Robert McKay A02080393
Xuecong Fan A01972388
October 9, 2017

Lab 2

1. **Introduction**
Students designed Sumo, a game in which two players race to press their button before their opponent and push the other player's LED outside of the bar LED range.

2. **Scope**

This document includes a design overview, design details, and testing of the Sumo game. It does not include a step-by-step process of how students accomplished the lab or how creative problem solving and troubleshooting was implemented to make the lab a success. It does not include a sob story of how difficult the lab was for students, nor does it include psalms about the greatness of the game of Sumo.

3. **Design Overview**

3.1. Requirements

Students were issued the following requirements:

1. The display shall consist of a 10-LED bar graph mounted horizontally.
2. There shall be 2 buttons, each in the proximity of a different end of the bar graph. Player 1 uses the button on the left and player 2 uses the button on the right.
3. There shall be a DIP switch to configure the speed of each player. The speed Sn for player n shall be interpreted as a 2-bit binary number, one switch per bit.
4. The buttons shall be sampled at least every 5 ms (milliseconds).
5. After the system is reset, the two center LEDs of the bar graph shall flash at a rate of 2 Hz. This rate will be controlled using a timer. The LED on the left represents player 1 and the LED on the right represents player 2.
6. Each player must press their button to indicate their readiness to play. Once a player presses their button, their LED shall be lit solidly.
7. At some random time at least 1 second but no more than 2 seconds after (a) both players indicate their readiness to play or (b) a move concludes that does not end the game, the leftmost lit LED shall move one spot to the left and the rightmost lit LED shall move one spot to the right. This event starts the move.
8. After the move starts, each player races to press their button. As soon as a button is pressed, the corresponding player's lit LED moves back to its prior position and a timer is started.
9. If the timer in (8) expires before the opponent presses their button (and moves their lit LED), the quicker player's lit LED shall move again and be adjacent to their opponent's lit LED. Otherwise, the move is a draw.
10. If the result of this move is that the two lit LEDs are on the leftmost or rightmost side of the bar graph, the game is over and the 2 lit LEDs shall flash at a rate of 2 Hz until the system is reset.
11. The delay time in (8) shall be based on the player's speed, Sn, and the number of contiguous drawn moves, d. If player n is the first to press their button, the delay in milliseconds shall be $2^{-min(\mathrm{d},4)}(320 - 80\mathrm{Sn})$.

3.2. Dependencies

• The system required a 3.3 volt power source to light the LEDs and supply voltage to the buttons
• The system required DIP switches to be set prior to running the program • The system required the code students wrote to be flashed to the Cortex-M4 microcontroller

3.3. Theory of Operation

For this section, it may be helpful to view Figure 2 (the flowchart for the system) in the appendix. The system began by initializing all ports, pins, and clocks to be used. As part of this process, the DIP switches were checked for their values. The reset position was then loaded. The program waited for player inputs to signify readiness on the part of both players. The first person to press his or her button called a function which grabbed the value from the system ticker at the moment the button was pressed. As soon as the other player pressed his or her button, that value was used to wait for a random time between 1 and 2 seconds before beginning the move. On beginning the move, the first player to press his or her button would advance his or her LED one position toward the opposite player. The first player to press his or her button also grabbed a value from the system ticker that would be used for the next random number. The faster player's button press began a timer based on the faster player's DIP switch setting, in which either the opposing player would press his or her button in time (resulting in a draw) or not (resulting in a round win for the first player). In the event of a draw, the consecutive draw counter would be incremented, followed by waiting for a random amount of time between 1 and 2 seconds before beginning the move once more. In the event of a round win, the faster player's LED was advanced once more. Victory conditions were checked in this case. If victory conditions were met (both LEDs end at one side of the bar) then the lights would stay in position and flash at 2 Hz. If victory conditions were not met, a random amount of time was waited before beginning the move again. In this manner play would proceed until victory conditions were met.

3.4. Design Alternatives

4. **Design Details**

The design began with initializing clocks, ports, and pins, and loading the DIP switch values. DIP switch values were stored in dedicated registers so that they could be called upon later. The rest of the design breaks down into sections. For schematics, see Figure 1 of the appendix. For the code, see Figures 10 to 16 of the appendix. The description follows the code, so it may be helpful as a reference.

4.1. Reset Position

In the reset position, the system waits for input from player 1 or player 2 while keeping the center LEDs flashing at 2 Hz. LEDs were managed using bit banding. When one of the buttons is pressed, the LED nearest that player stops flashing while the LED for the opposite player continues to flash until the opposing player's button is pressed. When the first button is pressed, the code branches to that player's ready state. At the beginning of this state, randomseed is called. randomseed simply reads the current value from the system ticker (a value between 0 and 3999999) and multiplies it by four, then stores it in a dedicated register. This is useful because the microcontroller clock runs at 16 MHz, and with one machine cycle per clock cycle, a value can be obtained to reload into the system ticker between 0 and 1 second for wait time (randomclock accomplishes this). By simply reloading system ticker with 15999999 after this wait, the system will wait for a full second in addition to the random wait time (secondwait accomplishes this). Once the player who has not pressed his or her button presses his or her button, the ready position is reached.

4.2. Ready Position

In the ready position, both players' LEDs are set to be solid. Two more registers are dedicated to the positions of players 1 and 2, respectively. These will be used to adjust the lights back and forth down the bar LED with minimal effort, using bit banding. Finally, move is reached.

4.3. Move

Move is where the random amount of time plus one second is waited. This is done immediately after the label. Next, each player's LED is moved back by 1. Logic shifts on the bits in the dedicated location registers accomplish this. The contents of those two registers are "orred" together, then stored into the output so that players can see the move begin. After this, players race to make a move. In waitformove, buttons are checked until one is pressed. Whoever moves first calls randomseed and starts the timer for the other player, based on the fastest player's DIP switch value. The faster player's LED is incremented by one position toward the other player by a logic shift on the value in the dedicated location register, followed by a store to the output. The clock is then set according to the values prescribed in requirement 11 (Section 3.1). This is accomplished by first multiplying the dedicated DIP switch register of the faster player by 80, then subtracting that value from 320. Next, the consecutive draw dedicated register is compared with 4 to find the minimum. After finding the minimum, the difference between 320 and 80 times the DIP switch is logic shifted right by that amount, accomplishing division by 2 to the power of the minimum of consecutive draws and 4. Finally, this result is multiplied by 16000 (multiplying first the result by 16 million, which is the reload value to wait a full second, followed by dividing by 1000 since the result is in milliseconds). Once this is done, the final count is loaded into the system ticker, and the clock begins to count down.

At this point, the system waits for an input from the slower player. If the input comes before the system ticker flags, the result of the move is a draw (shiftplayer(1 or 2)led) and the program branches back to move, after incrementing the consecutive draw dedicated register and showing the tie on the bar LEDs.

If the input does not come before the system ticker flags (p(1 or 2)roundwin), then the round winner advances one more LED, and the system checks for the victory condition–whether the round winner's LED is positioned at the second to last LED away from the opponent. In that event, the lights flash at 2 Hz until the program is reset (accomplished by victory(1 or 2). Otherwise, the program branches back to move.

4.4. Victory

Victory is almost identical to the reset position, except instead of lighting the two central LEDs, the extreme ones are flashing. The same value, 3,999,999, is loaded into the system ticker to achieve the desired 2 Hz.

5. **Testing**

The following list of tests corresponds to the requirements list detailed in section 3.1–Requirements.

1. The display consisted of a 10-LED bar graph mounted horizontally. The test was to put the LED into the board horizontally and connect it correctly to the microcontroller pins.
2. There were 2 buttons on opposite ends of the bar graph, one corresponding to player 1 and the other to player 2. Depending on the orientation, this could be left or right for player 1 or 2 respectively.
3. The DIP switch was used to configure the speed of each player as specified in the requirements. Testing to verify the correctness of Sn was attaching the logic analyzer across one of the pins crossed by the LED pressed by a player. The time delay was measured and recorded for each case, without consecutive draws. As expected, the screenshots verified 80, 160, 240, and 320 milliseconds for binary DIP switch inputs of 3, 2, 1, and 0 respectively. Screenshots to show this can be viewed in Figures 6 through 9 of the appendix.
4. The code is less than 500 lines. If each line had a branch operation (at most 3 machine cycles), that's less than 1500 machine cycles. Each loop of code checks input. Suppose the entire code was a single loop, and each time it only checked one button. Then both buttons would still be checked at least every $2 * 1500/16000000 = 1.875$ microseconds, which is much less than 5 milliseconds.
5. Students tested the initial 2 Hertz frequency by connecting the logic analyzer to one of the middle bar LEDs. The oscilloscope screenshot shown in Figure 3 of the Appendix verifies that the period of the signal is 500 milliseconds. Since the same code for timing was used in both victory conditions, this

test was sufficient to verify their functionality as well.
6. This was tested by students pressing the LED. When the button was pressed, the corresponding light turned solid.
7. This was tested using the logic analyzer. Two results are shown in Figure 4 and Figure 5 in the appendix.
8. This was tested by pressing the button during the move. The LED of each player moved back to its original position after the players' buttons were pressed individually.
9. This was verified using the logic analyzer in the same manner as in test (5). After the move, the button of one player was pressed. As it moved to, then through, where it originally was, the time that LED was high was measured. Results are shown in Figure blah of the appendix.
10. This was tested by moving the LEDs to the extremes of the bar LED. See test (5) for more details about the timing.
11. This formula was measured as described in test (9).

6. Conclusion

The game students produced met the requirements of the stakeholders; however, possibly undesirable issues persisted. Player 1 is given priority for checks in the game, so if he or she holds down his or her button, that player will win every game.
The wiring into the board was correct but somewhat faulty. It would occasionally move a player's LED as though the button was pressed when it was not. This was tested and was due to a set of female to female ribbon cables connecting poorly with a set of male to male cables. More male to female cables would easily fix this issue.
Students succeeded in developing the product and gained insight into timers on the Cortex-M4 microcontroller.

7. Appendix

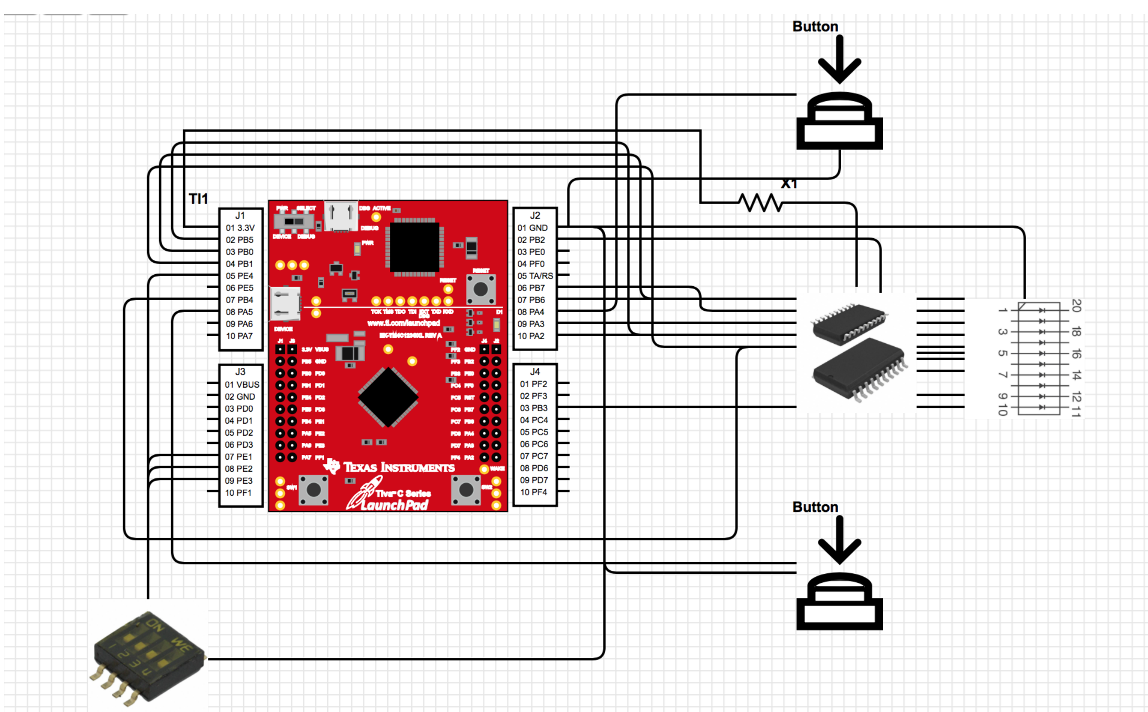

Figure 1: Schematic. All resistors are 220 ohms. Pins are connected sequentially to DIP switch and bar LED.

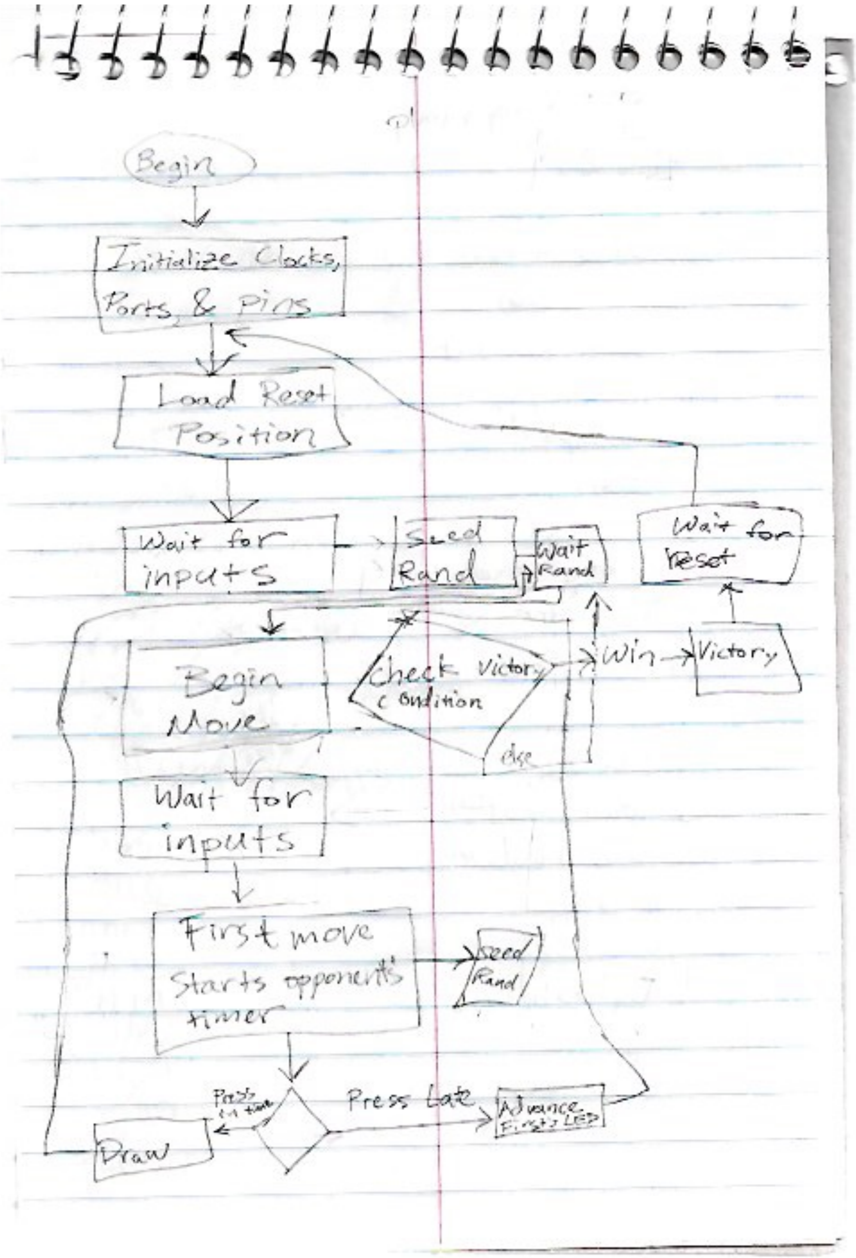

Figure 2: Flowchart

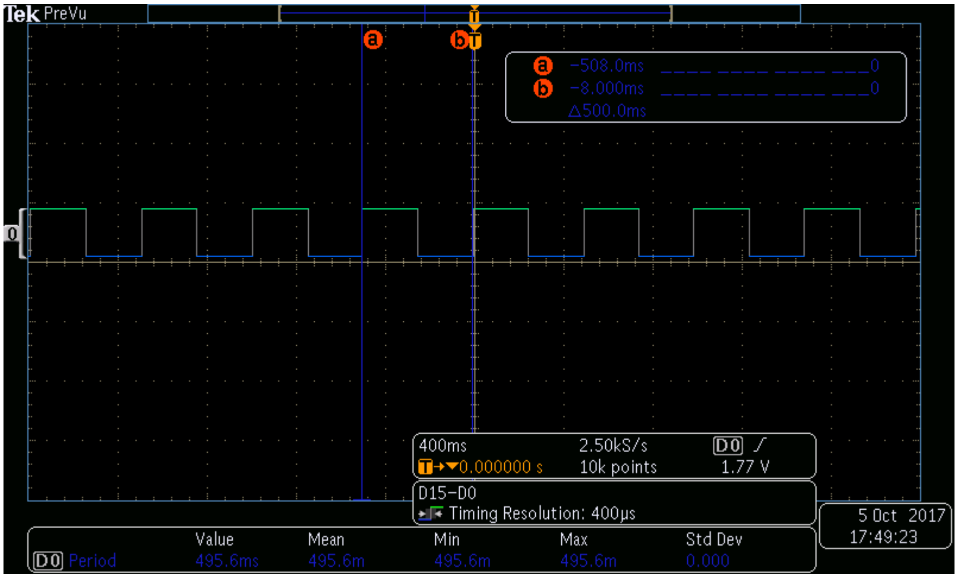

Figure 3: A demonstration of 2 Hz

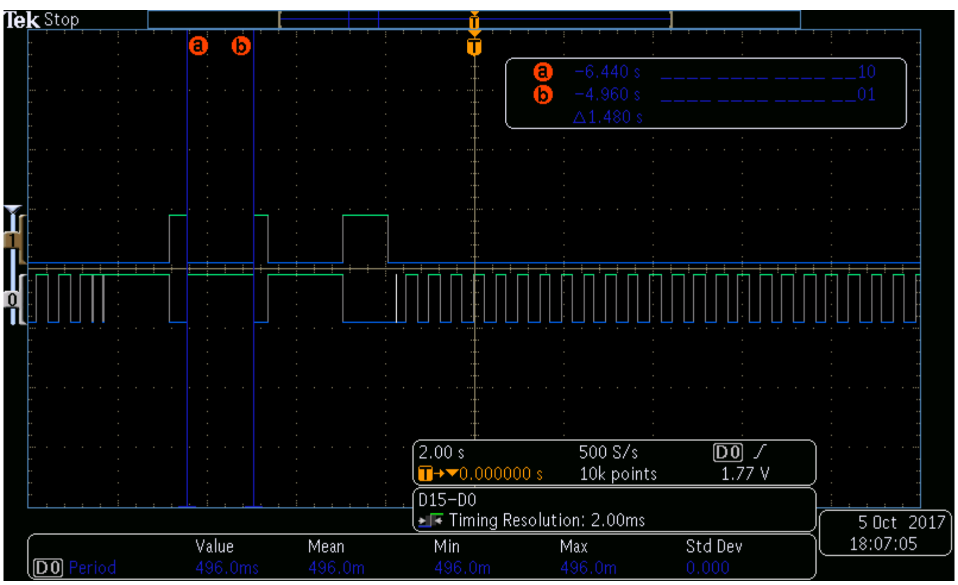

Figure 4: First example of a random time

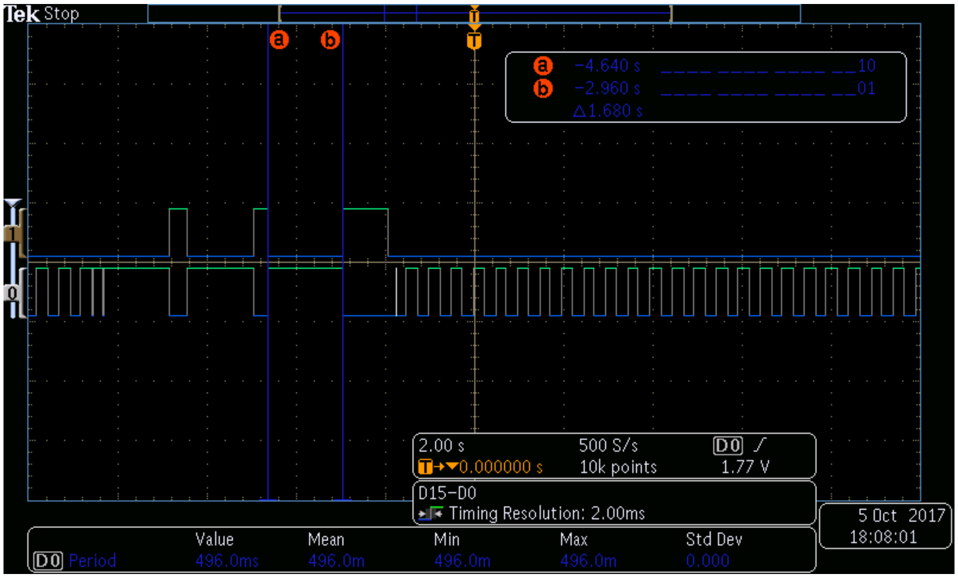

Figure 5: Second example of a random time

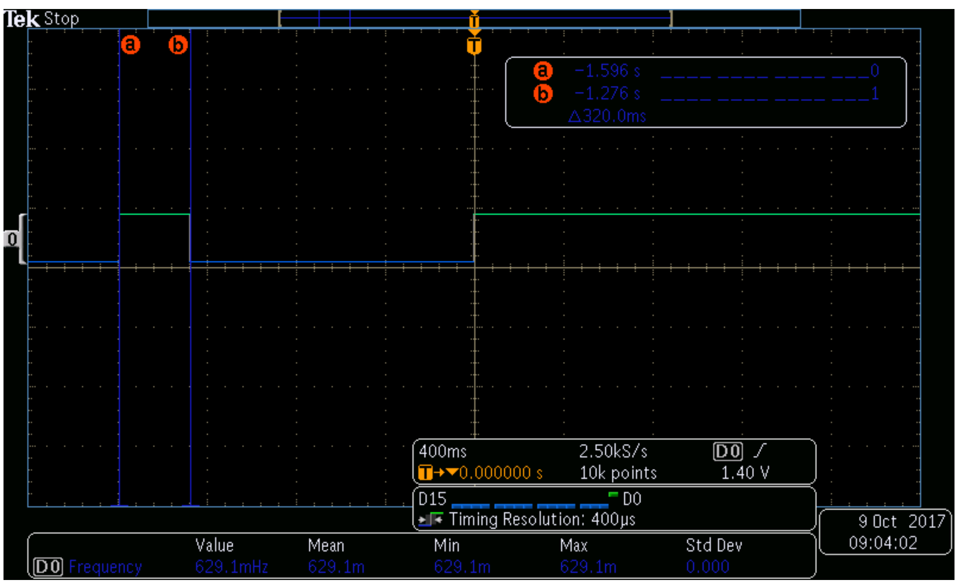

Figure 6: DIP switch set to 0

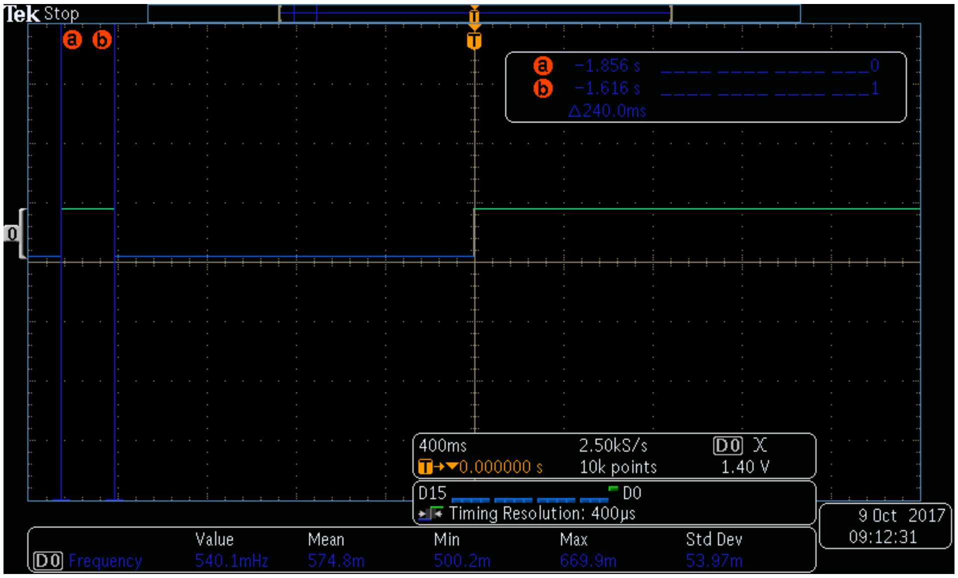

Figure 7: DIP switch set to 1

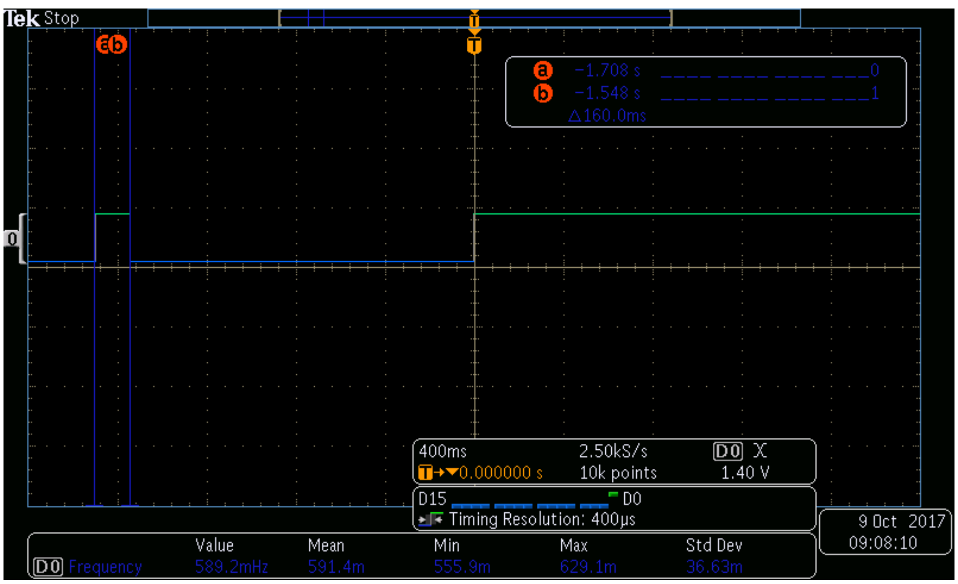

Figure 8: DIP switch set to 2

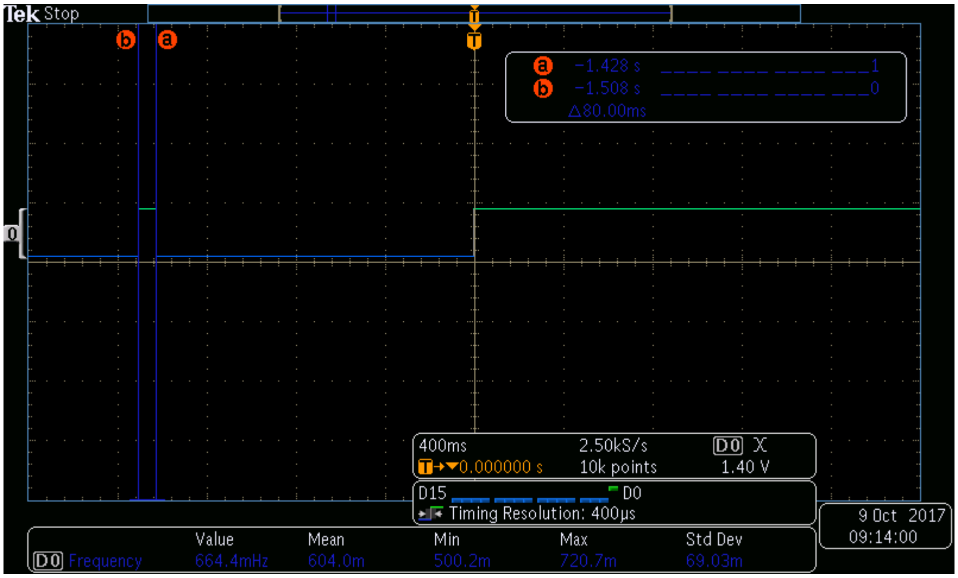

Figure 9: DIP switch set to 3

```
        THUMB
        AREA    |.text|, CODE, READONLY, ALIGN=2
        EXPORT Start

        ALIGN
Start
    mov32 R1, #0x400FE608 ;Load the clock address and prepare to turn it on
    mov R0, #0x13; 0001 0011, to open ports A, B, and E
    str R0, [R1]; write this to turn on the clock

    mov32 R1, #0x40004000 ;get address of port A
    mov32 R2, #0x40005000 ;get address of port B
    mov32 R3, #0x40024000 ;get address of port E

    mov R0, #0xFF ;1111 1111, used to enable pins
    str R0, [R1, #0x524] ;Lets us use pins in port A by writing 1s in GPIOCR
    str R0, [R2, #0x524] ;Lets us use pins in port B by writing 1s in GPIOCR
    str R0, [R3, #0x524] ;Lets us use pins in port E by writing 1s in GPIOCR

    mov R0, #0xFF ;will be used to configure all ports in B
    str R0, [R2, #0x400] ;configures pins in port B to be output
    mov R0, #0xC ;0000 1100
    str R0, [R1, #0x400] ;configures PA2 and PA3 to be output
    mov R0, #0x0 ;0000 0000
    str R0, [R3, #0x400] ;configures port E to be input

    mov R0, #0x1E ;0001 1110
    str R0, [R3, #0x514] ;Sets pull down resistors for PE1-PE4. These will be used for the DIP switch.
    mov R0, #0x50 ;0110 0000, sets pull down resistors for PA4, PA5. These will be used for the buttons.
    str R0, [R1, #0x514] ;

    mov R0, #0xFF ;1111 1111
    str R0, [R1, #0x51C] ;Digital enable for pins in A
    str R0, [R2, #0x51C] ;Digital enable for pins in B
    str R0, [R3, #0x51C] ;Digital enable for pins in E

    mov R0, #0x0 ;0000 0000
    str R0, [R1, #0x420] ;clear alternate functions in A
    str R0, [R2, #0x420] ;clear alternate functions in B
    str R0, [R3, #0x420] ;clear alternate functions in E

    ;begin the code

    ;DIP switch checker

    mov32 R0, #0x40024000 ;Load port E
    ldr R11, [R0, #0x60] ;Check PE3 and PE4 for player 1
    lsr R11, #3 ;shift it to grab Sn
    ldr R12, [R0, #0x18] ;Check PE1 and PE2 for player 2
    lsr R12, #1 ;shift it to grab Sn
    mov R6, #0x0 ;R6 will be dedicated to counting draws.

    ;this piece initializes SysTick. It will go at 2 Hz.
    ldr R0, =0xE000E000 ;base address for SysTick
    mov R1, #0x0 ;prepare to stop timer
    str R1, [R0, #0x10] ;sets ENABLE low on SysTick
    mov32 R1, #0x3D08FF ;Prepare to set count, which is 3,999,999
    str R1, [R0, #0x14]
    str R1, [R0, #0x18] ;set current value to the same as the count value
    mov R1, #0x4
    str R1, [R0, #0x10] ;disable interrupt and set clock source to be internal
    mov R1, #0x5
    str R1, [R0, #0x10] ;start counting, and enable the timer
```

Figure 10: Code

```
    str R1, [R0, #0x10] ;start counting, and enable the timer

    mov R0, #0x30 ;00 0011 0000 [00], set the middle LEDs to begin flashing
    str R0, [R2, #0xC0] ;turn on the middle LEDs

resetmode
    ldr R0, [R2, #0xC0]
    mvn R0, R0
    str R0, [R2, #0xC0]
    mov R1, #0x1
wait
    ;check player 1 ready
    ldr R0, =0x40004000 ;load A
    ldr R0, [R0, #0x80]
    mov R1, #0x1
    cmp R1, R0, lsr #5
    beq player1ready

    ldr R0, =0x40004000 ;load A
    ldr R0, [R0, #0x100]
    mov R1, #0x1
    cmp R1, R0, lsr #6
    beq player2ready

    ldr R0, =0xE000E000 ;load SysTick at count offset
    ldr R0, [R0, #0x10]
    cmp R1, R0, lsr #16 ;check the count bit
    bne wait
    b resetmode

player1ready
    bl randomseed ;grab the random value off of player 1's button press
    mov R0, #0x10 ;00 0001 0000 [00], set player 1's LED solid
    str R0, [R2, #0x40] ;store player 1's LED solid
    ldr R0, [R2, #0x80] ;only switch player 2's led now
    mvn R0, R0
    str R0, [R2, #0x80] ;switch it in
player1wait
    mov R1, #0x1
    ldr R0, =0x40004000 ;load A
    ldr R0, [R0, #0x100] ;check player 2 button
    mov R1, #0x1
    cmp R1, R0, lsr #6
    beq ready

    ldr R0, =0xE000E000 ;load SysTick at count offset
    ldr R0, [R0, #0x10]
    cmp R1, R0, lsr #16 ;check the count bit
    bne player1wait
    b player1ready

player2ready
    bl randomseed ;grab the random value off of player 1's button press
    mov R0, #0x20 ;00 0010 0000 [00], set player 2's LED solid
    str R0, [R2, #0x80] ;store player 2's LED solid
    ldr R0, [R2, #0x40] ;only switch player 1's led now
    mvn R0, R0
    str R0, [R2, #0x40] ;switch it in
player2wait
    mov R1, #0x1
    ldr R0, =0x40004000 ;load A
    ldr R0, [R0, #0x80] ;check player 1 button
    cmp R1, R0, lsr #5
    beq ready
```

Figure 11: Code

```
    beq ready

    ldr R0, =0xE000E000 ;load SysTick at count offset
    ldr R0, [R0, #0x10]
    cmp R1, R0, lsr #16 ;check the count bit
    bne player2wait
    b player2ready
ready
    mov R0, #0x30 ;00 0011 0000 [00], prepare to set both players' LEDs solid
    str R0, [R2, #0xC0] ;store both LEDs solid
    mov R8, #0x10 ;R8 will be dedicated to player 1's location.
    mov R9, #0x20 ;R9 will be dedicated to player 2's location
    b move

randomseed
    ldr R10, =0xE000E000 ;Grab the value for randomness
    ldr R10, [R10, #0x18] ;load the value for randomness into R10
    mov R4, #4
    mul R10, R4 ;multiplies the value in R10 by 4. This will give us the full range we want.
    mov32 R0, #0x7A1200
    add R10, R0 ;adds 1 second to the random delay time
    bx lr

randomclock
    ldr R0, =0xE000E000 ;base address for SysTick
    mov R1, #0x0 ;prepare to stop timer
    str R1, [R0, #0x10] ;sets ENABLE low on SysTick
    mov R1, R10 ;Prepare to set count, which is random
    str R1, [R0, #0x14]
    str R1, [R0, #0x18] ;set current value to the same as the count value
    mov R1, #0x4
    str R1, [R0, #0x10] ;disable interrupt and set clock source to be internal
    mov R1, #0x5
    str R1, [R0, #0x10] ;start counting, and enable the timer
    mov R1, #0x1
randwait
    ldr R0, =0xE000E000 ;load SysTick at count offset
    ldr R0, [R0, #0x10]
    cmp R1, R0, lsr #16 ;check the count bit
    bne randwait
    ldr R0, =0xE000E000 ;base address for SysTick
    mov R1, #0x0 ;prepare to stop timer
    str R1, [R0, #0x10] ;sets ENABLE low on SysTick
    mov R1, #0xFFFFFF ;prepare to wait for another second
    str R1, [R0, #0x14]
    str R1, [R0, #0x18] ;set current value to the same as the count value
    mov R1, #0x4
    str R1, [R0, #0x10] ;disable interrupt and set clock source to be internal
    mov R1, #0x5
    str R1, [R0, #0x10] ;start counting, and enable the timer
    mov R1, #0x1
secondwait
    ldr R0, =0xE000E000 ;load SysTick at count offset
    ldr R0, [R0, #0x10]
    cmp R1, R0, lsr #16 ;check the count bit
    bne secondwait
    bx lr

move
    bl randomclock
    ldr R1, =0x40004000 ;load port A
    lsr R8, #1 ;moves player 1 to the right by 1
    lsl R9, #1 ;moves player 2 to the left by 1
```

Figure 12: Code

```
    lsl R9, #1 ;moves player 2 to the left by 1
    mov R7, #0x0
    orr R7, R8, R9 ;puts the value we want to display into a register
    str R7, [R2, #0x3FC]
    ldr R1, =0x40004000
    lsr R7, #6 ;moves the register into position to store for A
    str R7, [R1, #0x3FC]
waitformove
    ;check player 1 button
    ldr R0, =0x40004000 ;load A
    ldr R0, [R0, #0x80] ;check if player 1's button is pressed
    mov R1, #0x1
    cmp R1, R0, lsr #5
    beq start2stimer
    ;check player 2 button
    ldr R0, =0x40004000 ;load A
    ldr R0, [R0, #0x100] ;check if player 2's button is pressed
    mov R1, #0x1
    cmp R1, R0, lsr #6
    beq start1stimer
    b waitformove
start1stimer
    bl randomseed

    lsr R9, #1 ;move player 2's LED up by one.
    mov R7, #0x0
    orr R7, R8, R9 ;prepare to store the result for the display
    str R7, [R2, #0x3FC]
    ldr R1, =0x40004000
    lsr R7, #6 ;moves the register into position to store for A
    str R7, [R1, #0x3FC]

    ldr R0, =0xE000E000 ;base address for SysTick
    mov R1, #0x0 ;prepare to stop timer
    str R1, [R0, #0x10] ;sets ENABLE low on SysTick
    mov R4, #80
    mul R5, R11, R4 ;R5 = 80*Sn, for player 2
    mov R4, #320 ;R4 = 320
    sub R4, R5 ;R4 = 320 - 80*Sn
    cmp R6, #4 ;compare consecutive draws with 4
    blt consecutivedrawcondition
    b nonconsecutivedrawcondition
consecutivedrawcondition
    lsr R4, R6 ;R4 = 2^(-R6)*(320-80*Sn)
    b restofcode
nonconsecutivedrawcondition
    lsr R4, #4 ;R4 = 2^(-4)*(320-80*Sn)
restofcode
    mov R5, #16000 ;used to convert ms to a count
    mul R5, R4
    ldr R0, =0xE000E000 ;base address for SysTick
    mov R1, #0x0 ;prepare to stop timer
    str R1, [R0, #0x10] ;sets ENABLE low on SysTick
    mov R1, R5 ;Prepare to set count, which is based on player 1's difficulty
    str R1, [R0, #0x14]
    str R1, [R0, #0x18] ;set current value to the same as the count value
    mov R1, #0x4
    str R1, [R0, #0x10] ;disable interrupt and set clock source to be internal
    mov R1, #0x5
    str R1, [R0, #0x10] ;start counting, and enable the timer

waitfor1
    ldr R0, =0x40004000 ;load A
    ldr R0, [R0, #0x80] ;check player 1 button
```

Figure 13: Code

```
    ldr R0, =0x40004000 ;load A
    ldr R0, [R0, #0x80] ;check player 1 button
    mov R1, #0x1
    cmp R1, R0, lsr #5
    beq shiftplayer1led
    ldr R0, =0xE000E000 ;load SysTick at count offset
    ldr R0, [R0, #0x10]
    cmp R1, R0, lsr #16 ;check the count bit
    bne waitfor1
    b p2roundwin
shiftplayer1led
    add R6, #1 ;increment consecutive draw
    lsl R8, #1
    mov R7, #0x0
    ldr R1, =0x40004000 ;load port A
    orr R7, R8, R9 ;puts the value we want to display into a register
    str R7, [R2, #0x3FC]
    ldr R1, =0x40004000
    lsr R7, #6 ;moves the register into position to store for A
    str R7, [R1, #0x3FC]
    b move
p2roundwin
    mov R6, #0x0
    lsr R9, #1 ;move player 2's LED up by one.
    mov R7, #0x0
    orr R7, R8, R9 ;prepare to store the result for the display
    str R7, [R2, #0x3FC]
    ldr R1, =0x40004000
    lsr R7, #6 ;moves the register into position to store for A
    str R7, [R1, #0x3FC]
    cmp R8, #0x1
    beq victory2b
    b move
start2stimer
    bl randomseed
    lsl R8, #1 ;move player 1's LED up by one.
    mov R7, #0x0
    orr R7, R8, R9 ;prepare to store the result for the display
    str R7, [R2, #0x3FC]
    ldr R1, =0x40004000 ;load port A
    lsr R7, #6 ;moves the register into position to store for A
    str R7, [R1, #0x3FC]
    ldr R0, =0xE000E000 ;base address for SysTick
    mov R1, #0x0 ;prepare to stop timer
    str R1, [R0, #0x10] ;sets ENABLE low on SysTick
    mov R4, #80
    mul R5, R12, R4 ;R5 = 80*Sn, for player 2
    mov R4, #320 ;R4 = 320
    sub R4, R5 ;R4 = 320 - 80*Sn
    cmp R6, #4 ;compare consecutive draws with 4
    blt cdraw
    b noncdraw
cdraw
    lsr R4, R6 ;R4 = 2^(-R6)*(320-80*Sn)
    b cont
noncdraw
    lsr R4, #4 ;R4 = 2^(-4)*(320-80*Sn)
cont
    mov R5, #16000 ;used to convert ms to a count
    mul R5, R4
    ldr R0, =0xE000E000 ;base address for SysTick
    mov R1, #0x0 ;prepare to stop timer
    str R1, [R0, #0x10] ;sets ENABLE low on SysTick
```

Figure 14: Code

```
    str R1, [R0, #0x10] ;sets ENABLE low on SysTick
    mov R1, R5 ;Prepare to set count, which is based on player 1's difficulty
    str R1, [R0, #0x14]
    str R1, [R0, #0x18] ;set current value to the same as the count value
    mov R1, #0x4
    str R1, [R0, #0x10] ;disable interrupt and set clock source to be internal
    mov R1, #0x5
    str R1, [R0, #0x10] ;start counting, and enable the timer
    mov R5, #16000 ;used to convert ms to a count
    mul R5, R4
    ldr R0, =0xE000E000 ;base address for SysTick
    mov R1, #0x0 ;prepare to stop timer
    str R1, [R0, #0x10] ;sets ENABLE low on SysTick
    mov R1, R5 ;Prepare to set count, which is based on player 1's difficulty
    str R1, [R0, #0x14]
    str R1, [R0, #0x18] ;set current value to the same as the count value
    mov R1, #0x4
    str R1, [R0, #0x10] ;disable interrupt and set clock source to be internal
    mov R1, #0x5
    str R1, [R0, #0x10] ;start counting, and enable the timer
waitfor2
    ldr R0, =0x40004000 ;load A
    ldr R0, [R0, #0x100] ;check player 2 button
    mov R1, #0x1
    cmp R1, R0, lsr #6
    beq shiftplayer2led
    ldr R0, =0xE000E000 ;load SysTick at count offset
    ldr R0, [R0, #0x10]
    cmp R1, R0, lsr #16 ;check the count bit
    bne waitfor2
    b p1roundwin
shiftplayer2led
    add R6, #1 ;increment consecutive draw
    lsr R9, #1
    ldr R1, =0x40004000 ;load port A
    mov R7, #0x0
    orr R7, R8, R9 ;puts the value we want to display into a register
    str R7, [R2, #0x3FC]
    ldr R1, =0x40004000
    lsr R7, #6 ;moves the register into position to store for A
    str R7, [R1, #0x3FC]
    b move
victory2b
    b victory2
p1roundwin
    mov R6, #0x0 ;reset consecutive draws
    lsl R8, #1 ;move player 1's LED up by one.
    mov R7, #0x0
    orr R7, R8, R9 ;prepare to store the result for the display
    str R7, [R2, #0x3FC]
    ldr R1, =0x40004000
    lsr R7, #6 ;moves the register into position to store for A
    str R7, [R1, #0x3FC]
    cmp R9, #0x200
    beq victory1
    b move
victory2
    ;this piece initializes SysTick. It will go at 2 Hz.
    ldr R0, =0xE000E000 ;base address for SysTick
    mov R1, #0x0 ;prepare to stop timer
    str R1, [R0, #0x10] ;sets ENABLE low on SysTick
    mov32 R1, #0x3D08FF ;Prepare to set count, which is 3,999,999
    str R1, [R0, #0x14]
```

Figure 15: Code

```
    str R1, [R0, #0x14]
    str R1, [R0, #0x18] ;set current value to the same as the count value
    mov R1, #0x4
    str R1, [R0, #0x10] ;disable interrupt and set clock source to be internal
    mov R1, #0x5
    str R1, [R0, #0x10] ;start counting, and enable the timer

victory2mode
    ldr R0, [R2, #0xC]
    mvn R0, R0
    str R0, [R2, #0xC]
    mov R1, #0x1
victory2loop
    ldr R0, =0xE000E000 ;load SysTick at count offset
    ldr R0, [R0, #0x10]
    cmp R1, R0, lsr #16 ;check the count bit
    bne victory2loop
    b victory2mode
victory1
    ;this piece initializes SysTick. It will go at 2 Hz.
    ldr R0, =0xE000E000 ;base address for SysTick
    mov R1, #0x0 ;prepare to stop timer
    str R1, [R0, #0x10] ;sets ENABLE low on SysTick
    mov32 R1, #0x3D08FF ;Prepare to set count, which is 3,999,999
    str R1, [R0, #0x14]
    str R1, [R0, #0x18] ;set current value to the same as the count value
    mov R1, #0x4
    str R1, [R0, #0x10] ;disable interrupt and set clock source to be internal
    mov R1, #0x5
    str R1, [R0, #0x10] ;start counting, and enable the timer

victory1mode
    mov32 R1, #0x40004000
    ldr R0, [R1, #0x30]
    mvn R0, R0
    str R0, [R1, #0x30]
    mov R1, #0x1
victory1loop
    ldr R0, =0xE000E000 ;load SysTick at count offset
    ldr R0, [R0, #0x10]
    cmp R1, R0, lsr #16 ;check the count bit
    bne victory1loop
    b victory1mode

    ALIGN
    END
```

Figure 16: Code