Theme Report - Act

Junbo(Frank) Wang - wangj430 - 400249823

Theme

We will talk about the third theme, called "Act." For the seventh and eighth lab exercises, we look at programming on interruption and how to communicate and collect data from a digital sensor. The theme "Act" normally describes how these actions work to handle problems,like a product's feature or the work of machines in industry and company. In the lectures related to this theme, we start by looking at how the interrupt and sensor perform the results from various tasks.

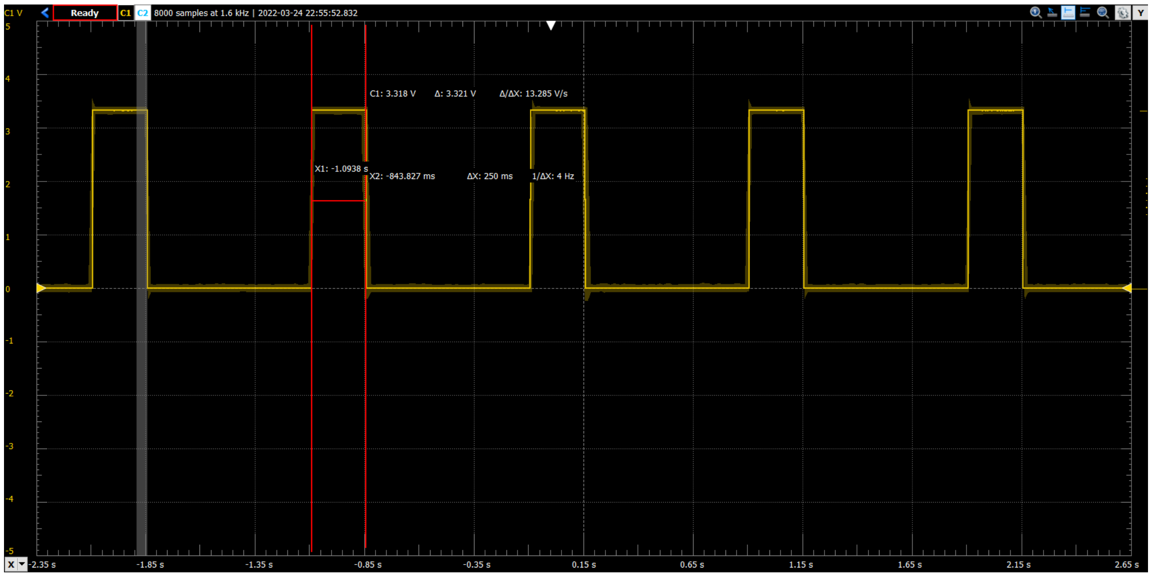
Background

For this theme, "Act" is the action to get an application and the result from hardware or software programming. The Time-of-Flight (ToF) sensor, which was introduced in our course, has an important role in various intelligent machines. ToF sensors and depth cameras are used for a range of applications, including robot navigation, vehicle monitoring, people counting, and object detection. For instance, "With the utmost precision guaranteed and unique considerations in place regarding personal privacy and data protection, ToF cameras enable people counting applications which offer superb insights into a variety of key business metrics, from periodic occupancy counting to space optimization." "ToF sensors can provide accurate stock monitoring, continuously measure fill and stock levels of solid, powder or liquid materials in silos, bins, tanks, stockpiles and more." **[1]** Also, there are several products related to ToF sensors, like cameras, fluid level monitoring, and so on.

Theme Exemplar

In lab 7 milestone 1, we ran the code successfully and saw the onboard LED D2 blink for 250ms with a 1 second 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 250 ms, which meets our requirement. In lab 8 milestone 2, we stuck the Time-of-Flight Sensor on the motor and ran the motor to measure the distance from a certain spot. We collect all the data from the Realterm, calculate the value for the x and y coordinates using trigonometry, and plot these points on Excel. **[2]**

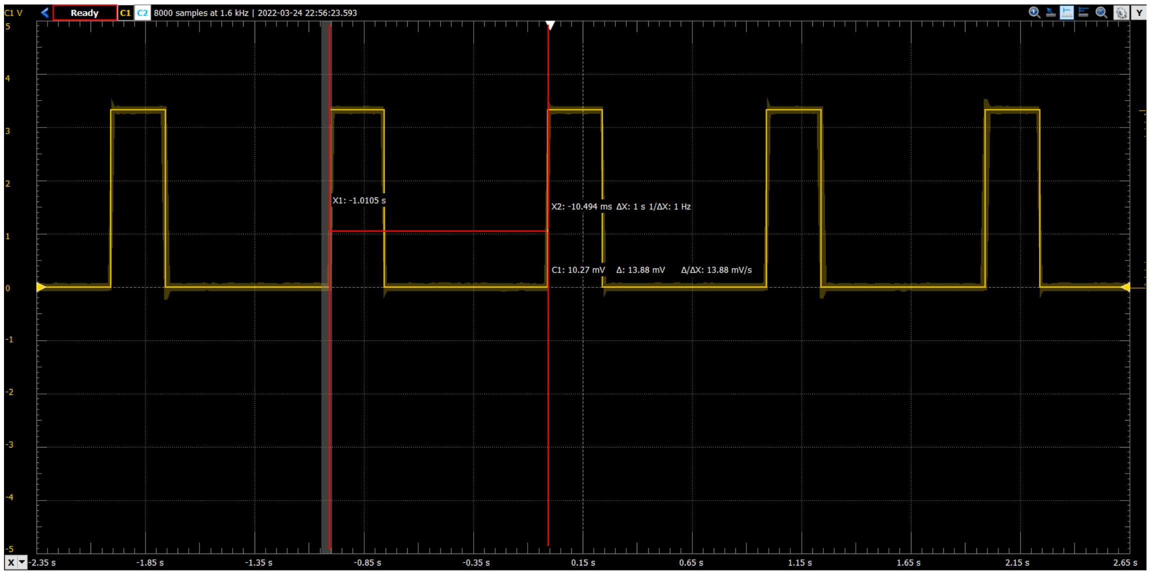
Lab 7 milestone 1



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

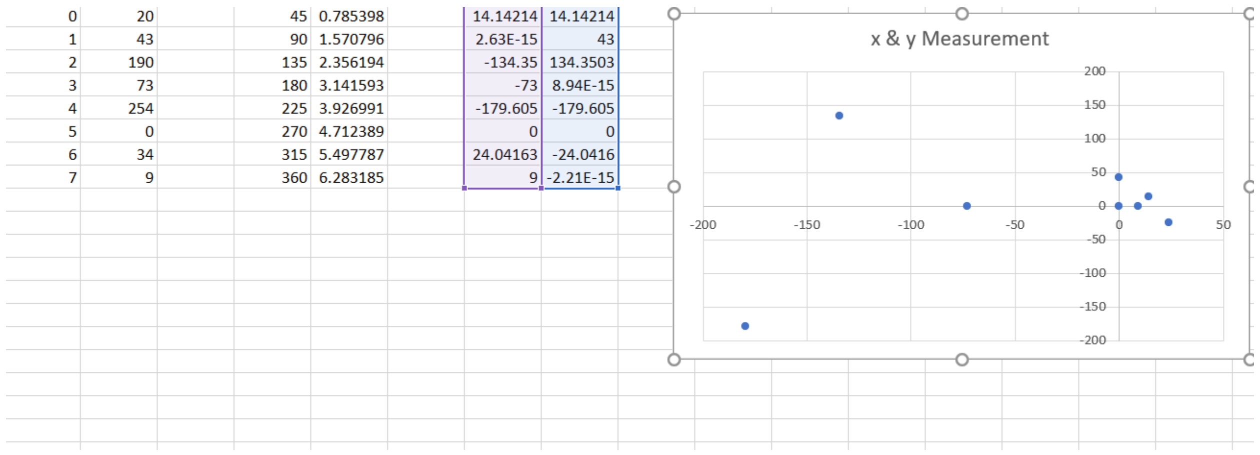


8000 samples at 1.6 kHz | 2022-03-24 22:56:23.593

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

Lab 8 milestone 2



0	20		45	0.785398		14.14214	14.14214
1	43		90	1.570796		2.63E-15	43
2	190		135	2.356194		-134.35	134.3503
3	73		180	3.141593		-73	8.94E-15
4	254		225	3.926991		-179.605	-179.605
5	0		270	4.712389		0	0
6	34		315	5.497787		24.04163	-24.0416
7	9		360	6.283185		9	-2.21E-15

Debugging Exemplar

For lab 7 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 the function SysTick_Wait to 3000000 to let the onboard LED flash for 250ms instead of the 10ms that corresponds to 1200000. For lab 8 milestone 2, we opened the Realterm to collect the data. In the debugging mode, we ran the code to get all the data and export it to Excel. Those points are the distance between the setup spot and the ToF sensor. **[3]**

Synthesis

The theme "Act" is shown in these milestones. According to the examples above, we can observe the periodic interruption in Waveform and the plots from the Time-of-Flight Sensor measurement. For the periodic interrupt, the resulting program produces a square wave with a period of 1s and a 25% duty cycle in the Waveform. The final result verifies that our initial setup is correct, which is 1 second interrupts and 250ms flash time. This shows the theme "Act," in which we get good final results and correctly perform the application. For the distance measurement of the sensor, a ToF sensor is combined with a stepper motor to capture 8 (45-degree intervals) distance measurements from a single 360-degree horizontal scan. These 8 measurements are converted to x and y coordinates and plotted on the graph in Excel. The theme "Act" is shown most vividly in this milestone. We connected the motor and sensor with the microcontroller, and added a function called "spin" in the code to rotate the motor. After the codes ran successfully, we collected all the data from the sensor in Realterm and exported it to Excel to plot the graph. Connecting motor and sensor, spinning the motor and exporting data to generate a plot proves that these actions are crucial to completing this task and showing the theme "Act" in this milestone. Both milestones clearly exhibit the process of all actions and confirm the theme "Act".

Reflection

In my opinion, the theme "Act" indicates the actions in problem-solving. From the examples above, there are several problem-solving actions in one milestone. For example, if we want to measure the distance with a ToF sensor, we use the motor to turn the sensor 360 degrees in both clockwise and counterclockwise directions. Also, I use Realterm to collect data from the ToF sensor and open Excel to calculate the coordinates and plot the graph. With the proper understanding of the theme "Act," I follow these three main actions to complete this milestone. In the project, I need to complete several tasks like operate the push button, build fixed-position ToF sensor, work serial communication with the PC, and so on. Those actions helped me to complete the final goal which is displaying data in 3D modeling. Therefore, this theme "Act" is related to our labs and project by using a microcontroller.

References

[1] www.terabee.com. “Terabee LoRa Level Monitoring XL - battery-powered, LoRaWAN, LED ToF, up to 60m” https://www.terabee.com/shop/level-monitoring/terabee-lora-level-monitoring/, [Accessed: April 4,2022].
[2] Junbo Wang & Yichen Lu. “Lab_07_wangj430_luy191”. Result, pp3, Apr. 2022
[3] Junbo Wang & Yichen Lu. “Lab_07_wangj430_luy191”. Method, pp2, Apr. 2022
[4] Junbo Wang & Yichen Lu. “Lab_07_wangj430_luy191”. Code Appendix, pp5-9, Mar. 2022

Code appendix [4]

```
/* 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))
```

```c
#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)
  }
}
```