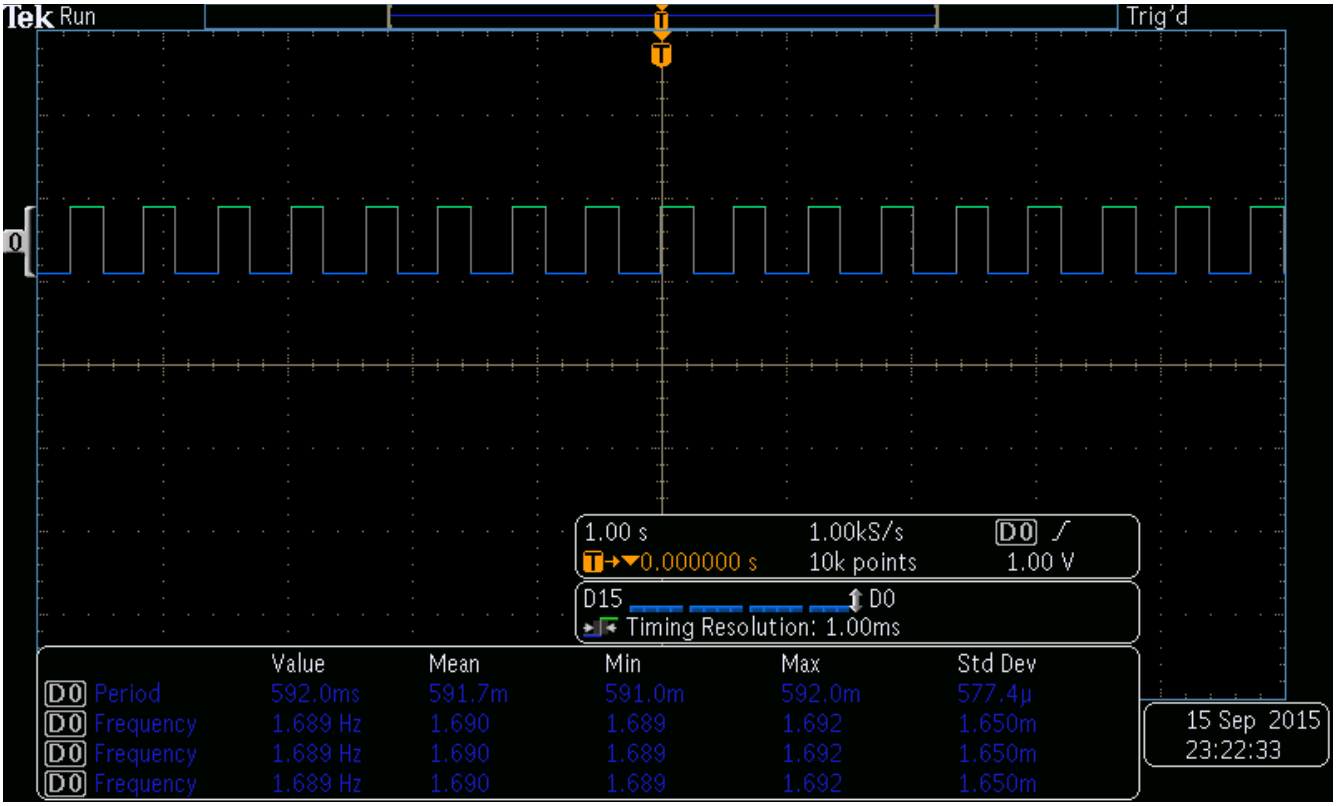
LAB REPORT 1
Dr. Ryan Gerdes
ECE 3710
By: Jonathan Tousley
Partner: Aaron Kunz

Blinky.asm:

For this section I got to analyze the assembly code that switched the LED between different colors. So that each state could be visible to the human eye, the code used a delay loop for each state. The time required for each loop was about 300ms as measured by the logic analyzer. Calculating the same with straight arithmetic yielded about 260ms – which is a fairly high amount of error but it's okay because of the amount of branching involved. Here's a screenshot of the logic analyzer:



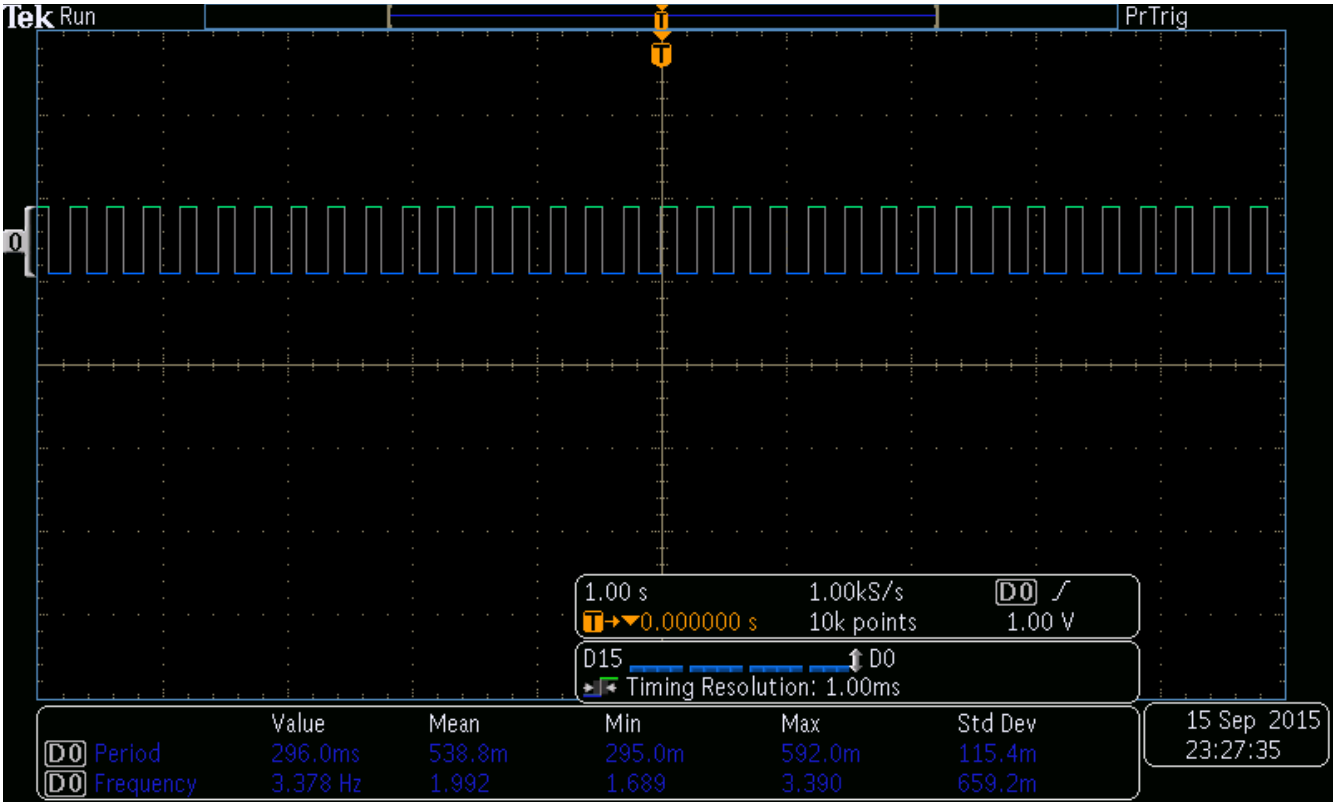
After this, the task was to make the frequency twice as fast as before. This was accomplished by modifying this line:

MOV32 R5, #0xFFFFF

into this line:

MOV32 R5, #0x7FFFF

The screenshot looks like this:



And the assembly code for the program looks like this:

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

GPIO_CLK    DCD 0x400FE018     ;address for GPIO Clock
CLKVAL              DCD 0x0020  ;value for enabling gpios clock
UNLOCK              DCD 0x4C4F434B ; gpio unlock code.
GPIOF       DCD 0x40025000  ;base address for one of the GPIOs
GPIOF_PINS  DCD 0xE       ;value for configuring the gpio
                    ;LIST the values toggled by the setup value.
                    ;i.e. gpio enable, pull up register, etc.
            ALIGN

Start
        mov32 R0, #0x400FE108 ; Enable GPIO Clock
        mov R1, #0x20
        str R1, [R0]

        mov32 R0, #0x40025000 ;GPIOF address

        ;unlock GPIOF
        mov32 R1, #0x4C4F434B; GPIO Unlock code.
        str R1, [R0,#0x520];
```

```
        mov R1, #0x1F
        str R1, [R0,#0x524];GPIOCR
        mov R1, #0x11
        str R1, [R0,#0x510]
        mov R1, #0x0F
        str R1, [R0,#0x400] ;GPIODIR
        mov R1, #0x1F
        str R1, [R0,#0x51C] ;digital enable
loop
        MOV32 R1, #0x02    ;load value for turning on LED color RED
        STR R1, [R0,#0x38]  ;write the above value to GPIOF ODR register.

        MOV R4, #0                  ;initial value for iteration loop
        MOV32 R5, #0x7FFFF          ;number of iterations for delay loops
delay1
        ADD R4, #1                  ; increment the loop counter
        CMP R4, R5                  ;check number of iterations
        BLE delay1                  ;continue if iterated less than 0xFFFFF + 1 times, otherwise repeat
delay loop

        MOV32 R1, #0x08    ;load value for turning on LED color BLUE
        STR R1, [R0, #0x38] ;write the above value to GPIOF ODR

        MOV R4, #0          ;initial value for iteration loop
        ;                   ; **** the tm4c123gh6pm has 16 MHz clock.
        ;;                   how long should the loop take with that clock? 260ms

delay2
        ADD R4, #1                  ;increment the loop counter
        CMP R4, R5                  ;check number of iterations
        BLE delay2                  ;continue if iterated less than 0xFFFFF + 1 times, otherwise repeat
delay loop

        MOV32 R1, #0x04    ;load value for turning on LED GREEN
        STR R1, [R0, #0x38] ;write the above value to GPIOF ODR

        B loop ;do it all over again, forever
        ALIGN
        END
```

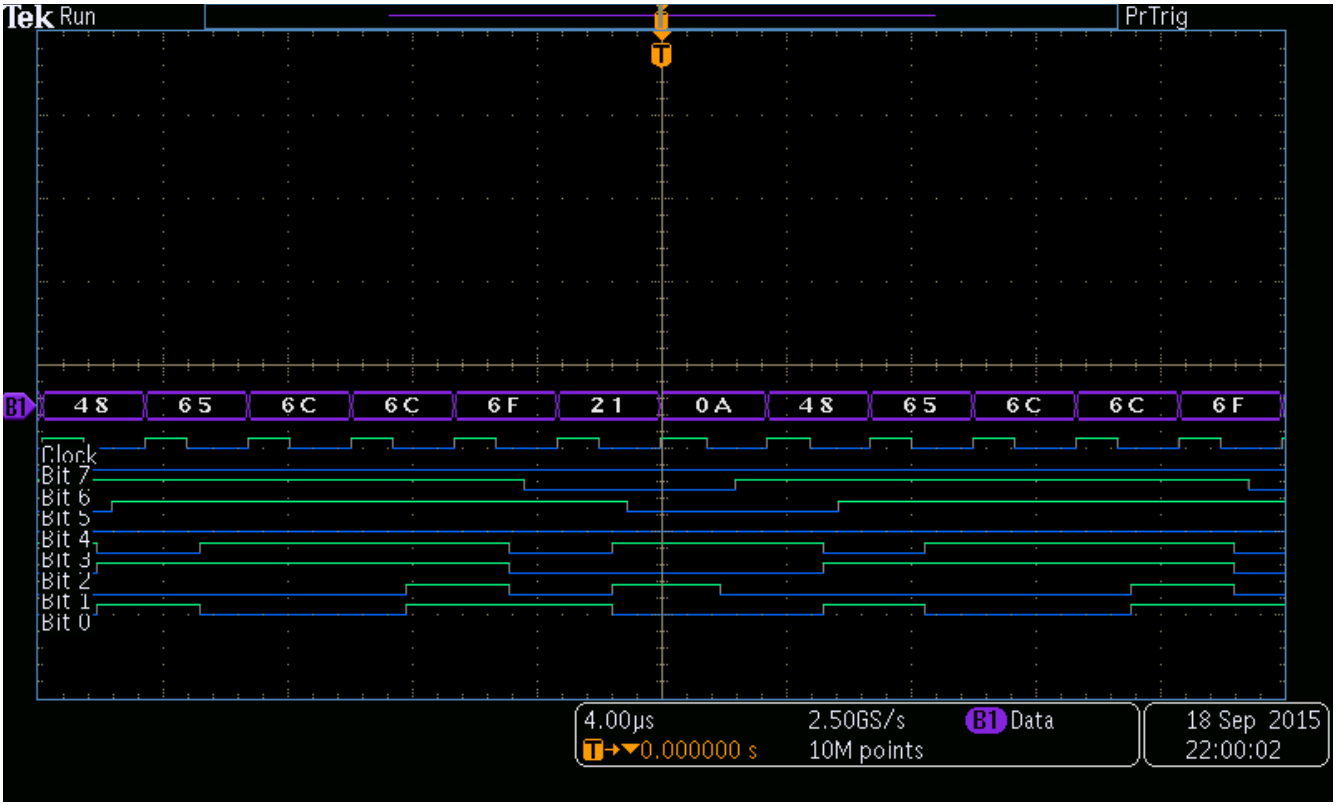
Code Debugging:

This program was intended to copy some data from read-only memory into the SRAM section. However, it failed on both accounts. The counter was initialized inside the loop (resulting in an infinite loop) and the destination in memory it was supposed to write to was the top of the stack (no write permission). Therefore, the code needed to be modified to fix both of those issues, the solution for the latter requiring the Axiom of Choice. The resulting code looked like this:

```
                AREA main, CODE, READONLY, ALIGN=2
                THUMB
                EXPORT Start


message         DCB     "Hello Students!",0             ; message stored readonly

                ALIGN ;pg149
Start           LDR   R0, =message          ; load address of the message

                MOV   R1, SP                ; load memory location to store
                SUB   R1, R1, #0x400        ; place in memory to write (#axiomOfChoice)
                MOV   R3, #4                ; used as a counter

load            LDR   R2,[R0]               ; load a word of the message
                ADD   R0, R0, #4            ; adds 4 to the pointer

                SUB   R3, R3, #1            ; decrements counter
                STR   R2,[R1,#4]            ; store the word to memory, inc R1 by 4
                ADD   R1, R1, #4            ; increment memory location
                CMP   R3, #0                ; Check for null
                BNE   load                  ; repeat if not null terminated

loop            B       loop

                ALIGN
                END
```

Logic Analyzer:

For the last section of this lab the string “Hello!\n” was loaded onto the board and then outputted sequentially into the logic analyzer. The analyzer was configured to trigger upon receiving '\n', causing the hex values of the string to be displayed on the logic analyzer. It looked like this:



“Hello\n” in hexadecimal is 48 65 6c 6c 6f 21 0a.