CSI 370 Computer Architecture
Esplora Puzzle Project

Tyler Chasse
Champlain College
tyler.chasse@mymail.champlain.edu

What:

I used the Arduino Esplora to create a puzzle game. These puzzles utilize different sensors on the Esplora to create a series of puzzles for the user to solve. The first puzzle uses the joystick to act similar to a safe dial. The user must put the joystick in the correct position and confirm that position three times in order to move to the next puzzle. An LED signifies how close they are to the correct position with the LED starting red and getting closer to green as they get closer. The second puzzle is a very similar concept but with the accelerometer. The user must find three different positions within the accelerometer range to move to the next puzzle. The last puzzle is a memory game as the LED flashes six colors and the user must input those colors in order through their corresponding buttons.

Why:

I wanted to do something with hardware because I haven't had much experience with it and I thought it would be a beneficial experience to have. I also remember seeing hardware experience as something mentioned on a job description that I applied for so that was another reason I thought it would be a good idea to get some more experience with hardware. For this specific idea, I decided upon it after looking online for some Arduino project inspiration and saw the title of a safecracking project that I thought could be fun and interesting. I've always had interest in puzzles and things of that nature so it was something that caught my attention and I'm glad it did because I enjoyed the new experience.

How:

To do this I used the Arduino Esplora and various sensors on it, specifically ones best suited for user input such as the joystick, accelerometer, and buttons. I used the LED and buzzer to relay information that the user might need to inform them on their progress and guide them through the puzzles. For coding, I used the Arduino IDE and implemented my ideas in C++ and then replaced some code with a function I implemented using the Esplora instruction set.

Challenges:

I haven't had much experience with hardware like this so learning how to navigate that was a challenge towards the beginning. Learning how to utilize the different sensors input and output was something I had to experiment with and learn to manage. Using assembly was the biggest challenge as I had to find something to implement with the Arduino instruction set that didn't include Esplora specific instructions as assembly information on those was much more difficult to find.

Solutions:

Sensor Input and Output:

I used the Serial Monitor to observe my inputs and to find the max values to generate a random number between. This also helped me tune values to find realistic ranges to accept as a correct input to find a fair difficulty.

```
xPos: 438       YPos: 512       Joystick X: -49 Y: -348 Button: 1       Attempt:1
xPos: 438       YPos: 512       Joystick X: -41 Y: -343 Button: 1       Attempt:1
xPos: 438       YPos: 512       Joystick X: -32 Y: -338 Button: 1       Attempt:1
xPos: 438       YPos: 512       Joystick X: -19 Y: -332 Button: 1       Attempt:1
xPos: 438       YPos: 512       Joystick X: -12 Y: -327 Button: 1       Attempt:1
```

Figure 1: Joystick Output Information

Assembly:

I had trouble finding information on assembly implementation for Esplora specific instructions so I began to look for Arduino Uno exapmles that I could I apply to general function within my code. I eventually found a series on Youtube that provided examples of assembly code for the Arduino Uno. From these videos I learned how to create and interact with a .S file that could hold assembly language with the Intel syntax. This made things more organized and easier for me to understand as it was more familiar to me. Within one specific video, I saw a delay function that I could work with as I was using the C++ delay function already to determine how long my lights and sounds should play. The function within the video uses an inner and outer loop to create a delay with an arbitrary and hard-coded length. To make the delay predictable, I made the inner loop always take just about ten milliseconds (I'll explain my process for this later) and then passed a value to the function to act as the outer loop counter, meaning if I passed the value 100 to the function in my C++ code, the delay would take just about one second (1020 ms) as the inner loop would run 100 times. In figure 3 you can see how length is passed to R21 within the Puzzle.ino C++ code so that it can be used when asmDelay() is called which is defined as an extern C function.

Puzzle.ino Puzzle.S

```
1   #define __SFR_OFFSET 0x00
2   #include "avr/io.h"
3
4   .global asmDelay
5
6   .equ delayVal, 40000        ; inner loop counter (16 bit value)
7
8   asmDelay:                   ; function name
9     outerLoop:
10      LDI R30, lo8(delayVal)  ; first 8 bits of inner loop counter
11      LDI R31, hi8(delayVal)  ; second 8 bits of inner loop counter
12    innerLoop:
13      SBIW R30, 1             ; sub 1 from inner loop counter
14      BRNE innerLoop          ; if counter != 0, branch to inner loop
15      SUBI R21, 1             ; sub 1 from outer loop counter passed to R21
16      BRNE outerLoop          ; if counter != 0, branch to outer loop
17    RET
```

Figure 2: Puzzle.S

```
void myDelay(uint8_t length) {
  asm volatile (
      "mov R21, %0\n"           // move length into R21 (%0 is a placeholder for input)
      :
      : "r" (length)            // signify length variable to be used as input
    );
  asmDelay();                   // call asm function that will access length via R21
}
```

Figure 3: Calling asmDelay() within Puzzle.ino

Explanations:

Puzzle 1:

The first puzzle uses random X and Y coordinates for the user to align the joystick with. The LED is red when furthest away, and then orange, and then yellow as you get closer. Once the light is green the joystick is in the correct position and the user can use switch (button) one to submit it. This will provide a high pitched beep signifying success. A low pitched beep signifies an incorrect guess. After this sequence has been completed three times the LED goes blue and is accompanied by a longer and higher pitched beep.

Puzzle 2:

The second puzzle is very similar to the first but uses the accelerometer for input instead of the joystick. Only the X and Y values are used as the Z axis made input less intuitive. The red, orange, yellow, green, and blue lights all hold the same meaning and the buzzer sounds and pitches also follow the same rules of puzzle 1.

Puzzle 3:

The third puzzle is a memory puzzle as it tests the users ability to remember a sequence of six colors, with four different possibilities. After puzzle 2 is completed six colors will flash on the LED and the user must pass the sequence back through the buttons corresponding to the correct color. If an incorrect color is entered, a lower pitched beep plays and the sequence shows again. If a correct color is entered a higher pitched beep plays. If all six colors are entered correctly then the next sequence begins. Once three sequences have been correctly entered a longer and higher pitched beep plays. After all of the puzzles are complete a brief light show plays and then everything resets to be played again.

Delay Timing:

To try to determine how long my inner loop within my assembly delay function would take I looked for the calculation to find how long one clock cycle would take. What I found was to divide 1 by the hertz of the clock speed of the CPU. I found that the Esplora has a 16 MHz clock speed so I divided 1 by 16 million and then multiplied that by 1000 to go from seconds to milliseconds. So one clock cycle for the Esplora takes about 0.0000625 ms. Next I needed to know how many clock cycles each instruction used within the loop. I found that both the SBIW subtraction instruction and the BRNE branch instruction take two clock cycles. Therefore each run through the inner loop takes four clock cycles, or about 0.00025 ms. Knowing this I set the counter for the inner loop to 40000 so that it would always take 10 ms. To scale up I simply pass the function how many times I want to run the outer loop. So if I want a 1000 ms delay, I pass 100 as the counter for the outer loop, which would run the inner loop 100 times, providing a 1000 ms delay. In reality, when using the millis() function in my .ino file to test how long the delay really was, I got a value of 1020 ms. I assume that extra 20 ms can be accounted to the overhead of the function call and the running of the outer loop that I do not account for, but for a situation like this, that 20 ms of precision isn't a huge deal.

Visualizations:

Video of completed puzzles: Youtube Link

$$10\,(\text{ms}) = \left(\frac{1\,(\text{sec})}{16,000,000\,(\text{hz})} \times 1,000\,(\text{ms})\right) \times 2\,(\text{SBIW}) \times 2\,(\text{BRNE}) \times 40,000\,(\text{counter})$$

Figure 4: Delay Timing Calculation

Table 1: An example of the LED going from red to orange to yellow to green as the user gets closer to correct position during puzzle 2.

```
// if in correct position (green)
if ((xAxisPos >= (xAxis - 12) && xAxisPos <= (xAxis + 12)) && (yAxisPos >= (yAxis - 12) && yAxisPos <= (yAxis + 12))) {
  Esplora.writeRGB(0, 255, 0);    // green light
  if (button == 0) {              // if button pressed to enter guess
    Esplora.tone(600);            // high pitch tone
    myDelay(20);
    Esplora.noTone();
    complete = true;              // complete, next attempt
    attempt += 1;
  }
// if in yellow position
} else if ((xAxisPos >= (xAxis - 25) && xAxisPos <= (xAxis + 25)) && (yAxisPos >= (yAxis - 25) && yAxisPos <= (yAxis + 25))) {
  Esplora.writeRGB(255, 255, 0);  // yellow light
  wrongGuess();                   // if guess entered, low pitch tone
// if in orange position
} else if ((xAxisPos >= (xAxis - 50) && xAxisPos <= (xAxis + 50)) && (yAxisPos >= (yAxis - 50) && yAxisPos <= (yAxis + 50))) {
  Esplora.writeRGB(255, 50, 0);   // orange light
  wrongGuess();                   // if guess entered, low pitch tone
// if in red position
} else if (!part2Done) {
  Esplora.writeRGB(255, 0, 0);    // red light
  wrongGuess();                   // if guess entered, low pitch tone
}
```

Figure 5: Puzzle 2 Code

Sources:

"Arduino Esplora Legacy Documentation." Docs.Arduino.Cc, 14 Mar. 2024, docs.arduino.cc/retired/boards/arduino-

esplora/. Accessed 08 Dec. 2024.

"Atmel AVR Instruction Set." Wikipedia, Wikimedia Foundation, 12 Aug. 2024, link. Accessed 07 Dec. 2024.

"AVR-GCC Inline Assembler Cookbook." AVR LibC, avrdudes.github.io/avr-libc/avr-libc-user-manual-2.2.0/inline-asm.html. Accessed 07 Dec. 2024.

Jfjlaros. "Random Function Not Randomising after Reset." Arduino Forum, 24 Aug. 2023, forum.arduino.cc/t/random-function-not-randomising-after-reset/1161545/7. Accessed 07 Dec. 2024.

Kuzechie, Anas. "Assembly Programming via Arduino." YouTube, 2021, www.youtube.com/playlist?list=PL09ZAP7-T-LmlX5vctZV4PFfZwMNzjX1F. Accessed 07 Dec. 2024.

"Math - Clock Speeds and Cpu Pipes." Northern Illinois University, faculty.cs.niu.edu/-berezin/463/Assns/clocks.html:-:text=Remember-20:-201-20Hertz-20=-201-20Hz,-5E-2D6-20secs/cycle. Accessed 08 Dec. 2024.

meslomp. "Help with Counting the Machine Cycles for This Assembly Program?" AVR Freaks, 24 Oct. 2019, www.avrfreaks.net/s/topic/a5C3l000000UaOUEA0/t153692. Accessed 07 Dec. 2024.