Sumo Showdown

Design Documentation

Trevor Welch and Aaron Pettit
October 12, 2017

Table of Contents

List of Figures

1. Introduction

The design of this project is a two-player game using the bar graph and two push buttons. Each player will control one of two sumo wrestlers and each will try to push the other out of the ring. As the match goes on, the wrestlers periodically shove each other apart, and the first to regain his balance is able to push the other a little closer to the edge of the ring.

2. Scope

This document covers the design requirements, dependencies, and operation. An explanation of each function in the code is given. The results of tests and design details are included. Schematics and code segments are included in the Appendix.

The functionality of each electrical component is not discussed in this report.

3. Design Overview

3.1 Requirements

The following are the requirement for Sumo Showdown

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 $2^{-\min(d,4)}(320 – 80Sn)$.

3.2 Dependencies

The following are dependencies

- 3.3 V power supply

3.3 Theory of Operation

When the board is first powered on or on reset, the center two LEDs (pins 4,5) will flash at a rate of 2 Hz. A player must push their button to indicate they are ready to begin and their LED will stop blinking. Once both player have pressed their buttons the game will begin.

After a random time between one and two seconds, the LEDs move move apart by one. The first player to press their button will advance their LED to the other player. The other player has a delay set by the DIP switches of 0-3 with 3 giving a shorter delay. If the other player does not hit their button before this delay expires, the player who hit their button first will advance one more space towards the slower player. If the 2nd player hits their button before the delay is finished, they will advance their LED towards the other player and that move will be considered a draw with subsequent draws reducing the delay.

This process is repeated until the lit LEDs are on either side of the LED bar (pins 0,1 or 8,9). Those LEDs will then flash until the board is reset.

4. Design Details

4.1 Hardware Design

Sumo Showdown is implemented on a TM4C123GH6PM. The display is a 10 position LED bargraph. The game has two buttons, player 1 on the left side of the bargraph, and player 2 on the right side of the bargraph. A 4 position DIP switch is used to control player delay. Pins 0-1 is used for player 1. Pins 2-3 is used for player 2.

4.2 Software Design

4.2.1 Setup

General purpose input/output ports B, C, and E are used in the program. Port B is used for LED 0-7. PE0 and PE1 are initialized to LED 8 and 9. PE2 and PE3 are initialized to player 1 switch and player 2 switch, respectively. Port C is configured for the dip switches using ports 4-7. The general purpose timer 1 is activated in the setup and configured as a 32 bit wide, periodic timer. After port initialization, the program will remain at in the Standby loop until both players are ready. This loop blinks the middle 2 LEDs at a frequency of 2 Hz. When one player presses their ready button, their LED becomes solid. When both players are ready, both LEDs remain solid for a random number between 1 and 2 seconds, then continues to START_GAME.

4.2.2 Timer

General purpose timer wide 0 is used here. Timer 0 is turned off and the following attributes are applied to it

- 32 bit value
- Periodic

The value that is loaded into the timer is specified before the timer function call in R3. The timer is then enabled and we return to our place in the program. To check if the timer is done, Register GPTMRIS bit one is polled. If this value is a one, this means the timer has completed and the done flag must be reset by writing a 1 to Register GPTMICR.

4.2.3 Update LED

The current state of registers 0-3 is saved as those are modified in this function. The updated locations of the players are stored into registers r11(player 1) and r12(player 2). These registers are ANDed together in R3 to find which LEDs to turn on. The first 8 bits are stored into port B pins 0-7 and R3 is shifted right by 8. The first two bits of the shifted R3 is then stored into port E pins 0-1. The old state of registers 0-3 are restored and the function returns to where it was called.

4.2.4 Get First Turn

```
timer(rand(1,2))
while(!timer.done)
        if(p1.pressed || p2.pressed) goto MainLoop
end while
move p2 right
move p1 left
if(p1.pressed)
        move p1 right
        playerturn = p2
        timer(2^(-min(draws,4))*(320 - 80*p2.sn))
```

```
else if(p2.pressed)
        move p2 left
        playerturn =p1
        timer(2^(-min(draws,4))*(320 - 80*p1.sn))
end if
```

A turn begins after a random time between 1 and 2 seconds. If any button is pressed during this time interval, the timer is reset with a new random value.

Next, the code will load the current value of the buttons and check which player has pressed their button first. After detection, the code moves to either PLAYER1_FIRST or PLAYER2_FIRST based on whose button was pressed first. If no buttons were pressed, the code will loop until there is a button pressed.

Inside of the PLAYER_FIRST loops, the respective LED is advanced one space toward the slower player by branching to UPDATE_LED. The program now loads the current DIP switch values and initiates COMPUTE_PLAYER_DELAY. This loop calculates the time that the other player has to press their button and then initiates the TIMER loop.

4.2.5 Wait for Slower Player

This code accomplishes the following psuedo code

```
while3 (!timer.done)
        if(p1.pressed && playerturn == p1)
                move p1 right
                draw++
                goto MainLoop
        end if
        if(p2.pressed && playerturn == p2)  //skip p1
                move p2 left
                draw++
                goto Main loop
        end if
end while
draw = 0
if(playerturn == p1) ;p1 missed their turn
        move p2 left
end if
else ;p2 missed their turn
        move p1 right
end if
```

While the timer set for the player delay is not done, the player's buttons are polled. If the program was waiting on player 1 and their button is pressed, the move is considered a draw. P1's LED is moved back to its prior position by shifting right by 1 and calling UPDATE_LED, the delay counter is incremented, and the main game loop starts the next turn. The same thing happens for P2.

If the timer expires, this means the slower player did not hit their button in time. The delay counter is reset to 0. The faster player's LED advances towards the slower player a second time using UPDATE_LED moving the pair closer to one of the sides of the LED bargraph.

4.2.6 End Game

Once the LEDs come back together at the end of a turn, the LEDs' positions are checked to see if they are on the far sides of the LED bargraph. Position of player 1 is compared to 0x200 which is the leftmost LED. Position of player 2 is compared to 0x1 which is the rightmost LED. If either of these statements are true, the function BLINK is called which loops until reset is pressed. If both of these statements are false, the game continues to the next turn.

5. Testing

5.1 Two Hz Blink

The first test is to verify requirement number 5. The LEDs will blink at a rate of 2 Hz. The test was set up by connecting a lead from the oscilloscope to the breadboard where the blinking LED was connected. By capturing a single positive pulse width, the screen capture for Figure 1 shows a value of 499.2 ms. An error of 0.16%.

5.2 Random Time

This test satisfies requirement 7. The test compares the time from a completed turn to the time the LEDs separate and initiate a new turn. The time interval needs to be a random value between 1 and 2 seconds. The first random value measured is 1.68 seconds as shown in Figure 2. The second random value is 1.40 seconds as shown in Figure 3.

5.3 DIP Switch Delay

This test satisfies requirements three and eleven. The DIP switch is used to configure the speed of each player. The value Sn can vary from 0-3. This test accomplishes this by evaluating two different scenarios.

<u>Scenario 1</u>
Draws: 1
Player 2 speed: 1
Expected time delay: 80 ms

When these conditions are set, and player 1 hits their button first the expected delay is 80 ms. This test was verified and can be seen in Figure 4.

Scenario 2
Draws: 2
Player 1 speed: 2
Expected time delay: 60 ms

This test was verified and can be seen in Figure 5.

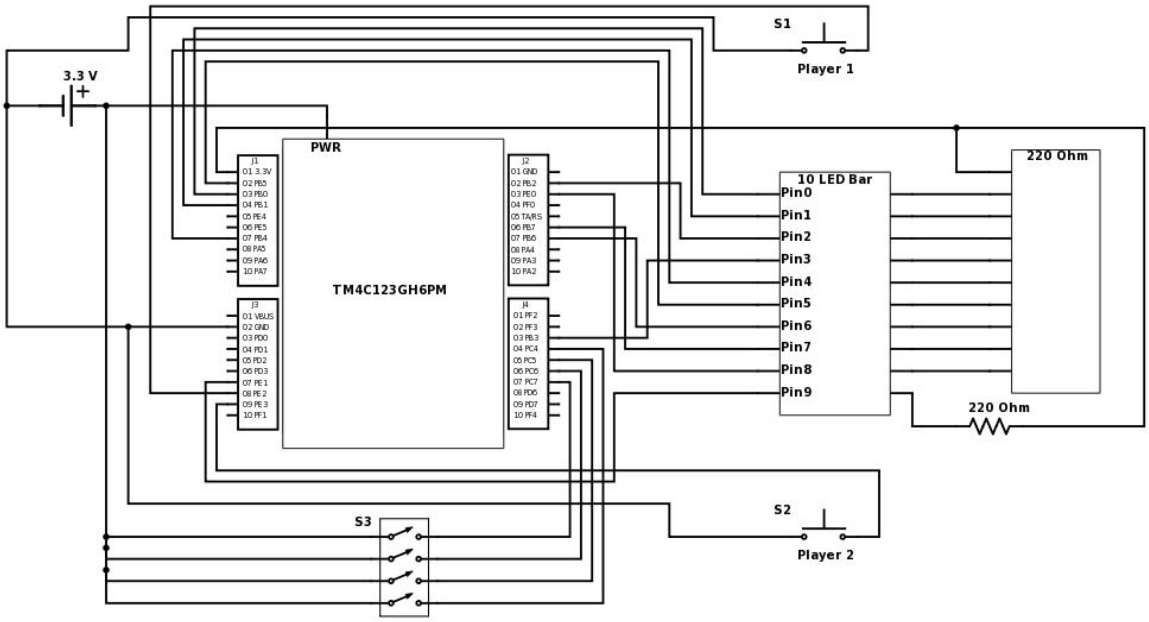
6. Conclusion

The design specified in this document creates a functional sumo wrestler game. 3 tests were performed to meet performance requirements 5, 7, 9, 10 and 11 (see sections 3.1 and 5)

This design could be better optimized. There are some inefficient uses of the registers where some variables could be condensed into a single register. Some better comments could also be created to describe each of the functions better. This code could also be rewritten in C to make it more readable.

Appendix A

Hardware Diagram



Appendix B

Figures

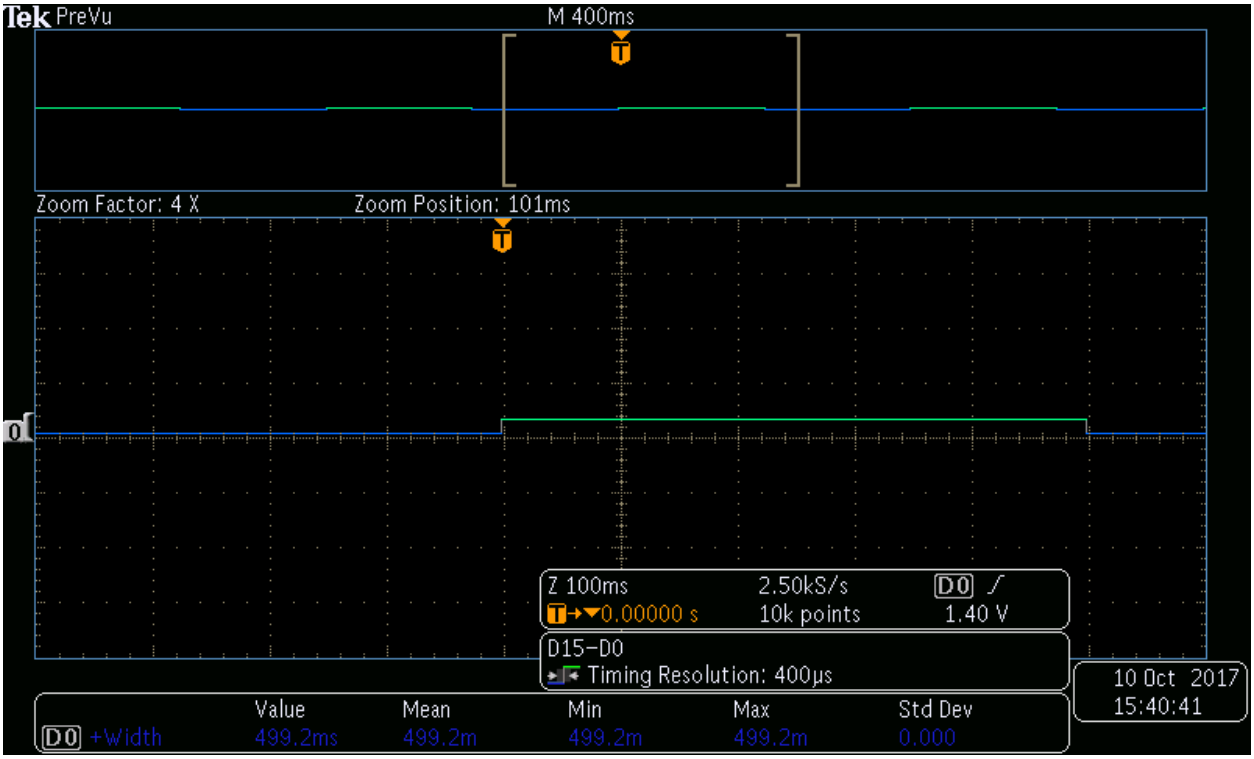


Figure 1: 499.2 ms delay time

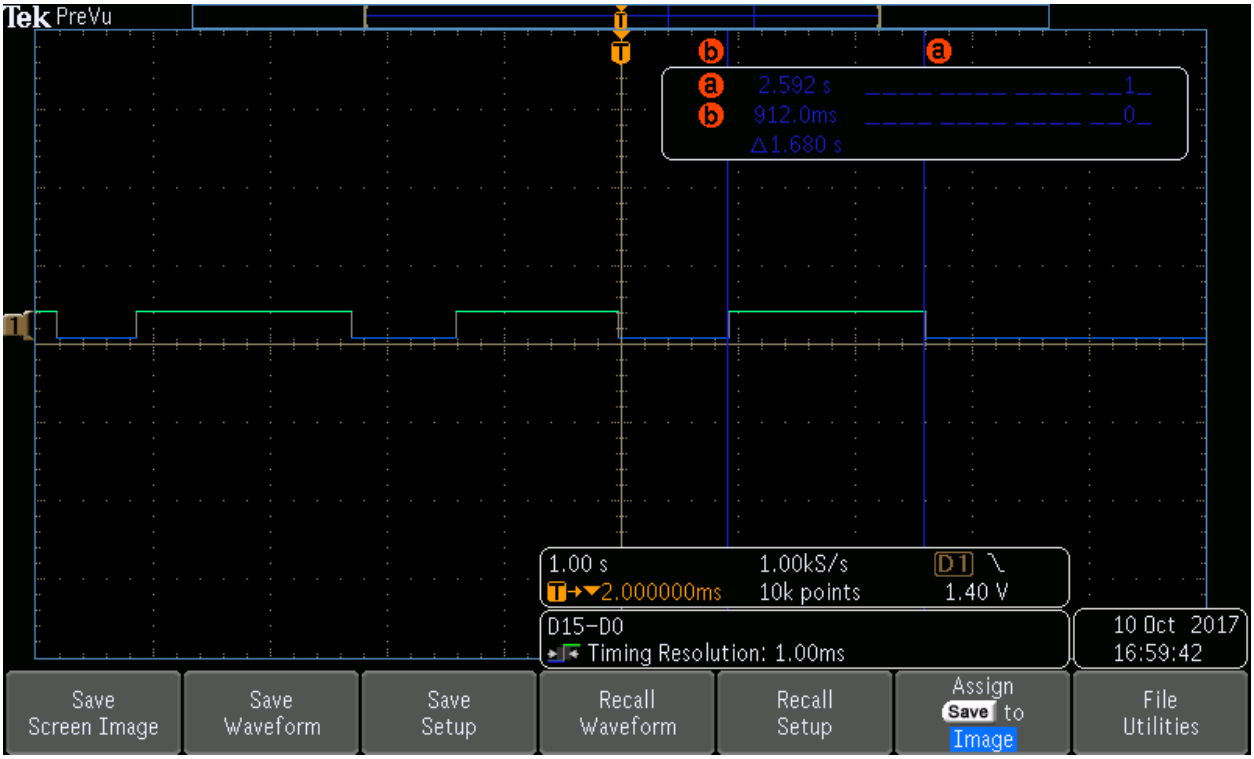


Figure 2: First random value with time of 1.680 seconds

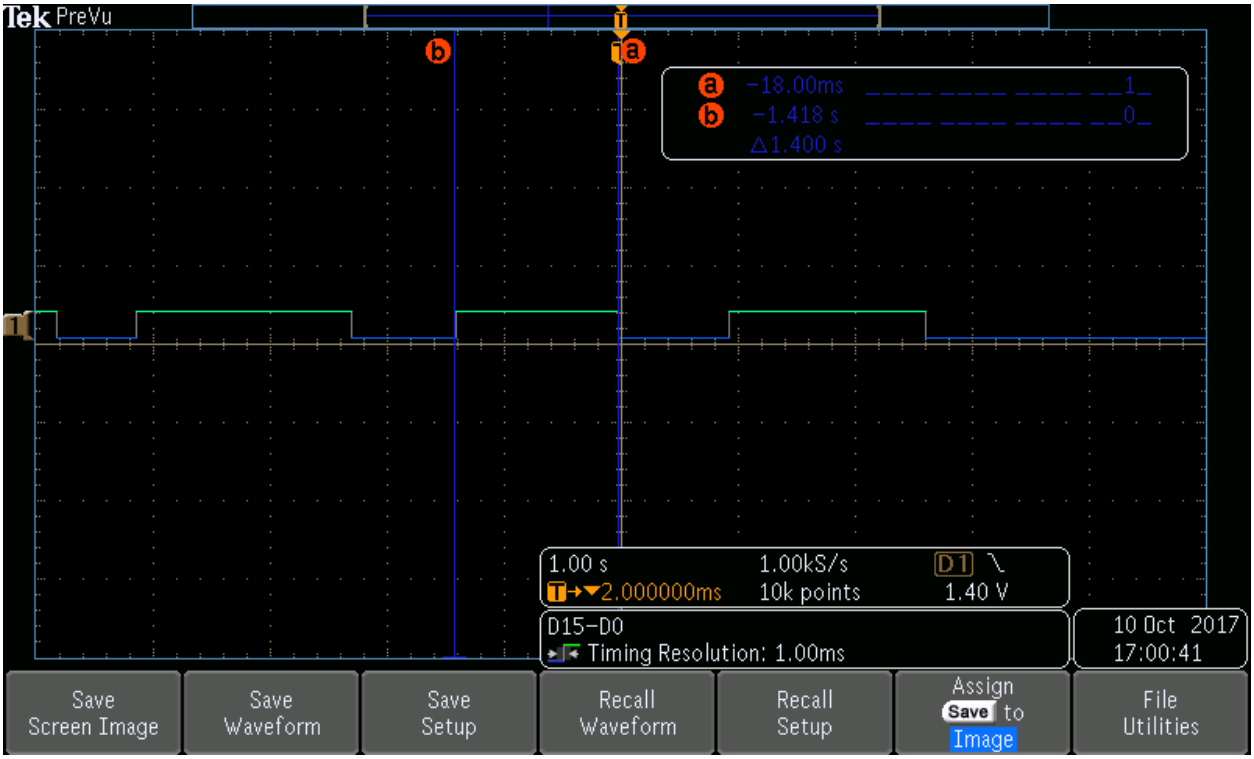


Figure 3: Second random value with time of 1.400 seconds

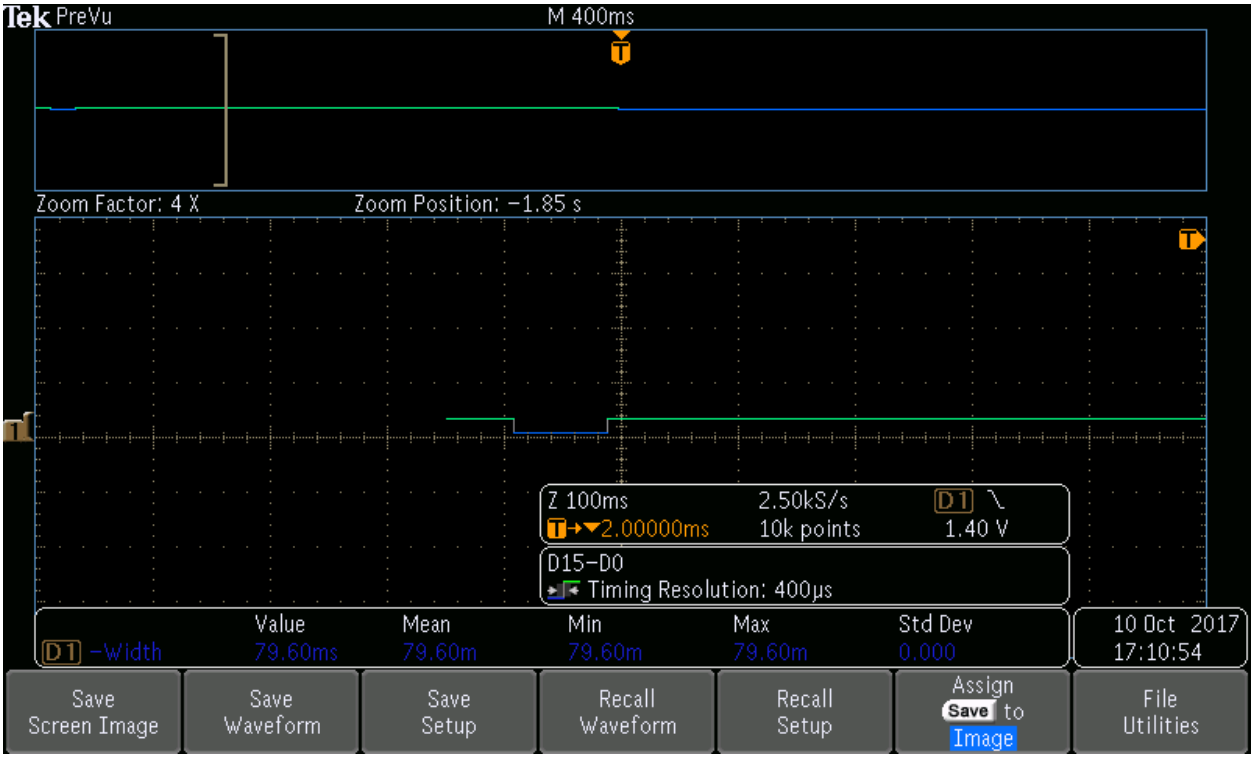


Figure 4: Player 2, 1 draw, DIP switch set 1, expected value of 80ms

Figure 5: Player 1, 2 draws, DIP switch set 2, expected value of 60ms

Appendix C

```
standby
        while p1!ready && p2!ready        ;p1 button pe2, p2
button pe3
                if(p1!ready) blink p1
                if(p2!ready) blink p2
                p1.sn = dip[0,1]
                p2.sn = dip[2,3] (lsr(2))
        end while
end standby

MainLoop
        timer(rand(1,2))
        while(!timer.done)
                if(p1.pressed || p2.pressed) goto MainLoop
        end while
        move p2 right
        move p1 left
        if(p1.pressed)
                move p1 right
```

```
                playerturn = p2
                timer(2^(-min(draws,4))*(320 - 80*p2.sn))
        else if(p2.pressed)
                move p2 left
                playerturn =p1
                timer(2^(-min(draws,4))*(320 - 80*p1.sn))
        end if
        while3 (!timer.done)
                if(p1.pressed && playerturn == p1)
                        move p1 right
                        draw++
                        goto MainLoop
                end if
                if(p2.pressed && playerturn == p2)  //skip p1
                        move p2 left
                        draw++
                        goto Main loop
                end if
        end while
        draw = 0
        if(playerturn == p1) ;p1 missed their turn
                move p2 left
        end if
        else ;p2 missed their turn
                move p1 right
        end if
        if(p1 on left edge || p2 on right edge) goto gameOver
        goto MainLoop
end MainLoop

gameOver
        blink p1, p2 current positions
```

Appendix D
Assembly Code

```
        THUMB
        ;AREA    DATA, ALIGN=2
        AREA    |.text|, CODE, READONLY, ALIGN=2
        EXPORT  Start
         ALIGN
         ;unlock 0x4C4F434B

         ;PF4 is SW1
         ;PF0 is SW2
         ;PF1 is RGB Red
         ;Enable Clock RCGCGPIO p338
         ;Set direction 1 is out 0 is in. GPIODIR
         ;DEN
         ; 0x3FC

         ;GPIO Port B (APB): 0x4000.5000
         ;GPIO Port E (APB): 0x4002.4000


sysTick equ 0xe000e000

port_B equ 0x40005000 ;Port B
port_E equ 0x40024000
port_C equ 0x40006000
TMW0 equ 0x40036000
TMW1 equ 0x40037000
RCGCTimer equ 0x400FE000

Start
     mov32 R0, #0x400FE108 ; Enable GPIO Clock
     mov R1, #0x16 ;clock for port b,c,e
     str R1, [R0]

     mov32 r0, RCGCTimer
     mov r1, #0x3 ;for timer 0 and 1
     str r1, [r0, #0x65c] ;enable clock for timerwide RCGCWTIMER
pg. 355
```

```
     mov32 R0, port_C
     mov R1, #0x0
     str R1, [R0, #0x420] ;turn off alt function
     mov R1, #0x0F
     str R1, [R0, #0x400] ;Port C input 4-7
     mov R1, #0xf0
     str R1, [R0, #0x51c] ;Port C pin 4-7 on

     mov32 r0, TMW1
     mov r1, #0x0
     str r1, [r0, #0x00c] ;turn off timer
     mov r1, #0x4
     str r1, [r0] ;32 bit wide timer
     mov r1, #0x2
     str r1, [r0, #0x004] ;set timer to periodic
     mov32 r1, #0x007a1200 ;load value, 0.5 second
     str r1, [r0, #0x028] ;write to set reload value, read to
get current value
     mov r1, #0x1
     str r1, [r0, #0x00c] ;enable timer

     mov32 R0, port_B ;Port B
     mov R1, #0x0
     str R1, [R0, #0x420] ;turn off alt function, may need
unlock
     mov R1,#0xFF
     str R1,[R0,#0x400] ;Port B output 0-7
     mov R1,#0xFF
     str R1,[R0,#0x51C] ;Port B pin 0-7 on


     mov32 R0, port_E ;Port E
     mov R1, #0x0
     str R1, [R0, #0x420] ;turn off alt function, may need
unlock
     mov R1,#0x03
     str R1,[R0,#0x400] ;Port E output 0-1, input 2-3
     mov R1, #0x0c
     str R1, [R0,#0x510] ;pull up
     mov R1,#0x0f
```

```
     str R1,[R0,#0x51C] ;Port E pin 0-3 on

     mov R9, #0x0 ;draw counter
     mov R11, #0x20 ;player 1 position
     mov R12, #0x10 ;player 2 position
     mov r2, #0 ;ready states of p1[0] and p2[1]
     mov32 r0, port_B
     mov r1, #0x30
     mvn r1, r1
     str r1, [r0, #0x3fc] ;turns on center leds
     mov32 r0, port_E
     mov r1, #0x3
     str r1, [r0, #0xc]

STANDBY

     mov32 r0, port_B
     ldr r1, [r0, #0xc0] ;load current values of led 4,5
     mvn r1, r1
     mvn r3, r2
     and r1, r3, lsl #2
     str r1, [r0, #0xc0] ;set led 4,5 to inverted value

     mov32 r3, #0x7a1200 ; 0.5 Seconds
     bl TIMER

blinkDelay
     mov32 r0, port_E
     ldr R1, [R0, #0x30] ;get current buttons pressed
     eor r1, #0xc
     orr r2, r1 ;refresh ready states of players
     cmp r2, #0x0c ;p1 and p2 are ready, might be #0xc
     beq START_GAME ;start game

    mov32 r0, TMW0
     ldr r1, [r0, #0x01c] ;poll this to check if done, 0th bit
     cmp r1, #0x1
     bne blinkDelay
     mov r1, #0x1
     str r1, [r0, #0x024] ;clear status(done) pin,
     b STANDBY
```

```
TIMER
     push {r0}
     push {r1}

     mov32 r0, TMW0
     mov r1, #0x0
     str r1, [r0, #0x00c] ;turn off timer
     mov r1, #0x4
     str r1, [r0] ;32 bit wide timer
     mov r1, #0x2
     str r1, [r0, #0x004] ;set timer to periodic
     str r3, [r0, #0x028] ;load r3 value into timer
     mov r1, #0x1
     str r1, [r0, #0x00c] ;enable timer

     pop {r1}
     pop {r0}
     bx lr

UPDATE_LED
     push{r0}
     push{r3}

     orr r3, r11, r12 ;which leds turn on
     mvn r3, r3
     mov32 r0, port_B
     str r3, [r0, #0x3fc]
     lsr r3, #8
     mov32 r0, port_E
     str r3, [r0, #0xc]

     pop{r3}
     pop{r0}
     bx lr

RND12
     push{r2}
    push{r1}
     push{r0}

     mov32 r0, TMW1
```

```
     ldr r1, [r0, #0x050] ;get current timer1 value
     mov r2, #0x2
     mul r1, r2
     mov32 r2, #0xf42400 ;1 sec
     add r3, r1, r2 ;rng 1-2

     pop{r0}
    pop{r1}
     pop{r2}
     bx lr

START_GAME
     mov32 r0, port_B
     mov r1, #0xcf ; cf=1100 1111, turn on center led
     str r1, [r0, #0xc0] ;load current values of led 4,5

Loop
     bl RND12 ;get random value 1-2 and store in r3
     bl TIMER ;use value of r3 from rnd12
WHILE2 ;while(!timer.done)
     mov32 r0, port_E
     ldr R1, [R0, #0x30] ;get current buttons pressed
     eor r1, #0xc
     cmp r1, #0x0000
     bne Loop ;player pressed button early

     mov32 r0, TMW0

     ldr r1, [r0, #0x01c] ;poll timer
     cmp r1, #0x1 ;is timer done
     bne WHILE2 ;end while(!timer.done)
     mov r1, #0x1
     str r1, [r0, #0x024] ;store 1 to this to clear status(done)
pin, do this after polling

     lsl r11, #1 ;move p1 left 1
     lsr r12, #1 ;move p2 right 1
     bl UPDATE_LED

GET_FIRST_TURN
     mov32 r0, port_E
```

```
     ldr R1, [R0, #0x30] ;get current buttons pressed
     eor r1, #0xc
     cmp r1, #0x8
     bne CHECK_P2 ;player 1 button not pressed
     bl PLAYER1_FIRST
    b WHILE3
CHECK_P2
     cmp r1, #0x4
     bne GET_FIRST_TURN ;recheck buttons, p2 button not pressed
     bl PLAYER2_FIRST

WHILE3 ;while(!timer.done)

     mov32 r0, port_E
     ldr r1, [r0, #0x20] ;get button p1
     eor r1, #0x8 ;inverse button, 1 on 0 off
     lsr r1, #3 ;move p1 bit to first position
    cmp r1, r10 ;check if it was p1's turn
    bne skipP1
    ;if(p1.pressed && playerturn ==p1)
     lsr r11, #1 ;move p1 right
    bl UPDATE_LED
    add r9, #1 ;draw++
    b Loop
skipP1
    ldr r1, [r0, #0x10] ;get button p2
     eor r1, #0x4 ;inverse button, 1 on 0 off
     lsr r1, #1 ;move p2 bit to second position
    cmp r1, r10 ;check if it was p2's turn
    bne skipP2
    ;if(p2.pressed && playerturn ==p2)
    lsl r12, #1 ;move p2 left
    bl UPDATE_LED
    add r9, #1 ;draw++
    b Loop
skipP2 ;no buttons pressed

     mov32 r0, TMW0
     ldr r1, [r0, #0x01c] ;poll this to check if done, 0th bit
     cmp r1, #0x1
     bne WHILE3
```

```
    mov r1, #0x1
     str r1, [r0, #0x024] ;store 1 to this to clear status(done)
pin, do this after polling

    mov r9, #0 ;reset draw to 0
    cmp r10, #0x1
    bne p2Turn
    lsl r12, #1 ;move p2 left
    bl UPDATE_LED
    b checkWin
p2Turn
    lsr r11, #1 ;move p1 right 1
    bl UPDATE_LED
    b checkWin

checkWin
    cmp r11, #0x200 ;check if p1 on left edge
    beq BLINK
    cmp r12, #0x1 ;check if p2 on right edge
     beq BLINK
     b Loop

BLINK
    mov r4, r11
    mov r5, r12
blink1
    eor r11, r4 ;flip r11
    eor r12, r5 ;flip r12
    bl UPDATE_LED
    mov32 r3, #0x7a1200 ;0.5 sec
    push{lr}
    bl TIMER ;delay of 0.5 sec
    pop{lr}
    mov32 r0, TMW0
blinkDelay1
     ldr r1, [r0, #0x01c] ;poll this to check if done, 0th bit
     cmp r1, #0x1
     bne blinkDelay1 ;delay of 0.5 sec
     mov r1, #0x1
     str r1, [r0, #0x024] ;store 1 to this to clear status(done)
pin, do this after polling
```

```
    b blink1 ;endlessly loop until reset


PLAYER1_FIRST

     lsr r11, #1 ;move p1 right 1
    push{lr}
    bl UPDATE_LED
    pop{lr}
     mov r10, #0x2 ;wait on p2
     mov32 r0, port_C
     ldr r3, [r0, #0x300] ;get pins 6,7, get p2 delay
     lsr r3, #6
    push{lr}
     bl COMPUTE_PLAYER_DELAY
    pop{lr}
     mov r3, r5 ;store value r5 from COMPUTE_PLAYER_DELAY into
R3 for timer
    push{lr}
     bl TIMER
    pop{lr}
     bx lr

PLAYER2_FIRST

     lsl r12, #1 ;move p2 left 1
    push{lr}
    bl UPDATE_LED
    pop{lr}
     mov r10, #0x1 ;wait on p1
     mov32 r0, port_C
     ldr r3, [r0, #0x0c0] ;get pins 4,5, get p1 delay
     lsr r3, #4
    push{lr}
     bl COMPUTE_PLAYER_DELAY
    pop{lr}
     mov r3, r5 ;store value r5 from COMPUTE_PLAYER_DELAY into
R3 for timer
    push{lr}
     bl TIMER
    pop{lr}
```

```
     bx lr

COMPUTE_PLAYER_DELAY
    push{r2}
     ;r3 is the current switch value
     ;r9 is current draw number
    mov r2, #0x50; 80
     mul r3, r2 ;p.sn*80
     mov r5, #0x140 ;320
     sub r5, r3 ;r5=320 - 80*p.sn

     mov r3, #0x4
     cmp r3, r9 ;if(4>=draws)
     ble continue1 ;else r3=4
     mov r3, r9 ; if(4>=draws) r3=num draws
continue1
     lsr r5, r3     ;left shift r5 by min(4,draws)

    mov32 r2, #0x3e80 ;16,000
    mul r5, r2 ;value of r5 * 16,000 to get time in ms
    pop{r2}
     bx lr ;END COMPUTE_PLAYER_DELAY, return value in r5

   ALIGN
   END
```