```
/*******************************************************************************
 Module
   TemplateService.c

 Revision
   1.0.1

 Description
   This is a template file for implementing a simple service under the
   Gen2 Events and Services Framework.

 Notes

 History
 When           Who     What/Why
 -------------- ---    --------
 01/16/12 09:58 jec      began conversion from TemplateFSM.c
*******************************************************************************/
/*---------------------------- Include Files ----------------------------*/
/* include header files for this state machine as well as any machines at the
   next lower level in the hierarchy that are sub-machines to this machine
*/
#include "ES_Configure.h"
#include "ES_Framework.h"
#include "UltrasonicService.h"
```

```
#include <sys/attribs.h>
#include "dbprintf.h"
#include <stdio.h>
#include "EventCheckers.h"
#include "ES_Events.h"

/*---------------------------- Module Defines ----------------------------*/
#define T2_PERIOD 0xFFFF
#define T3_PERIOD 0xFFFF
#define CLOSE_TREE_THRESHOLD_CM 5
#define FAR__TREE_THRESHOLD_CM 100
#define CLOSE_WALL_THRESHOLD_CM 20
#define FAR__WALL_THRESHOLD_CM 100

/*---------------------------- Module Functions ----------------------------*/
/* prototypes for private functions for this service.They should be functions
   relevant to the behavior of this service
*/
//void configUltrasonicInterrupts(void);
static void configTimer2(void);
//static void configTimer3(void);
static void configOC1(void);
static void configOC2(void);
//void configIC2(void);
//void configIC4(void);
```

```
/*---------------------------- Module Variables ----------------------------*/

// with the introduction of Gen2, we need a module level Priority variable

static uint8_t MyPriority;

static bool wallDetected = 0;

static bool treeDetected = 0;

static Period_32BitCount_t currentRecordedTimeWallUltra;

static Period_32BitCount_t currentRecordedTimeTreeUltra;

static volatile uint32_t lastRecordedTimeWallUltra = 0;

static volatile uint32_t lastRecordedTimeTreeUltra = 0;

static volatile uint16_t wallPeriodLength = 0; // currentRecordedTime.LowByteRolloverCount +
currentRecordedTime.ByBytes.HighByteRolloverCount

static volatile uint16_t treePeriodLength = 0; // currentRecordedTime.LowByteRolloverCount +
currentRecordedTime.ByBytes.HighByteRolloverCount

static uint16_t capturedTimeWallUltra;

static uint16_t capturedTimeTreeUltra;

static ES_Event_t UltraFired;

static int CountIC = 1;


static uint16_t dummyVariable4Clearing;// buffer emptying routine & flag clearing routine before setting
interrupt enable


static volatile float wallEchoPulse = 0.0;

static volatile float treeEchoPulse = 0.0;

static int wall_distance;

static int tree_distance;


/*---------------------------- Module Code ----------------------------*/
```

```
/****************************************************************************
 Function
    InitUltrasonicService

 Parameters
    uint8_t : the priorty of this service

 Returns
    bool, false if error in initialization, true otherwise

 Description
    Saves away the priority, and does any
    other required initialization for this service
 Notes

 Author
    A. Kelkelyan, 02/25/24, 20:04
****************************************************************************/
bool InitUltrasonicService(uint8_t Priority)
{
  ES_Event_t ThisEvent;

  MyPriority = Priority;
  /********************************************
   in here you write your initialization code
   *******************************************/
```

```
  DB_printf("Initializing Ultrasonic Sensor \n");
  // Interrupt Initialization
  //configTimer2(); // Setup Timer 2 for continuous PWM output, commented out for dev
  //configTimer3(); // Setup Timer 3 (same timer as Beacon) for Output Compare
  //configOC1();  // Setup Output Compare for OC1, commented out for dev
//  configIC2(); // Setup Input Capture for IC2
//  configIC4();
//  configUltrasonicInterrupts(); //configure IC interrupt, this isn't really needed though
  RunPsuedoPWM4Ultra4Wall();
  RunPsuedoPWM4Ultra4Tree();
  TRISBbits.TRISB9 = 0; // config PWM for IC2 as digital output
  TRISBbits.TRISB8 = 0; // config PWM for IC4 as digital output

  // start with interrupts disabled
  DisableUltra4Wall();
  DisableUltra4Tree();

  // post the initial transition event
  ThisEvent.EventType = ES_INIT;
  if (ES_PostToService(MyPriority, ThisEvent) == true)
  {
          return true;
  }
  else
  {
          return false;
```

```
  }
}

/****************************************************************************
 Function
    PostUltrasonicService

 Parameters
    EF_Event_t 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
 Notes

 Author
    J. Edward Carryer, 10/23/11, 19:25
****************************************************************************/
bool PostUltrasonicService(ES_Event_t ThisEvent)
{
  return ES_PostToService(MyPriority, ThisEvent);
}

/****************************************************************************
```

```
Function

   RunUltrasonicService


Parameters

  ES_Event_t : the event to process


Returns

  ES_Event, ES_NO_EVENT if no error ES_ERROR otherwise


Description

  add your description here

Notes


Author

   A. Kelkelyan, 02/25/2024, 20:58
*****************************************************************************/
ES_Event_t RunUltrasonicService(ES_Event_t ThisEvent)
{
  ES_Event_t ReturnEvent;
  ReturnEvent.EventType = ES_NO_EVENT; // assume no errors
  /*******************************************
   in here you write your service code
   *******************************************/
  switch (ThisEvent.EventType)
  {
          case ES_INIT:
```

```
{
DB_printf("In Run function of Ultrasonic Service\n\n");
}
break;

case ES_WALL_DETECTED://if new ultra sonic pulse
{
DB_printf("Wall distance in cm is = %d\r\n\n", wall_distance);
}
break;

case ES_TREE_DETECTED://if new ultra sonic pulse
{
DB_printf("Tree distance in cm is = %d\r\n\n", tree_distance);
}
break;

case ES_NEW_KEY:
{
switch(ThisEvent.EventParam)
{
case 'w':
{
DB_printf("Disabling wall sensor\n");
DisableUltra4Wall();
}
```

```
break;

case 'x':
{
DB_printf("Enabling wall sensor\n");
EnableUltra4Wall();
}
break;

case 't':
{
DB_printf("Disabling tree sensor\n");
DisableUltra4Tree();
}
break;

case 'u':
{
DB_printf("Enabling tree sensor\n");
EnableUltra4Tree();
}
break;

}

}
```

```
            break;


            case ES_TIMEOUT:
            {
            RunPsuedoPWM4Ultra4Wall();
            RunPsuedoPWM4Ultra4Tree();
            // DB_printf("Outputting PWM \n");
            }
            break;


            default:
            {DB_printf("You posted an event that was defaulted in Ultrasonic Service\n");}
            break;


            }
  return ReturnEvent;
}


/****************************************************************************
 private functions
 ****************************************************************************/
bool IsWallDetected(void)
{
            return wallDetected;
}
```

```
void configUltrasonicInterrupts(void)
{
        __builtin_disable_interrupts();

    INTCONbits.MVEC = 1; // Multi-vector mode enabled
    IPC2bits.IC2IP = 7; //set IC2 priority
    IPC4bits.IC4IP = 7; //set IC4 priority
    IPC3bits.T3IP = 6; //set Timer3 Priority

        //Clear any pending interrupt flags
    IFS0CLR = _IFS0_T3IF_MASK;
    IFS0CLR = _IFS0_IC2IF_MASK;
    IFS0CLR = _IFS0_IC4IF_MASK;

        // Enable T3, but NOT IC2, IC4 interrupts
    IEC0SET = _IEC0_T3IE_MASK;
//    IEC0SET = _IEC0_IC2IE_MASK;
//    IEC0SET = _IEC0_IC4IE_MASK;

        __builtin_enable_interrupts(); //enable global interrupts
}

static void configTimer2(void){
    T2CONbits.ON = 0;//disable timer 2
        //CONFIG TIMER 3
    T2CONbits.TGATE = 0;//disable gated time accumulation mode
```

```
    T2CONbits.TCS = 0; //select PBCLK as source

    T2CONbits.TCKPS = 0b000; //

            // Originally: 1:1 Pre-scale. Trigger pulses at 10 us. 10 us / 50 ns = 200 / 1 prescale = 200 <
65535 (16 bits)

    PR2 = T2_PERIOD;//set timer period

    TMR2 = 0; // start timer at zero

    T2CONbits.ON = 1;//enable timer 2

}



void configTimer3(void)

{

            // configure timer 3

    T3CONbits.ON = 0; // turn timer off

    T3CONbits.TCS = 0; // internal clock source

    T3CONbits.TCKPS = 0b101; // 1:32 Pre-scale. Echo suggests to use over 60 ms measurement cycle.
60 ms / 50 ns = 1,200,000 / 32 prescale = 37500 < 65535 (16 bits)

//    T3CONbits.TCKPS = 0b110; // 1:64 Pre-scale. Echo suggests to use over 60 ms measurement cycle.
60 ms / 50 ns = 1,200,000 / 64 prescale = 18750 < 65535 (16 bits)

//    T3CONbits.TCKPS = 0b000; // 1:1 Pre-scale. Echo returns a burst over 40 kHz measurement cycle.
40000 < 65535 (16 bits)

    T3CONbits.TGATE = 0; // no external gate

    PR3 = T3_PERIOD; // period setting

    TMR3 = 0; // start timer at zero

    IFS0CLR = _IFS0_T3IF_MASK; // clear any flag

    T3CONbits.ON = 1; // turn timer on

}
```

```
void configIC2(void)

{

    IC2CONbits.ON = 0;

    TRISBbits.TRISB10 = 1; // OG

    IC2Rbits.IC2R = 0b0011; // map IC2 to RB10

    IC2CONbits.ICTMR = 0; // Timer3 is the counter source for capture

    IC2CONbits.SIDL = 0; // Continue to operate in Idle mode ? let it run in idle mode

    IC2CONbits.ICM = 0b110; // Simple Capture Event mode - every edge but with specified edge (rising)
first

    IC2CONbits.C32 = 0; //  16-bit timer resource capture

    IC2CONbits.ICI = 00; //  Interrupt on every capture event

    IPC2bits.IC2IP = 7; // Interrupt priority level 7

    IC2CONbits.FEDGE = 1; // detect rising edge first

            // IEC0SET = _IEC0_IC2IE_MASK; // --> SHOULD NOT BE SET HERE

    IFS0CLR = _IFS0_IC2IF_MASK;

    IC2CONbits.ON = 1;

}


void configIC4(void)

{

    IC4CONbits.ON = 0;

    TRISBbits.TRISB4 = 1;

    IC4Rbits.IC4R = 0b0010; // map IC4 to RB4

    IC4CONbits.ICTMR = 0; // Timer3 is the counter source for capture

    IC4CONbits.SIDL = 0; // Continue to operate in Idle mode ? let it run in idle mode

    IC4CONbits.ICM = 0b110; // Simple Capture Event mode - every edge but with specified edge (rising)
first
```

```
    IC4CONbits.C32 = 0; //  16-bit timer resource capture
    IC4CONbits.ICI = 00; //  Interrupt on every capture event
    IPC4bits.IC4IP = 7; // Interrupt priority level 7, IS THIS OKAY???????
    IC4CONbits.FEDGE = 1; // detect rising edge first
        // IEC0SET = _IEC0_IC4IE_MASK; // --> SHOULD NOT BE SET HERE
    IFS0CLR = _IFS0_IC4IF_MASK;
    IC4CONbits.ON = 1;
}

void RunPsuedoPWM4Ultra4Wall(void)
{
    LATBbits.LATB9 = 1; // set line high
        for(int ii = 0; ii < 23; ii++) // blocking code recommended by Ed over OC to save a timer (approx
10 us)
        {
      LATBbits.LATB9 = 1; // include here so compiler doesn't try to optimize an empty loop
        }
    LATBbits.LATB9 = 0;
        ES_Timer_InitTimer(ULTRA_TIMER, 60);

}

void RunPsuedoPWM4Ultra4Tree(void)
{
    LATBbits.LATB8 = 1; // set line high
        for(int ii = 0; ii < 23; ii++) // blocking code recommended by Ed over OC to save a timer (approx
10 us)
```

```
            {
        LATBbits.LATB8 = 1; // include here so compiler doesn't try to optimize an empty loop
            }
    LATBbits.LATB8 = 0;
            ES_Timer_InitTimer(ULTRA_TIMER, 60);

}

void EnableUltra4Wall(void)
{
            do{
        dummyVariable4Clearing = (uint16_t)IC2BUF;
            }while(IC2CONbits.ICBNE != 0);
    IFS0CLR = _IFS0_IC2IF_MASK;
    IEC0SET = _IEC0_IC2IE_MASK;
}

void DisableUltra4Wall(void)
{
            do{
        dummyVariable4Clearing = (uint16_t)IC2BUF;
            }while(IC2CONbits.ICBNE != 0);
    IFS0CLR = _IFS0_IC2IF_MASK;
    IEC0CLR = _IEC0_IC2IE_MASK;
}
```

```
void EnableUltra4Tree(void)
{
            do{
        dummyVariable4Clearing = (uint16_t)IC4BUF;
            }while(IC4CONbits.ICBNE != 0);
    IFS0CLR = _IFS0_IC4IF_MASK;
    IEC0SET = _IEC0_IC4IE_MASK;
}

void DisableUltra4Tree(void)
{
            do{
        dummyVariable4Clearing = (uint16_t)IC4BUF;
            }while(IC4CONbits.ICBNE != 0);
    IFS0CLR = _IFS0_IC4IF_MASK;
    IEC0CLR = _IEC0_IC4IE_MASK;
}

//void Conversion2Centimeters(void)
//{
//    echoPulse = periodLength * 1.708 / 58; // my personally tested transfer function / conversion
//    distance = (int)echoPulse;
//    DB_printf("Distance in cm is = %d\r\n", distance);
//}
//
//void Conversion2Inches(void)
```

```
//{
//    echoPulse = periodLength * 1.708 / 148; // my personally tested transfer function / conversion
//    distance = (int)echoPulse;
//    DB_printf("Distance in cm is = %d\r\n", distance);
//}


void __ISR(_INPUT_CAPTURE_2_VECTOR, IPL7SOFT) UltrasonicInputCaptureWallISR(void)
{
            do{
        capturedTimeWallUltra = (uint16_t)IC2BUF;
        currentRecordedTimeWallUltra.ByBytes.LowByteRolloverCount = capturedTimeWallUltra;
//        printf("%d\n", periodLength);
            }while(IC2CONbits.ICBNE != 0);


   IFS0CLR = _IFS0_IC2IF_MASK; // we can't clear the flag before reading the IC buffer - but we can't only clear the flag sometimes!
            if((IFS0bits.T3IF == 1) && (capturedTimeWallUltra < 0x8000))
            {
        currentRecordedTimeWallUltra.ByBytes.HighByteRolloverCount++;
        IFS0CLR = _IFS0_T3IF_MASK; // clear again in case
            }


    wallPeriodLength = currentRecordedTimeWallUltra.TotalTime - lastRecordedTimeWallUltra;
    lastRecordedTimeWallUltra = currentRecordedTimeWallUltra.TotalTime;
//    if(CountIC % 2 == 0) // will become a threshold if check
//    {
```

```
//        UltraFired.EventType = ES_ULTRA_DETECTED_WALL; // post to leader SM
//        PostUltrasonicService(UltraFired);
//    }
    wallEchoPulse = wallPeriodLength * 1.708 / 58; // my personally tested transfer function / conversion to cm
    wall_distance = (int)wallEchoPulse;
    ES_Event_t WallEvent;
    WallEvent.EventType = ES_WALL_DETECTED;
            //PostUltrasonicService(WallEvent);
            if (wall_distance < 40) //ignore extraneous pulses
            {
            if(wall_distance <= CLOSE_WALL_THRESHOLD_CM)
            {
           ES_Event_t WallEvent;
           WallEvent.EventType = ES_WALL_DETECTED;
//            PostLeaderSM(WallEvent); //tell SPI follower of new distance
            PostMasterSM_Leader(WallEvent);
           IEC0CLR = _IEC0_IC2IE_MASK;
            }
            }
}

void __ISR(_INPUT_CAPTURE_4_VECTOR, IPL7SOFT) UltrasonicInputCaptureTreeISR(void)
{
            do{
       capturedTimeTreeUltra = (uint16_t)IC4BUF;
```

```
        currentRecordedTimeTreeUltra.ByBytes.LowByteRolloverCount = capturedTimeTreeUltra;
//        printf("%d\n", periodLength);
            }while(IC4CONbits.ICBNE != 0);

   IFS0CLR = _IFS0_IC4IF_MASK; // we can't clear the flag before reading the IC buffer - but we can't only clear the flag sometimes!
            if((IFS0bits.T3IF == 1) && (capturedTimeTreeUltra < 0x8000))
            {
        currentRecordedTimeTreeUltra.ByBytes.HighByteRolloverCount++;
        IFS0CLR = _IFS0_T3IF_MASK; // clear again in case
            }

   treePeriodLength = currentRecordedTimeTreeUltra.TotalTime - lastRecordedTimeTreeUltra;
   lastRecordedTimeTreeUltra = currentRecordedTimeTreeUltra.TotalTime;
//    if(CountIC % 2 == 0) // will become a threshold if check
//    {
//        UltraFired.EventType = ES_ULTRA_DETECTED_WALL; // post to leader SM
//        PostUltrasonicService(UltraFired);
//    }

            /* THIS IS THE BlOCK TO FIX AND I WILL FIX IT 3/6/2024 - 9:00am */
   treeEchoPulse = treePeriodLength * 1.708 / 58; // my personally tested transfer function / conversion to cm
   tree_distance = (int)treeEchoPulse;
   ES_Event_t TreeEvent;
   TreeEvent.EventType = ES_TREE_DETECTED;
            //PostMasterSM_Leader(TreeEvent);
```

```
//    */
            if (tree_distance < 40) //ignore extraneous pulses
            {
            if(tree_distance <= CLOSE_TREE_THRESHOLD_CM)
            {
           ES_Event_t TreeEvent;
           TreeEvent.EventType = ES_TREE_DETECTED;
            PostMasterSM_Leader(TreeEvent);
           IEC0CLR = _IEC0_IC4IE_MASK;
            }
            }
}

void __ISR(_TIMER_3_VECTOR,IPL6SOFT) Timer3Rollover(void){
            if (IFS0bits.T3IF == 1){
       currentRecordedTimeWallUltra.ByBytes.HighByteRolloverCount++; //increment roll-over counter
       currentRecordedTimeTreeUltra.ByBytes.HighByteRolloverCount++; //increment roll-over counter
       IFS0CLR = _IFS0_T3IF_MASK;
            }
}
/*------------------------------ Footnotes -------------------------------*/
/*------------------------------ End of file -------------------------------*/
```