Welcome To Illinois - UIUC ECE 385 Final Project

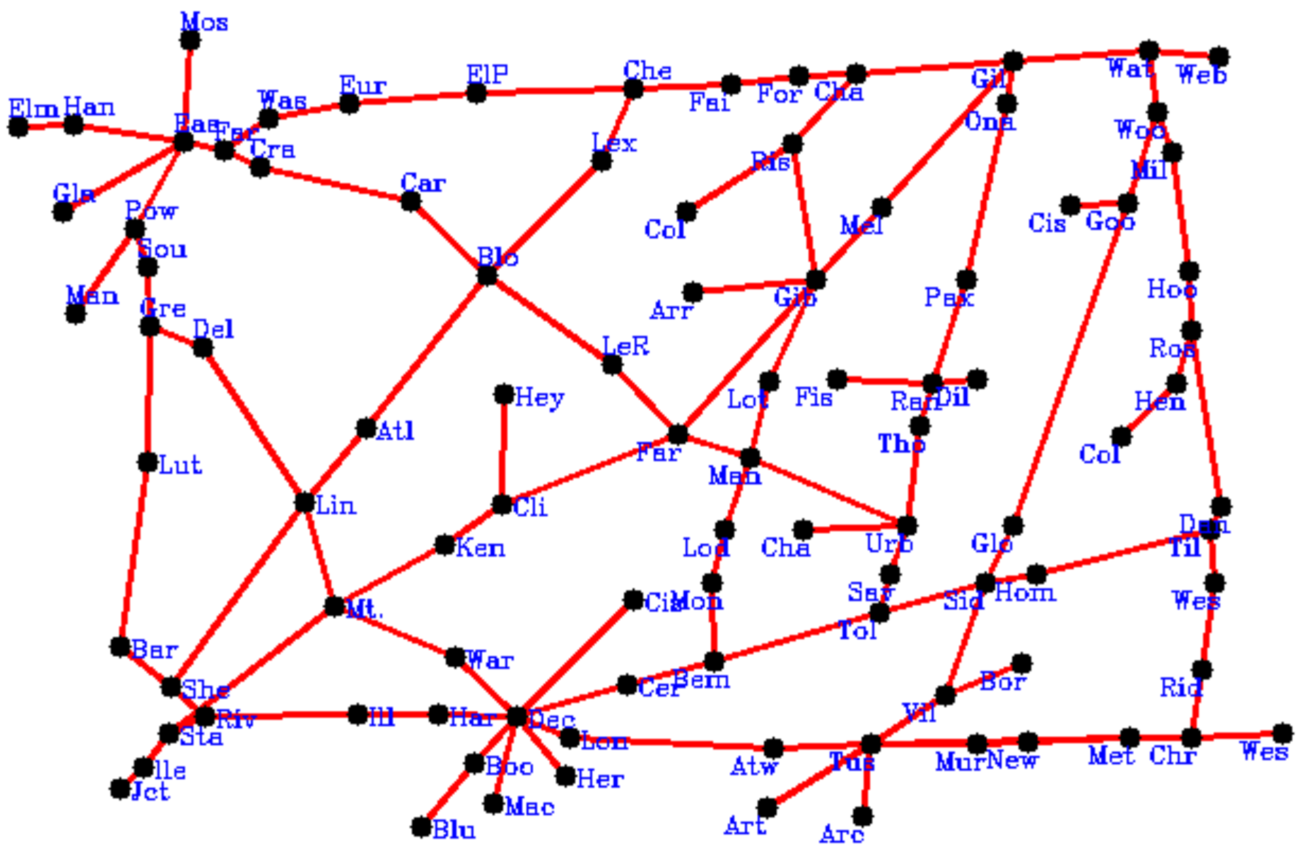
This is the public repository for the ECE385 Course Project in UIUC. Credits are given for *Travelling by Chinese Speed* by Alex Lian for initiating this project and we ameliorated his designs.

Project Github: https://github.com/BiEchi/welcome-to-illinois

Table of Contents

Introduction

With the rapid development of public communications, travelling by bus becomes an inevitable skill to command by both locals and foreigners. As we ZJUIer's are going to USA in one week, we introduced this project to act as a guide. We now introduce our project - Welcome to Illinois.



This project acts as the final project for ECE 385 in UIUC, with curtsey for Professor Zuofu Cheng and Professor Chushan Li, and all the TAs. We highly appreciate their high-quality efforts to this course.

Construction

If you want to reproduce our project, you need to construct the project first.

Construct the hardware part

To construct the hardware part, you need to go to `hardware_code/lab8.qpf` and open it with Quartus II. After opening the project, you need to compile the files and program the file onto the DE2-115 board.

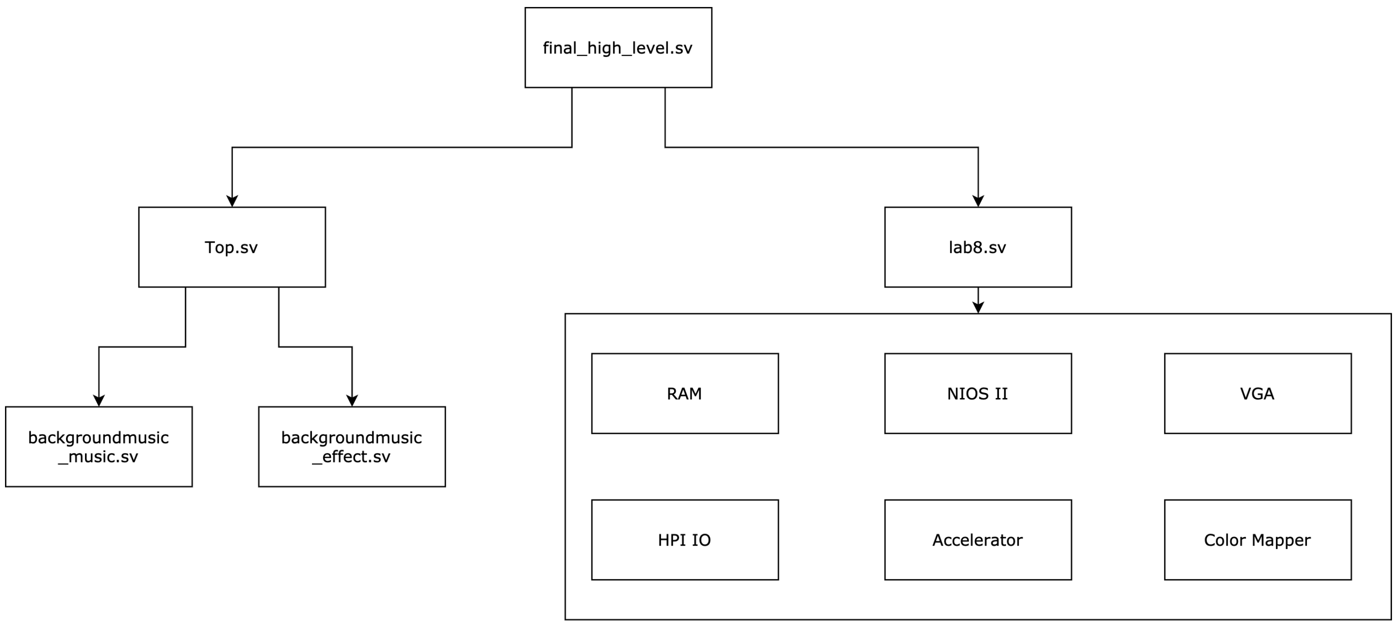
Construct the software part

As the hardware part contains a CPU, you need to program the CPU then. To program the CPU, you need to open Eclipse in Quartus II and construct a new software workspace. Then you need to copy and paste all the code in `software_code/` into the workspace and compile it. Before compilation, you need to use BSP editor to add timer0 as the timestamp generator for a precise timing behavior.

After construction, you can now travel using bus in Illinois!

General flow of circuit for different modes

High-level description

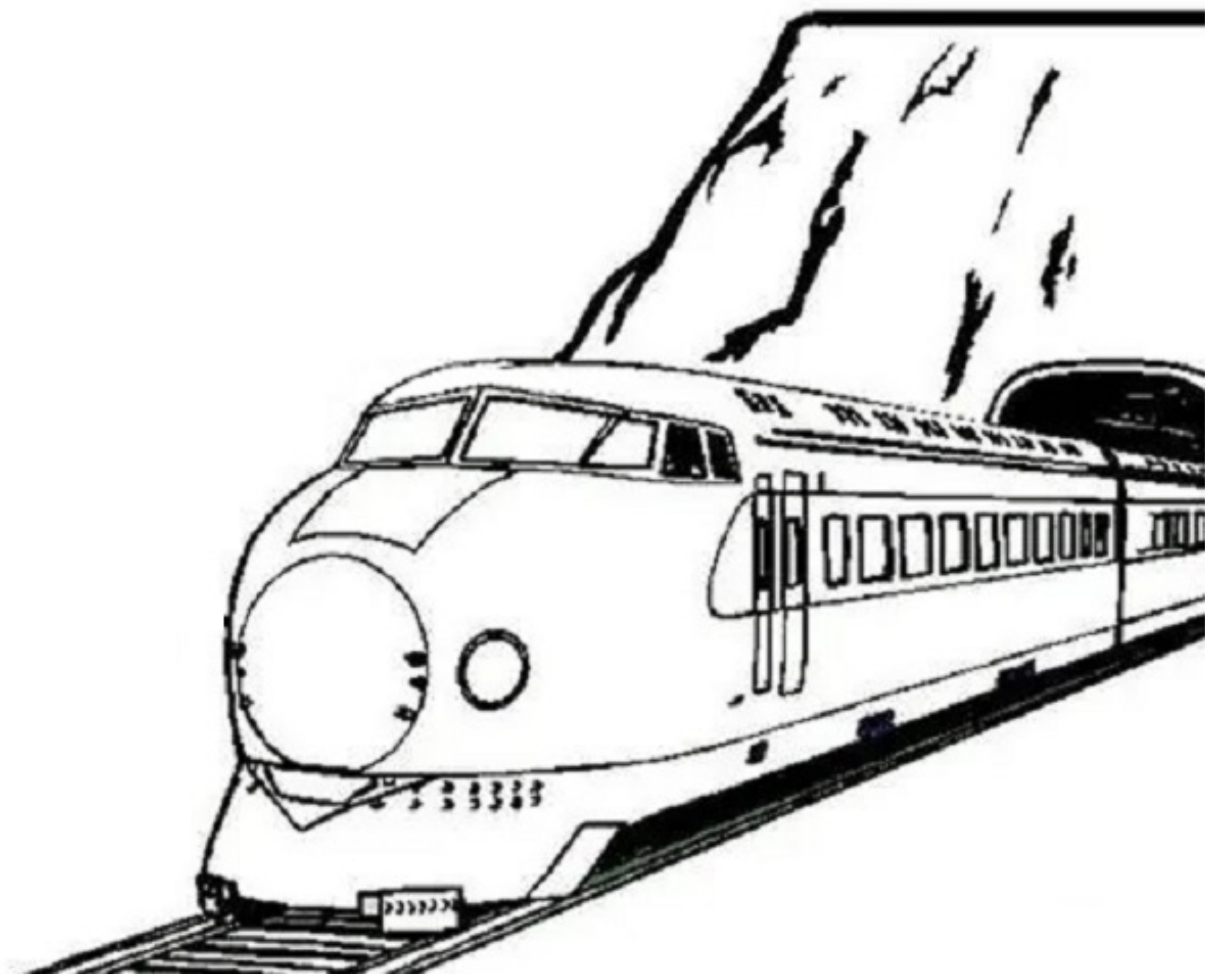


Process Description

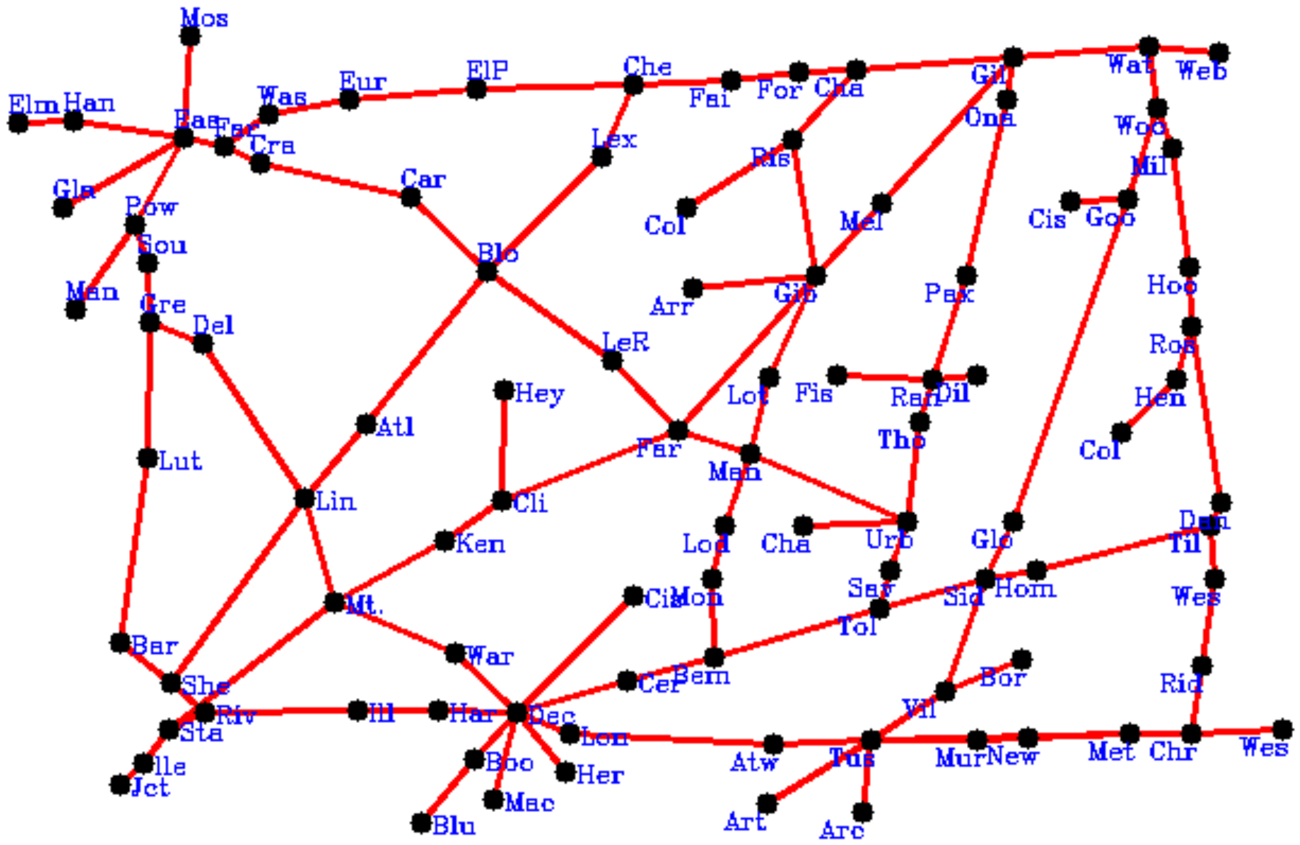
Since all other cluster have been discussed in Lab 8, we focus our explanation on the music part. The `Top.sv` module is able to receive information from `final_high_level.sv` such as clock signals, instantiation signals, and keyboard signals. When the keyboard SPACE button is not pressed, the module selects `backgroundmusic_music.sv` to play the background music, otherwise it plays the music effect to give the user some feedback.

Output Check

The output check of the path is illustrated on the screen. After the user goes inside the system, he will see the welcome interface, as shown below.



After he switches the button `SW[1]`, he will then be led to the second interface to play the path-seeking algorithm, as shown below.



After connecting the keyboard by software part, he will be able to see the keycode on the HEX display.

Module Description

`last_final_level.sv`

`accelerator.sv`

Hardware accelerated Dijkstra algorithm is implemented in this module. It takes the source city index and the target city index as input, and output the minimum path length. A adjacent list is used to store the connections between cities. A state machine is used to control the algorithm. Several PIOs are used to output the result from hardware to NIOS II software. The time complexity for the accelerated version of Dijkstra is $O(n)$.

`minimum_finder.sv`

Dijkstra needs to find the minimum current distance from source city among all possible cities in every round, so a minimum finder is implemented in this module. It applies hierarchical design to achieve a good trade-off between area and speed. A basic 8-elements minimum finder is first implemented, and it can select the minimum element among the 8 elements, both its value and index. Since we have 79 cities in total in our case, we use 16 basic 8-elements minimum finders as the first hierarchy. In the second hierarchy, two basic 8-elements minimum finders are built on top of the previous 16 basic 8-elements minimum finders, and finally a simple MUX is applied to find out the final minimum value among all cities as well as its index. This minimum finder system can work at a frequency up to 25 MHz.

`Top.v`

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.

`final_high_level.sv`

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) and sound playing part (Top).

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.

`ramstores`

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. We also added 2 modules that were not used in the original project.

This module is used to store the railway map of Illinois. 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.

`color_mapper.sv`

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 Illinois 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.sv`

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. 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.

`backgroundmusic.qip`

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.

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`

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.

Overview of the design procedure

Foundation

We develop our project using a hybrid C + SystemVerilog + Verilog programming approach. In order to convert the sound information and picture information into a form that is able to be put into the FPGA board, we also implemented several scripts to generate the code using Python. All the scripts are available in the directory `generator/`. In general, hybrid programming is a way to make our project implementation very flexible. Based on lab7-lab9, we implemented an interactable environment for user and machine to pass information. To show our project’s application prospect, we utilized the authorized map from part of Illinois and used Open CV to get the data.

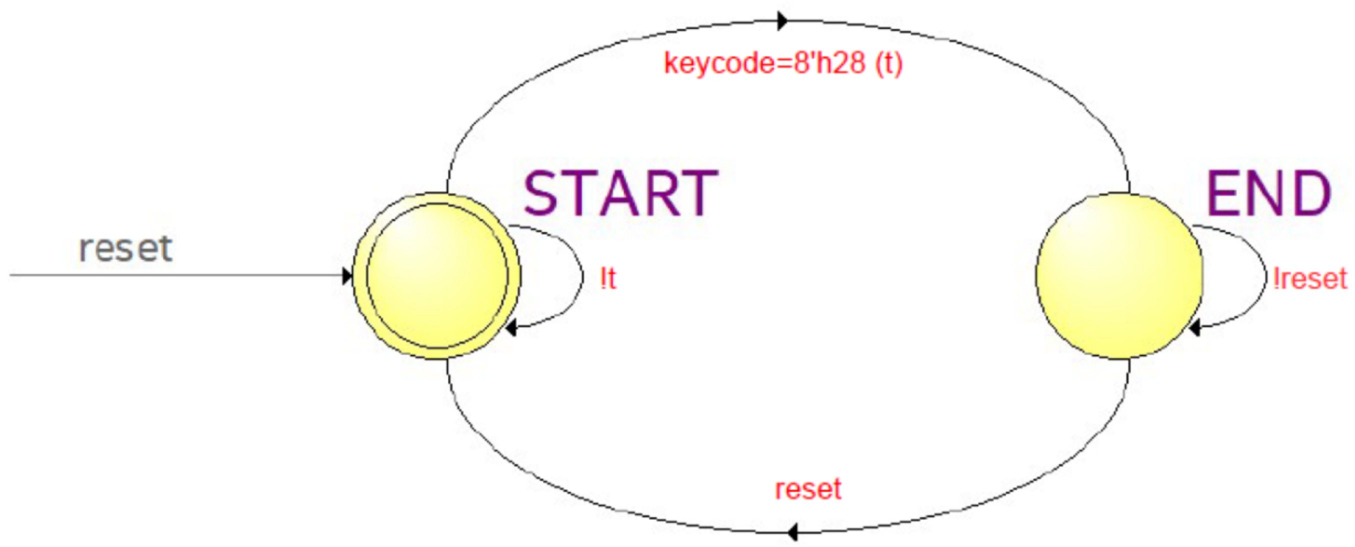
Functions of the sub-projects

- 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.

- Path finding algorithm. Although A* has a smaller time complexity, Dijkstra can be parallelized so that the apportioned time complexity can be reduced to faster than the A* algorithm.
- Sound player. Since we have not yet taken the DSP course, we refer to Alex's project (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.
- 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. 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. 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. When both cities are correctly selected (marked in blue and red), the CPU starts running the pathfinding algorithm and displays the optimal path.
- 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. We implement this function in the color mapper module using a MUX and VGA display ports.
- 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. 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.

State Machine for Lab8.sv

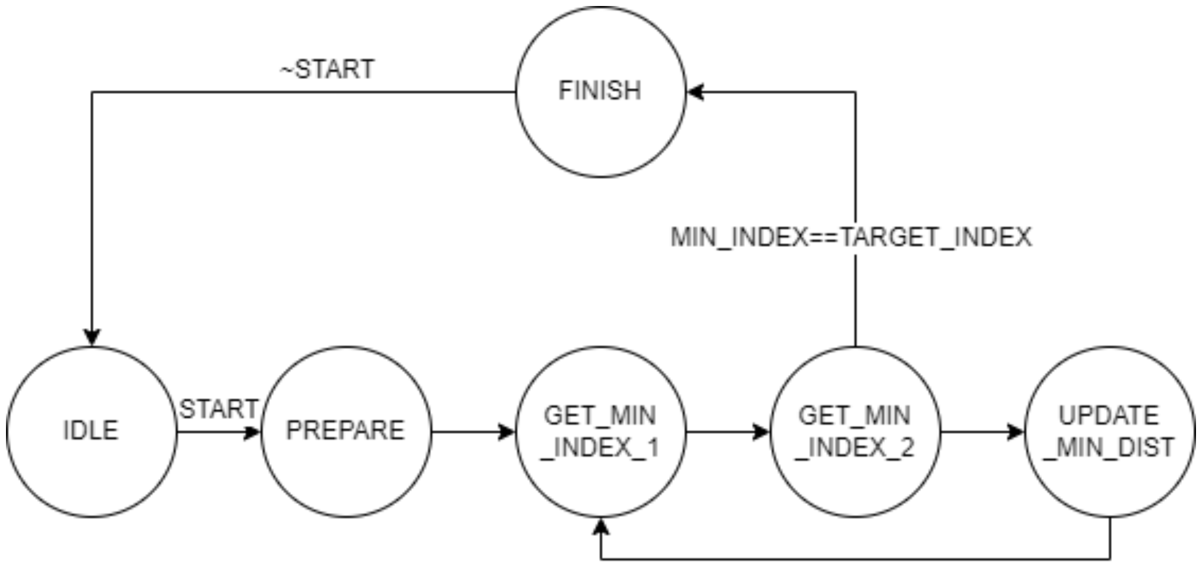
The state machine is implemented by `INPUTCONTROL.sv`.



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:

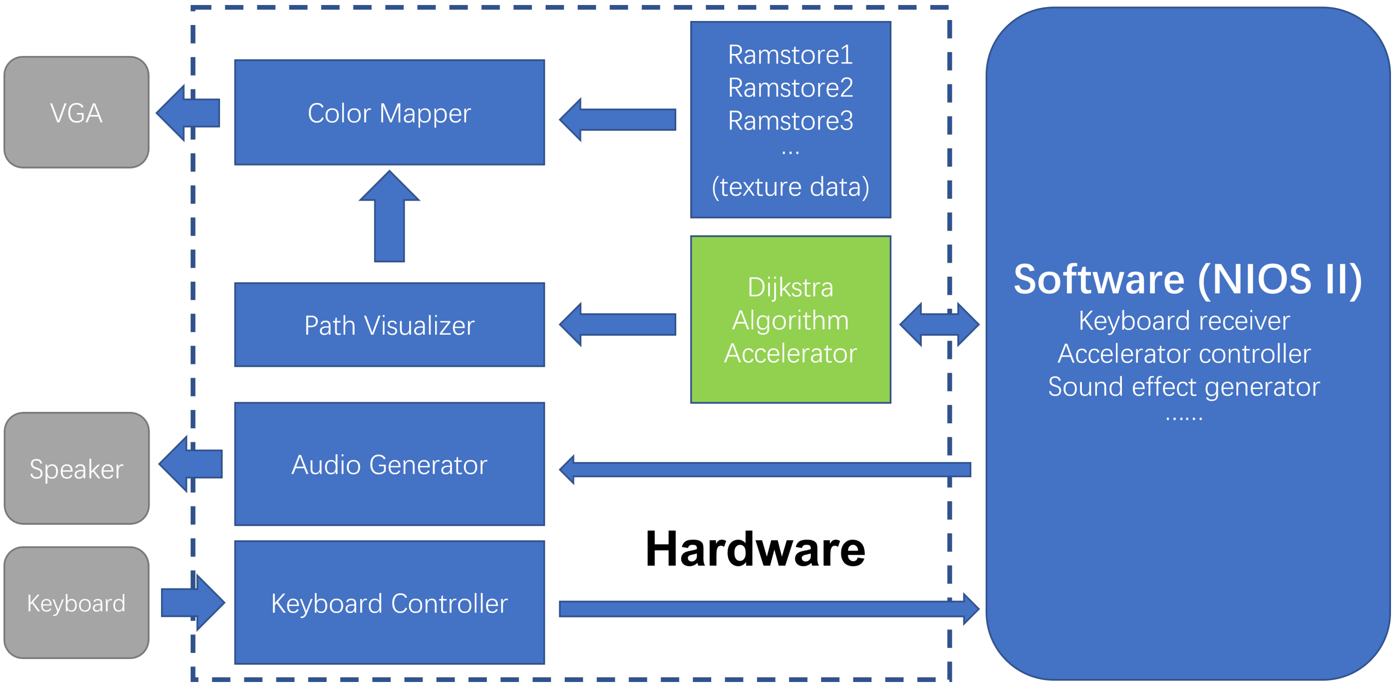
START: `controlsignal=1'd0`. **END**: `controlsignal=1'd1`.

State Machine for Accelerator



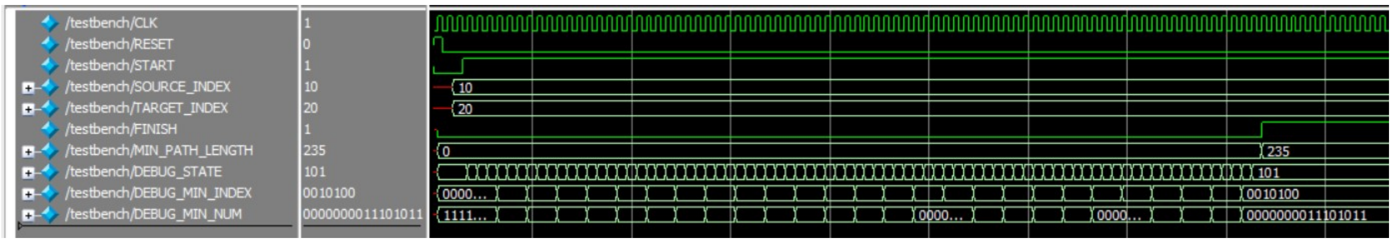
Initial state is IDLE. Then, when START signal is on, PREPARE state will be entered to clear the registers and set correct values. Then, a loop for the following three states is performed to iteratively solve the shortest path problem using dijkstra algorithm. When the target index is reached, the accelerator stops and the result will be outputted until the START signal is off. Then, it returns to the IDLE state and wait for next START signal.

Block Diagram



Simulation Waveform

When designing the hardware accelerator, RTL and Gate Level simulations are performed to make sure that the accelerator behaves as we expect. A sample waveform is shown below,



```
/testbench/CLK              1
/testbench/RESET            0
/testbench/START            1
/testbench/SOURCE_INDEX     10
/testbench/TARGET_INDEX     20
/testbench/FINISH           1
/testbench/MIN_PATH_LENGTH  235
/testbench/DEBUG_STATE      101
/testbench/DEBUG_MIN_INDEX  0010100
/testbench/DEBUG_MIN_NUM    000000001110 1011
```

We set the source index to be 10 and the target index to be 20, then set START to be 1 to trigger the accelerator. After several clock cycles, it finishes and output a constant FINISH signal, as well as the minimum path length which is 235, the correct value.

Analytical Data

Resource	Amount
LUT	60551
DSP	16
RAM	3227392
DFF	5041
Freq.	66.67
Static Power	114.64
Dynamic Power	132.37
IO Power	88.68
Total Power	337.76

Conclusion

Bugs & Solution

1. In the software dijkstra algorithm, firstly we found it hard to accelerate for the A* algorithm implemented by Alex. So we used Dijkstra algorithm, which is easier for acceleration.
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. At last, only one script is needed to generating the music.
3. When we wanted to load our map to the monitor, the image displayed was in a mass. After we searching the article, we realized that our understanding to the store method of the system Verilog.

General thoughts

The target of our project is to help foreigners find optimized railway paths using an easier approach. we provide two kinds of forms for our users to operate. Console mode and keyboard 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. We also add the music named *タヌくちの冒険* prelude which is the famous music relevant to VTubers. Finally, we record one demo video and upload to Bilibili. Through this project, through this course, we really learned a lot. Not only the use of the Quartus II and familiar with the software and hardware combined design, but also the importance of teamwork and the determination to overcome difficulties.

Contribution

We highly appreciate your idea to take a look at our project, if you think the project is worthwhile to go further on, you can consult your TA about your proposal and then start with our project. Note that it's crucial to contact us if you get any question. We'd like to help you out all the time! You can contact us by either creating an issue or sending an E-mail. Detailedly, the contribution of the team members are included below.

Jack Bai (haob2@illinois.edu)$(\dagger)$

- Hardware side of the project.
    - Display of title, cursor, number, city name, etc.
    - Design the RAM files `ramstore*.sv`.
    - Design the interfaces of hardwares and softwares.
- Audio displaying part.
    - Generation of audio.
    - Adding audio as the background music.
    - Adding the sound effects.
- Design the scripts of software and hardware.

Junyan Li (Junyanl3@illinois.edu)

- Implementation of the parallel dijkstra algorithm.
    - Fit the Dijkstra algorithm onto the board.
    - Use hardwares to accelerate the dijkstra algorithm.
- Generating the RAM files and design the map.
    - Seek data in Illinois to generate the data file.
    - Use the data file to construct a topographical map.