```
/****************************************************************************
 Module
   Ultrasonic.c

****************************************************************************/
/*----------------------------- Include Files -----------------------------*/
/* include header files for the framework and this service
*/
#include "ES_Configure.h"
#include "ES_Framework.h"
#include "ES_DeferRecall.h"
#include "Ultrasonic.h"

#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "inc/hw_gpio.h"
#include "inc/hw_sysctl.h"
#include "inc/hw_timer.h"
#include "inc/hw_nvic.h"
#include "inc/hw_ints.h"

#include "driverlib/sysctl.h"
#include "driverlib/pin_map.h" // Define PART_TM4C123GH6PM in project
#include "driverlib/gpio.h"
#include "driverlib/timer.h"
#include "driverlib/interrupt.h"
#include "driverlib/rom.h"

#include "ES_ShortTimer.h"

/*----------------------------- Module Defines ----------------------------*/
// these times assume a 1.000mS/tick timing
#define ONE_SEC 1000
#define HALF_SEC (ONE_SEC/2)
#define TWO_SEC (ONE_SEC*2)
#define FIVE_SEC (ONE_SEC*5)
#define ALL_PINS (0xff<<2)
#define DWT_O_CYCCNT 0x00000004
#define CLK_CYCLE_PER_SEC 40000000
#define TEN_US_LOAD 400
#define UPDATE_ULTRASONIC_PERIOD 20
#define SHORT_TIMER_PULSE_WIDTH 11 //may subject to tuning, nominal is 10us on the scope
#define ALL_BITS (0xff<<2)
#define FLIGHT_TIME_MULTIPLIER 1000000/56.5
/*---------------------------- Module Functions ---------------------------*/
/* prototypes for private functions for this service.They should be functions
   relevant to the behavior of this service
*/

//static void InitLED(void);

void StartOneShot_Ultrasonics(uint32_t oneshot_load);
void InitOneShotInt_Ultrasonics( void );
void InitInputCapture_Ultrasonics( void );

/*---------------------------- Module Variables ---------------------------*/
// with the introduction of Gen2, we need a module level Priority variable
static uint8_t MyPriority;
// add a deferral queue for up to 3 pending deferrals +1 to allow for ovehead
```

```
static ES_Event DeferralQueue[3+1];

static uint32_t EncoderPeriod;
static uint32_t NewTick;
static float TimeOfFlight;

static uint32_t LastCapture = 0;
static uint32_t tenus_oneshotload  =      400;
static uint32_t tenms_oneshotload   =     40000;
//one shot timer mode:
//    0 for 10us timer
//    1 for 10ms timer
static uint8_t oneshotmode = 1;

static uint8_t muxSelect = 2; //1 or 2 or 3

static uint8_t printCounter=0;
static const uint8_t maxPrintCounter=1;
static uint32_t latest_x_reading = 0;
static uint32_t latest_yf_reading = 0;
static uint32_t latest_yb_reading = 0;

/*------------------------------ Module Code ------------------------------*/
/****************************************************************************
 Function
     InitUltrasonic

 Parameters
     uint8_t : the priorty of this service

 Returns
     bool, false if error in initialization, true otherwise

****************************************************************************/
bool InitUltrasonic ( uint8_t Priority )
{
      ES_Event ThisEvent;
      MyPriority = Priority;
      /********************************************
      in here you write your initialization code
      ********************************************/
      // initialize deferral queue for testing Deferal function
      ES_InitDeferralQueueWith( DeferralQueue, ARRAY_SIZE(DeferralQueue) );
      // initialize the Short timer system for channel A
      ES_ShortTimerInit(MyPriority, SHORT_TIMER_UNUSED); //<---<MAKE SURE YOU INIT
      // initialize LED drive for testing/debug output
      printf("before init LED\n\r");
      // set up I/O lines for debugging
      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);
      GPIOPinTypeGPIOOutput(GPIO_PORTD_BASE, GPIO_PIN_3);
      //set up the I/O lines for MUX
      GPIOPinTypeGPIOOutput(GPIO_PORTA_BASE, GPIO_PIN_6);
      GPIOPinTypeGPIOOutput(GPIO_PORTA_BASE, GPIO_PIN_7);
      //init input capture
      InitInputCapture_Ultrasonics();
      // post the initial transition event
      // start with the lines low
      GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_3, BIT3HI);
```

```
      //HWREG(GPIO_PORTD_BASE+(GPIO_O_DATA + ALL_BITS)) &= ~(GPIO_PIN_3);
      ThisEvent.EventType = ES_INIT;

      if (ES_PostToService( MyPriority, ThisEvent) == true)
      {
            return true;
      }else
      {
            return false;
      }
}

/****************************************************************************
 Function
     PostUltrasonic

 Parameters
     EF_Event ThisEvent ,the event to post to the queue

 Returns
     bool false if the Enqueue operation failed, true otherwise

 Description
     Posts an event to this state machine's queue

****************************************************************************/
bool PostUltrasonic( ES_Event ThisEvent )
{
  return ES_PostToService( MyPriority, ThisEvent);
}

/****************************************************************************
 Function
    RunUltrasonic

 Parameters
   ES_Event : the event to process

 Returns
   ES_Event, ES_NO_EVENT if no error ES_ERROR otherwise

****************************************************************************/
ES_Event RunUltrasonic( ES_Event ThisEvent )
{
      ES_Event ReturnEvent;
      ReturnEvent.EventType = ES_NO_EVENT; // assume no errors
      static char DeferredChar = '1';

      switch (ThisEvent.EventType){
            case ES_INIT :
                  //start timer to kick it off
                  ES_Timer_InitTimer(SERVICE0_TIMER, HALF_SEC);
                  break;

            case ES_TIMEOUT :  // re-start timer & announce
                  printCounter = printCounter+1;
                  if (printCounter == maxPrintCounter){
                        printCounter=0; //reset counter
                        TimeOfFlight = ((float)EncoderPeriod)/CLK_CYCLE_PER_SEC;
```

```
                        // calculate and save reading according to the current mux
setting
                        if(muxSelect == 1){
                              latest_x_reading = TimeOfFlight *
FLIGHT_TIME_MULTIPLIER; // in cm
                        }
                        else if(muxSelect == 2){
                              latest_yf_reading = TimeOfFlight *
FLIGHT_TIME_MULTIPLIER; // in cm
                        }
                        else{
                              latest_yb_reading = TimeOfFlight *
FLIGHT_TIME_MULTIPLIER; // in cm
                        }

                        //switching between the mux, so we can sequencially get reading
from one of them on each pass
                        if(muxSelect == 1){
                              muxSelect = 2;
                        }
                        else if(muxSelect == 2){
                              muxSelect = 3;
                        }
                        else{
                              muxSelect = 1;
                        }
                  }
                  ES_Timer_InitTimer(SERVICE0_TIMER, UPDATE_ULTRASONIC_PERIOD);

                  if (muxSelect==1){
                        //choose the line of the mux
                        GPIOPinWrite(GPIO_PORTA_BASE, GPIO_PIN_6, BIT6LO);
                        GPIOPinWrite(GPIO_PORTA_BASE, GPIO_PIN_7, (uint8_t) BIT7LO);
                        //Lower both PA6 and PA7 to select X-X1, that is BIT7LO
                  }
                  else if(muxSelect==2){
                        //choose the line of the mux
                        GPIOPinWrite(GPIO_PORTA_BASE, GPIO_PIN_6, BIT6HI);
                        GPIOPinWrite(GPIO_PORTA_BASE, GPIO_PIN_7, (uint8_t)
BIT7LO);//Lower both PA6 and PA7 to select X-X1
                  }
                  else if(muxSelect==3){
                        //choose the line of the mux
                        GPIOPinWrite(GPIO_PORTA_BASE, GPIO_PIN_6, BIT6LO);
                        GPIOPinWrite(GPIO_PORTA_BASE, GPIO_PIN_7, (uint8_t) BIT7HI);
                        //Lower both PA6 and PA7 to select X-X1
                  }
                        //pulse the 10us for the trigger pin
                        ES_ShortTimerStart(TIMER_A, SHORT_TIMER_PULSE_WIDTH);
                        //raise the trigger pin
                        GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_3, BIT3HI);
                        break;

            case ES_SHORT_TIMEOUT :  // lower the line & announce
                  //printf("short timeout happens\n\r");
                  GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_3, BIT3LO);
                  break;

            //This is for debugging purposes
```

```
            case ES_NEW_KEY :  // announce
                  printf("ES_NEW_KEY received with -> %c <- in Service 0\r\n",
(char)ThisEvent.EventParam);
                  if( 'd' == ThisEvent.EventParam ){
                        ThisEvent.EventParam = DeferredChar++; //
                        if (ES_DeferEvent( DeferralQueue, ThisEvent )){
                              puts("ES_NEW_KEY deferred in Service 0\r");
                        }
                  }
                  if( 'r' == ThisEvent.EventParam ){
                        ThisEvent.EventParam = 'Q'; // This one gets posted normally
                        ES_PostToService( MyPriority, ThisEvent);
                        // but we slide the deferred events under it so it(they) should
come out first
                        if ( true == ES_RecallEvents( MyPriority, DeferralQueue )){
                              puts("ES_NEW_KEY(s) recalled in Service 0\r");
                        DeferredChar = '1';
                        }
                  }
                  if( 'p' == ThisEvent.EventParam ){
                        ES_ShortTimerStart(TIMER_A, 10);
                  }
                  if (ThisEvent.EventParam=='1'){
                        muxSelect=1;
                  }
                  if (ThisEvent.EventParam=='2'){
                        muxSelect=2;
                  }
                  if (ThisEvent.EventParam=='3'){
                        muxSelect=3;
                  }
                  break;

            default :
                  break;
  }
  return ReturnEvent;
}

/****************************************************************************
 private functions
 ****************************************************************************/

static void InitLED(void)
{
      volatile uint32_t Dummy;
      // enable the clock to Port F
      HWREG(SYSCTL_RCGCGPIO) |= SYSCTL_RCGCGPIO_R5;
      // kill a few cycles to let the peripheral clock get going
      Dummy = HWREG(SYSCTL_RCGCGPIO);
      // Enable pins for digital I/O
      HWREG(GPIO_PORTF_BASE+GPIO_O_DEN) |= (GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3);
      // make pins 1,2 & 3 on Port F into outputs
      HWREG(GPIO_PORTF_BASE+GPIO_O_DIR) |= (GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3);
      }

/*
*            George's code: one shot timer init, WT0
*/
```

```c
//original from Prof.Ed, I used subTimer B instead
void InitOneShotInt_Ultrasonics( void ){
	// start by enabling the clock to the timer (Wide Timer 0)
	HWREG(SYSCTL_RCGCWTIMER) |= SYSCTL_RCGCWTIMER_R0;
	// kill a few cycles to let the clock get going
	while((HWREG(SYSCTL_PRWTIMER) & SYSCTL_PRWTIMER_R0) != SYSCTL_PRWTIMER_R0)
	{
	}
	// make sure that timer (Timer B) is disabled before configuring
	HWREG(WTIMER0_BASE+TIMER_O_CTL) &= ~TIMER_CTL_TBEN; //TBEN = Bit8
	// set it up in 32bit wide (individual, not concatenated) mode
	// the constant name derives from the 16/32 bit timer, but this is a 32/64
	// bit timer so we are setting the 32bit mode
	HWREG(WTIMER0_BASE+TIMER_O_CFG) = TIMER_CFG_16_BIT; //bits 0-2 = 0x04
	// set up timer B in 1-shot mode so that it disables timer on timeouts
	// first mask off the TAMR field (bits 0:1) then set the value for
	// 1-shot mode = 0x01
	HWREG(WTIMER0_BASE+TIMER_O_TBMR) =
	(HWREG(WTIMER0_BASE+TIMER_O_TBMR)& ~TIMER_TBMR_TBMR_M)|
	TIMER_TBMR_TBMR_1_SHOT;
	// set timeout, by default 10 us
	HWREG(WTIMER0_BASE+TIMER_O_TBILR) = tenus_oneshotload;
	// enable a local timeout interrupt. TBTOIM = bit 8
	HWREG(WTIMER0_BASE+TIMER_O_IMR) |= TIMER_IMR_TBTOIM; // 8
	// enable the Timer B in Wide Timer 0 interrupt in the NVIC
	// it is interrupt number 95 so appears in EN2 at bit 30
	HWREG(NVIC_EN2) |= BIT31HI;
	// make sure interrupts are enabled globally
	__enable_irq();
	//set one shot timer mode
	oneshotmode = 1;
	//StartTime = ES_Timer_GetTime();
	// now kick the timer off by enabling it and enabling the timer to
	// stall while stopped by the debugger. TAEN = Bit0, TASTALL = bit1
	HWREG(WTIMER0_BASE+TIMER_O_CTL) |= (TIMER_CTL_TBEN | TIMER_CTL_TBSTALL);
}

void UltrasonicISR (void)
{
	HWREG(WTIMER0_BASE+TIMER_O_ICR) = TIMER_ICR_TBTOCINT; //service the interrupt
	//check the oneshot mode, start it again with 10ms load if mode = 0
	//lower the trigger line
	GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_2, BIT2LO);
}

void StartOneShot_Ultrasonics(uint32_t oneshot_load){
	//start the oneshot timer
	HWREG(WTIMER0_BASE+TIMER_O_CTL) |= (TIMER_CTL_TBEN | TIMER_CTL_TBSTALL);
}


void InitInputCapture_Ultrasonics( void ){
	// start by enabling the clock to the timer (Wide Timer 5)
	HWREG(SYSCTL_RCGCWTIMER) |= SYSCTL_RCGCWTIMER_R5;
	// enable the clock to Port D
	HWREG(SYSCTL_RCGCGPIO) |= SYSCTL_RCGCGPIO_R3;
	// since we added this Port D clock init, we can immediately start
	// into configuring the timer, no need for further delay
```

```
	// make sure that timer (Timer B) is disabled before configuring
	HWREG(WTIMER5_BASE+TIMER_O_CTL) &= ~TIMER_CTL_TBEN;
	// set it up in 32bit wide (individual, not concatenated) mode
	// the constant name derives from the 16/32 bit timer, but this is a 32/64
	// bit timer so we are setting the 32bit mode
	HWREG(WTIMER5_BASE+TIMER_O_CFG) = TIMER_CFG_16_BIT;
	// we want to use the full 32 bit count, so initialize the Interval Load
	// register to 0xffff.ffff (its default value :-)
	HWREG(WTIMER5_BASE+TIMER_O_TBILR) = 0xffffffff;
	// set up timer B in capture mode (TAMR=3, TAAMS = 0),
	// for edge time (TACMR = 1) and up-counting (TACDIR = 1)
	HWREG(WTIMER5_BASE+TIMER_O_TBMR) =
	(HWREG(WTIMER5_BASE+TIMER_O_TBMR) & ~TIMER_TBMR_TBAMS) |
	(TIMER_TBMR_TBCDIR | TIMER_TBMR_TBCMR | TIMER_TBMR_TBMR_CAP);
	// To set the event to rising edge, we need to modify the TAEVENT bits
	// in GPTMCTL. Rising edge = 00, so we clear the TAEVENT bits
	HWREG(WTIMER5_BASE+TIMER_O_CTL) &= ~TIMER_CTL_TBEVENT_M;
	//Capture Both Edges
	HWREG(WTIMER5_BASE+TIMER_O_CTL) |= TIMER_CTL_TBEVENT_BOTH;
	// Now Set up the port to do the capture (clock was enabled earlier)
	// First unlock PD7 first before setting it
	HWREG(GPIO_PORTD_BASE + GPIO_O_LOCK) = GPIO_LOCK_KEY;
	HWREG(GPIO_PORTD_BASE + GPIO_O_CR) = 0x80;
	// start by setting the alternate function for Port D bit 7 (WT0CCP0)
	HWREG(GPIO_PORTD_BASE+GPIO_O_AFSEL) |= BIT7HI;
	// Then, map bit 4's alternate function to WT0CCP0
	// 7 is the mux value to select WT5CCP1, 28 to shift it over to the
	// right nibble for bit 7 (4 bits/nibble * 7 bits)
	HWREG(GPIO_PORTD_BASE+GPIO_O_PCTL) =
	(HWREG(GPIO_PORTD_BASE+GPIO_O_PCTL) & 0x0fffffff) + (7<<28);
	// Enable pin on Port D for digital I/O
	HWREG(GPIO_PORTD_BASE+GPIO_O_DEN) |= BIT7HI;
	// make pin 7 on Port D into an input
	HWREG(GPIO_PORTD_BASE+GPIO_O_DIR) &= BIT7LO;
	//
	//Now modify the configuration of the pins that we unlocked.
	//
	//GPIOPinTypeGPIOInput(GPIO_PORTD_BASE, GPIO_PIN_7);
	// back to the timer to enable a local capture interrupt
	HWREG(WTIMER5_BASE+TIMER_O_IMR) |= TIMER_IMR_CBEIM;
	// enable the Timer B in Wide Timer 5 interrupt in the NVIC
	// it is interrupt number 105 so appears in EN3 at bit 9
	HWREG(NVIC_EN3) |= BIT9HI;
	// make sure interrupts are enabled globally
	__enable_irq();

	// now kick the timer off by enabling it and enabling the timer to
	// stall while stopped by the debugger
	HWREG(WTIMER5_BASE+TIMER_O_CTL) |= (TIMER_CTL_TBEN | TIMER_CTL_TBSTALL);
}


void InputCaptureResponse_Ultrasonics( void ){
	// start by clearing the source of the interrupt, the input capture event
	HWREG(WTIMER5_BASE+TIMER_O_ICR) = TIMER_ICR_CBECINT;
	uint32_t ThisCapture;
	ThisCapture = HWREG(WTIMER5_BASE+TIMER_O_TBR);
	EncoderPeriod = ThisCapture - LastCapture;
	LastCapture = ThisCapture;
```

```
}

/* Return x reading (mux 1) */
uint32_t get_Ultrasonic_X(void){
      return latest_x_reading;
}

/* Return y reading (front) (mux 2) */
uint32_t get_Ultrasonic_Y_F(void){
      return latest_yf_reading;
}

/* Return y reading (back) (mux 3) */
uint32_t get_Ultrasonic_Y_B(void){
      return latest_yb_reading;
}


/*------------------------------- Footnotes -------------------------------*/
/*------------------------------- End of file -------------------------------*/
```