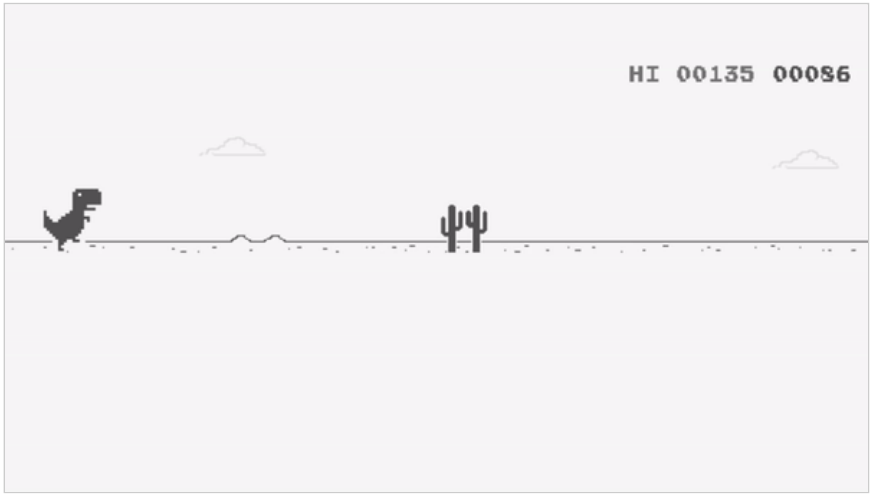
F19: T-Rex Run!

From Embedded Systems Learning Academy

T-Rex Run!

Contents



T-Rex Run

Abstract

T-Rex Run! game is an adaptation of popular chrome browser offline game in which the player steers a T-Rex dinosaur through increasingly dangerous landscapes to escape a tree of doom. The game uses simple pixel art and sound to replicate the style of 1980s arcade games. The player will be able to control the game by using switch buttons available on the Sjtwo-C microcontroller. The main goal behind the project is to understand the basics of graphics programming in the C language on a microcontroller.

Objectives & Introduction

The primary objective of this project is to write efficient code to develop a game using FreeRTOS and SJTwo Board. SJTwo Boards act as the heart of the project which acts as the controller of the game which we primarily test using onboard switch controlled with GPIO. The main objective which we are achieving with is to use the onboard switch as a controlling aspect. The onboard switch on the SJTwo board which is connected via GPIO is used to register the controller orientation and according to the orientation obtained by the sensor, game logic decides whether to make dinosaur jump up or not. The obstacles are generated randomly and are moved from left to right on the LED matrix. If dinosaur collides with any of the obstacles its game over. The speed of obstacle movement increases as the levels goes up.

The project includes the following modules:

1. Controller: One SJTwo board is used as the controller. It transmits orientation to the processor.

2. Display Module: Adafruit 32x32 Led Matrix display is used to display the game. This is driven by the GPIO pins on the master SJTwo Board.

Team Members & Responsibilities

- **Tina Ruchandani (https://www.linkedin.com/in/tina-ruchandani/)**
  - Wiki Page Updates.
- **Tushar Tarihalkar (https://www.linkedin.com/in/tushar-tarihalkar-2a5832120/)**
  - PCB and Hardware Design.
  - Software Design (Dinosaur Design and Control).
  - Collision Detection.
  - Wiki Page Updates.
- **Wenyan He (https://www.linkedin.com/in/wayan-he-80076a37/)**
  - Random Obstacle Generation.
  - Speed Controls for Obstacle Generation, Feature Addition.
- **Mahesh Mohan Shinde (https://www.linkedin.com/in/mahesh-shinde-39a251153/)**

- LED Matrix Interface and Design of base matrix code.
- Collision Detection.
- Testing.
- Wiki Page Updates.

Schedule

Week	Date	Task	Status
1	10/08	▪ T-Rex Run project proposal approved by instructor.	▪ Completed
2	10/10	▪ Project Wiki page creation. ▪ Github Repository creation. ▪ Create bill of materials. ▪ Selection and ordering parts.	▪ Completed.
3	10/15	▪ Research on software development for the game. ▪ Initial divide of project modules. ▪ Assigning responsibility to each group member.	▪ Completed.
4	10/29	▪ Understand working of LED matrix. ▪ Understand Accelerometer MMA8452Q data sheet.	▪ Completed.
5	11/13	▪ Start screen for Game on LED Matrix. ▪ Initial display testing and generation of patterns, names, shapes.	▪ Completed.
6	11/19	▪ Design and development of logic for Dinosaur Control using switch. ▪ PCB Design, Layout and Implementation. ▪ Finalizing the schematic.	▪ Completed.
7	11/26	▪ Design and development of logic for Obstacle Generation. ▪ Initial testing of Game for proper flow.	▪ Completed.
8	12/03	▪ Soldering components and hardware testing on PCB ▪ Packaging of hardware board and related components.	▪ Completed.
8	12/10	▪ Final Testing and Bug fixes ▪ Complete Wiki Report.	▪ Completed.
8	12/18	▪ Final Demo	▪ Completed.

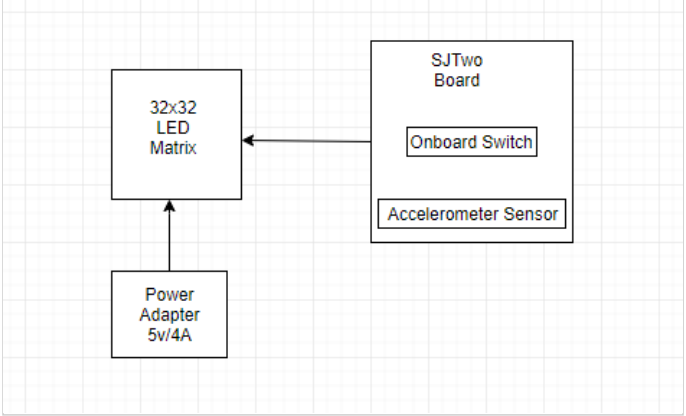
Parts List & Cost

Part	#	Cost	Source
SJ2 Board	1	$55.00	Preet
Adafruit RGB (32x32) LED Matrix Display	1	$60.80	Amazon (https://www.amazon.com/dp/B00KHBJPIK/ref=cm_sw_r_wa_api_i_WtYYDbXTNMGSK)
PCB Fabrication	1	$20.00	JLC PCB (https://jlcpcb.com/)
5V/4A Power Adapter	1	$8.99	Amazon (https://www.amazon.com/gp/product/B01N4HYWAM/ref=ppx_yo_dt_b_asin_title_o01_s00?ie=UTF8&psc=1)
Female DC Power adapter - 2.1mm jack	1	$2.99	Adafruit (https://www.adafruit.com/product/368)
Jumper Wires	1	$6.99	Amazon (https://www.amazon.com/Elegoo-EL-CP-004-Multicolored-Breadboard-arduino/dp/B01EV70C78/ref=sr_1_3?crid=126P4QWNMIBO3&keywords=connecting+wires+arduino&qid=1573612339&sprefix=connecting+wires%2Caps%2C181&sr=8-3)

- Total Cost: $154.77

Design & Implementation

The block diagram for the project given below depicts the flow of the game



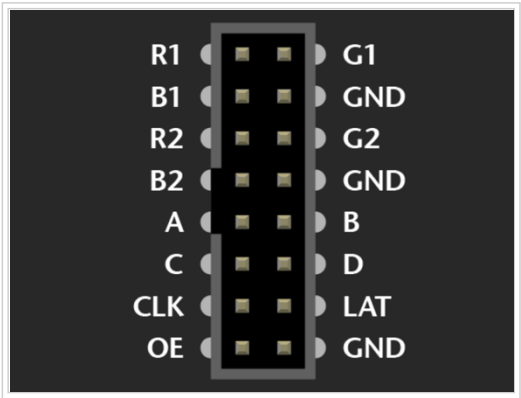
State_Diagram

Hardware Design

The hardware design for the project consists of a 32x32 RGB LED matrix operated using SJTWO Board (LPC4078). The RGB LED matrix is controlled using GPIO pins available on the microcontroller. The functioning of the LED matrix is basically controlled by the four data lines A, B, C and D which can be addressed and used to control each LED on the Matrix.

LED Matrix specifications:

- 190.5mm x 190.5mm x 14mm / 7.5" x 7.5" x 0.55"
- Panel weight with IDC cables and power cable: 357.51g
- 5V regulated power input, 4A max (all LEDs on)
- 5V data logic level input
- 2000 mcd LEDs on 6mm pitch
- 1/16 scan rate



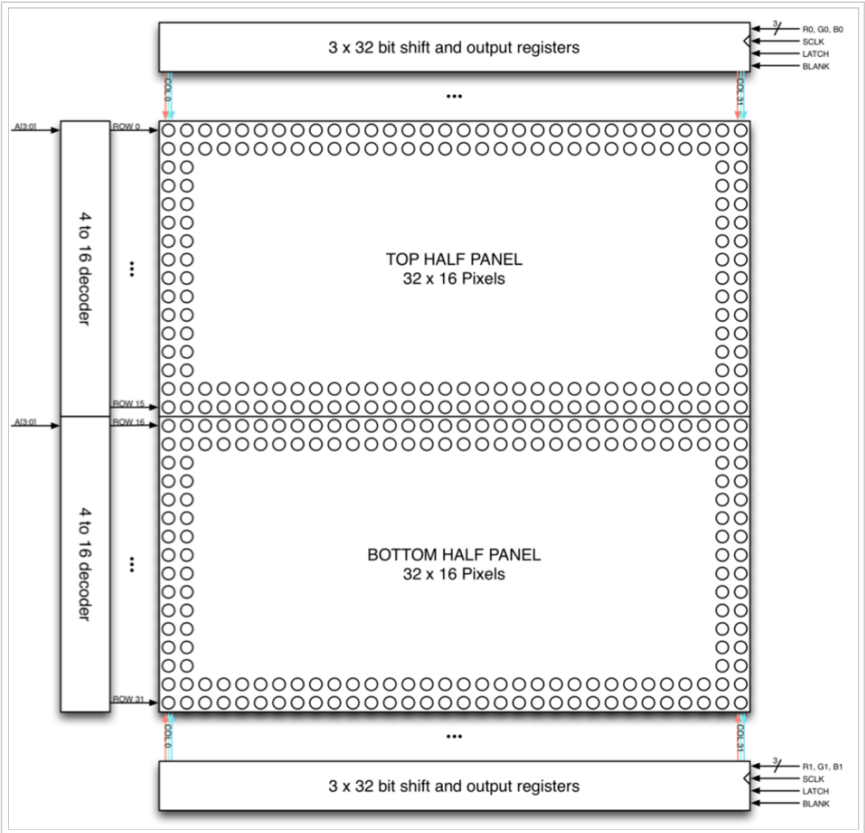
RGB LED Matrix connection ports

Label	Name	Pin Function
1	R1	Red data for top half of the LED Matrix
2	G1	Green data for top half of the LED Matrix
3	B1	Blue data for top half of the LED Matrix
4	A	Select line A
5	B	Select line B
6	C	Select line C
7	D	Select line D
8	R2	Red data for bottom half of the LED Matrix
9	B2	Blue data for bottom half of the LED Matrix
10	G2	Green data for bottom half of the LED Matrix
11	CLK	CLOCK Pin
12	LAT	LATCH
13	OE	Output Enable
14	GND	Ground

LED Matrix Control

The 32x32 LED matrix consists of 1024 LEDs arranged in 32 rows and 32 columns each, with each led having a separate Red, Green and Blue LED chips assembled together as a single unit. This 32x32 LED matrix comprises of 2 of 16x32 led matrices placed vertically to each other. There are different drivers for controlling the corresponding rows and columns each. In order to illuminate a particular led, one needs to access and control the corresponding row and column (just like in coordinate-axis). To change the color of a particular led, each led must be controlled using the driver for that row and column.

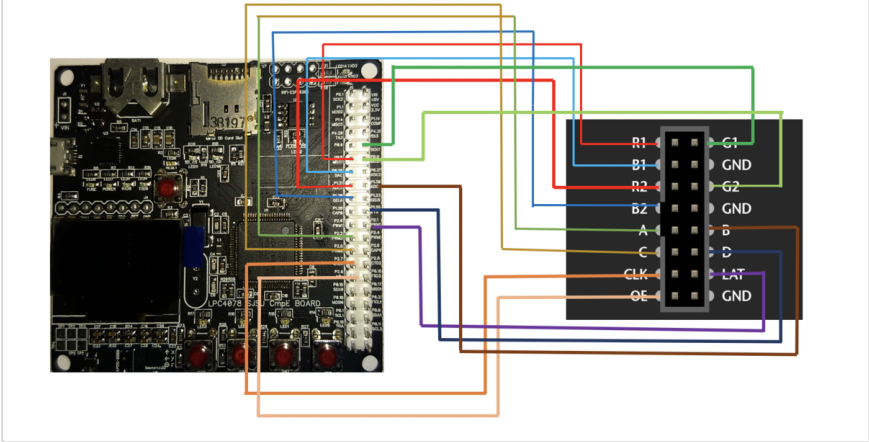
The matrix pixels are controlled by 6 drivers, 3 for the top half of the matrix(R1,G1,B1) and 3 for the bottom half(R2,G2,B2). The three drivers for R, G and B share a SCLK, Latch and BLANK signal. The top half pixels of the matrix controlled by R1, G1,B1 are rows 0-15 and the second half controlled by R2,G2,B2 are rows 16-31. The display is multiplexed at a duty cycle of 1/16, this means that one row in the top half and one row in the bottom half will be illuminated at a time. In order to display an image, the row and column LEDs for that image must be turned “ON”, therefore, the entire panel must be scanned at a rate(speed) at which it appears that the image is being displayed without flickering. To display different colors and different brightness levels, the LED chips of corresponding row and column must be adjusted by varying the amount and time that each LED chip is turned on during each cycle.



RGB LED Matrix Control

Hardware Interface

The connections to the REG LED matrix with the corresponding microcontroller (LPC4078) are as follows:

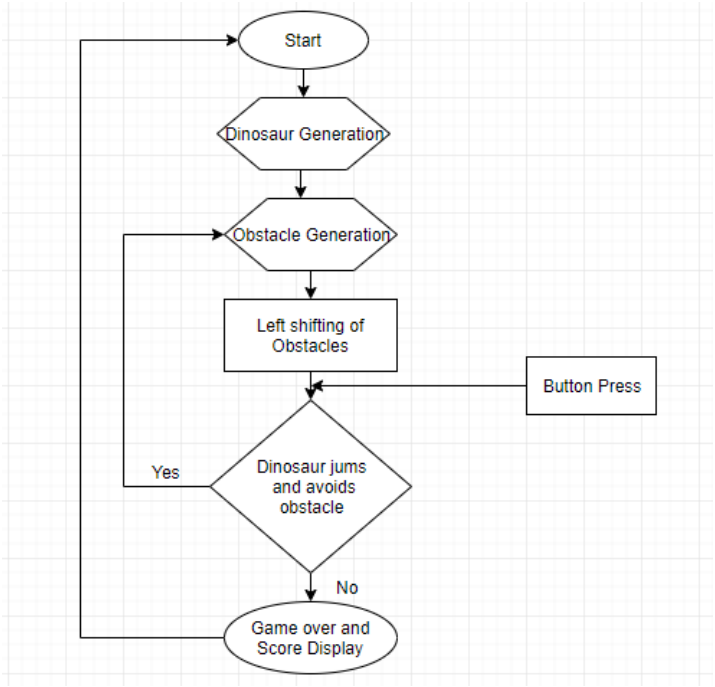


Interface of SJTWO board (LPC4078) with LED Matrix using GPIO

SJTWO Board pins	LED matrix pin	Pin Function
P0.8	R1	Red data for top half of the LED Matrix
P0.7	G1	Green data for top half of the LED Matrix
P0.26	B1	Blue data for top half of the LED Matrix
P2.2	A	Select line A
P1.30	B	Select line B
P2.5	C	Select line C
P1.29	D	Select line D
P1.31	R2	Red data for bottom half of the LED Matrix
P1.20	B2	Blue data for bottom half of the LED Matrix
P0.9	G2	Green data for bottom half of the LED Matrix
P2.7	CLK	CLOCK Pin
P2.1	LAT	LATCH
P2.9	OE	Output Enable
GND	GND	Ground

Software Design

The following state flow diagram illustrates the tasks for software implementation. The major thing behind the implementation of T-Rex Run game is a thorough understanding of the control of the LED Matrix.



Game Logic:

A matrixbuf array maps onto the LED Display Matrix. A value overwritten in this two-dimensional buffer updates the corresponding Pixel value of the LED Matrix. The drawPixel function maps onto the matrixbuf and updates the corresponding value.

There are different tasks that handle the generation of the Dinosaur, generation of obstacles, collision detection, shifting of obstacles as well as maintaining the scoring of the game. These tasks run in parallel and are scheduled with the help of of the task scheduler that updates the display regularly

There were three algorithms that were designed for the implementation of the game.

1) Dinosaur Generation and Hopping on Button Press:

The Dinosaur generation is simple and straight forward where the row and column are given as input parameters to the drawDino function which writes these values to the matrix buffer. For control of a dinosaur, we've implemented semaphore based logic which helps dinosaur hop up on switch press.

However, complexity increases when the input of the button determines the position of the dinosaur and the limit for the collision are checked with boundaries of the available screen estate.

```
void buttonpress(void *p) {
  while (1) {
    for (uint8_t col = 0; col < 32; col++) {
      matrixbuf[28][col] = 0x18;
      matrixbuf[29][col] = 0x18;
      matrixbuf[30][col] = 0x18;
    }
    LPC_GPIO2->PIN |= (1 << 3);
    if (0 == !(LPC_GPIO0->PIN & (1 << 30))) {
      HOP_FLAG = 1;
      cleardino();
      xSemaphoreGive(hop);

    } else {
      vTaskDelay(1000);
      drawdino();
    }
  }
}

void make_it_hop(void *p) {
  while (1) {
    xSemaphoreTake(hop, portMAX_DELAY);
    cleardino();
    hopdino();
    vTaskDelay(100);
    clearhopdino();
    drawdino();
    LPC_GPIO0->PIN |= (1 << 30);
    HOP_FLAG = 0;
  }
}

void drawdino() {
  matrixbuf[24 - 1][4] = 0x30;
  matrixbuf[24 - 1][5] = 0x30;
  matrixbuf[25 - 1][4] = 0x30;
  matrixbuf[26 - 1][4] = 0x30;
  matrixbuf[26 - 1][5] = 0x30;
  matrixbuf[27 - 1][3] = 0x30;
  matrixbuf[27 - 1][4] = 0x30;
  matrixbuf[27 - 1][3] = 0x30;
  matrixbuf[27 - 1][4] = 0x30;
  matrixbuf[27 - 1][5] = 0x30;
  matrixbuf[28 - 1][3] = 0x30;
  matrixbuf[28 - 1][4] = 0x30;
  matrixbuf[28 - 1][5] = 0x30;
}

void cleardino() {
  matrixbuf[24 - 1][4] = 0x00;
  matrixbuf[24 - 1][5] = 0x00;
  matrixbuf[25 - 1][4] = 0x00;
  matrixbuf[26 - 1][4] = 0x00;
  matrixbuf[26 - 1][5] = 0x00;
  matrixbuf[27 - 1][3] = 0x00;
  matrixbuf[27 - 1][4] = 0x00;
  matrixbuf[27 - 1][5] = 0x00;
  matrixbuf[28 - 1][3] = 0x00;
  matrixbuf[28 - 1][4] = 0x00;
  matrixbuf[28 - 1][5] = 0x00;
}

void hopdino() {
  matrixbuf[18][4] = 0x30;
  matrixbuf[18][5] = 0x30;
  matrixbuf[19][4] = 0x30;
  matrixbuf[20][4] = 0x30;
  matrixbuf[20][5] = 0x30;
  matrixbuf[21][3] = 0x30;
  matrixbuf[21][4] = 0x30;
  matrixbuf[21][5] = 0x30;
  matrixbuf[22][3] = 0x30;
  matrixbuf[22][4] = 0x30;
  matrixbuf[22][5] = 0x30;
}

void clearhopdino() {
  matrixbuf[18][4] = 0x00;
```

```
  matrixbuf[18][5] = 0x00;
  matrixbuf[19][4] = 0x00;
  matrixbuf[20][4] = 0x00;
  matrixbuf[20][5] = 0x00;
  matrixbuf[21][3] = 0x00;
  matrixbuf[21][4] = 0x00;
  matrixbuf[21][5] = 0x00;
  matrixbuf[22][3] = 0x00;
  matrixbuf[22][4] = 0x00;
  matrixbuf[22][5] = 0x00;
}
```

2) Obstacle Generation and Shifting to left:

A random number determines the placement of the gap between the obstacles for the dinosaur to escape through. Depending on the value of the random gap generated, the matrix buffer is updated and subsequently, the obstacle is displayed. The speed with which obstacles are moving to the left is varied with a fixed level which determines the complexity of T-Rex Run game.

```
void generate_obstacle_task(void *p) {

  while (1) {
    if (counter > rand_gap) {
      rand_wd = rand() % MAX_WD;
      rand_ht = rand() % MAX_HT;
      rand_gap = rand() % MAX_GAP + 5;
      for (col = 31; col >= 32 - rand_wd; col--) {
        for (row = 28; row >= 29 - rand_ht; row--) {
          obstacle_buf[row][col] = 0x20;
        }
      }
      delay__ms(40);
      counter = 0;
    }
    counter++;

    for (col = 0; col < 32; col++) {
      for (row = 28; row >= 29 - MAX_HT; row--) {
        obstacle_buf[row][col] = obstacle_buf[row][col + 1];
      }
    }

    // obstacles moving at different speeds in stages
    if (counter2 < 50) {
      vTaskDelay(200);
      counter2++;
    } else if (counter2 < 150) {
      vTaskDelay(100);
      counter2++;
    } else if (counter2 < 275) {
      vTaskDelay(80);
      counter2++;
    } else if (counter2 < 441) {
      vTaskDelay(60);
      counter2++;
    } else if (counter2 < 691) {
      vTaskDelay(40);
      counter2++;
    } else {
      counter2 = 0;
    }
  }
}
```

3) Collision Detection:

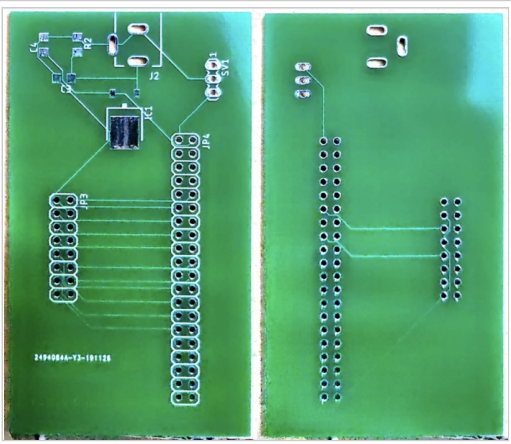
The Dinosaur continues to run and jump over the screen space unless it collides with the incoming obstacles or runs out of the screen space. In such a case, the game is stopped and the end game screen is displayed.

```
void collision_detection() {

  if (((matrixbuf[24][5] == 0x30) && (obstacle_buf[24][5] == 0x20)) ||
      ((matrixbuf[25][5] == 0x30) && (obstacle_buf[25][5] == 0x20)) ||
      ((matrixbuf[26][5] == 0x30) && (obstacle_buf[26][5] == 0x20)) ||
      ((matrixbuf[27][5] == 0x30) && (obstacle_buf[27][5] == 0x20)) ||
      ((matrixbuf[28][5] == 0x30) && (obstacle_buf[28][5] == 0x20))) {

    // for (uint8_t row = 0; row < 32; row++) {
    //   for (uint8_t col = 0; col < 32; col++) {
    //     matrixbuf[row][col] = GameOver[row][col];
    //   }
    gameover = true;
    for (uint8_t row = 12; row < 16; row++) {
      for (uint8_t col = 12; col < 16; col++) {
        matrixbuf[row][col] = 0x7;
      }
    }
  }

  else {
    for (uint8_t row = 12; row < 16; row++) {
      for (uint8_t col = 12; col < 16; col++) {
```

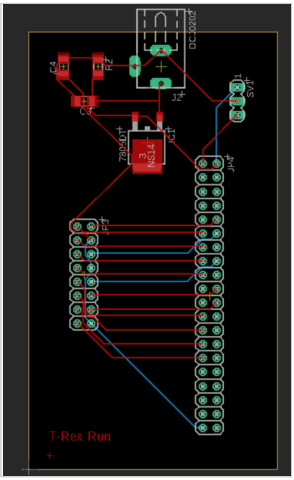
```
        matrixbuf[row][col] = 0x0;
      }
    }
  }
}
```

Printed Circuit Board Design

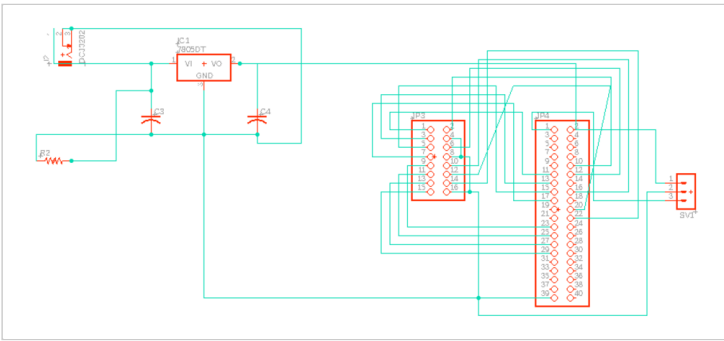
We have designed and developed a PCB in order to supply power for LPC4078 (SJTWO board) and RGB LED Matrix which is able to provide 5v and 1A supply efficiently. The PCB Layout is designed using the Eagle Software v9.5.1. The Power Supply circuit has an IC7805 voltage regulator IC and a voltage divider to fulfill the specific power requirements. IC7805 is a linear voltage regulator which has a variable output voltage ranging from 4.8 V to 5.2 V and is suitable for our application. We have used a 5V adapter in order to power our board.



Manufactured PCB



Board Layout

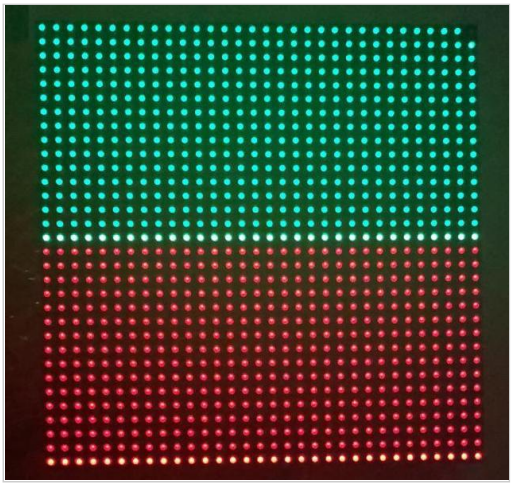


PCB Schematic

Implementation

This section includes implementation, but again, not the details, just the high level. For example, you can list the steps it takes to communicate over a sensor, or the steps needed to write a page of memory onto SPI Flash. You can include sub-sections for each of your component implementation.

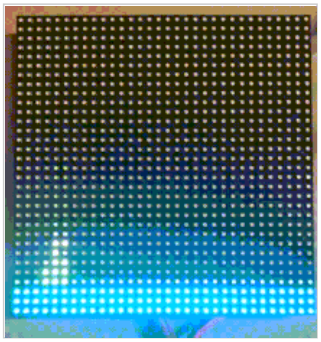
Testing



Initial LED matrix testing

Initial start screen

End Game screen



Dinosaur Hop testing

Technical Challenges

1) Display Driver Implementation:

When we started implementing an initial driver for testing LED matrix for basic shapes, figures, number as well as controlling individual pixel, we tried to replicate the Adafruit library since the library was primarily designed to work with Arduino. Hence, we were unsuccessful in porting the library entirely. Then we moved on with reading various Sparkfun libraries where we understood the basic working of RGB LED matrix and decided to control over a single row first and then over a single pixel. We first implemented a function to first select the row according to A, B, C and D signals and write the corresponding pixel values of all LEDs in those rows with the updated values from the frame buffer. The values of the pixels were set only within the clock trigger. In the end, the pixel values were latched before the output enable signal was asserted again. This function was called from a periodic task with the 2.5-millisecond delay in order to get the refresh rate of the display as 2.5 milliseconds. This helped us to get control over individual pixels of the matrix.

2) Obstacle generation:

i. Obstacle moving left is intended. But only left edge moves while right edge stays. So the obstacle expands to the left rather than moves to the left.

Solution: In the initial effort, obstacle generation is defined in a function. Then another function defined as a task calls the first function. After multiple efforts were made at solving the problem, we simply made the first function a task and got rid of the second function. Problem solved.

ii. It's easy to generate one obstacle and move it to the left. It's somewhat complex to randomly generate multiple obstacles in the same frame.

Solution: In the obstacle generation task function, a block of a random width and height is generated. A random gap is also generated. After the block moves to the left by this random gap, another block is randomly generated. This guarantees that any two adjacent blocks don't overlap and multiple blocks appear in the same frame properly.

iii. When randomly generated obstacles of different heights move to the left, taller obstacles in the same frame mess up the display.

Solution: Make sure to copy all rows within the maximum possible height (the constant MAX_HT in our code).

3)Collision Detection:

Identifying the incoming collisions at varying speed was a task since there were issues of time-lag. The issue was solved by separating different tasks adding delay wherever appropriate and setting their priority.

4) Code Integration:

i.Smooth transition: Starting screen is expected to pause for a few seconds and then the game screen follows moving/not moving: Dinosaur and obstacles move toward the left simultaneously even though the former is expected to be stationary horizontally.

ii. Game score display: We designed a nice progress bar to indicate the scoring rather than display a number. The End Game screen then displays the total score when the game ends.

Conclusion

We successfully designed the T-Rex Run game using the RGB LED Matrix and the SJ-Two board based on FreeRTOS. This project helped us in having a better understanding of the FreeRTOS scheduler tasks that were used to handle the various components of the game. Working for this project made us practice and implement all the game-related drivers from scratch and this way proved beneficial in resolving issues especially the generation of an obstacle generation and moving it to left with random space between them, integration of various task together as a working game. Even though there weren't many proper datasheets and reliable tutorials for the LED Matrix, some previous semester's project report on the same display helped us gain momentum in the initial stages. Not only did this project help us in understanding the practical possibilities with boards like SJ-Two board but also instilled in us a sense of teamwork and accountability for an individually assigned task that helped the team overall.

References

Acknowledgement

Firstly we would like to thank our Prof.Preetpal Kang for designing such a wonderful course which made us capable of developing our own game. Your consistent feedback and guidance were highly appreciated. Finally, the credit goes to our entire team, T-Rex Run. With full support and cooperation from each other, we were successfully able to complete this project as planned.

References Used

- 32x32 LED Matrix by Adafruit (https://learn.adafruit.com/32x16-32x32-rgb-led-matrix/overview)
- Adafruit Github Library (https://github.com/adafruit/Adafruit-GFX-Library)
- Adafruit Github Library (https://github.com/adafruit/Adafruit-GFX-Library)
- WikiPage by Preetpal Kang (http://socialledge.com/sjsu/index.php/Main_Page)

Appendix

Project Video

T-Rex Run Youtube (https://www.youtube.com/watch?v=UYlVX5-KrjM)

Project Source Code

- GitHub Repository (https://github.com/maheshshindekj/T-Rex-Run-FreeRTOS-Game)