ECE 385

Fall 2020

Final Project

Travelling by Chinese Speed!

Lian Xinyu & Liu Tianyu
TA: Li Shuren

Introduction

Imagine that a foreign friend has just arrived in China and wants to visit famous cities in China. Indeed the first thing to consider is the choice of transportation. Meanwhile, the train/high-speed railway is an ID card of Chinese transportation to the world. To help our foreign friends plan their route properly, also inspired by the MP9 of ECE220, "Walk Me There" [6], we generate a set of programs that can be used to optimize the route for them. This project utilizes A star algorithm using NIOS-II IDE and runs this algorithm on the CPU of the DE2-115 board to find the optimized path. Moreover, the display of route, the occurrence of description for different cities, and the sound play function are implemented using the on-chip memory and logic elements on the board. The development environment for this project is Quartus and Eclipse IDE.

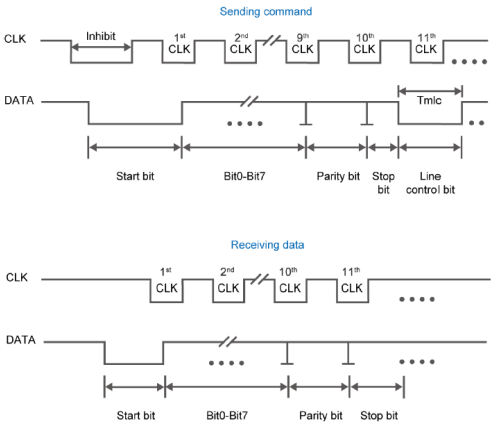
General flow of circuit for different modes.

1. High-level description

   We create a new file, final_high_level.sv, as our top-level element (See Module Description Part for the list of ports). In addition, they are grouped into six clusters: IO and standard clock, VGA Interface, CYC67200 Interface, SDRAM Interface for Nios II software, sound, and PS2.

2. Process description

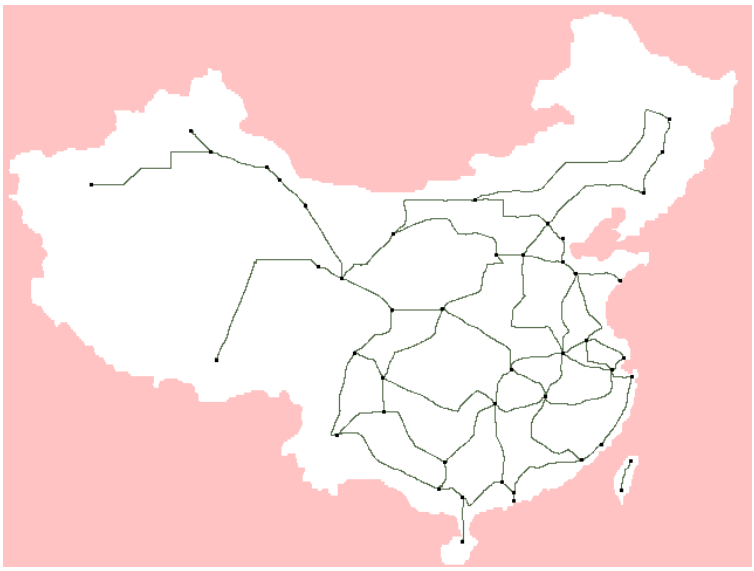
   Since the former four clusters of signals have been discussed in lab8, we focus our explanation on the last two clusters. The sound cluster is used to play the audio. The Top module's instantiation can receive information from the high-level input, such as clock signals, initialization signals, and switch signals. When the switch is closed (SW[0]=1), the top module will output the sound signal, and then the signal will be passed to the corresponding pin through the output of the top-level module, so after connecting the audio to the sound output, we can hear the beautiful music. The PS2 cluster will receive information from the high-level input, such as the clock signal, the reset signal, and packed data from the mouse (graph 0). It can also transfer the start message to the mouse. The PS2 mouse module then outputs the mouse displacement value to the HEX display and the mouse button information to the LEDs.



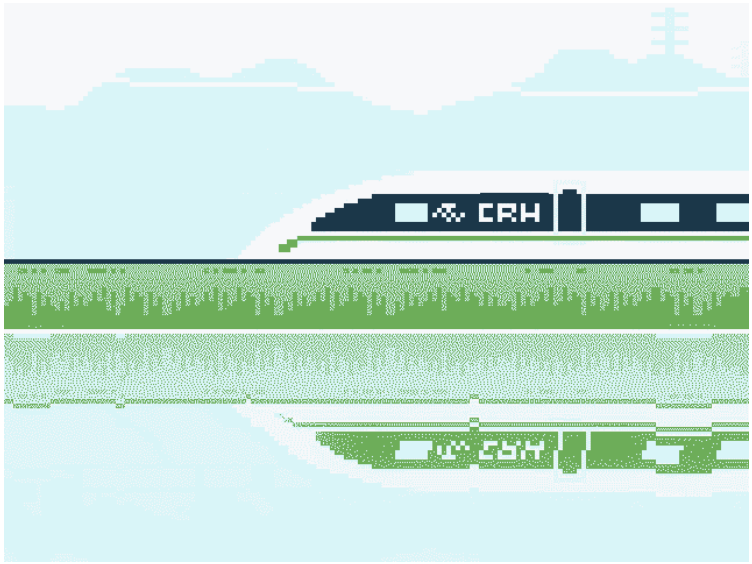
Graph 0 [1]

3. Output Check

   We have designed different ways to check the correctness of different outputs. For the display part, if the map (graph 1) and background graph (graph2) [2] are shown on the monitor clearly, the VGA output will be correct. For the sound playing part, if we can hear music with less noise, the sound output will be correct. For the mouse part, if the HEX display shows us the correct number, the mouse displacement output will be acceptable.



Graph 1 (modified from [3])



Graph 2 (modified from [2])

Module Description

1. **Hardware module without platform designer**

   I2C_Protocol

```
//Sound load
module I2C_Protocol(
    input clk,reset,ignition,
    input [15:0] MUX_input,
    inout SDIN,
    output reg finish_flag,
    output reg [2:0] ACK,
    output reg SCLK
);
```

Description: This protocol is used to control the sound play function, providing ACK to show successful data receiving. We create a counter to calculate the number of address being loaded. Once the address reaches the ending, the module will control the sound play to stop playing and reset. This module can also prevent the program from overflow.

Port: Input ignition is used to work as a trigger for the restart operation. Inout SDIN works as a tristate buffer and receives the serial data. Output ACK means Acknowledge Character, which is used to ensure that the sending data has been correctly received. Output SCLK means serial clock signal, whose negedge can be used to trigger the 28 clock cycles needed to complete a configuration cycle from the I2C bus.

Top

```
//Music top module, change some places which will be used for our implement.
module Top (
    input clk,reset,SW0,
    inout SDIN,
    output SCLK,USB_clk,BCLK,
    output reg DAC_LR_CLK,
    output DAC_DATA,
    output [2:0] ACK_LEDR
);
reg [3:0] counter; //selecting register address and its corresponding data
reg counting_state,ignition,read_enable;
reg [15:0] MUX_input;
reg [17:0] read_counter;
reg [3:0] ROM_output_mux_counter;
reg [4:0] DAC_LR_CLK_counter;
wire [15:0]ROM_out;
wire finish_flag;
assign DAC_DATA = (SW0)?ROM_out[15-ROM_output_mux_counter]:0;
```

Description: It is the top-level module for the music player. This module can read the audio data from RAM and transfer it into the music player port (LINE OUT port). Moreover, there exists one MUX we use to switch different registers. For more details implement, please refer to the commends in this file.

Port: Output BCLK is the clock signal for the input signal. Output DAC_LR_CLK is used for synchronizing the left channel and the right channel for the audio player. Output DAC_DATA is the sound data finished digital-to-analog conversion.

Lab8

```
module lab8( input              CLOCK_50,PS2_CLK,
             input        [3:0]  KEY,          //bit 0 is set up as Reset
             input [15:0] SW,
             input [9:0] XMOV_MOUSE,YMOV_MOUSE,// the position data of PS2 mouse
             input LEFT,RIGHT,                 //Left button and right button active signal of PS2 mouse
             //input [7:0] X_CHANGE,Y_CHANGE,
             //output logic [6:0]  HEX0, HEX1,
             // VGA Interface
             //output logic [7:0] LEDG,           //LED used to show the direction key

             output logic [7:0]  VGA_R,         //VGA Red
                                 VGA_G,         //VGA Green
                                 VGA_B,         //VGA Blue
             output logic        VGA_CLK,       //VGA Clock
                                 VGA_SYNC_N,    //VGA Sync signal
                                 VGA_BLANK_N,   //VGA Blank signal
                                 VGA_VS,        //VGA virtical sync signal
                                 VGA_HS,        //VGA horizontal sync signal
             // CY7C67200 Interface
             inout  wire  [15:0] OTG_DATA,      //CY7C67200 Data bus 16 Bits
             output logic [1:0]  OTG_ADDR,      //CY7C67200 Address 2 Bits
             output logic        OTG_CS_N,      //CY7C67200 Chip Select
                                 OTG_RD_N,      //CY7C67200 Write
                                 OTG_WR_N,      //CY7C67200 Read
                                 OTG_RST_N,     //CY7C67200 Reset
             input               OTG_INT,       //CY7C67200 Interrupt
             // SDRAM Interface for Nios II Software
             output logic [12:0] DRAM_ADDR,     //SDRAM Address 13 Bits
             inout  wire  [31:0] DRAM_DQ,       //SDRAM Data 32 Bits
             output logic [1:0]  DRAM_BA,       //SDRAM Bank Address 2 Bits
             output logic [3:0]  DRAM_DQM,      //SDRAM Data Mast 4 Bits
             output logic        DRAM_RAS_N,    //SDRAM Row Address Strobe
                                 DRAM_CAS_N,    //SDRAM Column Address Strobe
                                 DRAM_CKE,      //SDRAM Clock Enable
                                 DRAM_WE_N,     //SDRAM Write Enable
                                 DRAM_CS_N,     //SDRAM Chip Select
                                 DRAM_CLK       //SDRAM Clock
                   );
```

Description: The architecture of the lab8 module is very similar to the former lab, but we add new functions and input ports to implement our final project. This module is responsible for performing position data, preparing for the interaction between NIOS-II core and HW code, and storing useful data. It also contains some modules we create to implement the keyboard operation mode and keyboard plus mouse operation mode.

Port: Please see the reference in the above graph.

Ramstore

In our final project, we create five different modules to store different data we need to utilize. Since the principles of five different modules are similar, we only choose to illustrate one module.

```
module ramstore(input [9:0] DrawX, DrawY,
                output [1:0] data);       // Current pixel coordinates
```

Description: This module is used to store the railway map of China. The form of store data is one matrix whose size is 480*640*3. In this module, we can output the code of color (data) for the coordinates corresponding to DrawX and DrawY, and the module color_mapper finishes the decipher step.

Final_high_level

```
module final_high_level(
            //IO and normal clock
            input               CLOCK_50,
            input        [3:0]  KEY,          //bit 0 is set up as Reset
            input [15:0] SW,
            output logic [6:0]  HEX0,HEX1,HEX2,HEX3,
            // VGA Interface
            output logic [7:0] LEDG,LEDR,          //LED used to show the direction key
            output logic [7:0]  VGA_R,         //VGA Red
                                VGA_G,         //VGA Green
                                VGA_B,         //VGA Blue
            output logic        VGA_CLK,       //VGA Clock
                                VGA_SYNC_N,    //VGA Sync signal
                                VGA_BLANK_N,   //VGA Blank signal
                                VGA_VS,        //VGA virtical sync signal
                                VGA_HS,        //VGA horizontal sync signal
            // CY7C67200 Interface
            inout  wire  [15:0] OTG_DATA,      //CY7C67200 Data bus 16 Bits
            output logic [1:0]  OTG_ADDR,      //CY7C67200 Address 2 Bits
            output logic        OTG_CS_N,      //CY7C67200 Chip Select
                                OTG_RD_N,      //CY7C67200 Write
                                OTG_WR_N,      //CY7C67200 Read
                                OTG_RST_N,     //CY7C67200 Reset
            input               OTG_INT,       //CY7C67200 Interrupt
            // SDRAM Interface for Nios II Software
            output logic [12:0] DRAM_ADDR,     //SDRAM Address 13 Bits
            inout  wire  [31:0] DRAM_DQ,       //SDRAM Data 32 Bits
            output logic [1:0]  DRAM_BA,       //SDRAM Bank Address 2 Bits
            output logic [3:0]  DRAM_DQM,      //SDRAM Data Mast 4 Bits
            output logic        DRAM_RAS_N,    //SDRAM Row Address Strobe
                                DRAM_CAS_N,    //SDRAM Column Address Strobe
                                DRAM_CKE,      //SDRAM Clock Enable
                                DRAM_WE_N,     //SDRAM Write Enable
                                DRAM_CS_N,     //SDRAM Chip Select
                                DRAM_CLK,       //SDRAM Clock
```

```
//sound
inout I2C_SDAT,                    //I2C Data
output I2C_SCLK,AUD_XCK,AUD_BCLK,//I2C clock, USB clock and BCLK
output reg AUD_DACLRCK,            //synchronizing the left channel and right channel
output AUD_DACDAT,                 //DAC data for CODEC chip

//PS2
inout PS2_CLK,                     //PS2 clk
inout PS2_DAT                      //PS2 data
);
```

Description: This is the top-level module for our final project. This module is obtained by modifying lab8. Specifically, to implement our mouse control needs and sound output needs, we added signals about the sound and the mouse work. In total, there are three different instances in this module: display and the pathfinding part (lab8), sound playing part (Top), and mouse control part (ps2).

Port: Sound playing: Inout I2C_SDAT contains the I2C Data, while I2C_SCLK works as I2C clock. Output AUD_XCK, AUD_BCLK and AUD_DACLRCK are all the clock signal between the cyclone and CODEC chip. Inout AUD_DACDAT contains the DAC data for CODEC chip. Mouse control part: Inout PS2_CLK is the standard PS2/Clock, and inout PS2_DAT contains standard PS/2 data.

SEG_LUT

```
module SEG7_LUT    (  oSEG
                     ,iDIG);
input [3:0] iDIG;
output reg  [6:0] oSEG;
```

Description: this module works same as hex driver, but it serves for the ps2 mouse module.

PS2

```
//PS2 mouse module, modify details for our use.
module ps2(
           iSTART,   //press the button for transmitting instrucions to device;
           iRST_n,   //FSM reset signal;
           iCLK_50,  //clock source, connected with final high level;
           PS2_CLK,  //ps2_clock signal inout;
           PS2_DAT,  //ps2_data  signal inout;
           oLEFBUT,  //left button press display;
           oRIGBUT,  //right button press display;
           oMIDBUT,  //middle button press display;
           oX_MOV1,  //lower SEG of mouse displacement display for X axis.
           oX_MOV2,  //higher SEG of mouse displacement display for X axis.
           oY_MOV1,  //lower SEG of mouse displacement display for Y axis.
           oY_MOV2,   //higher SEG of mouse displacement display for Y axis.
           XMOV_MOUSE, //the horizontal coordinate of mouse.
           YMOV_MOUSE, //the vertical coordinate of the mouse.
           X_CHANGE, //not use
           Y_CHANGE, // not use
           );
           //interface;
```

Description: We have implemented a module for controlling mouse movement based on some open-source materials [1]. PS2 mouse and USB keyboard work a little differently since the former one chooses to adopt a bidirectional synchronous serial protocol is used. That is, every time a pulse is sent on the clock line, a bit of data is sent on the data line. Therefore, the host will receive the data stream when it works. This module also creates one FSM for the PS2 mouse to ensure that the mouse performs properly, including listen state, pullclk state, pulldata state and trans state. This state machine ensures that the left and right mouse buttons are correctly entered into the corresponding registers and that the position of the mouse is updated and output in real-time.

Port: Please see the above commends.

Mouse&UsbMouse

In the project we implemented both selecting cities with the keyboard and selecting cities with the mouse, because the logic of both cursors is very similar, we choose one of them to introduce.

```
module  mouseclick (input         Clk,                // 50 MHz clock
                                  Reset,              // Active-high reset signal
                                  frame_clk,          // The clock indicating a new frame (~60Hz)
                    input [9:0]   DrawX, DrawY,       // Current pixel coordinates
                    input logic [9:0] XMOV_MOUSE,YMOV_MOUSE, //position of our mouse
//                  input logic STOREBEGIN,STOREEND,
//                  input logic [7:0] X_CHANGE,Y_CHANGE,
                    output logic  is_mouse,           // Whether current pixel belongs to ball or background
                    output logic  [9:0] Ball_X_OUT,Ball_Y_OUT
                   );
```

Description: Based on the little ball in lab8, this module is used to create an instance for the cursor. It can receive the position of the mouse and generate a round cursor controlled by users.

Port: Output is_mouse is used to represent the existence of our cursor in the color_mapper module. Output Ball_X_OUT and Ball_Y_OUT will output the coordinates of the cursor center.

color_mapper (The keyboard and mouse share a color mapper module)

```
// color_mapper: Decide which color to be output to VGA for each pixel.
module  color_mapper_mouse ( input        [1:0]       data,data4,           // Whether current pixel belongs to ball
                                                       //   or background (computed in ball.sv)
                             input is_mouse,is_usbmouse, data5, controlsignal,
                             input [31:0] path1_export,path2_export,beginpoint,endpoint,
                             input [15:0] SW,
                             input [7:0] keycode,
                             input logic [9:0] Ball_X_OUT,Ball_Y_OUT,Ball_X_OUT1,Ball_Y_OUT1,
                             input LEFT,RIGHT,
//                           input key1,
                             input        [9:0] DrawX, DrawY,       // Current pixel coordinates
                             input cityname,numbercode1,numbercode2,
//                           output logic [9:0] pos_x_out,pos_y_out,
                             output logic [7:0] VGA_R, VGA_G, VGA_B, // VGA RGB output
//                           output logic STOREBEGIN,STOREEND,
                             output logic [31:0] storedata
                            );
```

Description: This module is the most critical part of the hardware language. In this module, we accomplish the following tasks:1. Display the cursor on the monitor. 2. Display the map of china and the background graph of railway on the monitor. 3. Display the selected start and end cities with their name. 4. Complete the dynamic display of the optimal path. 5. Display the time spend on train travel.

Port: Input Ball_X_OUT, Ball_Y_OUT, Ball_X_OUT1, Ball_Y_OUT1 represent the position of the cursor. Input cityname is used to decide whether to display the name of the chosen city. Input numbercode1 and numbercode2 are the time spent on train travel. Output storedata represents the code of the chosen city, which will be transferred to the ramstore2 module.

detect_mouse

```
module  detect_mouse (       input logic  LEFT,
                             input logic [7:0] keycode,
                             input logic [9:0] Ball_X_OUT,Ball_Y_OUT,Ball_X_OUT1,Ball_Y_OUT1,
                             input        [9:0] DrawX, DrawY,       // Current pixel coordinates
                             output logic [9:0] pos_x_out,pos_y_out
                            );
```

Description: This module deals with the interaction between NIOS-II and hardware modules. After the user selects the corresponding city, the module will

pass the output to the two PIO modules, and then the NIOS-II CPU will complete the reading of the data.
Port: output pos_x_out and pox_y_out means the position the user chooses as departure city or arrival city, which will be passed to two PIO modules.

INPUTCONTROL

```
//This state machine is used to provide stable signal.
module INPUTCONTROL(input logic Clk, Reset,
                    input logic [7:0] keycode,
                    output logic controlsignal);
```

Description: This module is a simple state machine that can provide a stable signal. In our project, a background image is displayed before the user enters the operator interface, and the state machine is in the START state. When the user presses the start button, the state machine switches to the END state and stays there, and the user can start looking for the path they want.
Port: Output controlsignal is a control signal for the display of the background image. Its default value is 0, and it will be assigned 1 when the current state is END.

2. Qip files

Backgroundmusic

```
module Backgroundmusic(
   address,
   clock,
   rden,
   q);

   input [17:0]  address;
   input   clock;
   input   rden;
   output   [15:0]  q;
```

Description: This is a RAM file based on altsyncram [5] for storing the data of sound. Altsyncram is an IP core owned by the company Altera for synchronous RAM. The module can read and output the data stored in the address after receiving a determined address.
Port: Input address is the address we used to store the data of sound. Since we intend to satisfy the design of music looping, we have done this in the Top module using the loop input address. The input clock is the RAM reading clock. Input rden means “read enable.” It determines if the RAM can currently be read. Output q is the specific data.

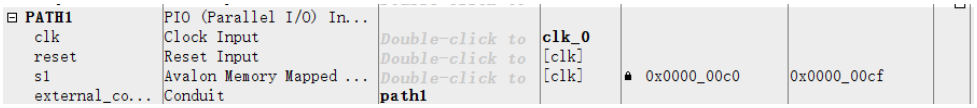
USB_Clock_PLL

```
module USB_Clock_PLL (
   inclk0,
   c0,
   c1);

   input   inclk0;
   output     c0;
   output     c1;
```

Description: This module is used to implement a phase-locked loop (PLL) circuit based on altpll module. PLL circuit is a kind of feedback control system that automatically adjusts the phase of the generated signal to match the phase of the

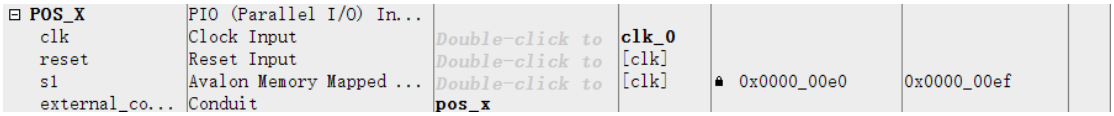
input signal. It works by first selecting an oscillator frequency that matches the input signal frequency as the reference frequency, and when the input signal changes, a phase difference is generated between the input signal and the standard signal, and this is where the PLL gets its name.
Port: Input inclk0 is the input signal. Output c0 and c1 will be the USB clock (which is not used in this module) and BCLK (which has been explained by us before).

3. Platform designer
We do not focus on the modules that have been implemented in lab7-lab9.

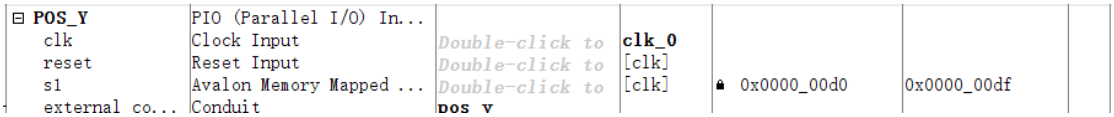


This PIO module is used to store the path that need to display. These paths are numbered between 0 and 31 in the HW code.

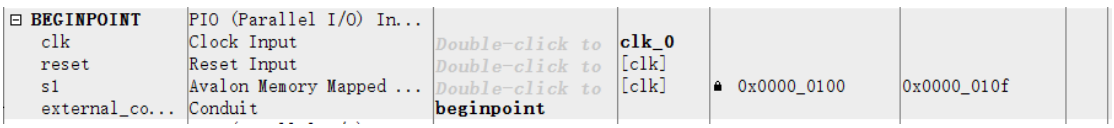
This PIO module is used to store the path that need to display. These paths are numbered between 32 and 63 in the HW code.



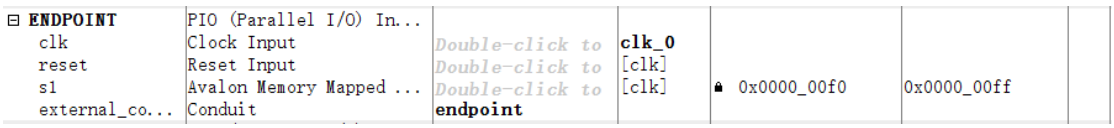
This PIO module is used to store the horizontal position of the city being chosen by the users.



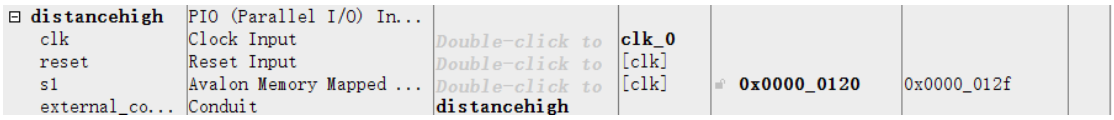
This PIO module is used to store the vertical position of the city being chosen by the users.



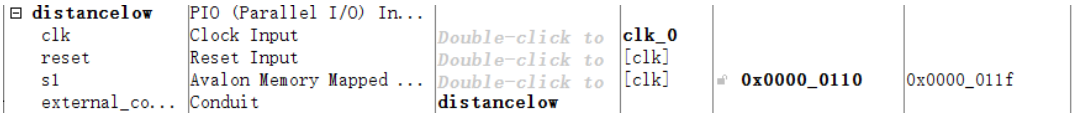
This PIO module is used to store the code of the departure city.



This PIO module is used to store the code of the destination city.



This PIO module is used to store the tens place of train travel time.



This PIO module is used to store the ones place of train travel time.

Overview of the design procedure

1. Foundation

   We develop our project using a hybrid C/System Verilog/Verilog programming approach. Moreover, in order to convert the sound information and picture information into a form that we can input, we also implement several scripts in python. In general, hybrid programming is a way to make our project implementation very flexible. Based on the knowledge and routines of lab7-lab9, we have implemented a shortest path search project with human-machine interaction, image display, and sound playback. Furthermore, to show our project's application prospect, we search the running time of railroads between major cities of China from the 12306 website [7] and construct the adjacency matrix using the running time as the path length.

2. Functions of the subprojects

   Based on the technical means of modular design, we have created different modules to perform different functions.

   a. Graph display

      Image display is one of the essential functions of our project. The solution we use is to output images (in a broad sense, including maps, background images, text, and numbers) into arrays via scripts, and the elements inside the arrays are numbered with color information. These arrays are stored in the “parameter” part of the ramstore module. This method of storage significantly reduces the space footprint and improves compilation speed. According to the if statement in the color_mapper's judgment requirements, we can display different images. It is worth mentioning that we have also implemented the function of dynamically displaying the optimal path, which is relatively difficult to implement. Its principle is using the usleep() function that comes with the C language in NIOS-II to create a typical delay in the PIO modules that store the code of path, and NIOS-II core can update the value in PIO modules in real-time.

   b. Path finding algorithm.

      [4] A* is one of the most famous AI algorithms that we will learn in EC448. It is better than Dijkstra algorithm because the time complexity is

smaller. Because Dijkstra is used to find the smallest path from the original point to all other point while A* just find the shortest path between two points.

It does the target by maintaining a tree of paths originating at the start node and extending those paths one edge at a time until its termination criterion is satisfied. We will iterate for all the possible path from the begin point to end. For every iteration and for every passing vertical, A* will determine which path from the current vertical to the end will get the smallest possible distance. The formula is $f(n) = g(n) + h(n)$. where n is the next node on the path, g(n) is the cost of the path from the start node to n, and h(n) is a heuristic function that estimates the cost of the cheapest path from n to the goal. A* terminates when the path it chooses to extend is a path from start to goal or if there are no paths eligible to be extended. The heuristic function is problem specific. If the heuristic function is admissible, meaning that it never overestimates the actual cost to get to the goal, A* is guaranteed to return a least-cost path from start to goal.

The following is the pseudo-code:

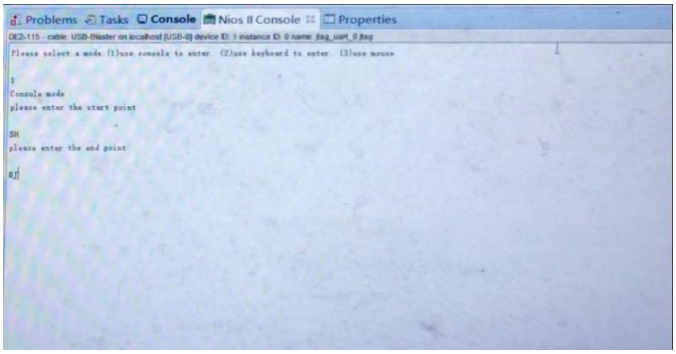
```
First create a data structure Node:
struct Node{
  int g;   // the g value of the node
  int h;   // the h value of the node
  int f;   // the f value of the node
  Node*pre; // pre node of the node
};

AStar_Search(){
  struct Node start_node;
  start_node.g = 0;
  start_node.h = H(start);
  start_node.f = start_node.h;
  start_node.pre = NULL;
  OPEN = [start_node]; CLOSE = [];

  while ( OPEN is not empty ) {
    The node with the smallest F value is obtained from the OPEN linked list,
    which is called x_node, and the corresponding node is called x;
    Remove x_node from the OPEN list;
    if (x is the end node){
      Return the path
      according to the pre pointer of the node structure corresponding to each node;
    }
    for (each successor node y of x){
      struct  Node  y_node;
      y_node.g = x_node.g+w(x,y);
      y_node.h = H(y);
      y_node.f = y_node.g+y_node.h;
      y.pre = x_node;

      if(y is not in the OPEN table and not in the CLOSE table){
        Put y_node in the OPEN table;
      }else  if(y is in the OPEN table){
        Take out the Node structure corresponding to the y node in the OPEN table,
        which is called y_open;
        if( y_node.f < y_open.f) {
          y_open.g = y_node.g;
          y_open.h = y_node.h;
          y_open.f = y_node.f;
          y_open.pre = y_node.pre;
        }
      }else{
        Take out the Node structure corresponding to the y node in the CLOSE table,
        which is called y_close;
        if(y_node.f < y_close.f){
          Remove y_close from the CLOSE list
          Put y_node in the OPEN table;
        }
      }
    } //end for

    Put x_node into the CLOSE table;
  } //end while
} // end AStar_Search
```

c. Sound player

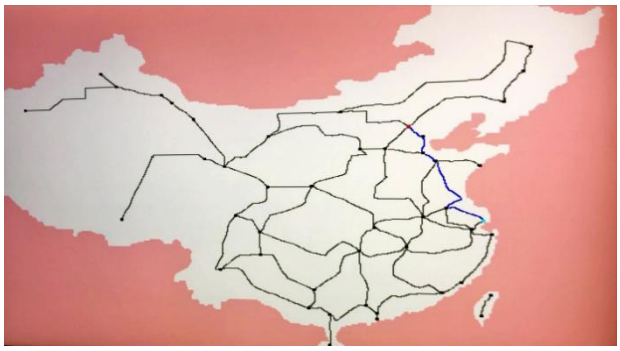
Since we have not yet taken the DSP course, we refer to Lai Xinyi's project [5] (which has been licensed) to implement the sound playback function. Firstly, we need to find the audio file in .wav format and convert it to .mif format by our own script. Moreover, Altera provides us with IP cores for sound storage and processing, and we can generate the corresponding instance of the module and connect it to the final_top_level module. The principle of sound playback is that after receiving the start signal, the sound module starts to read the sound information from the RAM and pass it to the sound player. When the counter value reaches the upper limit, the player pauses and completes the initialization of the sound part, and then the sound starts playing again. Furthermore, the sound module repeats these steps repeatedly until the switch is open.

d. Human–Computer Interaction

HCI is one of the most critical and essential functions in our project, which refers specifically to program entry and cursor control. In general, we give users three selectable modes of operation. The first mode is console-based input. This mode is mainly implemented in software. The user can enter the name of the departure city and the destination city in the console, and the program will automatically identify both cities and display the optimal route on the monitor (see graphs 1&2). The second mode is the USB keyboard input. This mode allows the user to use the “WASD” on the keyboard to move the cursor in the program and use “space” to control the cursor, including seeing the name of the city and selecting the starting and ending points. When both cities are correctly selected (marked in blue and red), the CPU starts running the pathfinding algorithm and displays the optimal path (see graph 3). The third mode is the mouse-keyboard hybrid input. This mode allows the user to bring the program from protected mode to working mode using the enter key. The user can then move the cursor by moving the mouse and select the start and end points with the left mouse button (see graph 4). When both cities are correctly selected (marked in blue and red), the CPU starts running the pathfinding algorithm and displays the optimal path.



Graph 1



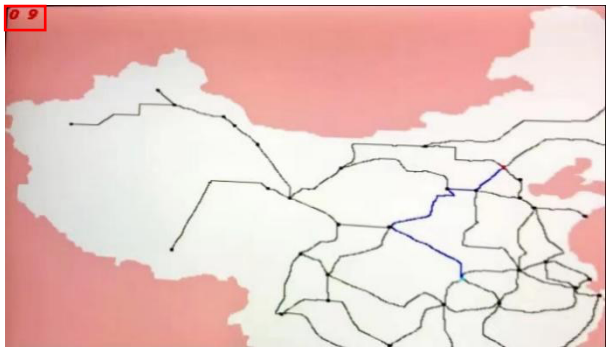
Graph 2

Graph 3

Graph 4

e. City name display
This feature allows the user to see the name of the city in the monitor after moving the cursor to the city's location (graph 4). We implement this function in the color mapper module using a MUX and VGA display ports.

f. Running time display
This feature allows the user to view the actual running time of a real-life train on the display once the optimal path has been obtained (graph 5). To implement this function, we record the running time of trains between different cities when searching for the shortest path and add them up. After the CPU finishes searching for the optimal path, it will pass the tens place and ones place of running time to different PIO modules, and the data in the modules are then passed to the color mapper to complete the display.

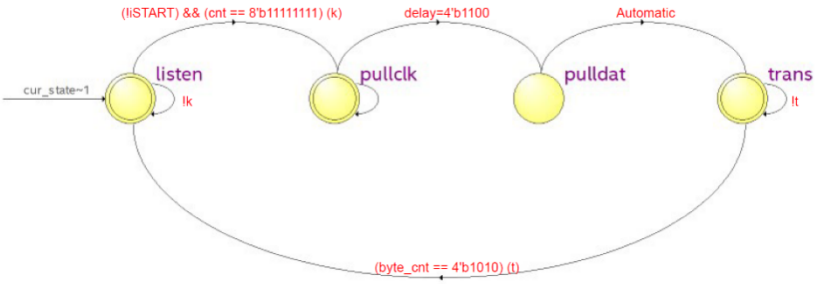


Graph 5

3. State machine

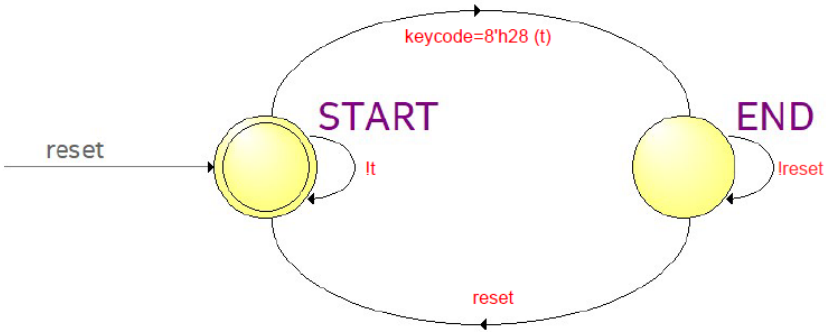
   In this project, in addition to the state machines required for SDRAM in lab7-9, we generated two groups of state machines. One group is used to control the PS2 mouse, and the other group is used to start the project.

   a. PS2



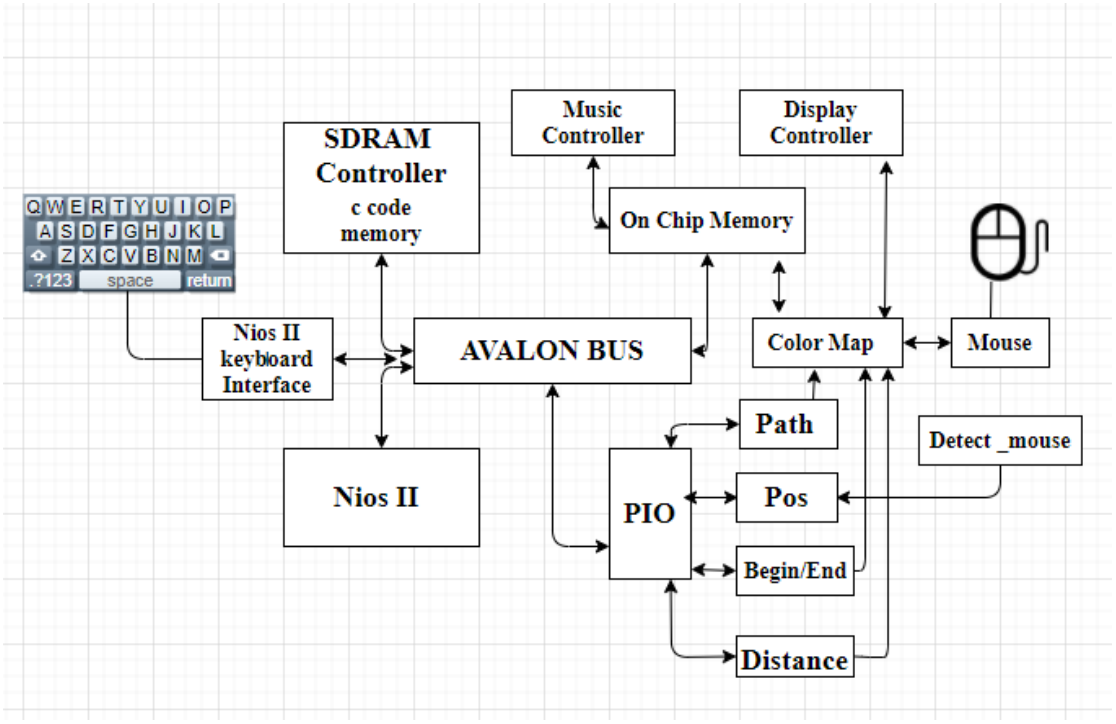
   This state machine is used to control the PS2 interface data reading. For the corresponding output value, they are:
   Listen: ce=1'b0, de=1'b0. **Pullclk**: ce=1'b1, de=1'b0. **Pulldat**: ce=1'b1, de=1'b1. **Trans**: ce=1'b0, de=1'b1.

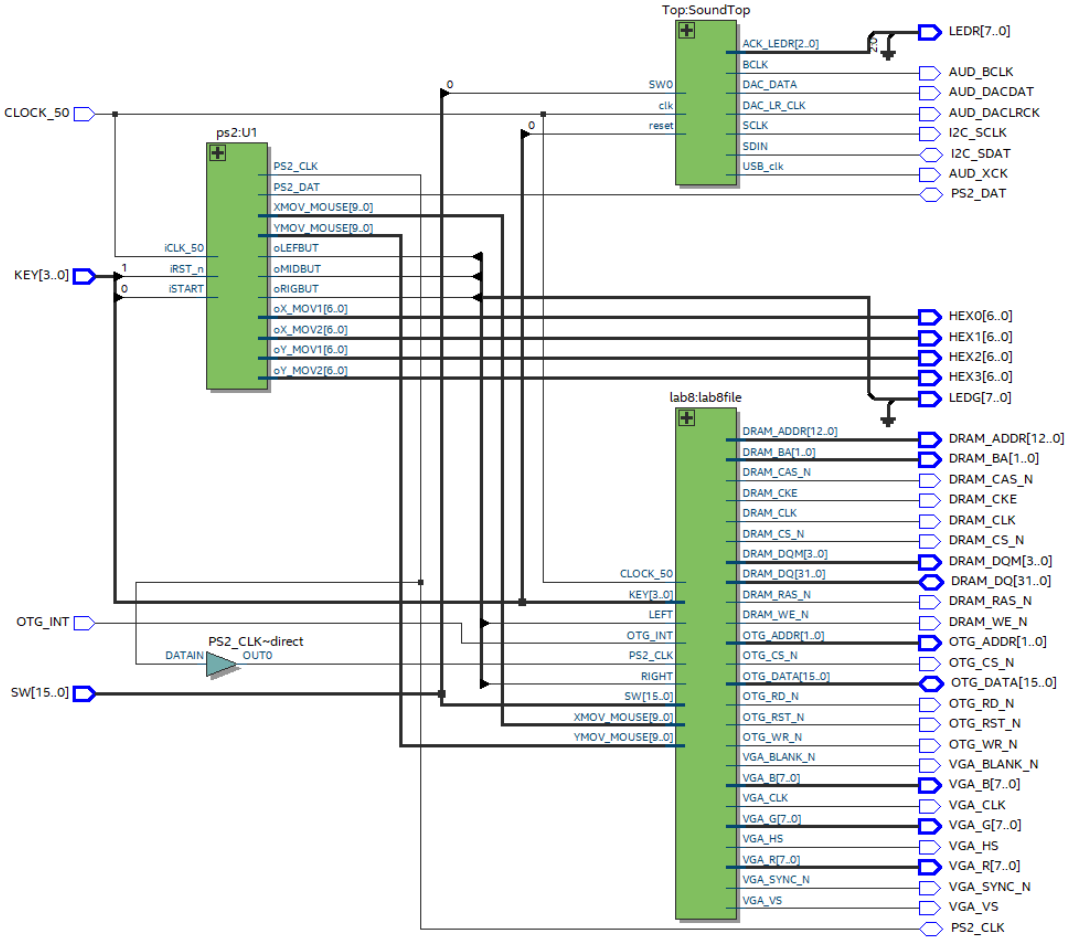
   b. INPUTCONTROL

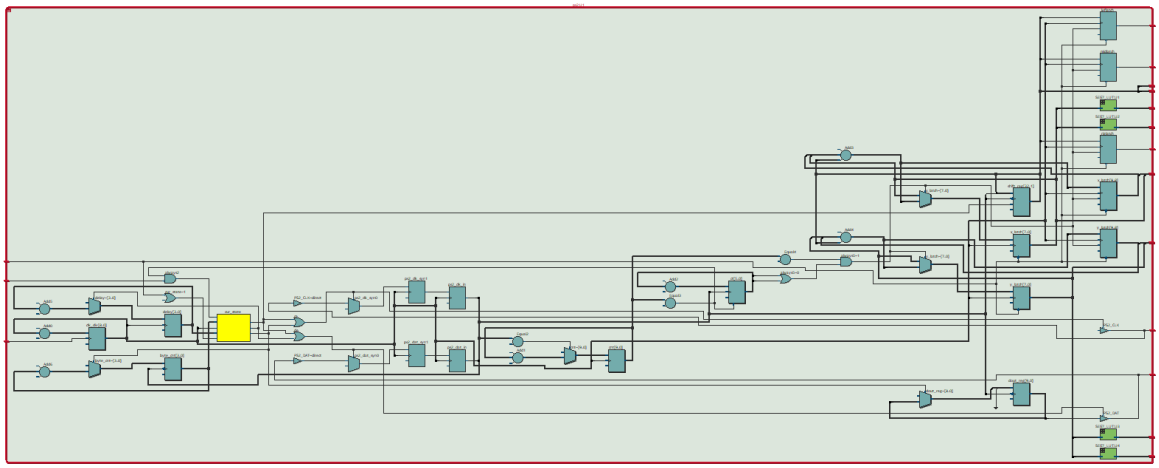


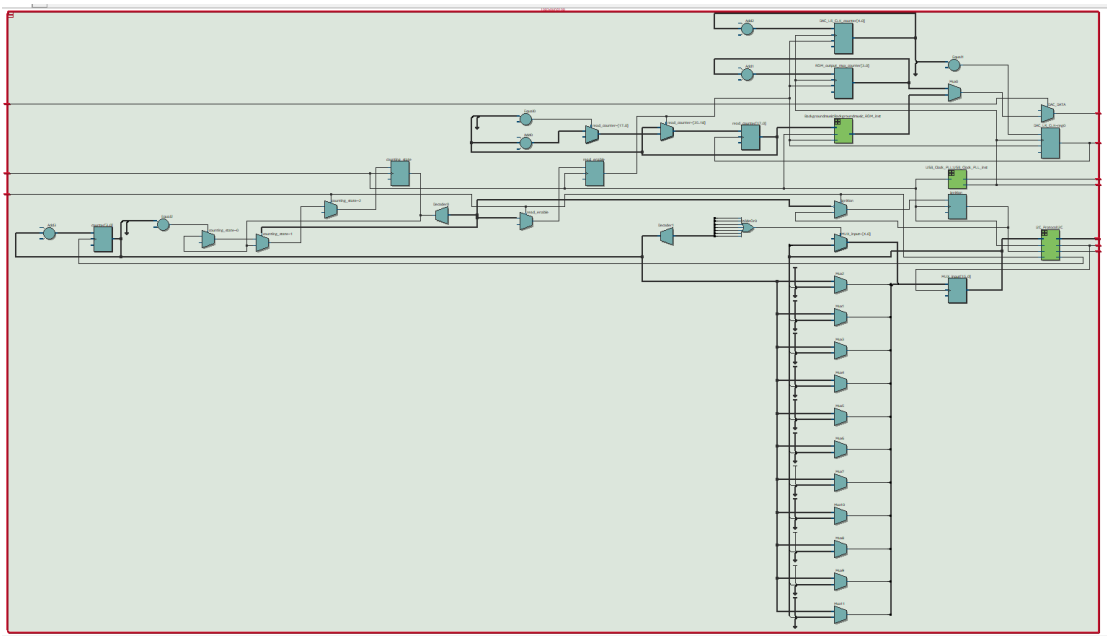
   This state machine is similar to a gate switch that ensures that the user can start using our project after pressing enter. In addition, by pressing the reset button, we can put our project into protected mode. For the corresponding output value, they are:
   START: controlsignal=1'd0. **END**: controlsignal=1'd1.

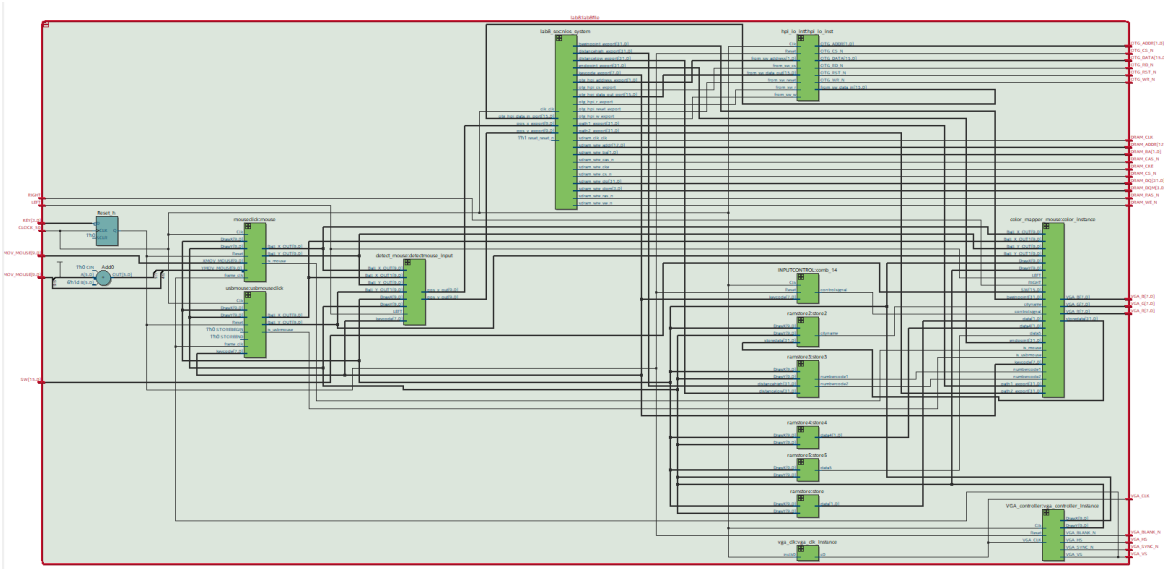
Block Diagram:











Post Lab Data

LUT	**29984**
DSP	**16**
Memory (BRAM) (bits)	**3211392**
Flip-Flop	**2837**
Frequency(MHz)	**69.58**
Static Power(mW)	**111.62**
Dynamic Power(mW)	**136.35**
I/O Thermal Power(mW)	**88.18**
Total Power(mW)	**336.16**

Conclusion:

Bugs and solution:

1. In the software A* algorithm, sometimes the algorithm cannot give the optimized path because the speed of the train is not same in all China area. I solve it by changing the original time data and using neural network to get the correct parameters.
2. When we first try to implement the music function. The must we displayed is not clear and has much noise. After we searching the article, we solve this problem by increasing the sample rate.
3. When we want to load our map to the monitor, the image displayed is in a mass. After we searching the article, we realized that our understanding to the store method of the system Verilog.
4. At the first time we want to use PS-II mouse to control the cursor, the cursor is blinking all the time. After we search the mouse control module, we find that the one array is overflow.

The target of our project is to help foreigners find optimized railway paths using an easier approach. we provide three kinds of forms for our users to operate. Console mode, keyboard mode and mouse mode. In addition, we also consider the human-computer interaction. When we finish one path search, the console will ask you whether you want to try again. And if you choose the city in Taiwan and the city in mainland, the console will tell you that the Taiwan and mainland is not connected yet. Moreover, for the mouse part, we combine the keyboard and mouse. We can use keyboard to enter the main window and use mouse to select the city. We also add the music named *The Spring Festival prelude* which is the famous music relevant to Spring Festival. We choose this music owing to the coming of the Spring Festival and many people will by the ticket to go back home and will use our project to find the shortest path. Finally, we record one demo video and upload to the YouTube. Through this project, through this course, we really learned a lot. Not only the use of the Quartus and familiar with the software and hardware combined design, but also the importance of teamwork and the determination to overcome difficulties.

Appendix

ECE 385 final project demo video.
https://www.youtube.com/watch?v=ISPvGTYFDGc

Reference

[1] Altera. DE2-115 user_manual_chinese.pdf.
[2] Background graph, https://www.zcool.com.cn/work/ZMjI5NTA3NDQ=.html.
[3] Chinese railway, https://www.pinterest.com/pin/633387417144642/.
[4] https://en.wikipedia.org/wiki/A*_search_algorithm.
[5] Lai, Xinyi. Yu, Yuqi. ECE385_FinalProject, https://github.com/Xinyi-Lai/ECE385_FinalProject (authorized).
[6] Lumetta, Steven. Computer Systems & Programming (ECE220), MP9 “Walk me there!”, http://lumetta.web.engr.illinois.edu/220-F18/.
[7] Running time, https://www.12306.cn/index/.