1. Introduction  
   In this lab the objective was to create a time delay subroutine without the use of sysTick then to use that to implement a blinking LED design on the MSP432. By doing accomplishing this task an understanding of clock cycle timing can be attained, which should be in bigger and more complex embedded systems. Most utility applications should have timing mechanisms of some sort.
2. Procedure/Discussion  
   For the time delay subroutine, the basic design concept was to create a loop that would continuously loop into itself until a certain amount of clock cycles were made, from which it would loop to the condition checks for LED logic. To count the amount of clock cycles, the amount of cycles needed to loop would divide the amount of cycles that would elapse in 100ms to attain the amount of loops needed to exit the subroutine. In this case, 4 cycles would elapse per loop. The condition check to exit the delay loop would check a register that would subtract once every loop. This register would have an initial value of 75000 and would reset upon exiting the loop.   
   To have a 1ms delay instead of 100ms, the register would have a value of 750 cycles. This modification is simply changing an assignment statement so it has the advantage of being low effort. This method however will require recalculation if the delay subroutine is changed in anyway.  
   The logic for the LED was derived from the previous lab, simple checking of a switch(P1.4) into modification of the specific LED pin for P1OUT (P1.0)
3. Conclusions  
   The proposed design ended up functioning exactly as required for the lab specifications. The LED blinked at an appropriate rate and the timing analysis within code composer showed that the delay was barely over 300 thousand cycles which was the target. In more sophisticated systems, the risks from improper timing are most definitely more severe thus it is important to have precise timings on functions.   
   Once again I found that modifying template code was frustrating due to lacking intuition on the code’s existing logic. I would definitely consider retyping using the template as a reference to tailor the code to personal tastes next time I work on a lab like this.
4. Appendix

Start

P1.4 Pressed

P1.4 not pressed

LED toggle

LED off

Delay Subroutine

Set LED as off and delay constant

Initialize Ports

Pseudocode:

Main initialize port 1;

P1.4 as input and P1.0 as output in P1DIR

P1.4 of P1REN to 1 for internal resistor

Set P1.4 as pull-up in P1OUT

Set register used for delay to calculated constant value for intended delay

Loop call delay subroutine

Subtract delay register by 1

Check to delay register to see if zero

Call return to loop function if zero, keep looping if not.

Read switch for input

If P1.4 = 0, toggle P1.0

Else P1.0 LED turned off

Jump to Loop

.thumb

**.text**

**.align** 2

P1IN .field 0x40004C00,32 ; Port 1 Input

P2IN .field 0x40004C01,32 ; Port 2 Input

P2OUT .field 0x40004C03,32 ; Port 2 Output

P1OUT .field 0x40004C02,32 ; Port 1 Output

P1DIR .field 0x40004C04,32 ; Port 1 Direction

P2DIR .field 0x40004C05,32 ; Port 2 Direction

P1REN .field 0x40004C06,32 ; Port 1 Resistor Enable

P2REN .field 0x40004C07,32 ; Port 2 Resistor Enable

P1DS .field 0x40004C08,32 ; Port 1 Drive Strength

P2DS .field 0x40004C09,32 ; Port 2 Drive Strength

P1SEL0 .field 0x40004C0A,32 ; Port 1 Select 0

P2SEL0 .field 0x40004C0B,32 ; Port 2 Select 0

P1SEL1 .field 0x40004C0C,32 ; Port 1 Select 1

P2SEL1 .field 0x40004C0D,32 ; Port 2 Select 1

Delay .field 0x124F8, 32 ; Amount of loops needed to hit 100ms

;RED .equ 0x01

;GREEN .equ 0x02

;BLUE .equ 0x04

SW1 .equ 0x02 ; on the left side of the LaunchPad board

SW2 .equ 0x10 ; on the right side of the LaunchPad board

LED .equ 0x01

**.global** main

.thumbfunc main

**main:** .asmfunc

BL Port1\_Init ; initialize P1.1 and P1.4 and make them inputs (P1.1 and P1.4 built-in buttons)

LDR R4, Delay

; BL Port2\_Init ; initialize P2.2-P2.0 and make them outputs (P2.2-P2.0 built-in LEDs)

loop

BL DelayWait

LDR R4, Delay

BL Port1\_Input ; read both of the switches on Port 1

**CMP** R0, #0x02 ; R0 == 0x02?

BEQ sw2pressed ; if so, switch 2 pressed

**CMP** R0, #0x12 ; R0 == 0x12?

BEQ nopressed ; if so, neither switch pressed

; if none of the above, unexpected return value

;MOV R0, #(RED+GREEN+BLUE) ; R0 = (RED|GREEN|BLUE) (all LEDs on) ; turn all of the LEDs on

**B** loop

sw2pressed

LDR R1, P1OUT

LDRB R0, [R1]

EOR R0, #0x01

BL Port1\_Output ; turn the red LED on

**B** loop

nopressed

;MOV R0, #0 ; R0 = 0 (no LEDs on)

;BL Port2\_Output

BIC R0, #0x01

BL Port1\_Output ; turn all of the LEDs off

**B** loop

DelayWait

**SUB** R4, R4, #0x01

**CMP** R4, #0x00

BNE DelayWait

BX LR

.endasmfunc

;------------Port1\_Init------------

; Initialize GPIO Port 1 for negative logic switches on P1.1 and

; P1.4 as the LaunchPad is wired. Weak internal pull-up

; resistors are enabled.

; Input: none

; Output: none

; Modifies: R0, R1

**Port1\_Init:** .asmfunc

LDR R1, P1SEL0

**MOV** R0, #0x00 ; configure P1.4 and P1.1 as GPIO

STRB R0, [R1]

LDR R1, P1SEL1

**MOV** R0, #0x00 ; configure P1.4 and P1.1 as GPIO

STRB R0, [R1]

LDR R1, P1DIR

**MOV** R0, #0x01 ; make P1.4 and P1.1 inputs (and P1.0 as output)

STRB R0, [R1]

LDR R1, P1REN

**MOV** R0, #0x12 ; enable pull resistors on P1.4 and P1.1

STRB R0, [R1]

LDR R1, P1OUT

ORR R0, #0x12 ; P1.4 and P1.1 are pull-up

STRB R0, [R1]

BX LR

.endasmfunc

;------------Port1\_Input------------

; Read and return the status of the switches.

; Input: none

; Output: R0 0x10 if only Switch 1 is pressed

; R0 0x02 if only Switch 2 is pressed

; R0 0x00 if both switches are pressed

; R0 0x12 if no switches are pressed

; Modifies: R1

**Port1\_Input:** .asmfunc

LDR R1, P1IN

LDRB R0, [R1] ; read all 8 bits of Port 1

**AND** R0, R0, #0x12 ; select the input pins P1.1 and P1.4

BX LR

.endasmfunc

;---------------------------------------

**Port1\_Output:** .asmfunc

LDR R1, P1OUT

ORR R0, #0x12

STRB R0, [R1]

BX LR

.endasmfunc

.end