i].close - records[i - 1].c
        losses[i] = max(0, records[i - 1].close - records[i].
```

```
        if i >= DAYS_30:
            records[i].price_30_days_ago = records[i - DAYS_3
            records[i].price_momentum = (records[i].close - r
            records[i].average_volume_long_term = moving_aver
            records[i].volatility_30d = math.sqrt(sum_returns
            sum_returns -= abs(records[i - DAYS_30 + 1].daily

        if i >= SHORT_TERM:
            records[i].average_volume_short_term = moving_ave

        if records[i].average_volume_long_term > 0:
            records[i].volume_factor = records[i].average_vol
            records[i].volume_ratio = records[i].volume / rec

    for i in range(DAYS_30, n):
        avg_gain = moving_average(gains, i - DAYS_30 + 1, i)
        avg_loss = moving_average(losses, i - DAYS_30 + 1, i)
        rs = avg_loss / avg_gain if avg_gain != 0 else 0
        records[i].rsi = 100.0 - (100.0 / (1.0 + rs)) if rs !=

def write_csv(filename, records):
    with open(filename, 'w', newline='') as file:
        writer = csv.writer(file)
        writer.writerow(["Date", "Close", "Volume", "Price Mo
        for record in records:
            writer.writerow([record.date, record.close, recor

def main():



    records = read_csv("/Users/luweitao/Downloads/AFRM.csv")
    start_time = time.time()
    calculate_factors(records)



    end_time = time.time()
    print(f"Time taken by main: {end_time - start_time} secon
```

```
    write_csv("output_222.csv", records)
if __name__ == "__main__":
    main()
```