```c
#include "ES_Configure.h"
#include "ES_Framework.h"
#include "ES_DeferRecall.h"
#include "TestHarnessService0.h"

#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "inc/hw_gpio.h"
#include "inc/hw_sysctl.h"
#include "driverlib/sysctl.h"
#include "driverlib/pin_map.h"      // Define PART_TM4C123GH6PM in project
#include "driverlib/gpio.h"
#include "ES_ShortTimer.h"
#include "driverlib/i2c.h"
#include "inc/hw_i2c.h"
#include <math.h>
#include "inc/hw_nvic.h"

#include "Hardware.h"
#include "Constants.h"
#include "I2C_Service.h"

static uint8_t MyPriority;
static I2C_State CurrentState = I2C_Init;
static int16_t Accel_X = 0;
static int16_t Accel_Y = 0;
static int16_t Accel_Z = 0;
static int16_t Gyro_X = 0;
static int16_t Gyro_Y = 0;
static int16_t Gyro_Z = 0;
static int16_t Accel_X_OFF = 0;
static int16_t Accel_Y_OFF = 0;
static int16_t Accel_Z_OFF = 0;
static int16_t Gyro_X_OFF = 0;
static int16_t Gyro_Y_OFF = 0;
static int16_t Gyro_Z_OFF = 0;


static bool read = 0;
static uint8_t Send_Registers[2] = {ACCELEROMETER_POWER_REGISTER,
GYROSCOPE_POWER_REGISTER};
static uint8_t Send_Data[2] = {ACCELEROMETER_POWER_SETTING,
GYROSCOPE_POWER_SETTING};
static uint8_t Receive_Registers[12] = {GYROSCOPE_X_REGISTER_BASE,
GYROSCOPE_X_REGISTER_BASE + 1, GYROSCOPE_Y_REGISTER_BASE,
GYROSCOPE_Y_REGISTER_BASE + 1,
GYROSCOPE_Z_REGISTER_BASE,GYROSCOPE_Z_REGISTER_BASE + 1,
ACCELEROMETER_X_REGISTER_BASE, ACCELEROMETER_X_REGISTER_BASE + 1,
ACCELEROMETER_Y_REGISTER_BASE, ACCELEROMETER_Y_REGISTER_BASE + 1,
ACCELEROMETER_Z_REGISTER_BASE, ACCELEROMETER_Z_REGISTER_BASE + 1};
static uint16_t Receive_Data[12] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};

bool Init_I2C(uint8_t Priority)
{
      // set local priority
  MyPriority = Priority;
  // set timer to allow I2C to hookup
      ES_Timer_InitTimer(IMU_TIMER, I2C_DELAY_TIME);
      // state is init
      CurrentState = I2C_Init;
      return true;
}

bool Post_I2C(ES_Event ThisEvent)
```

```
{
  return ES_PostToService( MyPriority, ThisEvent);
}

ES_Event Run_I2C(ES_Event ThisEvent)
{

      I2C_State NextState = CurrentState;
  ES_Event ReturnEvent;
  ReturnEvent.EventType = ES_NO_EVENT; // assume no errors

      // loop through states
      switch (CurrentState)
      {
            // if state is init
            case (I2C_Init):
            {

                  // if event is IMU_Timeout
                  if ((ThisEvent.EventType == ES_TIMEOUT) && (ThisEvent.EventParam
== IMU_TIMER))
                  {
                        // initialize Gyro/accelerometer power settings
//                      printf("\r\nGyro X\tGyro Y\tGyro Z\tAccel X\tAccel
Y\tAccel Z\r\n");
                        HWREG(I2C2_BASE + I2C_O_MDR) = Send_Registers[1];
                        HWREG(I2C2_BASE + I2C_O_MCS) = I2C_MCS_START_TX;

                        // set IMU Timer
                        ES_Timer_InitTimer(IMU_TIMER, CALIBRATION_TIME);
                        // next state is calibrate
                        NextState = I2C_Poll_IMU;
                  }
                  break;
            }
            // else if state is poll
            case (I2C_Poll_IMU):
            {
                  // if event is timeout
                  if ((ThisEvent.EventType == ES_TIMEOUT) && (ThisEvent.EventParam
== IMU_TIMER))
                  {
                        // reset timer
                        ES_Timer_InitTimer(IMU_TIMER, IMU_POLL_TIME);

                        // set addr to send
                        HWREG(I2C2_BASE + I2C_O_MSA) = IMU_SLAVE_ADDRESS;
                        HWREG(I2C2_BASE + I2C_O_MSA) &= ~I2C_MSA_RS;
                        // load register to read
                        HWREG(I2C2_BASE + I2C_O_MDR) = Receive_Registers[11];
                        // load START TX
                        HWREG(I2C2_BASE + I2C_O_MCS) = I2C_MCS_START_TX;
                  }
                  break;
            }
      }

      CurrentState = NextState;
  return ReturnEvent;
}

//get the upper 8 bits of the X acceleration data
uint8_t getAccelX_MSB(void)
{
```

```
	uint8_t AccelX_MSB;
	//to get the upper 8 bits, bit shift 8 times to the right
	AccelX_MSB = (Accel_X >> 8);
	//return the X MSB byte
	return AccelX_MSB;
}

//get the lower 8 bits of the X acceleration data
uint8_t getAccelX_LSB(void)
{
	uint8_t AccelX_LSB;
	//to get the lower 8 bits, and with 0xff
	AccelX_LSB = (Accel_X & 0xff);
	//return the X LSB byte
	return AccelX_LSB;
}

//get the upper 8 bits of the Y acceleration data
uint8_t getAccelY_MSB(void)
{
	uint8_t AccelY_MSB;
	//to get the upper 8 bits, bit shift 8 times to the right
	AccelY_MSB = (Accel_Y >> 8);
	//return the Y MSB byte
	return AccelY_MSB;
}

//get the lower 8 bits of the Y acceleration data
uint8_t getAccelY_LSB(void)
{
	uint8_t AccelY_LSB;
	//to get the lower 8 bits, and with 0xff
	AccelY_LSB = (Accel_Y & 0xff);
	//return the Y LSB byte
	return AccelY_LSB;
}

//get the upper 8 bits of the Z acceleration data
uint8_t getAccelZ_MSB(void)
{
	uint8_t AccelZ_MSB;
	//to get the upper 8 bits, bit shift 8 times to the right
	AccelZ_MSB = (Accel_Z >> 8);
	//return the Z MSB byte
	return AccelZ_MSB;
}

//get the lower 8 bits of the Z acceleration data
uint8_t getAccelZ_LSB(void)
{
	uint8_t AccelZ_LSB;
	//to get the lower 8 bits, and with 0xff
	AccelZ_LSB = (Accel_Z & 0xff);
	//return the Z LSB byte
	return AccelZ_LSB;
}

//get the upper 8 bits of the X gyro data
uint8_t getGyroX_MSB(void)
{
	uint8_t GyroX_MSB;
	//to get the upper 8 bits, bit shift 8 times to the right
	GyroX_MSB = (Gyro_X >> 8);
	//return the X MSB byte
```

```
	return GyroX_MSB;
}

//get the lower 8 bits of the X gyro data
uint8_t getGyroX_LSB(void)
{
	//to get the lower 8 bits, and with 0xff
	uint8_t GyroX_LSB = (Gyro_X & 0xff);
	//return the X LSB byte
	return GyroX_LSB;
}

//get the upper 8 bits of the Y gyro data
uint8_t getGyroY_MSB(void)
{
	//to get the upper 8 bits, bit shift 8 times to the right
	uint8_t GyroY_MSB = (Gyro_Y >> 8);
	//return the Y MSB byte
	return GyroY_MSB;
}

//get the lower 8 bits of the Y gyro data
uint8_t getGyroY_LSB(void)
{
	//to get the lower 8 bits, and with 0xff
	uint8_t GyroY_LSB = (Gyro_Y & 0xff);
	//return the X LSB byte
	return GyroY_LSB;
}

//get the upper 8 bits of the Gyro Z data
uint8_t getGyroZ_MSB(void)
{
	//to get the upper 8 bits, bit shift 8 times to the right
	uint8_t GyroZ_MSB = (Gyro_Z >> 8);
	//return the Z MSB byte
	return GyroZ_MSB;
}

//get the lower 8 bits of the gyro Z data
uint8_t getGyroZ_LSB(void)
{
	//to get the lower 8 bits, and with 0xff
	uint8_t GyroZ_LSB = (Gyro_Z & 0xff);
	//return the X LSB byte
	return GyroZ_LSB;
}



void I2C_ISR(void)
{
	static uint8_t Read_Index = 0;
	static uint8_t Send_Index = 0;
	static uint8_t Sends_Left = 1;
	static uint8_t Reads_Left = 11;
	//clear the source of the interrupt
	HWREG(I2C2_BASE + I2C_O_MICR) = I2C_MICR_IC;
	//if read is set
	if (read == 1)
	{
		// if index is 0
		if (Read_Index == 0)
		{
```

```
                        for (int i = 0; i<400; i++);
                        // set addr to read
                        HWREG(I2C2_BASE + I2C_O_MSA) = IMU_SLAVE_ADDRESS;
                        HWREG(I2C2_BASE + I2C_O_MSA) |= I2C_MSA_RS;
                        // load START RX
                        HWREG(I2C2_BASE + I2C_O_MCS) = I2C_MCS_SINGLE_RX;
                        // increment index
                        Read_Index ++;
                }
                // else if index is 1
                else if (Read_Index == 1)
                {
                        // read data from buffer
                        Receive_Data[Reads_Left] = (HWREG(I2C2_BASE + I2C_O_MDR) & 0xff);
                        // if reads left is 0
                        if (Reads_Left == 0)
                        {
                                // update Accel/Gyro vals
                                Gyro_X = ((Receive_Data[0]) | (Receive_Data[1] << 8)) -
Gyro_X_OFF;
                                Gyro_Y = ((Receive_Data[2]) | (Receive_Data[3] << 8)) -
Gyro_Y_OFF;
                                Gyro_Z = ((Receive_Data[4]) | (Receive_Data[5] << 8)) -
Gyro_Z_OFF;
                                Accel_X = ((Receive_Data[6]) | (Receive_Data[7] << 8)) -
Accel_X_OFF;
                                Accel_Y = ((Receive_Data[8]) | (Receive_Data[9] << 8)) -
Accel_Y_OFF;
                                Accel_Z = ((Receive_Data[10]) | (Receive_Data[11] << 8)) -
Accel_Z_OFF;

                                Reads_Left = 11;
                                Read_Index = 0;
                        }
                        else
                        {
                                // decrement Reads left
                                Reads_Left --;
                                // reset index to 0
                                Read_Index = 0;
                                // start next read
                                // set addr to send
                                HWREG(I2C2_BASE + I2C_O_MSA) = IMU_SLAVE_ADDRESS;
                                HWREG(I2C2_BASE + I2C_O_MSA) &= ~I2C_MSA_RS;
                                // load register to read
                                HWREG(I2C2_BASE + I2C_O_MDR) =
Receive_Registers[Reads_Left];
                                // load START TX
                                HWREG(I2C2_BASE + I2C_O_MCS) = I2C_MCS_START_TX;
                        }
                }
        }
        // else if not read (send)
        else if (read == 0)
        {
                // if send index is 0
                if (Send_Index == 0)
                {
                        // load Data
                        HWREG(I2C2_BASE + I2C_O_MDR) = Send_Data[Sends_Left];
                        // load LAST TX
                        HWREG(I2C2_BASE + I2C_O_MCS) = I2C_MCS_LAST_TX;
                        // increment send index
                        Send_Index ++;
```

```
            }
            // else if send index is 2
            else if (Send_Index == 1)
            {
                  // if sends left is 1
                  if (Sends_Left != 0)
                  {
                        // decrement sends left
                        Sends_Left --;
                        // load register to write
                        HWREG(I2C2_BASE + I2C_O_MDR) = Send_Registers[Sends_Left];
                  }
                  // else if sends left is 0
                  else if (Sends_Left == 0)
                  {
                        // set read
                        read = 1;
                        // load register to read
                        HWREG(I2C2_BASE + I2C_O_MDR) =
Receive_Registers[Reads_Left];
                  }
                  // set send index to 0
                  Send_Index = 0;
                  // load START TX
                  HWREG(I2C2_BASE + I2C_O_MCS) = I2C_MCS_START_TX;
            }
      }
}
```