Larry Sun

Ada Wong

EC450 A1

April 29, 2015

**Whack-A-Mole**

**Goal & Design**

The purpose of our project is to recreate the arcade game “Whack-a-mole” on a smaller scale. Typically, there are somewhere around ten moles for the game, but we toned it down to four.

**How to Play**

There are four buttons corresponding to four LEDs on the board. The premise is to correctly hit the buttons with lit LEDs, or “moles,” and to not miss. Pressing the button with the LED lit up counts as a correct hit, while pressing the button with a non-lit LED counts as a miss. It also counts as a miss if the LED turns off by itself, and you did not hit the button corresponding to it.

The player begins by pressing the play button as prompted by the LCD panel. The LEDs begin to light up in a random order and the game begins. Each time the player correctly hits a certain number of “moles,” the level of the game increases, meaning that the LEDs light up more frequently and for a shorter time. The number of points the player receives increases with the level of the game. The player has a certain number of misses allowed. Once the player misses the maximum allowed of times, the game is over and the final score is display, followed by the high score. Then the game restarts.

**Description of the implementation**

**Components:**

* 5 Pushbuttons
* 4 LEDs
* 1 Speaker
* 1 16x2 [LCD Panel](https://www.sparkfun.com/products/255)
* 1 [SparkFun Serial Enabled LCD Backpack](https://www.sparkfun.com/products/258)
* 4 100 ohm resistors
* 5 200 ohm resistors
* 1 20 ohm resistor
* ~12 female-male wires
* ?? Breadboard wires

**Random mole generation:**

The random generator used Timer A0. The way the random number generated was by using the VLO to source ACLK. The ACLK was then used as an input to the Timer A0. The Timer A0 is on continuous mode, constantly checking the capture, and by doing this at various times of the VLO clock, the result is that the result is a fairly random number.

**Sound production:**

The sound producer also used Timer A0. It was set up similarly to the many examples from class. The reason why this doesn’t intersect and cause issues with the random generator is because the random generator is a function that just captures the Timer A0. If one was being used, the other wouldn’t be because the random function was called in the watchdog interrupt while the sound was handled in the Timer A0 interrupt. The MSP430 takes care of the prioritizing.

**LCD Panel:**

The LCD Panel used Timer A1. Our LCD Panel came with a microchip (the LCD Backpack) that allowed us to use serial communication. That cut down the number of pins to simply three pins: Vcc, Rx, and Gnd. In order to control the settings and behavior of the LCD panel, the microchip had a defined set of special numbers it deemed as commands. For example, to clear the screen, the MSP430 needed to first send a byte “0xFE” to tell the microchip to take the next byte as a command, not an ASCII character. Then the byte “0x01” is sent to clear the screen. The reason why we used Timer A1 was because there were issues with UART being unable to handle these special numbers necessary for the microchip. We also chose Timer A1 over Timer A0 because of priority level. It seemed that since the watchdog interrupt and a higher priority than the Timer A0, it was being prioritized before the LCD panel could finish.

**Input buttons and LEDs:**

The buttons and LEDs were controlled by the watchdog timer. The WDT used the SMCLK as the clock source calibrated at 8MHz with a divisor of 32768. This meant that the WDT interrupt handler was called every ~4.1ms. The WDT handler also acted as the debouncer for our input buttons.

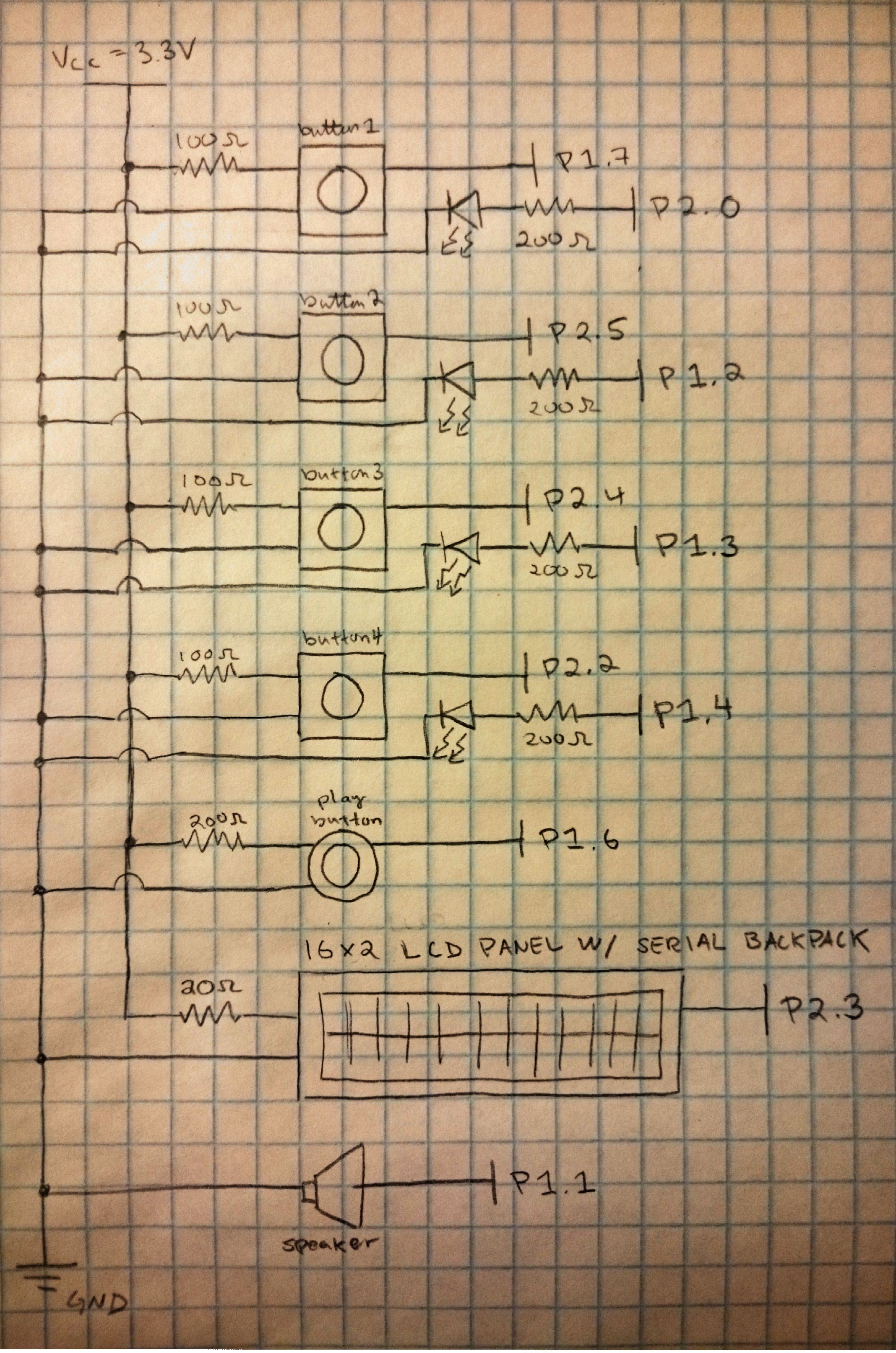
There were two global variables used to keep track of the button presses. The variable “hits” stored the number of times the player correctly pressed a button when the button LED, or “mole,” was lit up. The variable “misses” stored the number of times the player incorrectly pressed a button when the LED was not on.

A global variable “stage” kept track of the current level the player reached. After a certain numbers of “hits,” the stage would be incremented. Each time the stage incremented, the LEDs lit up faster and they stayed lit for a shorter duration.

**Flash memory high score:**

The high score was stored in flash memory. Each time the player achieved a new high score, the new high score replaced the old one in flash memory.

**Schematics**



**Assessment of success of the project**

Overall, the project was a success. We wanted a fun and operational Whack-a-mole game, and we achieved that.

**Improvements**

One of the things we could do to improve our game is to implement more moles. It would prove to be more of a challenge for users.

A simple thing we could do is to have the speaker play a tune when the game started and/or ended to further give the user feedback. We could also create more tones or short sound effects for some different states such as a game over sound, opening screen sound, etc.

A rather challenging improvement would be to implement a two-player mode. Perhaps one player could program the mole generation for the other player or both players have their own moles and try to beat one other’s score.

**Team Member Contributions**

Larry implemented the serial communication of sending data to the LCD Panel. He also made the buttons operational and did the logic behind the game.

Ada implemented the sound handler, sound production, and random generator. She also configured the hardware and implemented flash memory storage for the high score.