2DX4: Microprocessor System
Lab 7

Instructor: Drs.Boursalie, Doyle, Haddara

Lab TAs: Fatemeh Derakshani (derakhsf), Han Zhou (zhouh115)

Junbo(Frank) Wang – L01 – wangj430

Yichen Lu – L01 – luy191

As a future member of the engineering profession, the student is responsible for performing the required work in an honest manner, without plagiarism and cheating. Submitting this work with my name and student number is a statement and understanding that this work is our own and adheres to the Academic Integrity Policy of McMaster University and the Code of Conduct of the Professional Engineers of Ontario. Submitted by **[Junbo Wang wangj430 400249823]**

As a future member of the engineering profession, the student is responsible for performing the required work in an honest manner, without plagiarism and cheating. Submitting this work with my name and student number is a statement and understanding that this work is our own and adheres to the Academic Integrity Policy of McMaster University and the Code of Conduct of the Professional Engineers of Ontario. Submitted by **[Yichen Lu luy191 400247938]**

1. Purpose

The purpose of this lab is to gain experience using event based programming. With editing the codes in software, we can view different outcomes from the devices.

2. BackGround

In this lab, two milestones are related to the interrupt. In studios 7A and 7B, it introduces the GPIO interrupts and periodic interrupts. Interrupt is an event that stops program execution on the CPU and performs the services associated with that event. In GPIO interrupt, we arm the interrupt source, enable interrupt in NVIC and the Global interrupt for the specific port, and set interrupt priority of that register. For the periodic interrupt, it is requested periodically on a fixed time basis. We can generate these interrupts by editing the SysTick in software, and the results are shown through the waveform generator.

3. Method

Milestone 1

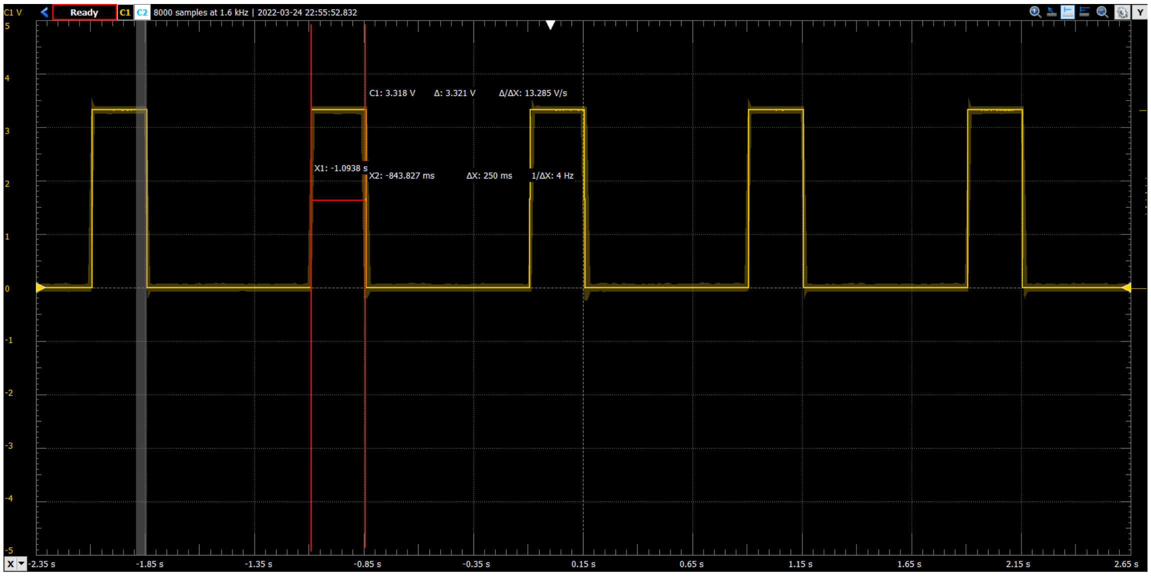
In milestone 1, we implement a periodic interrupt that triggers every 1s and flashes an onboard LED for 250 ms. Through the calculation, we get the period will be in 1us units, so the period value should be 1000000 to satisfy the 1 second interrupt. Also, we change the value of function SysTick_Wait to 3000000 to let the onboard LED flash 250ms due to 10ms corresponding to 1200000.

Milestone 2

In milestone 2, we use the relevant studio code to transmit data to the ToF device in order to implement the GPIO interrupt. We created a function called **Button_Init()** that contains all the data values for GPIO PortJ. Initially, the peripheral device (Time-of-Flight) address is set to 0x29 and one byte of data is 0x00. For the data transmission, we modify the one byte value to 0x56.

4. Results

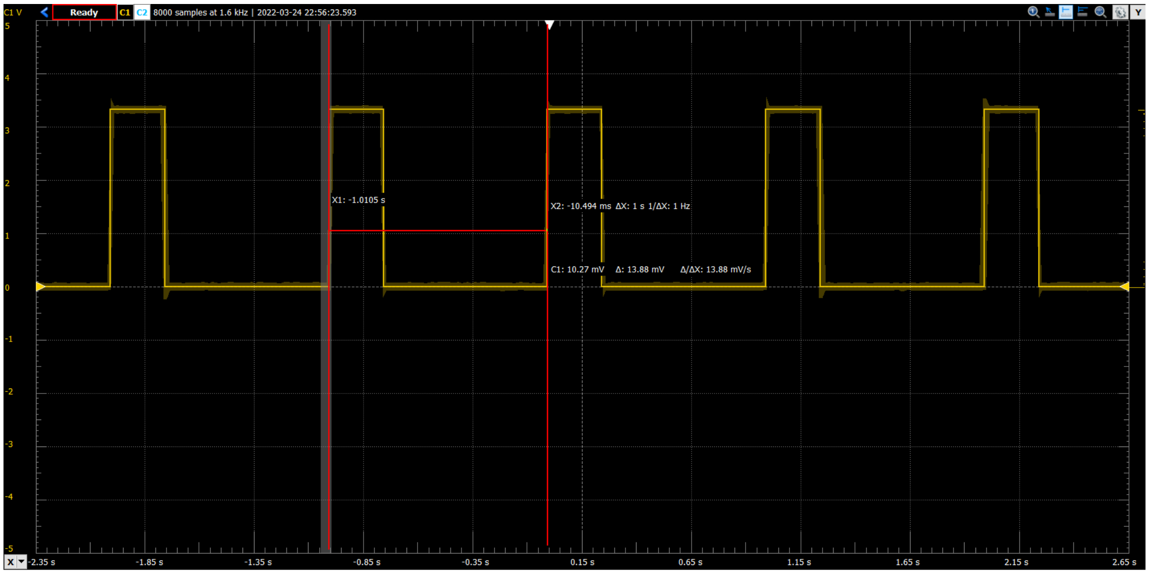
In milestone 1, we ran the code successfully and saw the onboard LED D2 blink for 250ms with 1s interrupt. Also, we generated a square wave on the Waveform. According to the measurement, we saw that the interrupt time was 1 second and the LED flash time was 250ms which meets our requirement. For milestone 2, we opened the protocol and logic sections in the Waveform after the code was run. We pushed the onboard button PJ1 to toggle the state of an LED. In Waveform, we observe that data 0x56 was transmitted in the protocol and logic.



Ready | 8000 samples at 1.6 kHz | 2022-03-24 22:55:52.832

C1: 3.318 V Δ: 3.321 V Δ/ΔX: 13.285 V/s

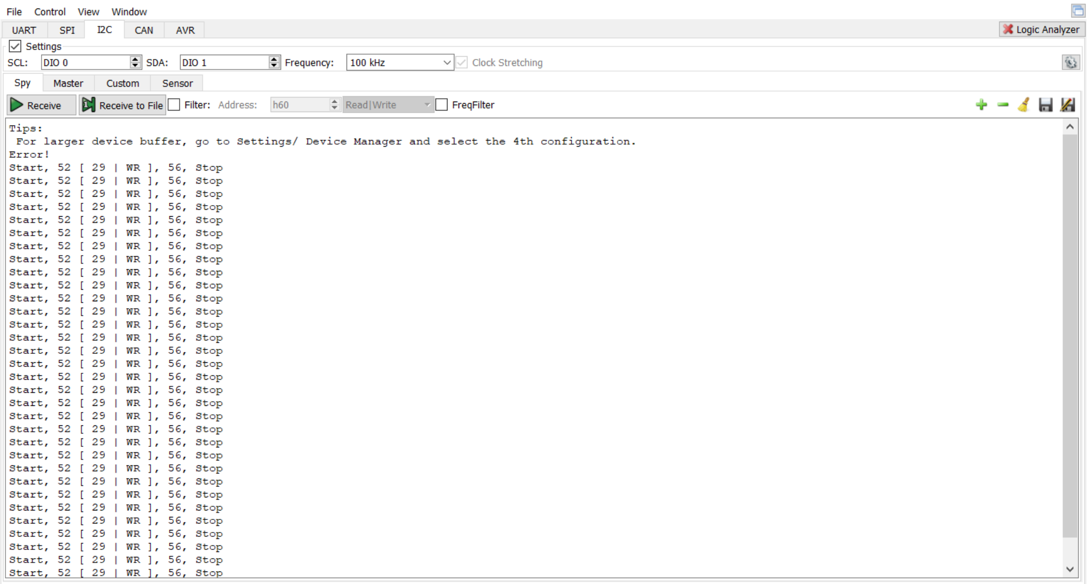
X1: -1.0938 s X2: -843.827 ms ΔX: 250 ms 1/ΔX: 4 Hz



Ready | 8000 samples at 1.6 kHz | 2022-03-24 22:56:23.593

X1: -1.0105 s X2: -10.494 ms ΔX: 1 s 1/ΔX: 1 Hz

C1: 10.27 mV Δ: 13.88 mV Δ/ΔX: 13.88 mV/s

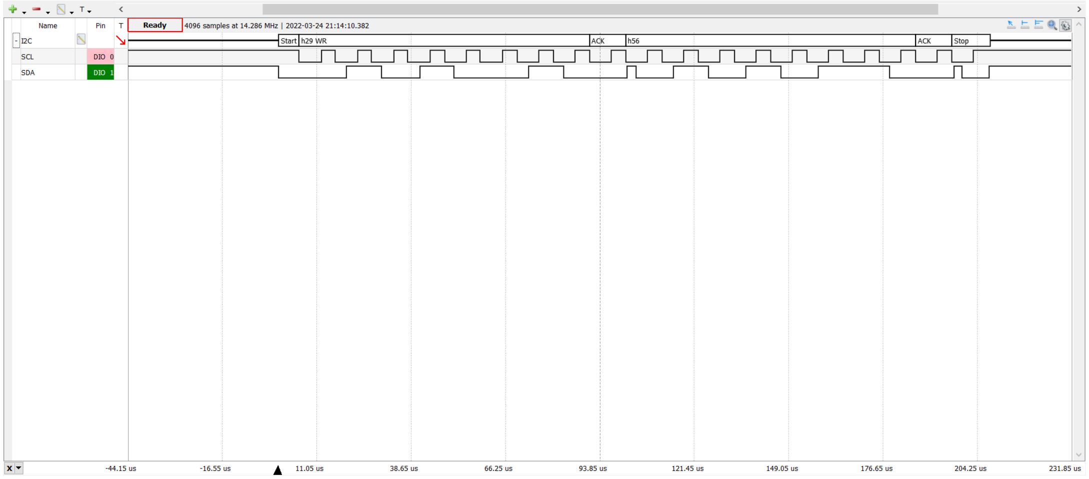


☑ Settings

SCL: DIO 0 SDA: DIO 1 Frequency: 100 kHz ☑ Clock Stretching

□ Filter: Address: h60 Read|Write □ FreqFilter

```
Tips:
 For larger device buffer, go to Settings/ Device Manager and select the 4th configuration.
Error!
Start, 52 [ 29 | WR ], 56, Stop
Start, 52 [ 29 | WR ], 56, Stop
Start, 52 [ 29 | WR ], 56, Stop
Start, 52 [ 29 | WR ], 56, Stop
Start, 52 [ 29 | WR ], 56, Stop
Start, 52 [ 29 | WR ], 56, Stop
Start, 52 [ 29 | WR ], 56, Stop
Start, 52 [ 29 | WR ], 56, Stop
Start, 52 [ 29 | WR ], 56, Stop
Start, 52 [ 29 | WR ], 56, Stop
Start, 52 [ 29 | WR ], 56, Stop
Start, 52 [ 29 | WR ], 56, Stop
Start, 52 [ 29 | WR ], 56, Stop
Start, 52 [ 29 | WR ], 56, Stop
Start, 52 [ 29 | WR ], 56, Stop
Start, 52 [ 29 | WR ], 56, Stop
Start, 52 [ 29 | WR ], 56, Stop
Start, 52 [ 29 | WR ], 56, Stop
Start, 52 [ 29 | WR ], 56, Stop
Start, 52 [ 29 | WR ], 56, Stop
Start, 52 [ 29 | WR ], 56, Stop
Start, 52 [ 29 | WR ], 56, Stop
Start, 52 [ 29 | WR ], 56, Stop
Start, 52 [ 29 | WR ], 56, Stop
Start, 52 [ 29 | WR ], 56, Stop
Start, 52 [ 29 | WR ], 56, Stop
Start, 52 [ 29 | WR ], 56, Stop
Start, 52 [ 29 | WR ], 56, Stop
Start, 52 [ 29 | WR ], 56, Stop
Start, 52 [ 29 | WR ], 56, Stop
Start, 52 [ 29 | WR ], 56, Stop
Start, 52 [ 29 | WR ], 56, Stop
```



5. Observations and conclusions

In this lab, we created GPIO interrupts and periodic interrupts for these two milestones. For the periodic interrupt, we set the time interrupt and observe the onboard LED blinking in each period. Also, the graph from scope shows the correct interrupt time and flashing time which meets the requirement. For the GPIO interrupt, we use the value in GPIO PortJ to transmit data to the ToF device through I2C in Waveform. When we press the button on the microcontroller, we can see that one byte of data 0x56 was transmitted into the protocol and logic section. Therefore, we can verify the data transmission of GPIO interrupt and periodic interrupt through the scope, the protocol, and the logic in the Waveform.

6. Reference

Code Appendix

Milestone 1

```
/* Studio 7A-0 Project Code
	Sample code provided for studio to demonstrate periodic interrupt
	Based upon interrupt texxtbook code by Valvano.

	This code will use an onboard 16-bit timer to trigger an inteerupt.  The trigger
	will cause the timer ISR to execute.  The timer ISR will
	flash the onboard LED on/off.  However, you can adapt this code
	to perform any operations simply by changing the code in the ISR

	The program is written to flash the onboard LED at a frequency of 10Hz

	Written by Tom Doyle
	Last Updated: Feb 20, 2022, Hafez: Added some comments and reordered the initialization steps.
*/


// Name: Yichen Lu Junbo Wang
// Student id: 400247938 400249823
// Date: March, 21th, 2022

#include <stdint.h>
#include "tm4c1294ncpdt.h"
#include "PLL.h"
#include "SysTick.h"


//Flash D2
void FlashLED2(int count) {
	while(count--) {
	   GPIO_PORTN_DATA_R ^= 0b00000101;				//hello world!
	   SysTick_Wait10ms(10);					//.1s delay
	   GPIO_PORTN_DATA_R ^= 0b00000101;
	}
}

//Flash D1
void FlashLED1(int count) {
	while(count--) {
	   GPIO_PORTN_DATA_R ^= 0b00000010;				//hello world!
	   SysTick_Wait10ms(10);					//.1s delay
	   GPIO_PORTN_DATA_R ^= 0b00000010;
    }
}

// Initialize onboard LEDs
void PortN_Init(void){
  //Use PortN onboard LED
  SYSCTL_RCGCGPIO_R |= SYSCTL_RCGCGPIO_R12;              // activate clock for Port N
  while((SYSCTL_PRGPIO_R&SYSCTL_PRGPIO_R12) == 0){};  // allow time for clock to stabilize
  GPIO_PORTN_DIR_R |= 0x07;                             // make PN0 out (PN0 built-in LED1)
    GPIO_PORTN_AFSEL_R &= ~0x07;                          // disable alt funct on PN[0:2]
 GPIO_PORTN_DEN_R |= 0x07;                              // enable digital I/O on PN[0:2]
                                                          // configure PN[0:2] as GPIO
 //GPIO_PORTN_PCTL_R = (GPIO_PORTN_PCTL_R&0xFFFFFF00)+0x00000000;
 GPIO_PORTN_AMSEL_R &= ~0x07;                          // disable analog functionality on PN[0:2]
  FlashLED1(1);
  return;
}

// Enable interrupts
void EnableInt(void)
{    __asm("    cpsie   i\n");
}

// Disable interrupts
void DisableInt(void)
{    __asm("    cpsid   i\n");
}

// Low power wait
void WaitForInt(void)
{    __asm("    wfi\n");
}




//Assumes 120MHz bus, bus period = 1/(120*10^6)
// If bus clock = timer clock, then max interrupt period = 1/(120*10^6) * 2^32 = 35.8s
//We *choose* for this example each count period to be 0.1s so timer counting freq is 1MHz
//Sincd we multiply the period by 120 to match the units of 1 us, the period will be in 1us units
// 0.25s = 250,000
void Timer3_Init(void){

  uint32_t period=1000000;                          // 32-bit value in 1us increments

  // Part I: Activate timer
  SYSCTL_RCGCTIMER_R = 0x0008;                   //Activate timer
  SysTick_Wait10ms(1);                              // Wait for the timer module to turn on
```

```
    // Part II: Arm and configure at Timer module side
    TIMER3_CTL_R = 0x00000000;                    // (step 1) Disable timer3 during setup (Timer stops counting)
    TIMER3_CFG_R = 0x00000000;                    // (step 2) Configure for 32-bit timer mode
    TIMER3_TAMR_R = 0x00000002;                   // (step 3) Configure for periodic mode
    TIMER3_TAPR_R = 0;                            // (step 4) Set prescale value to 0; i.e. Timer3 works with Maximum Freq = bus clock freq (120MHz)
    TIMER3_TAILR_R = (period*120)-1;              // (step 5) Reload value (we multiply the period by 120 to match the units of 1 us)
    TIMER3_ICR_R = 0x00000001;                    // (step 6) Acknowledge the timeout interrupt (clear timeout flag of Timer3 )
    TIMER3_IMR_R = 0x00000001;                    // (step 7) Arm timeout interrupt


    // Part III: Enable interrupt at Processor side
    NVIC_EN1_R = 0b1000;                     //Enable IRQ 35 in NVIC
    NVIC_PRI8_R = 0x40000000;                  //Priority 2
    EnableInt();                                                // Global Interrupt Enable


    // Part IV: Enable the timer to start counting
    TIMER3_CTL_R = 0x00000001;                    // Enable timer3
}



//This is the Interrupt Service Routine.  This must be included and match the
//  interrupt naming convention in startup_msp432e401y_uvision.s (Note - not the
//  same as Valvano textbook).
void TIMER3A_IRQHandler(void){

    TIMER3_ICR_R = 0x00000001;                    // acknowledge timer3 timeout
    FlashLED2(1);                                 // execute user task -- we simply flash LED
}


// The main program -- notice how we are only initializing the micro and nothing else.
// Our configured interrupts are being handled and tasks executed on an event drive basis.
int main(void){
  PLL_Init();             // set system clock to 120 MHz
    SysTick_Init();
    PortN_Init();
    Timer3_Init();

    while(1){
        WaitForInt();
    }
```

```
}
```

SysTick.c

```
// SysTick.c
// Runs on TM4C1294
// Provide functions that initialize the SysTick module, wait at least a
// designated number of clock cycles, and wait approximately a multiple
// of 10 milliseconds using busy wait.  After a power-on-reset, the
// TM4C1294 gets its clock from the 16 MHz precision internal oscillator,
// which can vary by +/- 1% at room temperature and +/- 3% across all
// temperature ranges.  If you are using this module, you may need more
// precise timing, so it is assumed that you are using the PLL to set
// the system clock to 120 MHz.  This matters for the function
// SysTick_Wait10ms(), which will wait longer than 10 ms if the clock is
// slower.
// Daniel Valvano
// April 3, 2014

/* This example accompanies the books
   "Embedded Systems: Introduction to ARM Cortex M Microcontrollers",
   ISBN: 978-1469998749, Jonathan Valvano, copyright (c) 2014
   Volume 1, Program 4.7

   "Embedded Systems: Real Time Interfacing to ARM Cortex M Microcontrollers",
   ISBN: 978-1463590154, Jonathan Valvano, copyright (c) 2014
   Program 2.11, Section 2.6

 Copyright 2014 by Jonathan W. Valvano, valvano@mail.utexas.edu
    You may use, edit, run or distribute this file
    as long as the above copyright notice remains
 THIS SOFTWARE IS PROVIDED "AS IS".  NO WARRANTIES, WHETHER EXPRESS, IMPLIED
 OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF
 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE.
 VALVANO SHALL NOT, IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL,
 OR CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.
 For more information about my classes, my research, and my books, see
 http://users.ece.utexas.edu/~valvano/
 */

#include <stdint.h>
#define NVIC_ST_CTRL_R          (*((volatile uint32_t *)0xE000E010))
#define NVIC_ST_RELOAD_R        (*((volatile uint32_t *)0xE000E014))
#define NVIC_ST_CURRENT_R       (*((volatile uint32_t *)0xE000E018))
#define NVIC_ST_CTRL_COUNT      0x00010000  // Count flag
#define NVIC_ST_CTRL_CLK_SRC    0x00000004  // Clock Source
#define NVIC_ST_CTRL_INTEN      0x00000002  // Interrupt enable
#define NVIC_ST_CTRL_ENABLE     0x00000001  // Counter mode
#define NVIC_ST_RELOAD_M        0x00FFFFFF  // Counter load value
```

```
// Initialize SysTick with busy wait running at bus clock.
void SysTick_Init(void){
  NVIC_ST_CTRL_R = 0;                   // disable SysTick during setup
  NVIC_ST_RELOAD_R = NVIC_ST_RELOAD_M;  // maximum reload value
  NVIC_ST_CURRENT_R = 0;                // any write to current clears it
                                        // enable SysTick with core clock
  NVIC_ST_CTRL_R = NVIC_ST_CTRL_ENABLE+NVIC_ST_CTRL_CLK_SRC;
}
// Time delay using busy wait.
// The delay parameter is in units of the core clock. (units of 8.333 nsec for 120 MHz clock)
void SysTick_Wait(uint32_t delay){
  volatile uint32_t elapsedTime;
  uint32_t startTime = NVIC_ST_CURRENT_R;
  do{
    elapsedTime = (startTime-NVIC_ST_CURRENT_R)&0x00FFFFFF;
  }
  while(elapsedTime <= delay);
}
// Time delay using busy wait.
// This assumes 120 MHz system clock.
void SysTick_Wait10ms(uint32_t delay){
  uint32_t i;
  for(i=0; i<delay; i++){
    SysTick_Wait(3000000);  // wait 10ms (assumes 120 MHz clock)
  }
}
```

Milestone 2

```
/* Studio W7-1 Project Code
	Sample code provided for studio to demonstrate I2C function and debugging
	Uses modified I2C code by Valvano.

	Written by Tom Doyle
	Last Updated: March 4, 2020
*/

// Name: Yichen Lu Junbo Wang
// Student id: 400247938 400249823
// Date: March, 21th, 2022

#include <stdint.h>
#include "tm4c1294ncpdt.h"
#include "PLL.h"
#include "SysTick.h"
#include "i2c0.h" //modified version of Valvano i2c0.c
```

```
//Flash D2
void FlashLED2(int count) {
        while(count--) {
            GPIO_PORTN_DATA_R ^= 0b00000001;                              //hello world!
            SysTick_Wait10ms(10);                                         //.1s delay
            GPIO_PORTN_DATA_R ^= 0b00000001;
        }
}

//Flash D1
void FlashLED1(int count) {
        while(count--) {
            GPIO_PORTN_DATA_R ^= 0b00000010;                              //hello world!
            SysTick_Wait10ms(10);                                         //.1s delay
            GPIO_PORTN_DATA_R ^= 0b00000010;
        }
}

// Initialize onboard LEDs
void PortN_Init(void){
   //Use PortN onboard LED
   SYSCTL_RCGCGPIO_R |= SYSCTL_RCGCGPIO_R12;              // activate clock for Port N
   while((SYSCTL_PRGPIO_R&SYSCTL_PRGPIO_R12) == 0){};  // allow time for clock to stabilize
   GPIO_PORTN_DIR_R |= 0x03;                             // make PN0 out (PN0 built-in LED1)
  GPIO_PORTN_AFSEL_R &= ~0x03;                            // disable alt funct on PN0
  GPIO_PORTN_DEN_R |= 0x03;                             // enable digital I/O on PN0
                                                          // configure PN1 as GPIO
  //GPIO_PORTN_PCTL_R = (GPIO_PORTN_PCTL_R&0xFFFFFF00)+0x00000000;
  GPIO_PORTN_AMSEL_R &= ~0x03;                            // disable analog functionality on PN0
   FlashLED1(1);
   return;
}

// Enable interrupts
void EnableInt(void)
{    __asm("    cpsie   i\n");
}

// Disable interrupts
void DisableInt(void)
{    __asm("    cpsid   i\n");
}

// Low power wait
void WaitForInt(void)
{    __asm("    wfi\n");
}
```

```
// global variable visible in Watch window of debugger
// increments at least once per button press
// GPIO Port J = Vector 67
// Bit in interrupt register = 51
volatile unsigned long FallingEdges = 0;
void Button_Init(void){
   SYSCTL_RCGCGPIO_R |= SYSCTL_RCGCGPIO_R8;               // activate clock for Port J
   while((SYSCTL_PRGPIO_R&SYSCTL_PRGPIO_R8) == 0){};   // allow time for clock to stabilize
  FallingEdges = 0;            // (b) initialize counter
  GPIO_PORTJ_DIR_R &= ~0x02;    // (c) make PJ1 in

  GPIO_PORTJ_DEN_R |= 0x02;     //     enable digital I/O on PJ1
   GPIO_PORTJ_PCTL_R &= ~0x000000F0; //  configure PJ1 as GPIO
   GPIO_PORTJ_AMSEL_R &= ~0x02;    //   disable analog functionality on PJ1
   GPIO_PORTJ_PUR_R |= 0x02;          //  enable weak pull up resistor
  GPIO_PORTJ_IS_R &= ~0x02;     // (d) PJ1 is edge-sensitive
  GPIO_PORTJ_IBE_R &= ~0x02;    //     PJ1 is not both edges
  GPIO_PORTJ_IEV_R &= ~0x02;    //     PJ1 falling edge event
  GPIO_PORTJ_ICR_R = 0x02;      // (e) clear flag1
  GPIO_PORTJ_IM_R |= 0x02;      // (f) arm interrupt on PJ1
  NVIC_PRI13_R = (NVIC_PRI13_R&0xFF00FFFF)|0x000A0000; // (g) priority 5
  NVIC_EN1_R |= 0x00080000;             // (h) enable interrupt 67 in NVIC
  EnableInt();                          // lets go
}

//This is the Interrupt Service Routine.  This must be included and match the
//  interrupt naming convention in startup_msp432e401y_uvision.s (Note - not the
//  same as Valvano textbook).
void GPIOJ_IRQHandler(void){
  GPIO_PORTJ_ICR_R = 0x02;      // acknowledge flag4
  FallingEdges = FallingEdges + 1; // Observe in Debug Watch Window
   FlashLED2(1);
}


int main(void){
  PLL_Init();              // set system clock to 120 MHz
   SysTick_Init();
   PortN_Init();
   I2C_Init();
   Button_Init();

   while(1){
      WaitForInt();
      I2C_Send1(0x29, 0x56);
   }
}
```