SE3910 Lab 3: The Game of Anticipation

Due: April 2, 2014 (Schilling)

Due: Week 4, Monday, 11pm (Yoder)

1. Introduction

Games are fun. Games allow us to compete with others and practice humility when we win and congratulate our opponents when we face the agony of defeat. Many software engineers play games. Many software engineers are interested in games. But game design is not easy. In fact, the field itself is extremely competitive. This makes writing games even more fun!

In this lab, working with a partner, you are going to write a game for the Beaglebone. The game is the game of anticipation.

2. Lab Objectives

- Construct software which uses multiple POSIX threads.
- Construct simple embedded software which controls GPIO devices.
- Use GPIO interrupts to manage a given software package.
- Design your own hardware interface.

3. Laboratory Equipment Needed

- One Beaglebone Black embedded systems board
- One 10W 5 V power supply
- 2 ethernet networking cables
- 1 Beaglebone protoboard

4. Prelab: Watching the video

Before coming to lab, it is very important that you watch the video for the lab. The video explains how GPIO works on the beaglebone as well as provides an overview of the lab.

You also will most likely want to watch the tutorial on using gdb, which is the gnu debugger. The GNU debugger helps to debug programs.

5. Prelab: Designing our hardware

This lab requires you to design a simple hardware system. In essence, your embedded software will be performing the role for the software shown in Figure 1. Before you can build this software, you will need to design your hardware. Start by sketching a schematic showing how you will wire the two switches and 2 LEDS. Be specific, including which pins you will connect into in order to make your circuit work. Once you have sketched the schematic, using the Beaglebone prototyping kit, draw how you would wire your circuit.

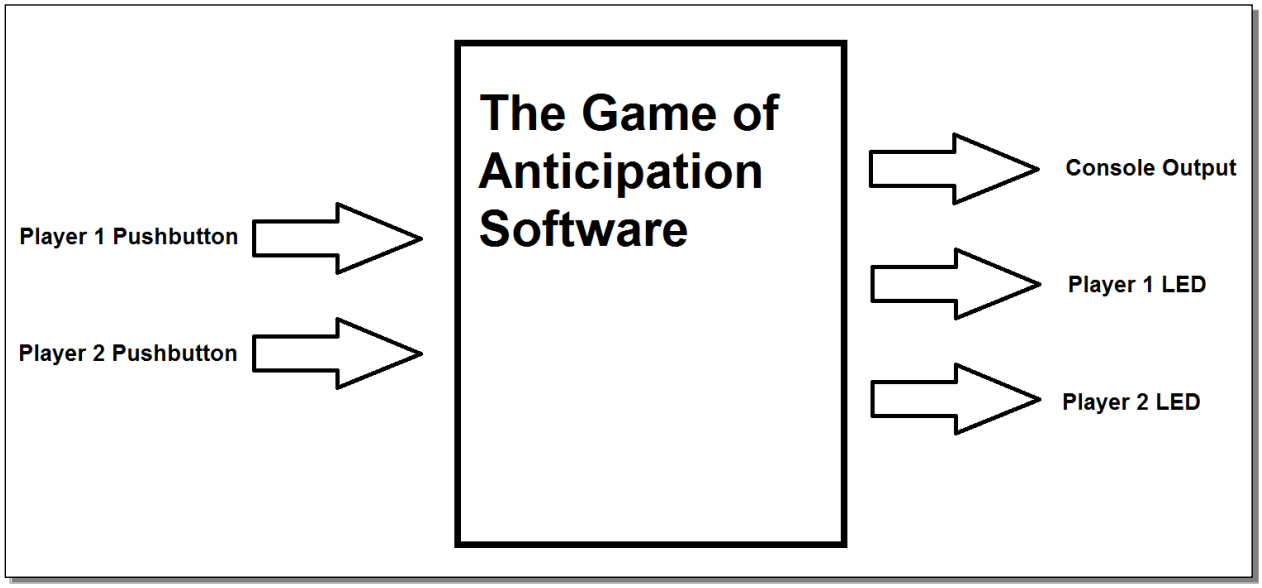

Figure 1: System Block Diagram

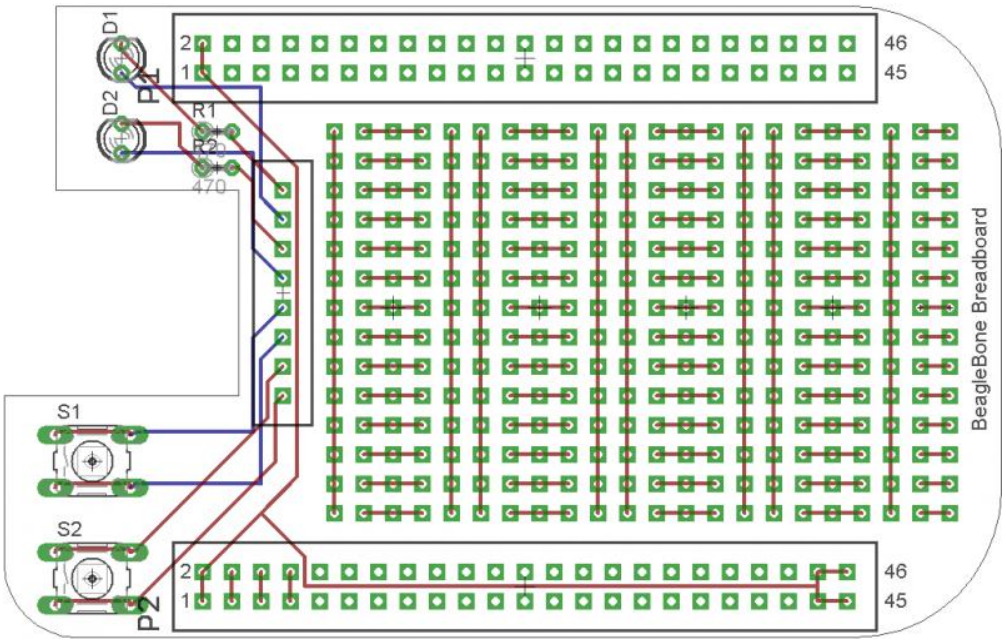

Figure 2: Prototype Cape hardware layout.

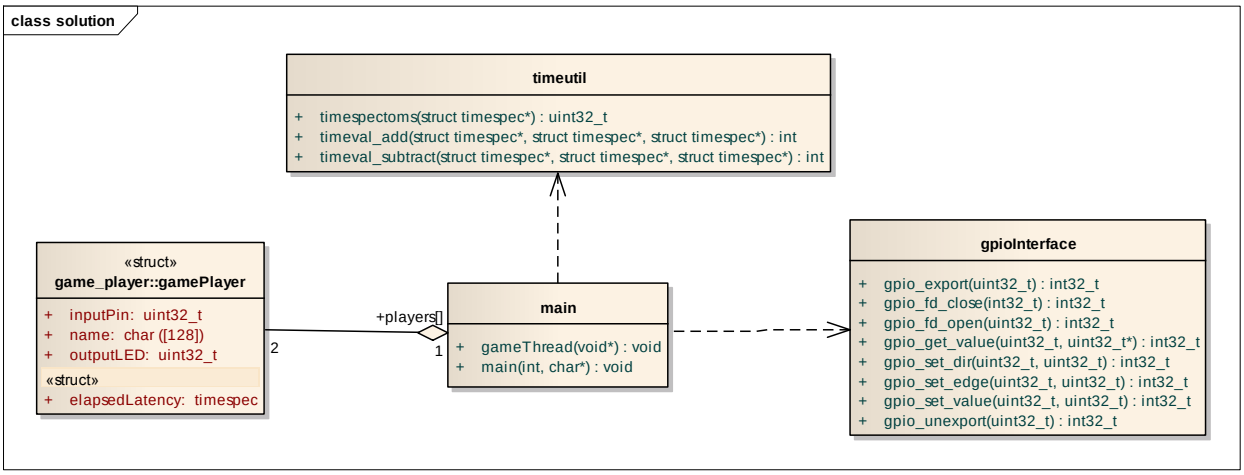

Figure 3: Basic UML for the project.

6. Designing our software

The first thing you will need to do is to design your software. To make this easier, you are being provided with a set of routines. The first set of routines manages time. The time utility uses the struct timespec stores time in seconds and nanoseconds. You can view its usage by typing "man clock_gettime" from the unix shell. The utility provides you with mechanisms to determine the difference between two timestamps.

You also are provided with a library that allows you to manipulate the gpio interface. This library was described in the video and it contains the necessary material to allow you to access gpio in an easy fashion.

You will need to write two routines. The first routine, main, will manage the entire program, spawning two child threads[1] and waiting for them to complete. The second method will actually run the game for a single player. The algorithms for these are presented in Appendix A.

On the course webpage, there is a tar file which contains starter code as well as makefile for your project.

7. Deliverables / Submission

Each person will be responsible for submitting one report with the following contents:

1. Introduction -> What are you trying to accomplish with this lab? This section shall be written IN YOUR OWN WORDS. DO NOT copy directly from the assignment.
2. Screen capture of a clean build of your software
   a. Include a screen capture of your software building without warnings. To do this, do a "make clean" and then a "make all" invocation of the makefile.

[1] Note: Review your CS3844 Operating Systems materials for details on multithreading. Sample code is available at http://www.walterschilling.us/msoe/winter20132014cs3844/Lectures/Lecture9PThreadExample1.zip

3. Wiring diagram
   a. Include the wiring diagram for your hardware. This can be a scan or a picture of your hardcopy inserted into the write-up.
4. Things gone right / Things gone wrong -> This section shall discuss the things which went correctly with this experiment as well as the things which posed problems during this lab.
5. Conclusions -> What have you learned with this experience? What improvements can be made in this experience in the future?

This material should be submitted as a single pdf file.

In addition, you shall submit a tar.gz file of your complete beaglebone code. Your program must cleanly compile simply by typing "make –f makefile.bb all" without any compilation warnings.

If you have any questions, consult your instructor.

8. Appendix A: Algorithms

12.1 Main Function

1. Declare all variables
2. Check to see that the arguments include both player names. If not, print out the usage to the console and exit the program.
3. Print a welcome message, including the 2 players names and which switches they will use.
4. Initialize the gameplayer structures with the player names and IO pins, and set the time to be 0.
5. Initialize and spawn two threads, passing along the proper player structure as an argument to each of the two threads.
6. Wait for the threads to complete (hint: man pthread_join)
7. Print that the game is over.
8. Print the total latency for each player.
9. Determine which player had less latency and announce him as the winner.
10. Tell everyone goodbye.

12.2 gameThread

1. Declare and initialize all variables
2. Setup the input port on the appropriate pins. The input port should be setup as an interrupt with an appropriate rising or falling edge.
3. Setup the output port to drive the LED.
4. Loop for the number of rounds in the game.
    a. Wait a random amount of time.
    b. Turn on the LED.
    c. Print that the Led is turned on.
    d. Grab the current time and store it in the starting time structure.
    e. Configure an interrupt to fire on the input port.
    f. Go into a loop of up to 10 times
        i. Block waiting up to 1 second for the user to press the appropriate button.
        ii. Determine if a failure occurred on the input. If it did, abort the program.
        iii. Determine if a timeout occurred. If it did, print to the console a reminder that we are waiting for a given person to press their button and increment a counter such that we will loop no more than 10 times.
        iv. If the interrupt was the push button being pressed, read the data from the pin and abort out of the inner loop of 10.
    g. Get the ending clock time.
    h. Determine the elapsed time
    i. Add the time to the total latency for the player.
    j. Print out the player's response time in ms.
    k. Turn the LED off.
    l. Wait 2 seconds and go back to the top of the loop.
5. Close out the pins by unexporting them.