**Sonic Drift**

EE 319K Lab 10 by Kenneth Lin, Kunpeng Qin

*Basic Game Playback*

* Race against Eggman as Sonic or Tails. Steer using a potentiometer sampled by the ADC. Press the “1” button to brake, and “2” button to reset the game. There are two tracks playable in increasing complexity.

*Implementation Summary*

1. The Game Engine is updated by Timer0A interrupt. The Game Engine routine **UpdateGame()** is in **DriftMain.c**. The update performs the following steps:
   1. Check the game state, if the game is in a menu, or in a track.
   2. Update the menu/track.
   3. Reset the graphics buffer and prepare for write to buffer.
   4. Write the game state menu/track graphics into the buffer.
   5. Output the buffer to the LCD screen.
2. State changes are handled by a call to **ChangeState(newState, inputs)** in **DriftMain.c**. State changes are called in the corresponding menus, and race tracks. For instance, in a menu’s **Update()** function, it will check for a input, and then change state.
3. Sound is handled via UART, on a separate microcontroller. The main microcontroller (the one running the Game Engine) sends a message periodically corresponding to the music currently being played. This message send is called in the **Update()** function of each menu or track. On the receiving end, it receives the message, sees if the message indicates the same song. If there is a song change, then the song played is changed. If not, the current song is looped. The message is in the format of 8 frames: 0x12, 0x34, Sound, Sound, Sound, Sound, 0x56, 0x78.
4. The Sound playback on the secondary microcontroller uses a MIDI-like system that can play more than one note at a time with digital signal processing. DSP performs superposition of frequencies, and sound enveloping.
5. The graphics are full resolution double buffered. The double buffer scheme can be found at the end of this file.

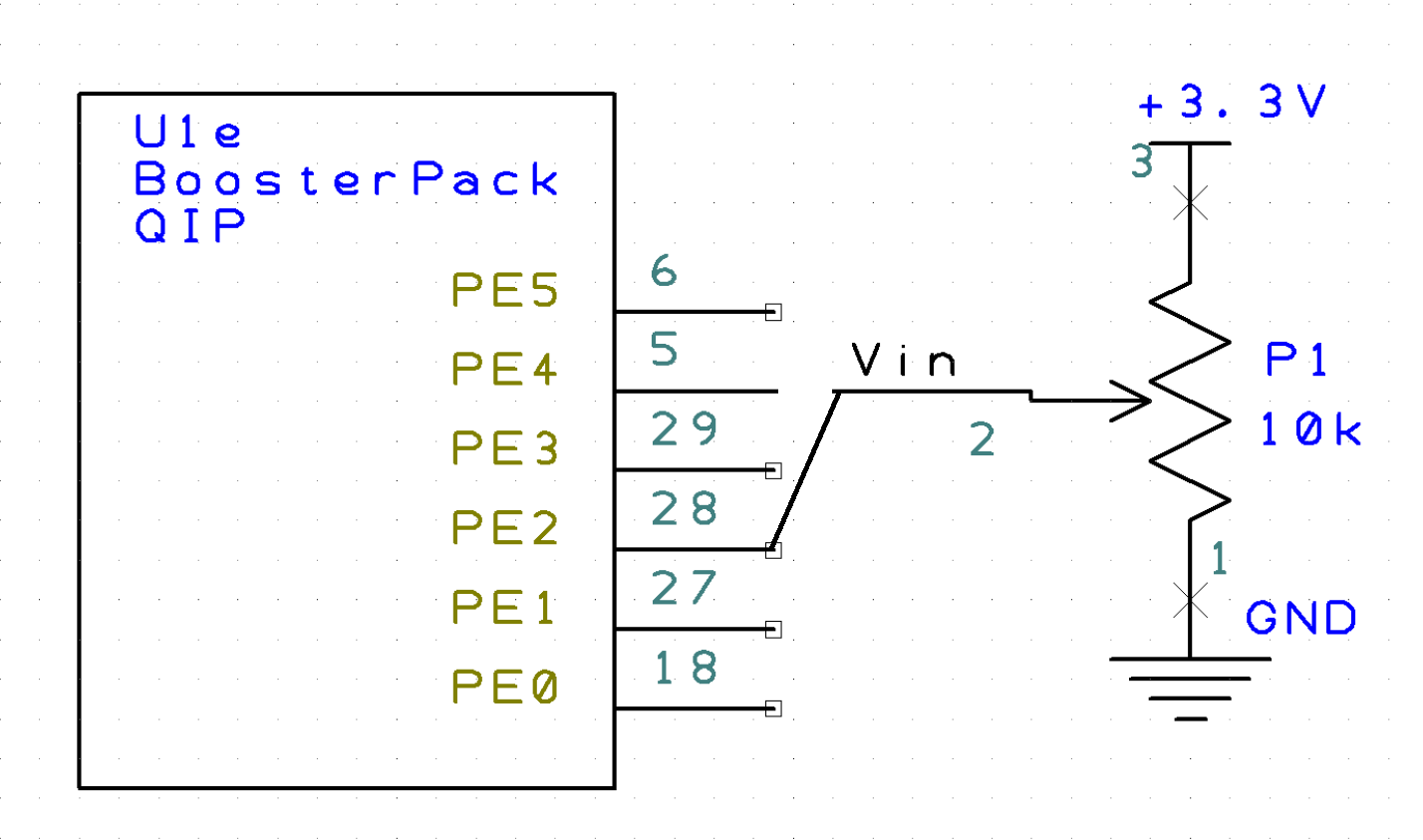
*Criteria Met*

1. **Requirement**: There must be two buttons and a slide pot. **Implementation**: Slide pot for steering, two buttons to choose character, brake, and quit game.
2. **Requirement**: There must be at least three images on the LCD that move. **Implementation**: minimap has three images, car, opponent, road, etc.
3. **Requirement**: There must be sounds appropriate for the game. **Implementation**: title sound, character select sound, Green Hill Zone sound, Scrap Brain Zone sound, and race result sound.
4. **Requirement**: The score should be displayed on the screen. **Implementation**: the time is displayed, as well as ranking on the minimap.
5. **Requirement**: At least two interrupt ISRs. **Implementation**: Timer0A in the DriftMain, and SysTick, Timer0A, and UART receive interrupts in the UART Sound.
6. **Requirement**: The game must be both simple to learn and fun to play. **Implementation**: see the game.

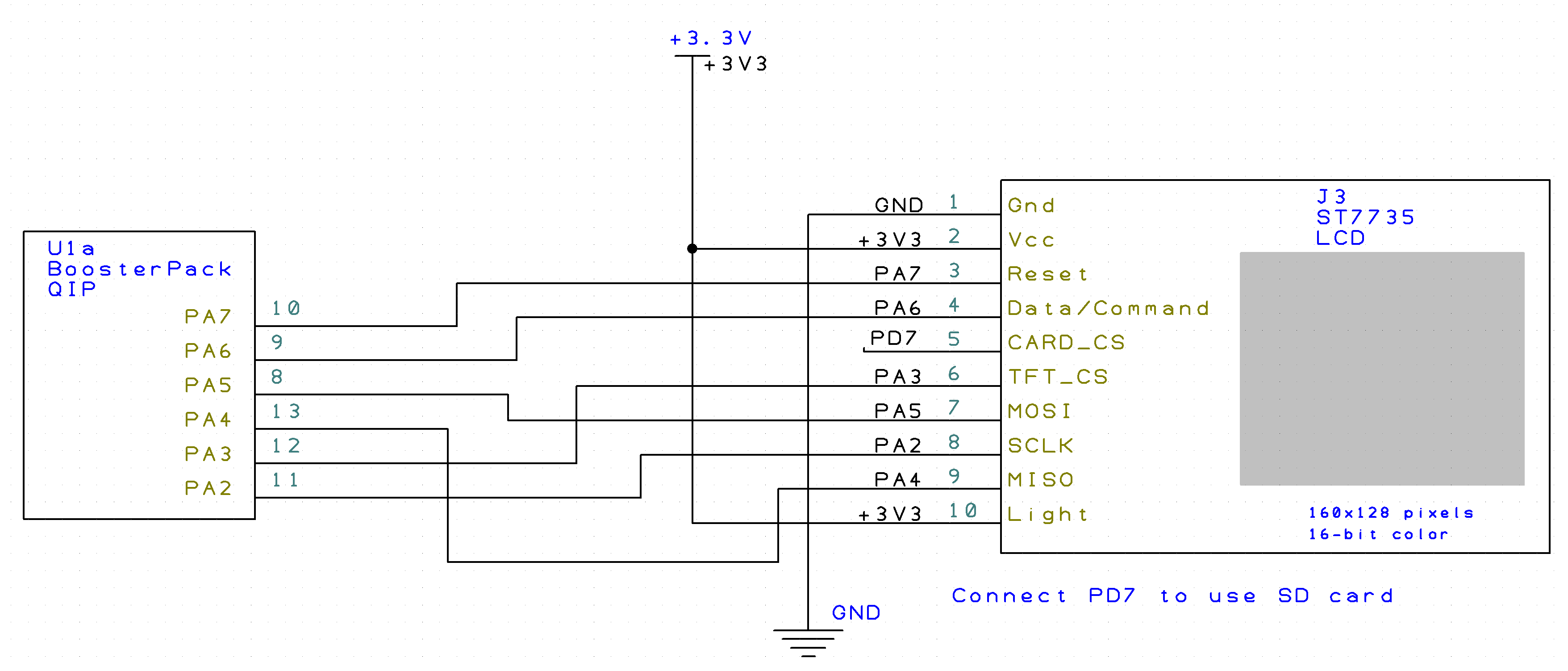
*Extra Credit Met*

1. UART used for communication – Sound system.
2. More than 4 Sound Effects connected to game events – See sound from above.
3. Multiple levels with demonstrable rise in intelligence and difficulty – Green Hill Zone and Scrap Brain Zone.
4. Double buffering – see Graphics Buffer.
5. Layering of Graphics – see Graphics Buffer, RaceCourseX.c, MenuX.c.
6. Digital Signal Processing – See Sound.c in the UART Sound module, DSP is handled with envelope and sound superposition.

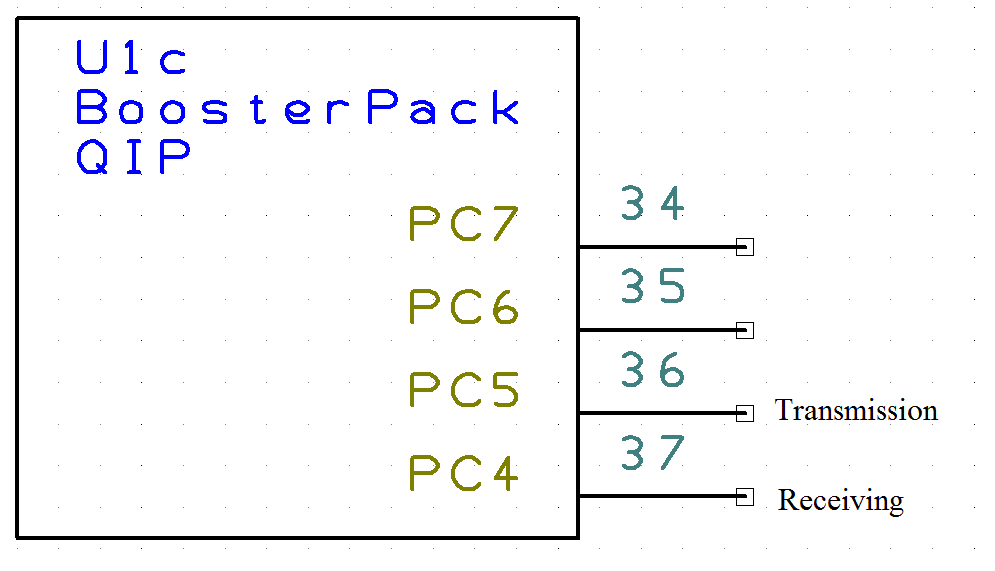
*Hardware Connections*



ADC Input (Slide Pot)

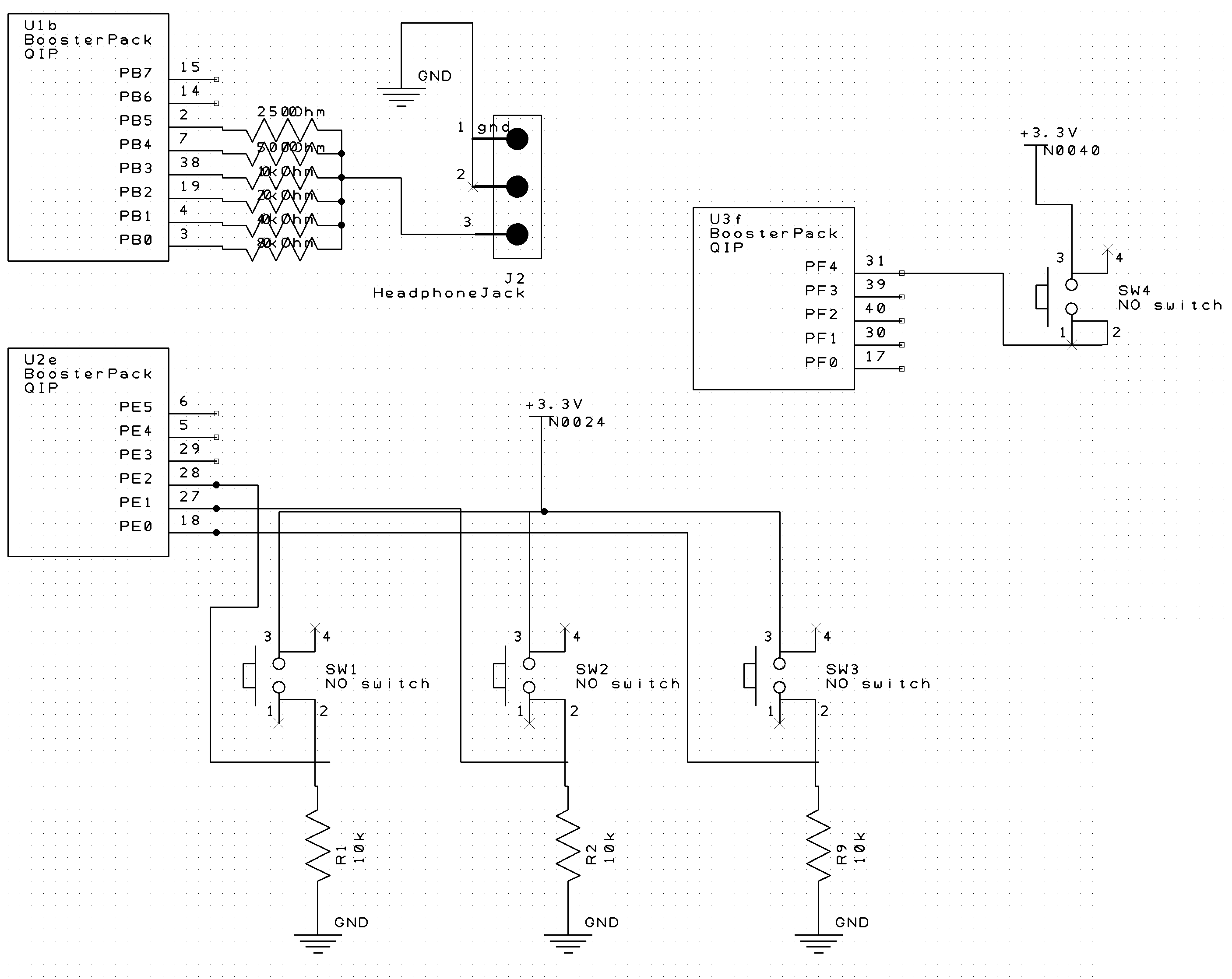


LCD and ST7735 Controller



UART Connection

Note: PC5 is connected on the Primary System, and PC4 is connected on the Secondary System.



DAC

Note: This is connected on the Secondary System.

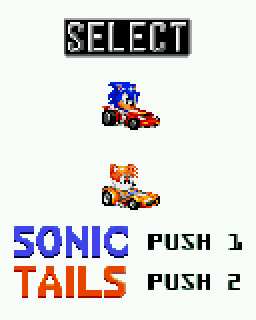
|  |  |
| --- | --- |
| Secondary Microcontroller (On Left) | Primary Microcontroller (On Right)    Top Button is Button 2.  Bottom Button is Button 1. |

Button and Hardware Visualization

*Screenshots*



Title Screen (Menu1.c)



Character Select (Menu2.c)

Note: the Sonic and Tails cars have a 3D rotating animation.



Splash Card Before Level (RaceCourse1.c)



Sonic in Green Hill Zone (RaceCourse1.c)



Race Results (RaceCourse1.c/RaceCourse2.c)



Tails in Scrap Brain Zone (RaceCourse2.c)

**Double Buffer System – Graphics.c**

*Concept*

The screen will be divided into 8px by 8px squares:



Full Screen Image



Top-most, Left-most 8x8 Square

Each 8x8 square has 16 color possibilities, so each pixel can be represented by 4 bits. This means a total of 64\*4 bits = 256 bits = 32 bytes are needed for each square. There are 320 squares on the 128\*160 display, which totals at 10240 bytes for the LCD screen.

Each square will be associated to its own color palette. Color palettes are generated dynamically. The color palette is 16 colors \* 2 bytes per color = 32 bytes, and the total RAM use of the color palette is 10240 bytes.

The total size of the buffer is 20480 bytes, which fits in the RAM easily.

The buffer will be filled with a **DrawPixel(x, y, Color)** function, which does:

1. Looks up the square in which (x, y) should be put.
2. If that square’s color palette is not full, then add the color into the palette.
3. If that square’s color palette is full, then replace Color with the closest color.
4. Fill in the buffer pixel information with the (x, y) and color index.

The buffer will be drawn onto the screen with a **DrawScreen()** function, which is atomic to prevent the buffer to be overwritten too quickly.

1. Draw each square’s pixel information, looking up the color palette in the process.

*Implementation*

The buffer array will be structured:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 |
| 25 | 26 | 27 | … |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  | 64 |

8 by 8 Square in GraphicsBuffer

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | … | n |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 320 |

128 by 160 Display Numbering for n in GraphicsBuffer

|  |  |
| --- | --- |
| GraphicsBuffer[0] – | Color Index for Pixels  2, 1 (1 is least significant). The index takes values 0000, 0001, 0010, 0011, …, 1111. |
|  | Color Index for Pixels  4, 3 |
|  | Color Index for Pixels  6, 4 |
|  | … |
| GraphicsBuffer[32] | Color Index for Pixels  64, 63 |
| GraphicsBuffer[32 \* n] | Color Index for Pixels  2, 1 (1 is least significant). |
|  | … |
| GraphicsBuffer[32 \* n + 31] | Color Index for Pixels  64, 63 |

Data Structure for GraphicsBuffer

|  |  |
| --- | --- |
| ColorPalette[0] – | Color 1 for Square 1 |
|  | Color 2 for Square 1 |
|  | Color 3 for Square 1 |
|  | Color 4 for Square 1 |
|  | … |
| ColorPalette[16 \* n] | Color 1 for Square n |

Data Structure for ColorPalette

**High Speed OutScreen in Graphics.c**

ST7735.c was modified so that Graphics.c could set the addressing window and write in the screen pixel by pixel to reduce the latency of the screen. This allows the LCD to only receive one 11-byte addressing window message and many 2-bytes color messages, which is faster than DrawPixel, which sends both an addressing window message of 11-bytes and color message of 2-bytes.