```
//#define TEST

#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_nvic.h"
#include "inc/hw_uart.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 "driverlib/uart.h"
#include "ES_ShortTimer.h"
#include "inc/hw_i2c.h"
#include "inc/hw_pwm.h"

#include "ADMulti.h"
#include "Constants.h"

static void IO_Init(void);
static void AD_Init(void);
static void UART_Init(void);
static void I2C_Init(void);
static void PWM_Init(void);

void Hardware_Init(void)
{
	IO_Init();
	AD_Init();
	UART_Init();
	I2C_Init();
	PWM_Init();
}

static void IO_Init(void)
{
	// connect clock to ports B and D
	HWREG(SYSCTL_RCGCGPIO) |= (SYSCTL_RCGCGPIO_R1 | SYSCTL_RCGCGPIO_R3);
	// wait for clock to connect to ports B and F
	while ((HWREG(SYSCTL_PRGPIO) & (SYSCTL_PRGPIO_R1 | SYSCTL_PRGPIO_R3)) !=
(SYSCTL_PRGPIO_R1 | SYSCTL_PRGPIO_R3)) {}
	// digitally enable IO pins
	HWREG(GPIO_PORTB_BASE + GPIO_O_DEN) |= (R_BUTTON_B | L_BUTTON_B | Y_LED_1_B
| G_LED_1_B | Y_LED_2_B | G_LED_2_B | Y_LED_3_B | G_LED_3_B);
	HWREG(GPIO_PORTD_BASE + GPIO_O_DEN) |= (REVERSE_BUTTON_D |
PERIPHERAL_BUTTON_D | SPEAKER_PIN_D);
	// set direction of IO pins
	HWREG(GPIO_PORTB_BASE + GPIO_O_DIR) &= ~(R_BUTTON_B | L_BUTTON_B);
	HWREG(GPIO_PORTB_BASE + GPIO_O_DIR) |= (Y_LED_1_B | G_LED_1_B | Y_LED_2_B |
G_LED_2_B | Y_LED_3_B | G_LED_3_B);
	HWREG(GPIO_PORTD_BASE + GPIO_O_DIR) &= ~(REVERSE_BUTTON_D |
PERIPHERAL_BUTTON_D);
	HWREG(GPIO_PORTD_BASE + GPIO_O_DIR) |= (SPEAKER_PIN_D);
	//enable internal pullup s
	HWREG(GPIO_PORTB_BASE + GPIO_O_PUR) |= (R_BUTTON_B | L_BUTTON_B);
	HWREG(GPIO_PORTD_BASE + GPIO_O_PUR) |= (REVERSE_BUTTON_D |
PERIPHERAL_BUTTON_D);

}
```

```
static void AD_Init(void)
{
	// Connect clock to port E
	HWREG(SYSCTL_RCGCGPIO)|=SYSCTL_RCGCGPIO_R4;
	// wait for clock to connect to port E
	while((HWREG(SYSCTL_PRGPIO)& SYSCTL_PRGPIO_R4)!=SYSCTL_PRGPIO_R4){}
	// digitally enable Anaolog Pins (I realize this doesn't make any sense, it's
2 am leave me alone)
	HWREG(GPIO_PORTE_BASE + GPIO_O_DEN) |= (SOUND_PIN_E);
	// set direction of Analog Pins
	HWREG(GPIO_PORTE_BASE + GPIO_O_DIR) &= ~(SOUND_PIN_E);
	ADC_MultiInit(NUMBER_OF_ANALOG_PINS);
}

static void UART_Init(void)
{
	//Enable the clock for the UART module
	HWREG(SYSCTL_RCGCUART) |= SYSCTL_RCGCUART_R1;

	//Wait for the UART to be ready
	while((HWREG(SYSCTL_PRUART)& SYSCTL_PRUART_R1)!=SYSCTL_PRUART_R1){}

	//Enable the clock to the appropriate gpio module via the RCGCGPIO - port C
	HWREG(SYSCTL_RCGCGPIO) |= SYSCTL_RCGCGPIO_R2;

	//Wait for the GPIO module to be ready
	while((HWREG(SYSCTL_PRGPIO)& SYSCTL_PRGPIO_R2)!=SYSCTL_PRGPIO_R2){}

	//Configure the GPIO pins for in/out/drive-level/drive-type
	HWREG(GPIO_PORTC_BASE+GPIO_O_DEN) |= (GPIO_PIN_4 | GPIO_PIN_5);
	HWREG(GPIO_PORTC_BASE+GPIO_O_DIR) |= GPIO_PIN_5;
	HWREG(GPIO_PORTC_BASE+GPIO_O_DIR) &= ~GPIO_PIN_4;

	//Select the Alternate function for the UART pins
	HWREG(GPIO_PORTC_BASE+GPIO_O_AFSEL) |= (BIT4HI | BIT5HI);

	//Configure the PMCn fields in the GPIOPCTL register to assign the UART pins
	HWREG(GPIO_PORTC_BASE+GPIO_O_PCTL) = (HWREG(GPIO_PORTC_BASE+GPIO_O_PCTL) &
0Xfff0ffff) + (RX_ALT_FUNC<<(RX_PIN*BITS_PER_NIBBLE));
	HWREG(GPIO_PORTC_BASE+GPIO_O_PCTL) = (HWREG(GPIO_PORTC_BASE+GPIO_O_PCTL) &
0Xff0fffff) + (TX_ALT_FUNC<<(TX_PIN*BITS_PER_NIBBLE));

	//Disable the UART by clearing the UARTEN bit in the UARTCTL register
	HWREG(UART1_BASE+UART_O_CTL) = HWREG(UART1_BASE + UART_O_CTL) &
~UART_CTL_UARTEN;

	//Write the integer portion of the BRD
	HWREG(UART1_BASE + UART_O_IBRD) = BAUD_RATE_INT;

	//Write the fraction portion of the BRD
	HWREG(UART1_BASE + UART_O_FBRD) = BAUD_RATE_FRAC;

	//Write the desired serial parameters
	HWREG(UART1_BASE + UART_O_LCRH) = HWREG(UART1_BASE + UART_O_LCRH) |
UART_LCRH_WLEN_8;

	//Enable RX and TX interrupts in mask
	HWREG(UART1_BASE + UART_O_IM) = HWREG(UART1_BASE + UART_O_IM) | UART_IM_RXIM;

	//Configure the UART operation
	//Enable the UART
	HWREG(UART1_BASE + UART_O_CTL) = HWREG(UART1_BASE + UART_O_CTL) |
UART_CTL_UARTEN;
```

```
	//Enable interrupt in the NVIC
	HWREG(NVIC_EN0) |= BIT6HI;

	//Enable interrupts globally
	__enable_irq();

	//Print successful initialization
	printf("UART 1 Successfully Initialized! :)\r\n");
}

static void I2C_Init(void)
{
	// enable the I2C clock for I2C module 2
	HWREG(SYSCTL_RCGCI2C) |= SYSCTL_RCGCI2C_R2;
	while ((HWREG(SYSCTL_PRI2C) & SYSCTL_PRI2C_R2) != SYSCTL_PRI2C_R2) {}
	// enable clock to GPIO pins on I2C2 (E4, E5)
	HWREG(SYSCTL_RCGCGPIO) |= SYSCTL_RCGCGPIO_R4;
	while ((HWREG(SYSCTL_PRGPIO) & SYSCTL_PRGPIO_R4) != SYSCTL_PRGPIO_R4) {}
	//enable internal pullups
	HWREG(GPIO_PORTE_BASE + GPIO_O_PUR) |= (I2C_SDA_PIN | I2C_SCL_PIN);
	// digitally enable maybe?
	HWREG(GPIO_PORTE_BASE + GPIO_O_DEN) |= (I2C_SDA_PIN | I2C_SCL_PIN);
	// select alternate functions for B2, B3
	HWREG(GPIO_PORTE_BASE + GPIO_O_AFSEL) |= (I2C_SDA_PIN | I2C_SCL_PIN);
	// set SDA to Open Drain
	HWREG(GPIO_PORTE_BASE + GPIO_O_ODR) |= I2C_SDA_PIN;
	// select I2C function
	HWREG(GPIO_PORTE_BASE + GPIO_O_PCTL) = ((HWREG(GPIO_PORTE_BASE + GPIO_O_PCTL)
& I2C_PIN_M) | ((3 << (I2C_SDA_BIT*BitsPerNibble)) | (3 <<
(I2C_SCL_BIT*BitsPerNibble))));
	// initialize the TIVA as Master
	HWREG(I2C2_BASE + I2C_O_MCR) |= I2C_MCR_MFE;
	// set the SCL clock (there is a fancy equation, I'm just using the provided
10KBPS val given)
	HWREG(I2C2_BASE + I2C_O_MTPR) = ((HWREG(I2C2_BASE + I2C_O_MTPR) &
~(I2C_MTPR_TPR_M)) | I2C_COMM_SPEED);
	// Load Slave address
	HWREG(I2C2_BASE + I2C_O_MSA) = IMU_SLAVE_ADDRESS;
	// set up ISR
	HWREG(NVIC_EN2) |= BIT4HI;
	HWREG(I2C2_BASE + I2C_O_MIMR) |= I2C_MIMR_IM;
}

static void PWM_Init(void)
{

	// Enable the clock to the PWM Module
	HWREG(SYSCTL_RCGCPWM) |= (SYSCTL_RCGCPWM_R1);
	while ((HWREG(SYSCTL_PRPWM) & (SYSCTL_PRPWM_R1)) != (SYSCTL_PRPWM_R1)) {}
	// Enable the clock to Port B and F
	HWREG(SYSCTL_RCGCGPIO) |= (SYSCTL_RCGCGPIO_R5);
	while ((HWREG(SYSCTL_PRGPIO) & (SYSCTL_PRGPIO_R5)) != (SYSCTL_PRGPIO_R5)) {}
	// digitially enable the PWM pins

	HWREG(GPIO_PORTF_BASE+GPIO_O_DEN) |= (RIGHT_VIBRATION_MOTOR_F |
LEFT_VIBRATION_MOTOR_F);
	HWREG(GPIO_PORTF_BASE+GPIO_O_DIR) |= (RIGHT_VIBRATION_MOTOR_F |
LEFT_VIBRATION_MOTOR_F);
	// Select the system clock/32
	HWREG(SYSCTL_RCC) = (HWREG(SYSCTL_RCC) & ~SYSCTL_RCC_PWMDIV_M) |
(SYSCTL_RCC_USEPWMDIV | SYSCTL_RCC_PWMDIV_32);
	// Disable the PWM generator while initializing
	HWREG(PWM1_BASE + PWM_O_3_CTL) = 0;
```

```
	// Set initial generator values: motors should be stopped, servos at idle
	HWREG(PWM1_BASE + PWM_O_3_GENA) = PWM_1_GENA_ACTZERO_ZERO;
	HWREG(PWM1_BASE + PWM_O_3_GENB) = PWM_1_GENB_ACTZERO_ZERO;
	// Set the load to ½ the desired period since going up and down
	HWREG(PWM1_BASE + PWM_O_3_LOAD) = ((MOTOR_PWM_PERIOD) >> 1);
	// Enable the PWM outputs
	HWREG(PWM1_BASE + PWM_O_ENABLE) |= (PWM_ENABLE_PWM6EN | PWM_ENABLE_PWM7EN);
	// Select the alternate function for PWM Pins
	HWREG(GPIO_PORTF_BASE + GPIO_O_AFSEL) |= (LEFT_VIBRATION_MOTOR_F |
RIGHT_VIBRATION_MOTOR_F);
	// Choose to map PWM to those pins
	HWREG(GPIO_PORTF_BASE + GPIO_O_PCTL) = (HWREG(GPIO_PORTF_BASE + GPIO_O_PCTL)
& PWM_PIN_M_F) + (5<<(RIGHT_VIBRATION_MOTOR_BIT*BitsPerNibble)) +
(5<<(LEFT_VIBRATION_MOTOR_BIT*BitsPerNibble));
	// Set the up/down count mode
	// Enable the PWM generator
	// Make generator updates locally synchronized to zero count
	HWREG(PWM1_BASE + PWM_O_3_CTL) = (PWM_3_CTL_MODE | PWM_3_CTL_ENABLE |
PWM_3_CTL_GENAUPD_LS | PWM_3_CTL_GENBUPD_LS);

}

void SetDutyRightVibrationMotor(uint8_t duty)
{

	// Motor starts at rest
	static bool restoreRVM = true;

	// New Value for comparator to set duty cycle
	static uint32_t newCmp;
	// set new comparator value based on duty cycle
	newCmp = HWREG(PWM1_BASE + PWM_O_3_LOAD)*(100-duty)/100;
	if (duty == 100 | duty == 0)
	{
		restoreRVM = true;
		if (duty == 100)
		{
			// To program 100% DC, simply set the action on Zero to set the
output to one
			HWREG(PWM1_BASE + PWM_O_3_GENA) = PWM_3_GENA_ACTZERO_ONE;
		}
		else
		{
			// To program 0% DC, simply set the action on Zero to set the output
to zero
			HWREG(PWM1_BASE + PWM_O_3_GENA) = PWM_3_GENA_ACTZERO_ZERO;
		}
	}
	else
	{
		// if returning from 0 or 100
		if (restoreRVM)
		{
			restoreRVM = false;
			// restore normal operation
			HWREG(PWM1_BASE + PWM_O_3_GENA) = GenA_1_3_Normal;
		}
		// write new comparator value to register
		HWREG(PWM1_BASE + PWM_O_3_CMPA) = newCmp;
	}

}

void SetDutyLeftVibrationMotor(uint8_t duty)
```

```
{

	// Motor starts at rest
	static bool restoreLVM = true;

	// New Value for comparator to set duty cycle
	static uint32_t newCmp;
	// set new comparator value based on duty cycle
	newCmp = HWREG(PWM1_BASE + PWM_O_3_LOAD)*(100-duty)/100;
	if (duty == 100 | duty == 0)
	{
		restoreLVM = true;
		if (duty == 100)
		{
			// To program 100% DC, simply set the action on Zero to set the
output to one
			HWREG(PWM1_BASE + PWM_O_3_GENB) = PWM_3_GENB_ACTZERO_ONE;
		}
		else
		{
			// To program 0% DC, simply set the action on Zero to set the output
to zero
			HWREG(PWM1_BASE + PWM_O_3_GENB) = PWM_3_GENB_ACTZERO_ZERO;
		}
	}
	else
	{
		// if returning from 0 or 100
		if (restoreLVM)
		{
			restoreLVM = false;
			// restore normal operation
			HWREG(PWM1_BASE + PWM_O_3_GENB) = GenB_1_3_Normal;
		}
		// write new comparator value to register
		HWREG(PWM1_BASE + PWM_O_3_CMPB) = newCmp;
	}

}

#ifdef TEST
int main(void)
{
	SysCtlClockSet(SYSCTL_SYSDIV_5 | SYSCTL_USE_PLL | SYSCTL_OSC_MAIN
			| SYSCTL_XTAL_16MHZ);
	TERMIO_Init();
	Hardware_Init();
	SetDutyLeftVibrationMotor(100);
	SetDutyRightVibrationMotor(100);
//	HWREG(GPIO_PORTB_BASE + (ALL_BITS + GPIO_O_DATA)) |= (R_BUTTON_B | L_BUTTON_B
| Y_LED_1_B | G_LED_1_B | Y_LED_2_B | G_LED_2_B | Y_LED_3_B | G_LED_3_B);
//	HWREG(GPIO_PORTD_BASE + (ALL_BITS + GPIO_O_DATA)) |= (SPEAKER_PIN_D);
	while(1) {};
	return 0;
}
#endif
```