EGR 326 - 903

Lab 3
Capture and Compare with the MSP 432

Corey Moura
Xue Hua

Date performed : 13 September 2019
Date submitted : 20 September 2019

Professor Kandalaft

Objective

The overall objective of this lab was to develop a C program to detect distance and control the intensity of an LED. Additional objectives to complete this task were required such as the Implementation timer A in different configurations, generating interrupts and using ISRs.

Equipment

- MSP432 P401R Microcontroller
- Blue LED
- 2n7000 MOSFET
- 1kohm resistor x 2
- 680 ohm resistor
- SRH-Ultrasonic Sensor

Results

Part I – "Measuring distance using the proximity sensor"

In the first part, the MSP432 was used in the capture mode to determine the length of a pulse generated by a proximity sensor. The proximity sensor was pre-triggered in order to generate a pulse so that distance to an object can be determined.

The equations used to determine the distance can be seen below:

$$CLK\ cycles\ in\ the\ interval\ =\ |\ beginning\ of\ interval\ CCR\ val\ -\ end\ of\ interval\ CCR\ val\ |$$

eq.1

Then the duration of the interval in seconds was then computed:

$$Period\ of\ the\ 3.3MHz\ CLK\ =\ \frac{1}{3.3MHz}\ =\ 3.03E-7\ \ (s)$$

eq.2

Using the time of a CLK period and the number of cycles, the time taken to receive the echo back to the sensor was determined:

$$Time(s) = number\ of\ cycles\ in\ the\ interval * 3.03E-7$$

eq.3

The distance of the object can then be obtained from the equation below using 340m/s as the reletive speed of the pulse wave.

$$Distance\ (m) = (\ Time\ (s) * 340\ (m/s))\ / 2$$

eq.4

The final equation is divided by two to compensate for the wave to travel to the object and then back to the echo senor.

The proximity sensor was connected to the MSP432 I/O port pins. Figure 1. A voltage divider circuit was built to ensure that the proximity sensors echo pin could not output more than 3V to the MSP432 input pin, connected to the timer. The sensor was powered by 5V and the trigger was controlled by a GPIO output pin.

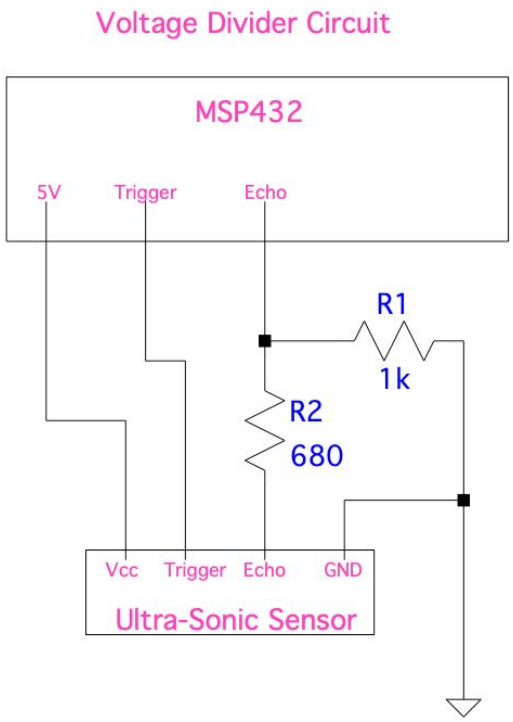

Figure 1. Voltage divider circuit

Another option was to use a “level-shifter” circuit to interface the echo pin to the MSP432. The “level-shifter” circuit would allow the pin to theoretically change state from high to low, using a MOSFET. This circuit can be seen below.

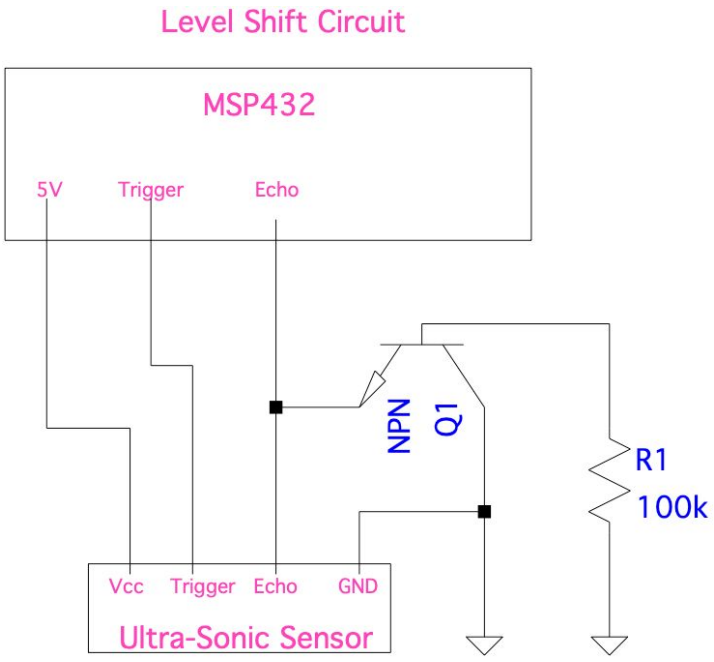

Figure 2. Level shift circuit

While the shift level circuit is a better option, it requires further configuration of the timer to capture from a high signal to a low signal.

The pulse was generated to the trigger pin of the sensor module. This pulse was ~10uS as per the specifications of the sensor. The GPIO output pin was assigned high and a delay function was placed immediately after. Following the 10uS delay, the GPIO output pin was reset to LOW.

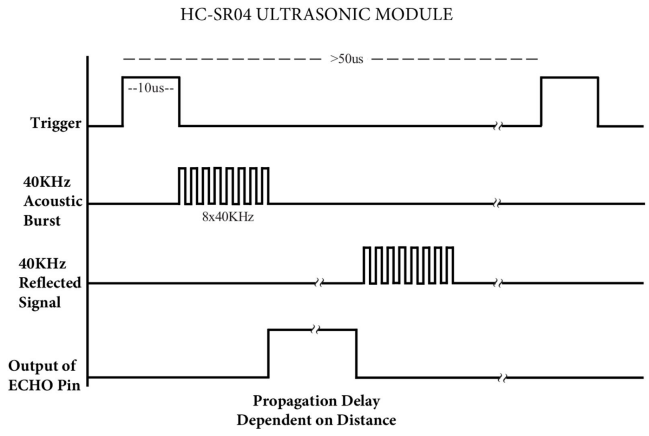

Figure 3. Timing module of sensor
https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf

The program outputs the distance to the object in the console window, in 2 second time intervals. This interval was accomplished via the SYSTICK Timer, configured in interrupt mode. The pin-out for this can be seen below.

Table 1. Pin Table

PINS	MODE	ASSIGNMENT
5.5	GPIO (output)	Trigger
2.5	Timer A0.2 (input)	Echo

The sensor data sheet specifies that the sensor will produce accurate readings to an object up to the object having a 15 degree offset. While this seemed to be accurate it produced inconsistent readings. If the sensor was required to operate in a variety of situations, then the program would need to handle the outlier measurements. Additionally, if the object had a flat surface near it and was angled toward the flat surface, this also generated inaccurate readings.

The working program was demonstrated and the working code can be seen in the appendix. Note that the program uses a custom library to initialize pins and the Systick timer.

Part II – "Creating an alert for proximity"

In this part, the program from part 1 was used to control the intensity of an LED.

An external LED was connected to one of the TIMER_A output pins. This pin was set to the alternate pin mode of TA2.1. Timer A2.1 was configured for PWM without interrupts. The PWM configuration requires the timers CCR0 value to be set along with the CCR1 value. This can be seen in the code provided in the appendix. Below is the pin-out used.

Table 2. Pin Table

PINS	MODE	ASSIGNMENT

5.5	GPIO (output)	Trigger
5.6	Timer A2.1 (output)	PWM LED Control
2.5	Timer A0.2 (input)	Echo

The CCR0 value directly determines the end of the interval for which the output changes. How it changes is determined by the timer count mode. Set / Reset mode was used to allow the CCR0 value to directly control the duty cycle of the LED.

$$Duty\ Cycle = \frac{T_{on}}{Period}$$

eq.5

Using the distance value generated from part 1, the CCR0 value is reassigned to the timer which changes the duty cycle, and results in the change in LED intensity.

The output pin controlling the PWM of the LED is switched to a standard GPIO pin for the toggling of the LED. This SEL change is based on the current distance that the sensor is reading. If it is closer than 1" then it will set the pin to GPIO. The Systick timer interrupt flag is used to control the LED toggle, which prevented having a third timer to control a specific condition.

NOTE: The Timer A configuration table can be seen in the Appendix for further reference.

Conclusion

Timer A is a versatile timer which can be used and configured in many ways. This lab proved two different configurations of the timer can be useful for generating input interrupts, as seen in part 1, and also generating a PWM, as seen in part 2. Additionally, the configuration of the interrupts for the timer are important to link into the start-up file as they will not generate interrupts if the two files do not agree.

Appendix

TIMER A CONFIGURATION REGISTERS:

```
/************************************************************************************************************
 * TIMER A INITIALIZATIONS:
 *
 *
 *          PWM Pin Config:         1. Set the pin as the alternate SEL mode for the timer (SEL0 & SEL1)
 *                      2. Set the pin as an output (Px->DIR  |=  BITx)
 *                      3. Set the pin high (Px->OUT  |=  BITx)
 *
 *          PWM Timer Config:  1. TIMER_A2->CTL  =  0x0214;                     Set the configuration of the clock
 *                      2. TIMER_A2->CCR[0] = 60000;                    PWM Period (# cycles of the clock)
 *                      3. TIMER_A2->CCTL[1] = TIMER_A_CCTLN_OUTMOD_7;  Output mode, count up
 *                      4. TIMER_A2->CCR[1] = 0;                        CCR1 PWM duty cycle in 10ths of percent
 *
 *
 *          Intpt Pin Config:   1. Set the pin as the alternate SEL mode for the timer (SEL0 & SEL1)
 *                      2. Set pin as input
 *
 *          Intpt Timer Config: 1. Set the CTL reg (TIMER_A0->CTL  =  0x0220)      SMCLK, interrupts disabled, continuous mode, no divider
 *                      2. Set the CCTL reg (TIMER_A0->CCTL[2] = 0xCD78)   Intrpt en, set/rst, capture, sync, CCIxA signal, rising edge
 *
 *
 *  -------------------------------------REGISTER INFORMATION-----------------------------------------------
 *
 *
 *  TAxCTL Register -   Timer Control Register: The control register will setup the clock souorce configuration.
 *                      This clock source configuration will be consistent throughout all of the timer instances.
 *
 *          TIMER_A0->CTL           BIT0 = 0    TAIFG   -   0 = No interrupt pending 1 = Interrupt pending
 *          TIMER_A0->CTL           BIT1 = 0    TAIE    -   Timer_A interrupt enable 0 = Interrupt dis. 1 = Interrupt enabled
 *          TIMER_A0->CTL           BIT2 = 1    TACLR   -   Timer_A clear,  bit is automatically reset and is always read as zero
 *          TIMER_A0->CTL           BIT3 = 0    RESERVED
 *
 *          TIMER_A0->CTL           BIT4 = 1    MC      -   00 = Stop mode 01 = Up mode 10 = Continuous mode 11 = Up/down mode
 *          TIMER_A0->CTL           BIT5 = 0    MC
 *          TIMER_A0->CTL           BIT6 = 0    ID      -   CLK input divider (00b = /1) (01b = /2) (10b = /4) (11b = /8)
 *          TIMER_A0->CTL           BIT7 = 0    ID
 *
 *          TIMER_A0->CTL           BIT8 = 0    TASSEL  -   Timer_A CLK source select
 *          TIMER_A0->CTL           BIT9 = 1    TASSEL  -   (00b = TAxCLK) (01b = ACLK) (10b = SMCLK) (11b = INCLK)
 *          TIMER_A0->CTL           BIT10 = 0   RESERVED
 *          TIMER_A0->CTL           BIT11 = 0   RESERVED
 *
 *
 *
 *
 *  TAxCCTL Register - Capture/Compare Control Register: Controls the configurations of each timer instance.  The instance will
 *                     also be driven by the COMMON timer control register above.
 *
 *          TIMER_A0->CCTL[1]       BIT0  = 0   CCIFG       -   Capture compare: 0 = No interrupt pending 1 = Interrupt pending
 *          TIMER_A0->CCTL[1]       BIT1  = 0   COV           -   Indicates a capture overflow occurred overflow = 1, rst in software
 *          TIMER_A0->CCTL[1]       BIT2  = 0   OUT         -   Output. For output mode 0, (output low = 0) (output High = 1)
 *          TIMER_A0->CCTL[1]       BIT3  = 1   CCI         -   Capture/compare input. The selected input signal can be read by this bit.
 *
 *          TIMER_A0->CCTL[1]       BIT4  = 1   CCIE        -   Capture/compare interrupt enable, (0 = disabled) (1 = enabled)
 *          TIMER_A0->CCTL[1]       BIT5  = 1   OUTMOD   -   Output mode. (000 = OUT bit value) (001 = Set) (010 = Toggle/reset)
 *          TIMER_A0->CCTL[1]       BIT6  = 1   OUTMOD   -     (011 = Set/reset) (100 = Toggle)(101 = Reset) (110 = Toggle/set)
 *          TIMER_A0->CCTL[1]       BIT7  = 0   OUTMOD   -  (111 = Reset/set) (011 = Set/reset)
 *
 *          TIMER_A0->CCTL[1]       BIT8  = 1   CAP         -   Capture mode (0 = Compare mode) (1 = Capture mode)
 *          TIMER_A0->CCTL[1]       BIT9  = 0   RESERVED
 *          TIMER_A0->CCTL[1]       BIT10 = 1    SCCI       -   Synchronized capture/compare input. The selected CCI input signal can be read via this bit.
 *          TIMER_A0->CCTL[1]       BIT11 = 1    SCS        -   Synchronize capture source. This bit is used to synchronize the capture
 *                                                  input signal with the timer clock (0 = Asynchronous capture) (1 = Synchronous capture)
 *
 *          TIMER_A0->CCTL[1]       BIT12 = 0    CCIS       -   Capture/compare input select. These bits select the TAxCCR0 input signal.
 *          TIMER_A0->CCTL[1]       BIT13 = 0    CCIS       -           00b = CCIxA 01b = CCIxB 10b = GND 11b = VCC
 *          TIMER_A0->CCTL[1]       BIT14 = 0    CM         -   Capture mode 00b = No capture (01b = Capture on rising edge) 10b = Capture on falling edge
 *          TIMER_A0->CCTL[1]       BIT14 = 1    CM         -           11b = Capture on both rising and falling edges
 *
 *
 *
 *  TAxCCR Register - Timer_Ax Capture/Compare Register: Capture mode is usually used for inpt signals, compare is usually used for PWM outputs
 *
 *          Compare mode: TAxCCRn holds the data for the comparison to the timer value in the Timer_A Register, TAxR
 *          Capture mode: The Timer_A Register, TAxR, is copied into the TAxCCRn register when a capture is performed.
 *
```

```
 *
 *   TAxIV Register - Timer_Ax Interrupt Vector Register: Used when many interrupts need priority control
 *
 *          Timer_A interrupt vector value 00h = No interrupt pending 02h = Interrupt Source: Capture/compare 1;
 *          Interrupt Flag: TAxCCR1 CCIFG;
 *          Interrupt Priority: Highest 04h   = Interrupt Source: Capture/compare 2;
 *          Interrupt Flag: TAxCCR2 CCIFG 06h = Interrupt Source: Capture/compare 3;
 *          Interrupt Flag: TAxCCR3 CCIFG 08h = Interrupt Source: Capture/compare 4;
 *          Interrupt Flag: TAxCCR4 CCIFG 0Ah = Interrupt Source: Capture/compare 5;
 *          Interrupt Flag: TAxCCR5 CCIFG 0Ch = Interrupt Source: Capture/compare 6;
 *          Interrupt Flag: TAxCCR6 CCIFG 0Eh = Interrupt Source: Timer overflow;
 *          Interrupt Flag: TAxCTL TAIFG; Interrupt Priority: Lowest *
 *
 *
 *   TAxEX Register - Timer_Ax Expansion 0 Register: Use when the timer max value needs to be increased
 *
 *          15-3 Reserved R 0h Reserved. Reads as 0. 2-0 TAIDEX RW 0h Input divider expansion. These bits along with the ID bits select the divider for the
input clock.
 *          000b = Divide by 1
 *          001b = Divide by 2
 *          010b = Divide by 3
 *          011b = Divide by 4
 *          100b = Divide by 5
 *          101b = Divide by 6
 *          110b = Divide by 7
 *          111b = Divide by 8
 *
 ************************************************************************************************************************/
```

PART 1 CODE:

```
/************************************************************************************************************************
 * Author:      Corey Moura & Xue Hue
 * Lab:         3.1  -  "Measuring distance using the proximity sensor"
 * Date:        9/4/19
 * Instructor: Dr. Kandalaft
 *
 * Description:                 In this part, you will use the MSP432 in capture mode to determine the length of a pulse generated by a
 *              proximity sensor included in your kit.  The proximity sensor must be pre-triggered in order to generate a
 *              pulse so that distance to an object can be determined.Connect your proximity sensor to the MSP432 I/O port
 *              pins as designed in your pre-lab exercise. BECAREFUL- the proximity sensor must be powered by 5V- so a
 *              “level shifter” circuit must be used to interface it to the MSP432. Run your program that will output to
 *              the monitor in 2 second intervals, the distance to the proximity sensor when the object is directly in front
 *              of the monitor.  Use “printf” to display the value on your CCS window. Repeat your program when an object is
 *              off axis from the sensor by approximately 20 degrees. Repeat your program one last time when an object is off
 *              axis from the sensor by approximately -20 degrees.
 *
 * Notes:       No Driver Lib, use of custom library
 *
 *
 ************************************************************************************************************************/
#include <stdint.h>                     // A set of typedefs that specify exact-width integer types
#include <stdbool.h>                    // Allow boolean variable to be used
#include <stdio.h>                      // Standard input output for printf etc.
#include <stdlib.h>                     // Standard C-programming library
#include <string.h>                     // Allows the use of strings
#include "msp.h"                        // Another TI library of some sort
#include <EGR326Lib.h>
#include <math.h>                       // Another TI library of some sort

void inititialize();                    // Called to initialize all of the MSP features used in the program.
void timerA_Init();                     // Initializes an instances of timer A
void triggerPulse();                    // Pulses the output trigger pin for 10uS
void calcDistance();                    // Calculates the distance to the object using the time interval of captures

volatile float numClkCycles = 0;                        // Number of clock cycles in the captured interval
volatile float totalTime = 0;           // Total time elapsed in the interval
volatile float distanceInMeters = 0;                    // Distance in meters
volatile float speedOfSound = 340;                      // Speed of sound constant ~340 m/s
volatile float distanceInCm = 0;                        // Converted distance to cm
volatile uint16_t pulseLength = 0;                      // Number of cycles the clock took

volatile uint8_t  numCapture = 0;                       // Incremented to track the incoming interrupt values for the CCR reg.
volatile uint16_t captureVal = 0;                       // Assigned the value from the CCR register every interrupt
volatile uint16_t capturedVal_1 = 0;                    // First captured value from the CCR reg is stored here
volatile uint16_t capturedVal_2 = 0;                    // second captured value from the CCR reg is stored here
volatile uint32_t timeout = 0;          // Flag set from the syystic interrupt
volatile uint32_t sys_loadVal = 1500000;                // 1/2Hz = 6600000: Initial value assigned to the systic timer
```

```
void main(void){

        inititialize();

        __enable_irq ( );                                   // enable global interrupts
        NVIC->ISER[0] = 1 << ((TA0_N_IRQn) & 31);           // Enable interrupt in NVIC vector

        while(1){

        if(timeout){
        timeout = 0;                        // Reset the flag
        triggerPulse();                     // Send the trigger pulse
        }

        if(numCapture == 2){
        numCapture = 0;                     // Reset the counter
        calcDistance();                     // Calculate the distance
        printf("distanceInCm = %f\n\n", distanceInCm);      // Print the distance to the console
        }

        }
}

/*****************************************************************************************************/
/*  CALCULATING THE DISTANCE:  Calculates the distance from the timer interrupt values.  The number of clock cycles
 *  between reads is calculated into a time value by finding the period of a clock cycle.  The time is multiplied by
 *  the speed of sound and divided by two to compensate for the travel time to and from the object   */
/*****************************************************************************************************/
void calcDistance(){
        numClkCycles = abs(capturedVal_1 - capturedVal_2);          // The absolute value of the two captures
        totalTime = 3*pow(10,-7) * numClkCycles;                    // period of CLK cycle * number of CLK cycles
        distanceInMeters = (speedOfSound * totalTime) / 2;          // Speed of sound * total time / 2
        distanceInCm = distanceInMeters * 100;              // Conversion from m->cm for displaying
}

/*****************************************************************************************************/
/*  TIMER A0.2 ISR: This is called whenever the timer object detects an input signal changes.   */
/*****************************************************************************************************/
void TA0_N_IRQHandler(void){
        TIMER_A0->CCTL[2] &=~ BIT0;                 // Clear the interrupt flag

        captureVal = TIMER_A0->CCR[2];              // Read the value of the captured TAxR number

        numCapture++;                               // Track the begining and end of interval

        if(numCapture == 1) capturedVal_1 = captureVal;             // Beginning of interval CCR val is assigned here
        if(numCapture == 2) capturedVal_2 = captureVal;             // End of interval CCR val is assigned here
}

/*****************************************************************************************************/
/*  TRIGGER PULSE: It is required to toggle the trigger pin of the Ultrasonic to initiate a reading   */
/*****************************************************************************************************/
void triggerPulse(){
        P5->OUT  |=  BIT5;                          // Toggle bit ON
        _delay_cycles(33);                          // Delay set for 10uS as per data sheet of sensor
        P5->OUT  &=~  BIT5;                             // Toggle bit OFF
}


/*****************************************************************************************************/
/* SYSTICK INTERRUPT HANDLER:  This is called when the systick timer reaches zero from its load value, sets flag  */
/*****************************************************************************************************/
extern void SysTick_Handler(void){
        timeout = 1 ;                               // set flag for timeout of SysTick, rest in main
}


/*****************************************************************************************************/
/* Call out to the functions and initialize the pins and timers used in the program.  */
/*****************************************************************************************************/
void inititialize(){

        WDT_A->CTL = WDT_A_CTL_PW | WDT_A_CTL_HOLD;                 // stop watchdog timer

        timerA_Init();
        Init_SysTick(6600000, 7);                   // Systick initialization
```

```
        Init_OutputPin(55);                     //GPIO ouput pin
        Init_InputPin(25, 1, 0, 0, 0, 0, 0, 0); // Pin2.4, TA, input

}

/*****************************************************************************************************/
/*  TIMER A INITIALIZATIONS:
 *      Timer AO.2 used as the input interrupt timer
/*****************************************************************************************************/
void timerA_Init(){
        TIMER_A0->CTL  = 0x0220;        // SMCLK, interrupts disabled, continuous mode, no divider
        TIMER_A0->CCTL[2] = 0xCD78;     // Intrpt en, set/rst, capture, sync, CCIxA signal, rising edge

}
```

PART 2 CODE:

```
/*****************************************************************************************************************************************
 *  Author:         Corey Moura & Xue Hue
 *  Lab:            3.2  - "Creating alert for proximity"
 *  Date:           9/4/19
 *  Instructor:     Dr. Kandalaft
 *
 *  Description:    In this part, you will use the results from part one to indicate position by varying the intensity of an LED.
 *
 *                  1.  Connect an external LED to one of the TIMER_A output pins.
 *                  2.  Connect and run the proximity sensor as in part one of this lab
 *                  3.  Vary the intensity of the LED so that 1 inch distance will result in maximum intensity, while any
distance
 *                              greater than 10 inches will turn the LED off.
 *                                      1 inch or less  = max intensity (max duty cycle) (blinking)
 *                                      2 inches  = 9/10 max intensity (9/10 duty cycle)
 *                                      3 inches  = 8/10 max intensity (8/10 duty cycle)
 *                                      4 inches  = 7/10 max intensity (7/10 duty cycle)
 *                                      5 inches  = 6/10 max intensity (6/10 duty cycle)
 *                                      ...
 *                                      9 inches  = 2/10 max intensity (2/10 duty cycle)
 *                                      10 inches   = off
 *
 *                  4.  If an object is within 1 inch of the sensor, the LED should blink at 2 Hz.
 *
 *
 *  Notes:          No Driver Lib, Custom Lib used
 *
 *
 *****************************************************************************************************************************************/
#include <stdint.h>                             // A set of typedefs that specify exact-width integer types
#include <stdbool.h>                            // Allow boolean variable to be used
#include <stdio.h>                              // Standard input output for printf etc.
#include <stdlib.h>                             // Standard C-programming library
#include <string.h>                             // Allows the use of strings
#include "msp.h"                                // Another TI library of some sort
#include <math.h>                               // MathLib used for the calculation of distance
#include <EGR326Lib.h>                          // Custom made Library linked to the project

void inititialize();                            // Called to initialize all of the MSP features used in the program.
void timerA_Init();                             // Initializes two instances of timer A
void checkTimeout();                            // Check to see if the Systick Flag has been set
void checkNumCaptured();                        // Check to see if both captured numbers have been stored
void triggerPulse();                            // Pulses the output trigger pin for 10uS
void calcDistance();                            // Calculates the distance to the object using the time interval of captures
void assignPWM();                               // Used to modulate the intensity of the LED based on the distance value
void toggleControl();                           // Toggles an LED based on the current state

volatile float numClkCycles = 0;                // Number of clock cycles in the captured interval
volatile float totalTime = 0;                   // Total time elapsed in the interval
volatile float distanceInMeters = 0;            // Distance in meters
volatile float speedOfSound = 340;              // Speed of sound constant ~340 m/s
volatile float distanceInCm = 0;                // Converted distance to cm
volatile uint16_t pulseLength = 0;              // Number of cycles the clock took
```

```
volatile uint8_t  numCapture = 0;                // Incremented to track the incoming interrupt values for the CCR reg.
volatile uint16_t captureVal = 0;                // Assigned the value from the CCR register every interrupt
volatile uint16_t capturedVal_1 = 0;             // First captured value from the CCR reg is stored here
volatile uint16_t capturedVal_2 = 0;             // second captured value from the CCR reg is stored here
volatile uint32_t timeout = 0;                   // Flag set from the syystic interrupt
volatile uint32_t sys_loadVal = 1500000;         // 1/2Hz = 6600000: Initial value assigned to the systic timer

volatile uint32_t CCR1Value = 0;                 // Used to reassign the value of the second timerA
volatile uint8_t  toggleFlag = 0;                // If the object is too close it will set the flag


 void main(void){

          inititialize();                                    // Initialize all the components of the program

          while(1){

          checkTimeout();                                    // Check to see if the Systick Flag has been set
          checkNumCaptured();                                // Check to see if both captured numbers have been stored

          }
}

/*****************************************************************************************************************/
/*  DETERMINING IF THE INTERVAL HAS BEEN COMPLETED:  If the timer interupt has been called twice, then it can be
 *  implied that the interval is ready to be calculated, and the PWM value to be assigned from the new distance val.*/
/*****************************************************************************************************************/
void checkNumCaptured(){
          if(numCapture == 2){                               // Enter if the interval values have been captured
          numCapture = 0;                                    // Reset the counter

          calcDistance();                                    // Calculate the distance to the object
          assignPWM();                                       // Assign the brightness of the LED

          printf("distanceInCm = %f\n\n", distanceInCm);     // Display the distance to the object in the console

          }
}

/*****************************************************************************************************************/
/*  TRIGGER CONTROL / TOGGLE CONTROL / SYSTIC FLAG CONTROL           */
/*****************************************************************************************************************/
void checkTimeout(){
          if(timeout){
          timeout = 0;                                  // Reset the Systick timer flag
          triggerPulse();                               // Sends the pulse to the trigger every second, based on timer

          if(toggleFlag){
          P5->SEL0 &=~ BIT6;                            // Need to reset the definition of the output pin to toggle
          P5->SEL1 &=~ BIT6;                            // Set to GPIO
                P5->OUT  ^=  BIT6;                      // Toggle the LED
          }
          else{
          P5->SEL0 |=   BIT6;                           // Need to set back to alternate pin function for PWM
          P5->SEL1 &=~  BIT6;                           // Timer A0.2 re-assigned
          }
          }
}


/*****************************************************************************************************************/
/*  PWM CONTROL:  The led intensity is controlled by the value of the distance var.  The flag is set when the LED
 *  needs to be toggled and reset when the distqnce is greater than 1 inch ~ 2.5cm      */
/*****************************************************************************************************************/
void assignPWM(){

          if(distanceInCm <= 2.5)        {   TIMER_A2->CCR[1] = 60000;          toggleFlag = 1;     }  // less than 1 inch
          else if(distanceInCm <= 5)     {   TIMER_A2->CCR[1] = 54000;          toggleFlag = 0;     }  // Less than 2 inches
          else if(distanceInCm <= 7.5)   {   TIMER_A2->CCR[1] = 48000;          toggleFlag = 0;     }  // Less than 3 inches
          else if(distanceInCm <= 10)    {   TIMER_A2->CCR[1] = 42000;          toggleFlag = 0;     }  // Less than 4 inches
          else if(distanceInCm <= 12.5)  {   TIMER_A2->CCR[1] = 36000;          toggleFlag = 0;     }  // Less than 5 inches
          else if(distanceInCm <= 15)    {   TIMER_A2->CCR[1] = 30000;          toggleFlag = 0;     }  // Less than 6 inches
          else if(distanceInCm <= 17.5)  {   TIMER_A2->CCR[1] = 24000;          toggleFlag = 0;     }  // Less than 7 inches
          else if(distanceInCm <= 20)    {   TIMER_A2->CCR[1] = 18000;          toggleFlag = 0;     }  // Less than 8 inches
          else if(distanceInCm <= 22.5)  {   TIMER_A2->CCR[1] = 12000;          toggleFlag = 0;     }  // Less than 9 inches
          else if(distanceInCm <= 25)    {   TIMER_A2->CCR[1] = 6000;           toggleFlag = 0;     }  // Less than 10 inches
          else                           {   TIMER_A2->CCR[1] = 0;              toggleFlag = 0;     }  // Greater than 10
```

```
}


/*****************************************************************************************************************/
/*  CALCULATING THE DISTANCE:  Calculates the distance from the timer interrupt values.  The number of clock cycles
 *  between reads is calculated into a time value by finding the period of a clock cycle.  The time is multiplied by
 *  the speed of sound and divided by two to compensate for the travel time to and from the object   */
/*****************************************************************************************************************/
void calcDistance(){
          numClkCycles = abs(capturedVal_1 - capturedVal_2);                // The absolute value of the two captures
          totalTime = 3*pow(10,-7) * numClkCycles;                          // period of CLK cycle * number of CLK cycles
          distanceInMeters = (speedOfSound * totalTime) / 2;                // Speed of sound * total time / 2
          distanceInCm = distanceInMeters * 100;                            // Conversion from m->cm for displaying
}


/*****************************************************************************************************************/
/*  TIMER A0.2 ISR: This is called whenever the timer object detects an input signal changes.       */
/*****************************************************************************************************************/
void TA0_N_IRQHandler(void){
          TIMER_A0->CCTL[2] &=~ BIT0;                                       // Clear the interrupt flag

    captureVal = TIMER_A0->CCR[2];                                          // Read the value of the captured TAxR number

          numCapture++;                                                     // Track the beginning and end of interval

          if(numCapture == 1) capturedVal_1 = captureVal;                   // Beginning of interval CCR val is assigned here
          if(numCapture == 2) capturedVal_2 = captureVal;                   // End of interval CCR val is assigned here
}

/*****************************************************************************************************************/
/*  TRIGGER PULSE: It is required to toggle the trigger pin of the Ultrasonic to initiate a reading   */
/*****************************************************************************************************************/
void triggerPulse(){
          P5->OUT  |=  BIT5;                                                // Toggle bit ON
          _delay_cycles(33);                                                // Delay set for 10uS as per data sheet of sensor
          P5->OUT  &=~  BIT5;                                               // Toggle bit OFF
}


/*****************************************************************************************************************/
/* SYSTICK INTERRUPT HANDLER:  This is called when the systick timer reaches zero from its load value, sets flag  */
/*****************************************************************************************************************/
extern void SysTick_Handler(void){
          timeout = 1 ;                                                 // set flag for timeout of SysTick, rest in main
}


/*****************************************************************************************************************/
/*  INITIALIZATIONS: Initialize the pins and timers used in the program.  */
/*****************************************************************************************************************/
void inititialize(){

          WDT_A->CTL = WDT_A_CTL_PW | WDT_A_CTL_HOLD;                       // stop watchdog timer

          timerA_Init();
          Init_SysTick(sys_loadVal, 7);                                     // Systick init located in the custom library

          Init_OutputPin(55);                                               // GPIO ouput pin located in the custom library
          Init_InputPin(25, 1, 0, 0, 0, 0, 0, 0);                           // Pin2.4, TA, input


          P5->DIR  |=  BIT6;                                                // P2.4 set TA0.1 P2->SEL0 |= BIT4;
          P5->SEL0 |=  BIT6;                                                // Set to zero
          P5->SEL1 &=~ BIT6;                                                // Enable timer (set to 1)
          P5->OUT  |=  BIT6;                                                //turns on pin

          __enable_irq();                                                   // enable global interrupts
          NVIC->ISER[0] = 1 << ((TA0_N_IRQn) & 31);                         // Enable TA0_N_IRQn interrupt in NVIC vector
}

/*****************************************************************************************************************/
/*  TIMER A INITIALIZATIONS:
 *        Timer AO.2 used as the input interrupt timer
 *        Timer A2.1 used as the ouptut timer for LED PWM                   */
/*****************************************************************************************************************/
void timerA_Init(){
```

```
    TIMER_A0->CTL  =  0x0220;                       // SMCLK, interrupts disabled, continuous mode, no divider
    TIMER_A0->CCTL[2] = 0xCD78;                     // Intrpt en, set/rst, capture, sync, CCIxA signal, rising edge

    TIMER_A2->CTL  =  0x0214;                               // Set the configuration of the clock
    TIMER_A2->CCR[0] = 60000;                               // PWM Period (# cycles of the clock)
    TIMER_A2->CCTL[1] = TIMER_A_CCTLN_OUTMOD_7;             // Output mode, count up
    TIMER_A2->CCR[1] = 0;                                   // CCR1 PWM duty cycle in 10ths of percent

}
```

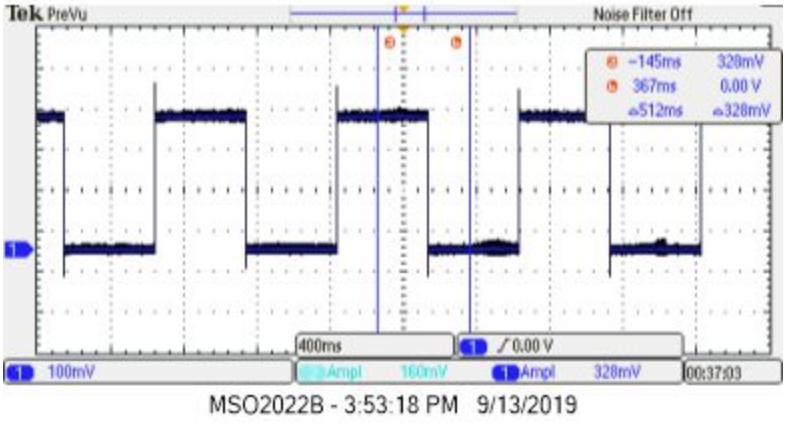

Figure A1. Confirmation of the trigger pulse