Embedded Yu-Gi-Oh Game

Yu-Gi-Oh Game Team
Kevin Chow: kevchow Shidong Sun: shidongs Leyang Yu: yly

Abstract — *Yu-Gi-Oh is a famous card game and anime show invented in Japan and is now popular all around the world. Our project is to design and program a Yu-Gi-Oh game on a on Gumstix Verdex Pro board. In this project, the Gumstix Verdex Pro board is used as the main process platform, and an LCD screen is used to show the user interface made with Qt. In order to make this game more engaging, an RFID reader connected with an Arduino Micro is used to read the cards. Some buttons are used to control the game instead of using keyboard input. In the end of the project, the game logic is able to handle one monster card for each side, and a Qt program is developed to make the game table for both players head-to-head. Also, a kernel module is implemented to help read from the button and push the game forward.*

1. Introduction

The project topic is to develop a Yu-Gi-Oh game machine. The game machine should scan the card, load the game rule logic and reflect the battlefield onto the screen. The game in the anime and in real life has various rules, but for our game, we have focused on the most basic aspect of the game: summoning monsters, performing battles, and damage to life points. The general idea is that each monster in the game have two values associated with them: attack points and defense points. In combat, when one player declares an attack with their monster, the strength of that attack is determined by how high the attack points of their monster is. Then, depending of what mode the other player's monster is in (attack mode or defense mode), destruction and/or damage is determined. The different situations encountered in battle are shown in the table below:

Situation	What happens
Monster 0 in attack mode VS Monster 1 in attack mode, Monster 0's attack points are greater than Monster 1's attack points	Monster 1 is destroyed, Player 1 loses life points equal to the difference in attack points of Monster 0 and Monster 1
Monster 0 in attack mode VS Monster 1 in attack mode, Monster 0's attack points are less than Monster 1's attack points	Monster 0 is destroyed, Player 0 loses life points equal to the difference in attack points of Monster 0 and Monster 1
Monster 0 in attack mode VS Monster 1 in attack mode, Monster 0 and Monster 1 have same attack points	Both Monster 0 and Monster 1 are destroyed, no player lose life points
Monster 0 in attack mode VS Monster 1 in defense mode, Monster 0 has higher attack points than Monster 1's defense points	Monster 1 is destroyed, but no life points are lost
Monster 0 in attack mode VS Monster 1 in defense mode, Monster 0 has less attack points than Monster 1's defense points	No monster is destroyed, but Player 0 loses life points equal to the difference in attack points of Monster 0 attack and the defense points of Monster 1

All other situations	No monster is destroyed, no player loses life points

In addition to these rules, in our version of the game, we have decided to have each player's monster only be able to survive for 4 turns before being automatically destroyed. This way, a player won't be able to keep around an extremely powerful monster for too long.
Usually this game can only be played in two ways. One is the physical way, where both people grab their cards and play it face to face. Another way is the software way, where play this game totally on a computer and control with mouse. The former way is not that popular because the game rules and card characters sometimes change over time. It is also hard to judge whether a card is allowed to use since there is a ban list that changes every month. The latter one, however, can avoid the problems above but at the cost of the feeling and excitement of playing with real cards. This can be a disappointment for some players, especially for players who are fans of the anime who wish they were able to have the same experience as the characters in the anime. Nevertheless, the original game in the anime was implemented as a machine that can read card and regulate the rules at the same time. We want to implement a machine that can fulfill the basic functions of the machine that in the anime. This will be a good product, if fully implemented, for Yu-Gi-Oh players.
There are 5 hardware components in this project: an RC522 RFID card reader, a Arduino Micro, a Gumstix, a button set and a LCD screen. The card reader is connected to the Arduino and the Arduino is connected to the Gumstix as a keyboard input. The buttons are connected to the Gumstix via GPIO pins and are read by kernel in the Gumstix. The LCD screen is also connected to the Gumstix, which makes the Gumstix a central unit. We will discuss the connection of the hardware later.
We have successfully implemented reading a card ID from a card reader, reading from buttons, some of the game logic and displaying battlefield onto the screen. The game logic we have gives us the ability to summon monsters, perform attacks, and calculate life points.

2. Design Flow

For the hardware, in the project there are a Gumstix board, a LCD screen, some buttons and a RFID reader connected by an Arduino. For the game, the main process is done in the Game Logic, and the LCD screen is controlled by a Qt module, which can communicate with the Game Logic part through a game file. Also the buttons, which is connected directly to the Gumstix with the GPIO ports, can communicate with the Game Logic by sending sigio when the Game Logic is sleeping and waiting for the signal with a Kernel Module. And the RFID reader can send signal to the Game Logic through a device file. The flow chart is shown in Figure 1.

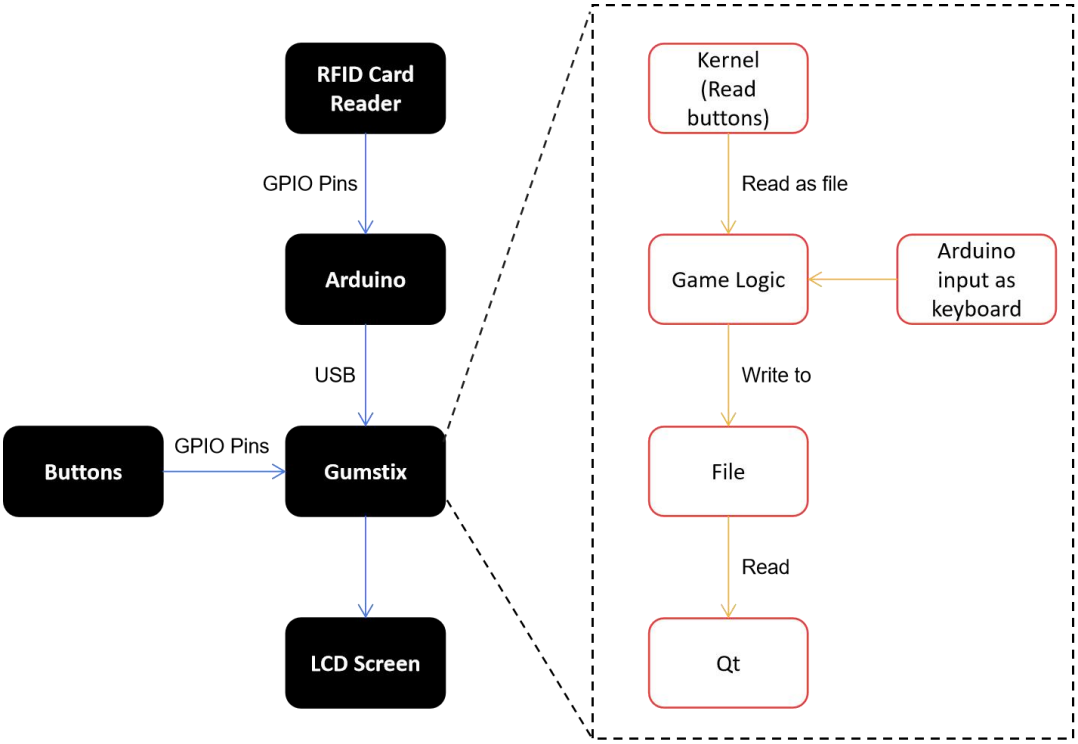

Figure 1. The main flow chart, hardware is listed in the left and software is listed in the right side.

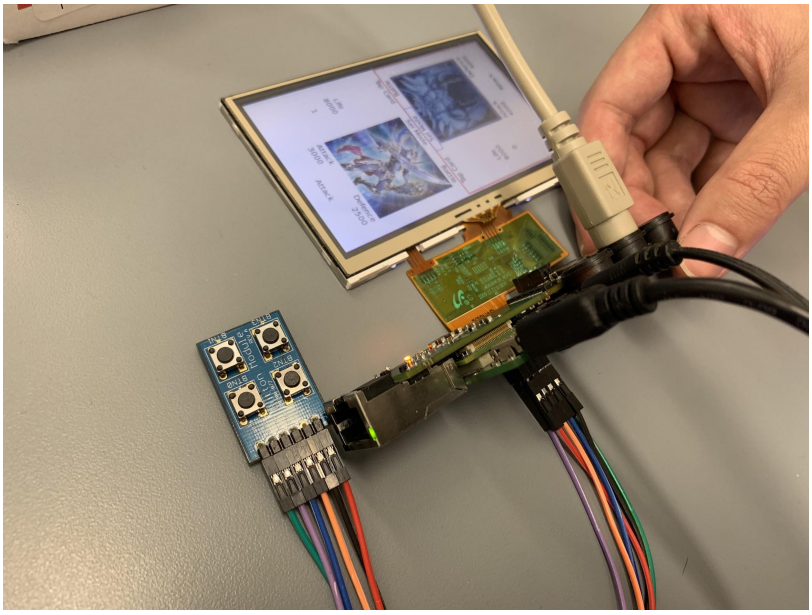

Figure 2. Link of buttons and Gumstix

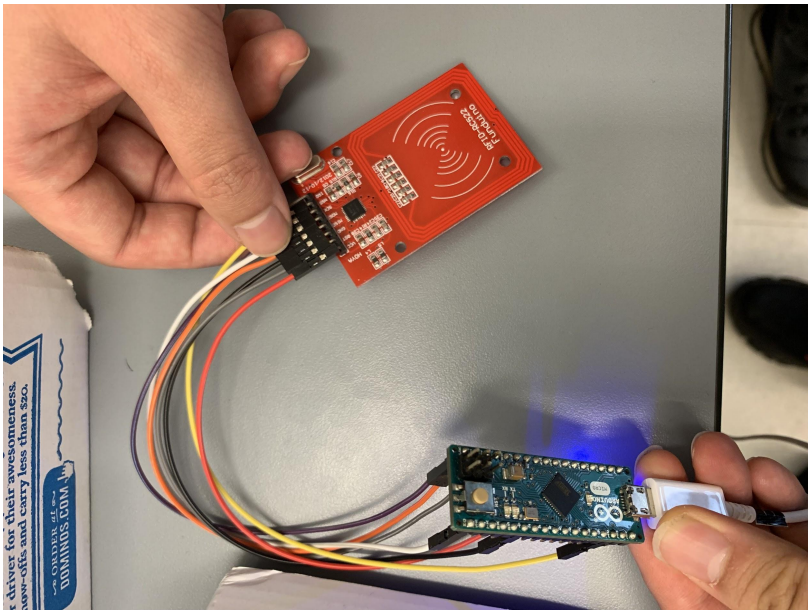

Figure 3. Link of RFID card reader and Arduino

As described above, what we have done can be divided into following parts.

Project Presentation Proposal	Shidong Sun, Kevin Chow, Leyang Yu	Gathering information needed by the project, doing basic design of the diagram and doing presentation.
Kernel Module and buttons	Shidong Sun	Designing a Kernel Module and setting up the way to connect to the Game Logic part.
Game Logic	Shidong Sun	Designing the Game Logic which is the main program of the project and can be used to play the game progress. Also, it is designed to be able to handle the RFID reader and Kernel Module's wake up.
Qt program	Leyang Yu, Kevin Chow	We designed the Qt program so that it can communicate with the Game Logic part, and it can read from the game.dat file to store the card information.
RFID reader and game card design	Kevin Chow, Shidong Sun	Researching on how to use the RFID reader, and writing the code to read from the RFID reader and transmitting information with Game Logic
UI design and LCD operation	Leyang Yu	Designing a User Interface that can show all necessary part of the game, and be able to fit all part of the UI into the LCD screen.
System integration	Shidong Sun, Leyang Yu	Putting all working modules together and make sure all parts are working properly.
Qt-only possibility	Kevin Chow	Attempting to make our game be able to run entirely as a

research		single Qt application instead of needing to rely on another C program; was not able to successfully get this working, so we decided to settle with our C and Qt combination
Project Progress Report	Shidong Sun, Kevin Chow	Wrote the progress report to make sure what has been done and what needs to be done.
Final Presentation and report	Shidong Sun, Kevin Chow, Leyang Yu	Wrote the final report, check if everything is fine, and do documentation for the project.

For the overall contributions of each team member, we considered that it can be divided as below: Shidong Sun 40%, Leyang Yu 30% and Kevin Chow 30%.

3. Project Details

1. **RFID Reader** *(see /rfidkeyboard/*)*
   Our RFID reader is an RC522 RFID module that is able to read 13.56 MHz Mifare cards. Originally, we planned to have our Gumstix interact directly with this RFID reader. However, this turned out to be incredibly difficult for us, as our RFID reader relied on the use of SPI libraries which did not seem to be readily available by default on the Gumstix we were given. Because of this, we resorted to using an Arduino Micro to operate our RFID reader. The Arduino Micro has the ability to emulate a keyboard, which gives us a convenient way of transferring the ID of a RFID card to the Gumstix. The Arduino Micro would read the ID of a scanned card from the RFID reader and then "type" the ID of the scanned card into the Gumstix. Since the Arduino Micro is emulating a keyboard, it is detected by the Gumstix as a keyboard and therefore can be read as a keyboard input via /dev/input/event0.

2. **Kernel module for buttons** *(see /km/*)*
   Unlike the RFID reader, which was controlled by an Arduino, the kernel module was written by ourselves and is a combination of our lab5 and lab3. When a button is pressed, it will enter the interrupt function that already settled for it, which will change the variable 'int button' to the one that pressed, and then sends a signal to wake up the game logic. The game logic will then reads this kernel, which will enter the read function inside the kernel and send the button number to game logic. This number will be reset to 0 right after the sending procedure. This is how we ensure the button to be correctly read by game logic. The design flow chart is as below.

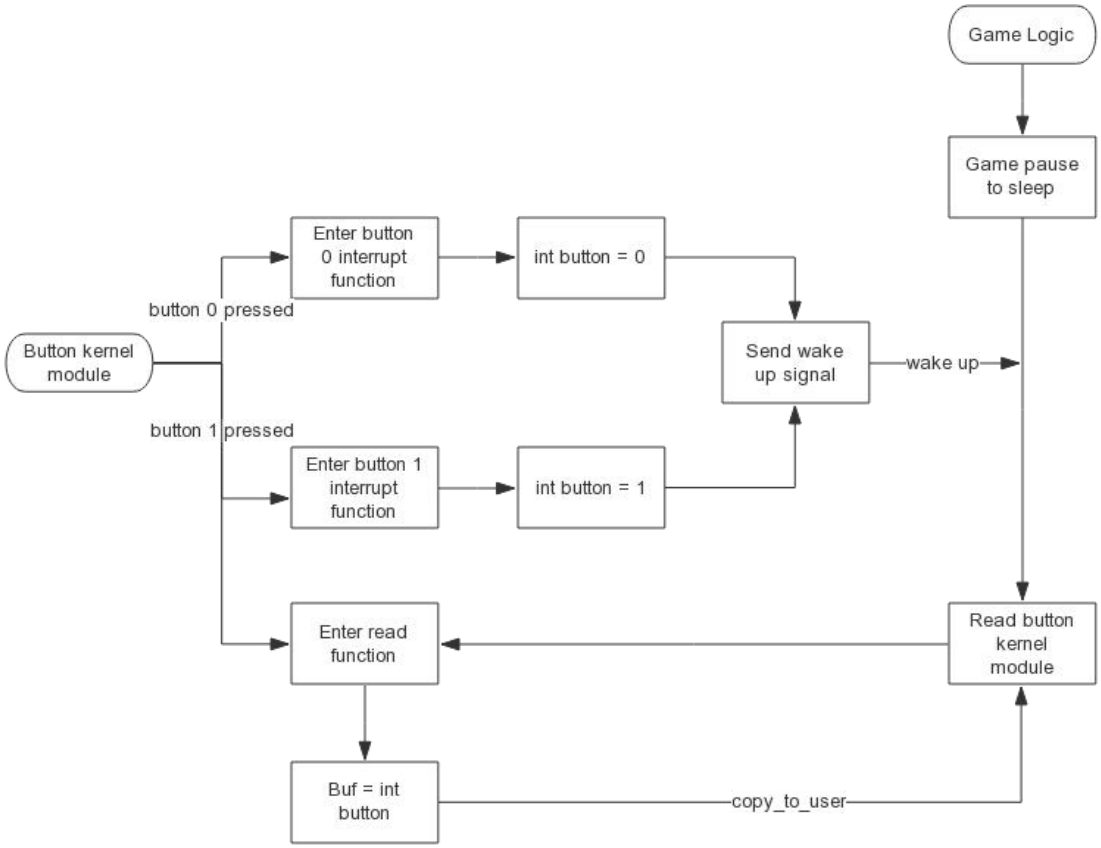

Figure 4. Button kernel module flow chart

3. **Game logic** *(see /ul/*)*

We implemented part of the whole game logic on Gumstix. It is written in C language and communicate with the two kernel modules mentioned above. It reads the 'data.txt' file which stores all the information of cards and create a file called 'game.dat' to tell the Qt what to print. In some stages of the game, it has to wait for the player to input something, that's where 'pause()' is used. The sequence of how the kernel and game logic work ensures that the game logic gets the button correctly (see Figure 1). The game will keep looping between 2 players until the life point of one of them goes down to 0 or below. At last, the game will decide the winner and after 5 seconds, delete the 'game.dat' file and exit. Below is a flow chart that explains how the game works. Every time when it waits for a button, it runs a whole process shown in Figure 4.

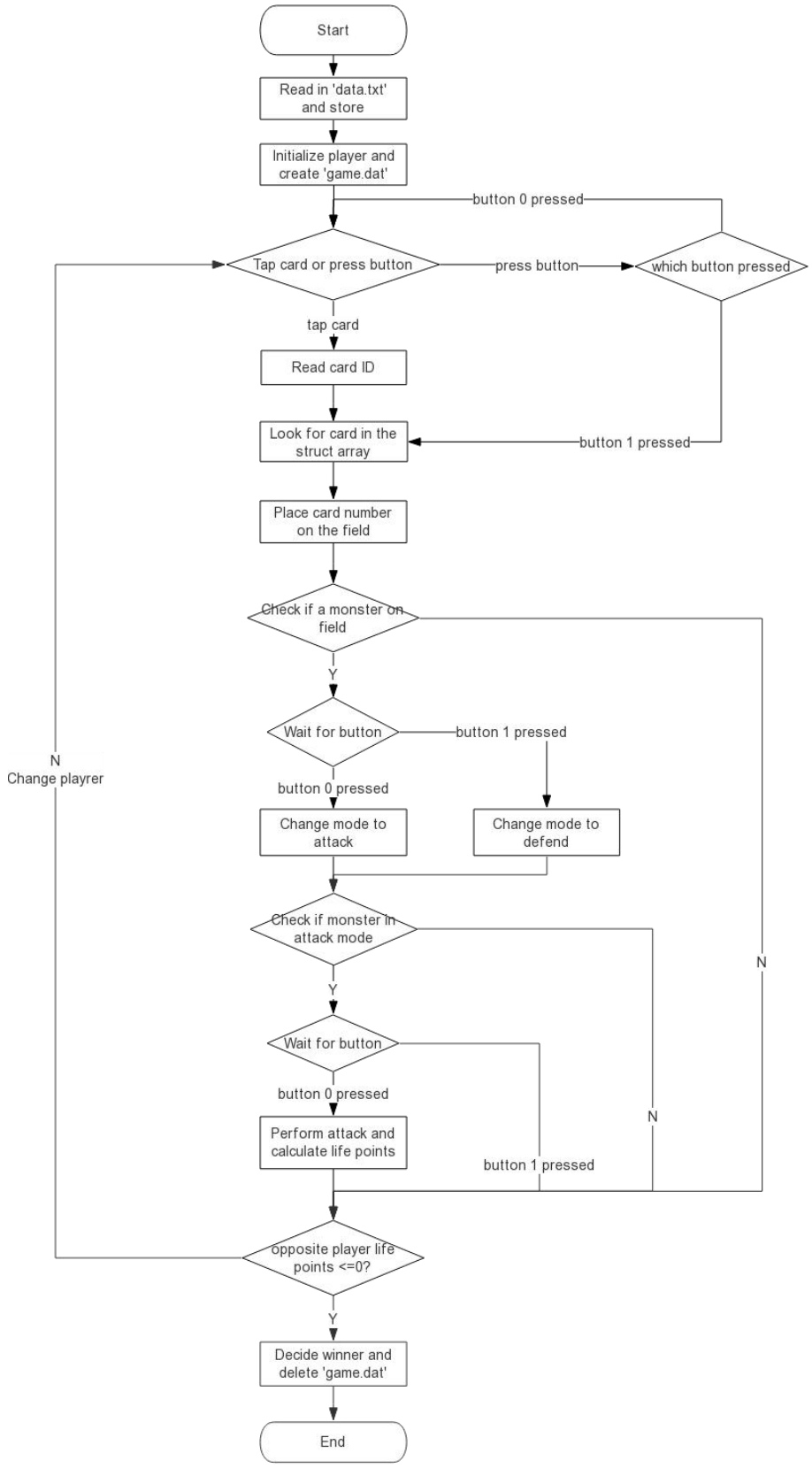

Figure 5. Game logic flow chart

4. Qt module and UI design *(See /Qt/*)*

In general, the Qt module is divided into three files: a main.cpp file which controls all Widgets, a custom.h file which contains all functions and global variables, and a custom.cpp file which is the main part of the Qt module.

In the custom.cpp file, there can also be divided into three main parts, one is to get ready of the printing job, including setting up the variables that will be useful, reading the card data file and store the path to each card to a list, and setting up a timer to update information read from file and update the screen. This part is located in the Custom::Custom() function.

The second part, which is the first half of Custom::paintEvent() function, is to read the game file. If the file is successfully open and the information is correct, it will put all strings into correct types, including converting some strings to qint8 to make it available to calculate as a number, and go into the third part. Otherwise it will assume that the game is not running and it will show Hello Screen.

For the third part, it is the most important part in this code, which is in charge of printing game information on the screen and make the UI looks organized and beautiful. In this part, we are using the Qpainter class to paint all strings and images. The process starts from setting up some important pens to paint information in different color and width. Then we rotate the painter 90 degree around the central point of the screen to make sure that two players can sit in front of each other.

After that, we start painting on the left side. First we need to make sure that the player has lost, and if he has lost, we paint the losing screen on the losing player's side. Otherwise we will paint the status bar and the status that is in by a blue rectangle (if it is this player's turn), and paint the red rectangle (also when it is this player's turn). Then we check if the monster is available, and print the image, attack value, defence value and status of the monster (if the monster is available). And after that, we rotate the painter 180 degrees and paint on the other side.

When painting the Rectangle, we are using painter.drawRect function to paint it. We give the coordinate (x, y) of the top-left point and the length of each side (dx, dy) to the function, and it will paint it out. And when painting the strings, we are using a function called painter.drawText function, and we give the function the coordinate (x, y) of the top-left point and the length of each side (dx, dy) of the bounding box to the function, and the string to the function. Also, we tell the function how to align the text to the bounding box (i.e, AlignCenter).

And in the end, we paint the image to the correct place by using a function called painter.drawPixmap. In order to use this function, we need to tell the function the coordinate (x, y) of the top-left point and the length of each side (dx, dy) of the image. Also, we will need to load the image and convert it into the function. Here since we are using a list to store the path to images, we just read the card number of the monster and extract the filename from the list according to the number we read. Since this module cannot communicate directly to the Game Logic module, it knows whether the game is still running by knowing that the game.dat file still exists. So the Game Logic will remove the game.dat after the game ended, and the Qt module will assume that the game is no longer running, and return to the Hello Screen. The Figure 6 shows the layout of both the hello screen and the battle screen. And Figure 7 to Figure 8 shows different screen displays when the game is in different stages.

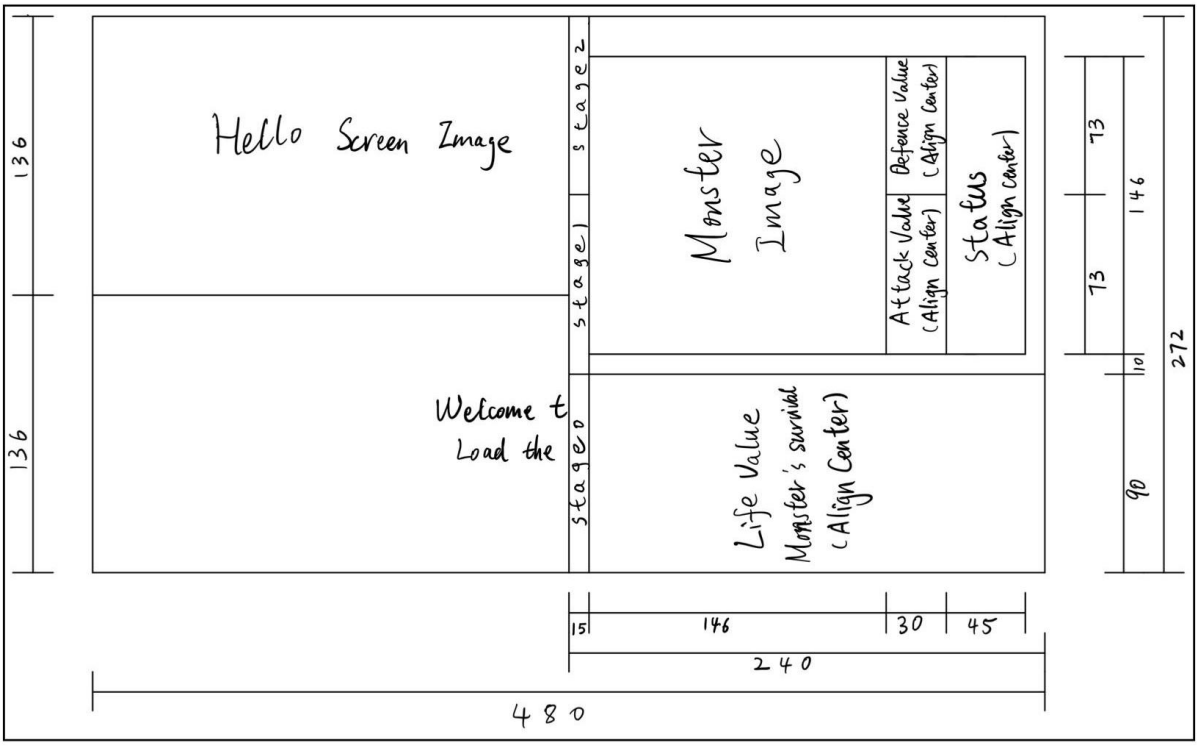

Figure 6. Layout of the LCD screen, left side is the Hello Screen layout, right side is the battle layout. All length are in pixels.

Figure 7. Used blue rectangle to locate the phase of the player, and red rectangle to locate the active user.

Figure 8. This figure shows Hello Screen and Defeated Screen

4. Summary

To summarize, we have successfully implemented the original thinking of our project, which is tap cards to summon monsters and perform basic game fights. We used what we learned from this course to accomplish our project, especially combining lab3 and lab5 to write our button kernel module. However, our project is just a prototype and can only perform a very limited Yu-Gi-Oh game. So it still remains a challenge to implement the whole game. It is the embedded system's advantage to link the physical and cyber world, and there are several good Yu-Gi-Oh games already implemented by software and run on the server. Thus, we believe that it will be a good idea to get the API of those games, employ their game logic and use Gumstix (or other embedded systems) as a linker between RFID card reader and the network. By this method, this prototype we created might be able to be developed into real products that benefit Yu-Gi-Oh players.

References

[1] Qt group. "Qt documentation ". https://doc.qt.io/qt-5/qtgui-module.html
[2] Linux Device Drivers, Third Edition, O'Reilly, 2005