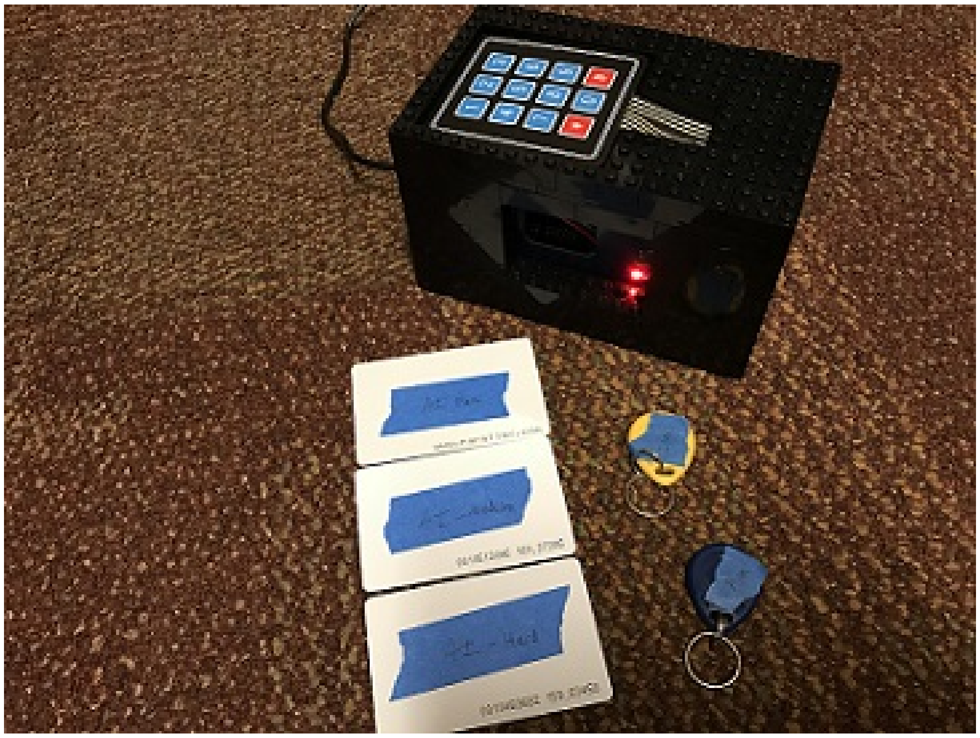
CS 3420 Lab 6 Report

Alexander Strandberg (ads286) and Alan Yan (ayy23)

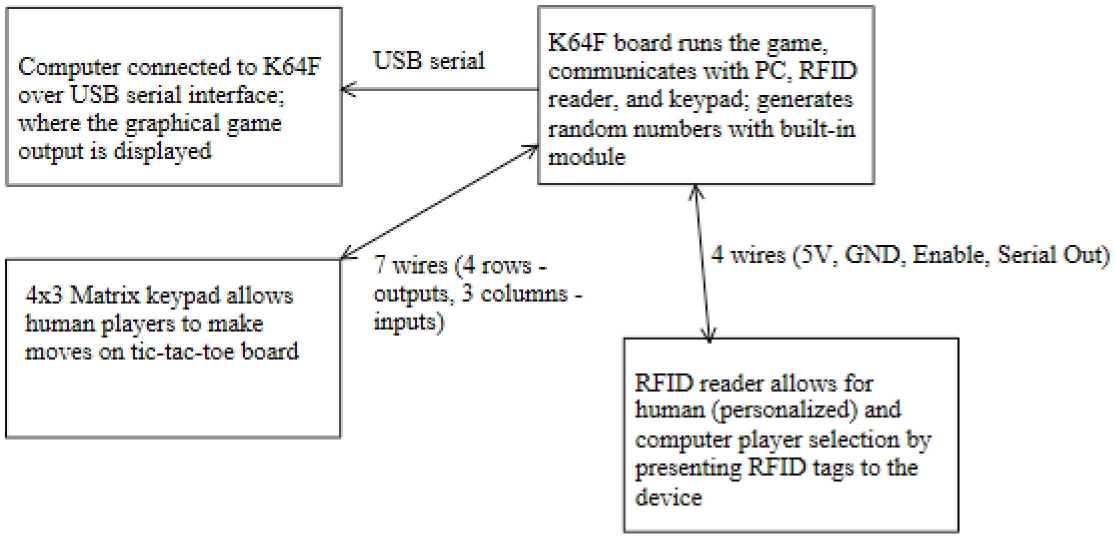
May 14, 2017



1 Overview

Our project allows 1-2 people to play Tic-Tac-Toe against each other or a computer from a command line interface. There are three different computer AI levels (easy, medium, hard) with different strategies to play the game. Players are selected by placing RFID tags near the device and people make their moves using a number keypad.

2 System Diagram


Computer connected to K64F over USB serial interface; where the graphical game output is displayed
USB serial
K64F board runs the game, communicates with PC, RFID reader, and keypad; generates random numbers with built-in module
4x3 Matrix keypad allows human players to make moves on tic-tac-toe board
7 wires (4 rows - outputs, 3 columns - inputs)
4 wires (5V, GND, Enable, Serial Out)
RFID reader allows for human (personalized) and computer player selection by presenting RFID tags to the device

The K64F board runs the code to play tic-tac-toe, outputting the game over a serial connection to a computer. To play the game, the board reads from the RFID module to obtain the ID of a RFID tag, retrieving the names of the players in the process. Each move is made by computer AI or keypad presses for human players.

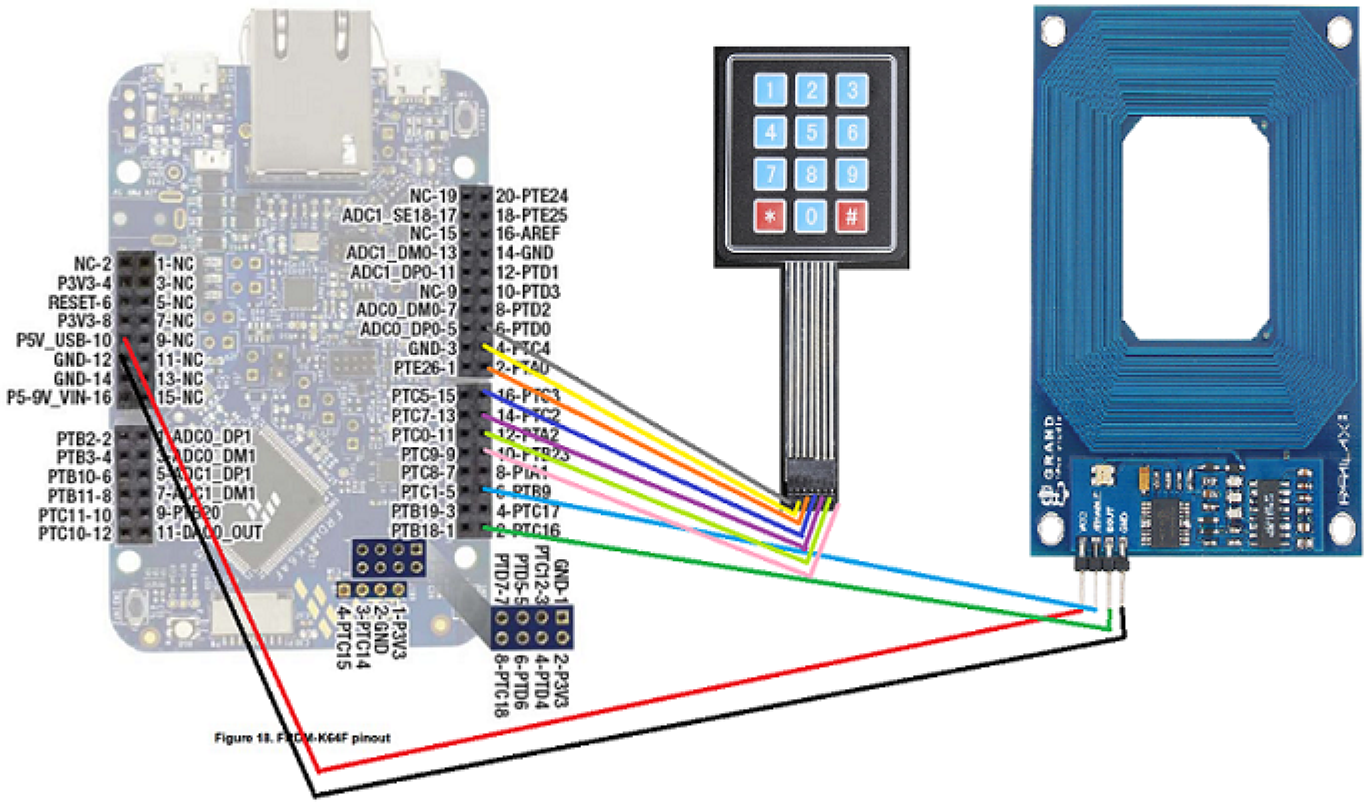
The reading of RFID or keypad data is a blocking operation - since our code runs in the following fashion: wait for user input, process user input, repeat. We therefore did not need to use any interrupts. We did use internal pulldown resistors instead of physical resistors on the keypad connections.

3 Hardware description

3.1 Bill of Materials

- Parallax RFID Reader - Serial Interface:
  $49.99, Product ID: 28140, `https://www.parallax.com/product/28140`
- Membrane 3x4 Matrix Keypad - Adafruit:
  $3.95, Product ID: 419, `https://www.adafruit.com/product/419`
- Premium Female/Female Jumper Wires - 40 x 6"
  $3.95, Product ID: 266, `https://www.adafruit.com/product/266`
- Extra-long break away 0.1" 16-pin strip male header (5 pieces):
  $3.00, Product ID: 400, `https://www.adafruit.com/product/400`

3.2 System Schematic Diagram



Images from: `https://chrischarleson.files.wordpress.com/2014/04/frdm-k64f_annotated_pinouts.jpg`, `https://www.parallax.com/product/28140`, `https://www.adafruit.com/product/419`

4 Detailed software description

We created a **utils.c** file to handle all of the hardware controls - including RFID, keypad, and the random number generator module. We wrote a method to initialize the GPIO - setting inputs/outputs, enabling clocks, and setting baud rates. We wrote methods to read RFID data and process the data. We also wrote a method to detect key presses on the keypad. The RFID module communicates over serial when an enable pin is set LOW. It sends a 12-bit message with the 10-bit RFID tag encoded with ASCII (we store the tag as a character array for simplicity).

The keypad's 4 rows are connected as outputs and the 3 columns are inputs with a pulldown resistor. Our keypad code sets each row pin to a HIGH value in a sequence while polling the column pins to determine if a key is pressed. For example, if row 1 is set HIGH and key 1 is pressed, the first column pin will register a HIGH signal, indicating that the '1' key was pressed.

We needed random number generation for the computer players, so we read the K64F documentation to see how to use the built-in random number generator, which outputs a random 32-bit unsigned integer when polled.

The **main.c** file is where the game code resides. After calling the GPIO setup function, the program runs in a while loop forever. The beginning of the loop asks for RFID tags to be placed on the sensor so that the players can be determined. At the top of the file, a database of RFID tag IDs and corresponding player names is stored. Then, the main game code exists in another while loop. The board waits for a keypad press or calculates the computer player's move. After a move is selected, the new state of the board is sent over serial and displayed on the computer. Additionally, the board checks to see if either player won the game. If a player won, then a message is displayed, and the board waits for RFID tags to determine players for the next game.

We primarily use various arrays for various aspects of the code. The tic-tac-toe board is stored in a 3x3 array. The RFID tag data - IDs and player names are stored in several arrays. These data structures are global variables, and are used throughout the code.

We have three separate AI's, each corresponding to an RFID tag. The Easy AI uses a random number generator and it checks to see if the number that is generated corresponds to a valid location (unchosen) and moves there. The medium AI has if statements checking to see if the opponent/it is about to win and moves there. The Hard AI uses an algorithm found online (referenced below), also using if statements to never allow the opponent to win.

5 Testing

We set and completed milestones to achieve desired features individually. First, we got the 4x3 keypad setup so that we could detect a key press. Next, we got the serial communication working with the computer and RFID reader. Then, we got 2-player tic-tac-toe working. Finally, we implemented computer AI players, starting with easy difficulty, then medium, and finally hard.

Before writing tic-tac-toe code, we simply used the debugger to make sure key presses and serial data was being sent/received properly. Once we began making the tic-tac-toe game, we played the game extensively to make sure the hardware and software worked properly. For example, we handled the case where a player tries to make a move for a spot that is already taken - the game waits until a valid move is picked. Since the game requires hardware components (keypad and RFID inputs), test cases could not be written without disabling code that interfaces with these components.

We determined that our implementation was correct by ensuring that the tic-tac-toe game plays as expected, while allowing for random number generation to make the gameplay different each successive time.

6 Results and Challenges

We achieved a working tic-tac-toe game which integrates hardware and software elements. A major challenge was getting the board's random number generator working, since a number of registers must be setup correctly for valid random numbers to be generated. The documentation did not clearly explain that since there are two different ways to enable the module, the corresponding set of registers must be used (we were using the incorrect registers located in the **fsl_device_registers.h** file.

The most complicated part of our design was the code to interface with the RFID reader, since we had to be able to identify RFID tags by ID and associate the tag with the corresponding player. Our final implementation is different than our proposal, since we could not successfully control a computer by emulating a USB keyboard. We did not want to use Python or another means to control the computer based on serial data sent from the board, so we instead pivoted to a different project idea using the same hardware. The only thing we would do differently next time is ensure that we could control the computer as a keyboard device before writing any additional code (we unsuccessfully tried to implement this functionality after writing RFID and keypad code).

Another challenge was developing the Hard AI algorithm. We initially wanted to develop it like we did Connect Four in CS 2110 with minimax recursively. However, that would force us to create new structs and data structures which we thought would be too time-consuming and unnecessary given that there are only 9 potential places to move at the start of each game of Tic-Tac-Toe. Thus, we decided on just using an algorithm we found online that only used if statements to ensure that the opponent never wins.

7 Work Distribution

To complete the lab, we met up to discuss how we would approach the lab and wrote the code together. We took turns writing different parts of the code, using a pair programming approach. We sought assistance from TAs on Piazza to help with properly connecting the hardware to the board. Our testing methodology is described above. We each thought of potential issues in our code, while explaining potential fixes to the other. Once our code was working, we met again to write the documentation, and continued adding our contributions remotely.

We both found a way to equally contribute to the assignment, with one of us (Alex) specializing in the hardware control, while the other (Alan) focused on the algorithms and data structures.

8 References, software reuse

`https://community.nxp.com/docs/DOC-102540`
Sample barebones code to interface with GPIO, set pullup resistors, and read/send serial data

`https://frankanya.wordpress.com/2013/04/16/non-recursive-tic-tac-toe-ai-algorithm/`
Procedure for determining Hard AI's next move

`http://en.cppreference.com/w/cpp/language/ascii`
ASCII chart - Converting non-character data into characters to send over serial

K64 Sub-Family Reference Manual (on CMS) used to understand how to set up and use external hardware on the board by controlling/setting various registers