```
////////////////////////////////////////////////////////////////////
// Lab 3 - PWM
//Colin & Daniel
// PWM Part #2
////////////////////////////////////////////////////////////////////

#include <clock.h>
#include <conf_clocks.h>
#include <fastmath.h>

//setup a the correct TC pointer for the corresponding pins. (use table 5-1 as a reference)
/* Set correct PA pins as TC pins for PWM operation */

Tc *TCptr = (Tc *)0x42003000UL;
Adc *ADC_Ptr = (Adc *)0x42004000UL;

int k=0;

void enable_port(PortGroup *porA, PortGroup *porB);
void enable_tc_clocks(void);
void enable_tc(int dutyCycle0, int dutyCycle1, PortGroup *porA, PortGroup *porB);

void enable_adc_clocks(void);
void init_adc(PortGroup *porA);

void display(int value, int decimal_place, PortGroup *porA, PortGroup *porB);
void digit(int position, int displayValue, int decimal_place, PortGroup *porA, PortGroup *porB);
void displayInit(PortGroup *porA, PortGroup *porB);
void wait(int t);
unsigned int read_adc(void);

void enable_port(PortGroup *porA, PortGroup *porB)                //setup pins
{
        //porA -> PMUX[6].bit.PMUXO = 0x4;                  //Port 13 - Peripheral Group
        //porA -> PINCFG[13].bit.PMUXEN = 0x1 ;  //Port 13 - Use PMUXEN

        porA->PMUX[11].bit.PMUXE = 0x5;                   //Port 22 - Peripheral Group
        porA->PMUX[11].bit.PMUXO = 0x5;                   //Port 23 - Peripheral Group
        porA->PINCFG[22].bit.PMUXEN = 0x1;                //Port 22 - Use PMUXEN
        porA->PINCFG[23].bit.PMUXEN = 0x1;                //Port 23 - Use PMUXEN
}

void enable_tc_clocks(void)
```

```
{
	/* Perform Clock configuration to source the TC
	1) ENABLE THE APBC CLOCK FOR THE CORREECT MODULE
	2) WRITE THE PROPER GENERIC CLOCK SELETION ID*/

	PM->APBCMASK.reg |=0x1 << 12;		// PM_APBCMASK_______ is in the ___
position <==== Different from Part 1
	uint32_t temp=0x15;					// ID for _________ is ___________
(see table 14-2)
	temp |= 0<<8;						// Selection Generic clock generator
0
	GCLK->CLKCTRL.reg=temp;					// Setup in the CLKCTRL
register
	GCLK->CLKCTRL.reg |= 0x1u << 14;	// enable it.
}
void enable_tc(int dutyCycle0, int dutyCycle1, PortGroup *porA, PortGroup *porB)
{
	/* Configure the basic timer/counter to have a period of_________ or a
	frequency of __________ */

	enable_port(porA, porB);
	enable_tc_clocks();

	TCptr->COUNT8.CTRLA.bit.MODE=0x1;
	TCptr->COUNT8.CTRLA.bit.PRESCALER=0x0;	//no prescaler
	TCptr->COUNT8.CTRLA.bit.PRESCSYNC=0x1;

	//TCptr->COUNT8.CTRLA.bit.WAVEGEN=0x3;	//match PWM mode
	//TCptr->COUNT8.CTRLA.bit.WAVEGEN=0x1;	//match frequency mode

	TCptr->COUNT8.CTRLA.bit.WAVEGEN=0x2;	//normal PWM mode

	TCptr->COUNT8.CC[1].reg = dutyCycle1;	//Set first duty cycle
	TCptr->COUNT8.CC[0].reg = dutyCycle0;	//Set second duty cycle

	TCptr->COUNT8.PER.reg = 125;			//1017 period for 0x5 prescaler for
500 Hz

	while(TCptr->COUNT8.STATUS.reg & TC_STATUS_SYNCBUSY) {}

	TCptr ->COUNT8.CTRLA.reg |= 0x2;

}
```

```
void enable_adc_clocks(void)
{
	struct system_gclk_chan_config gclk_chan_conf;

	gclk_chan_conf.source_generator = GCLK_GENERATOR_0;
	system_gclk_chan_set_config(ADC_GCLK_ID , &gclk_chan_conf);

	//Enable the generic clock for ADC
	system_gclk_chan_enable(ADC_GCLK_ID );
}

void init_adc(PortGroup *porA)
{
	ADC_Ptr -> CTRLA.reg = 0x0;  //adc disabled + reset operation ongoing

	ADC_Ptr -> REFCTRL.reg = 0x2;
	ADC_Ptr -> AVGCTRL.reg = 0x0;
	ADC_Ptr -> SAMPCTRL.reg =ADC_AVGCTRL_SAMPLENUM_1_Val;
	ADC_Ptr -> CTRLB.reg =
ADC_CTRLB_RESSEL_12BIT|ADC_CTRLB_PRESCALER_DIV32;	//(12 bit resolution
running with differential mode)
	ADC_Ptr -> INPUTCTRL.reg =
ADC_INPUTCTRL_GAIN_DIV2|ADC_INPUTCTRL_MUXNEG_GND|ADC_INPUTCTRL_MU
XPOS_PIN0 ; //(gain , muxneg, muxpos)

	// config PA02 to be owned by ADC Peripheral

	porA -> DIRSET.reg = PORT_PA13;
	porA -> OUTSET.reg = PORT_PA13;

	porA -> PMUX[1].bit.PMUXE = 0x1;
	porA -> PINCFG[2].bit.PMUXEN =0x1;


	ADC_Ptr -> CTRLA.reg = 0x2;  //adc enabled + no reset operation ongoing

}

void display(int value, int decimal_place, PortGroup *porA, PortGroup *porB){

	int u0,u1,u2,u3;
```

```
	u0 = value / 1000;
	u1 = (value - (u0*1000)) / 100;
	u2 = (value - (u0*1000) - (u1*100)) / 10;
	u3 = (value - (u0*1000) - (u1*100) - (u2*10));

	digit(1,u0,1,porA,porB);
	digit(2,u1,1,porA,porB);
	digit(3,u2,1,porA,porB);
	digit(4,u3,1,porA,porB);

}

void digit(int position, int displayValue, int decimal_place, PortGroup *porA, PortGroup *porB){
//Digit function: Scans for input, displays values, and performs mathematic operations

	porA->OUTSET.reg = PORT_PA04|PORT_PA05|PORT_PA06|PORT_PA07;
//Reset all used ports
	porB->OUTSET.reg =
PORT_PA00|PORT_PB01|PORT_PB02|PORT_PB03|PORT_PB04|PORT_PB05|PORT_PB0
6|PORT_PB07|PORT_PB09;

	switch(position){                                    //Determine which digit to illuminate
based on the passed "position" value
		case 1:
		porA->OUTCLR.reg = PORT_PA07;
		break;

		case 2:
		porA->OUTCLR.reg = PORT_PA06;
		break;

		case 3:
		porA->OUTCLR.reg = PORT_PA05;
		break;

		case 4:
		porA->OUTCLR.reg = PORT_PA04;
		break;
	}

	switch(displayValue){		//Displays numeric values 0-9, case '11' is blank
		case 1:
		if (decimal_place == position){
```

```
                porB->OUTCLR.reg = PORT_PB01 | PORT_PB02 | PORT_PB07;
            }
            else
            porB->OUTCLR.reg = PORT_PB01 | PORT_PB02;
            break;

            case 2:
            if (decimal_place == position){
                porB->OUTCLR.reg = PORT_PB00 | PORT_PB01 | PORT_PB03 |
PORT_PB04 | PORT_PB06 | PORT_PB07;
            }
            else
            porB->OUTCLR.reg = PORT_PB00 | PORT_PB01 | PORT_PB03 |
PORT_PB04 | PORT_PB06;
            break;

            case 3:
            if (decimal_place == position){
                porB->OUTCLR.reg = PORT_PB00 | PORT_PB01 | PORT_PB02 |
PORT_PB03 | PORT_PB06 | PORT_PB07;
            }
            else
            porB->OUTCLR.reg = PORT_PB00 | PORT_PB01 | PORT_PB02 |
PORT_PB03 | PORT_PB06;
            break;

            case 4:
            if (decimal_place == position){
                porB->OUTCLR.reg = PORT_PB01 | PORT_PB02 | PORT_PB05 |
PORT_PB06 | PORT_PB07;
            }
            else
            porB->OUTCLR.reg = PORT_PB01 | PORT_PB02 | PORT_PB05 |
PORT_PB06;
            break;

            case 5:
            if (decimal_place == position){
                porB->OUTCLR.reg = PORT_PB00 | PORT_PB02 | PORT_PB03 |
PORT_PB05 | PORT_PB06 | PORT_PB07;
            }
            else
```

```
                porB->OUTCLR.reg = PORT_PB00 | PORT_PB02 | PORT_PB03 |
PORT_PB05 | PORT_PB06;
                break;

                case 6:
                if (decimal_place == position){
                        porB->OUTCLR.reg = PORT_PB00 | PORT_PB02 | PORT_PB03 |
PORT_PB04 | PORT_PB05 | PORT_PB06 | PORT_PB07;
                }
                else
                porB->OUTCLR.reg = PORT_PB00 | PORT_PB02 | PORT_PB03 |
PORT_PB04 | PORT_PB05 | PORT_PB06;
                break;

                case 7:
                if (decimal_place == position){
                        porB->OUTCLR.reg = PORT_PB00 | PORT_PB01 | PORT_PB02 |
PORT_PB07;
                }
                else
                porB->OUTCLR.reg = PORT_PB00 | PORT_PB01 | PORT_PB02;
                break;

                case 8:
                if (decimal_place == position){
                        porB->OUTCLR.reg = PORT_PB00 | PORT_PB01 | PORT_PB02 |
PORT_PB03 | PORT_PB04 | PORT_PB05 | PORT_PB06 | PORT_PB07;
                }
                else
                porB->OUTCLR.reg = PORT_PB00 | PORT_PB01 | PORT_PB02 |
PORT_PB03 | PORT_PB04 | PORT_PB05 | PORT_PB06;
                break;

                case 9:
                if (decimal_place == position){
                        porB->OUTCLR.reg = PORT_PB00 | PORT_PB01 | PORT_PB02 |
PORT_PB05 | PORT_PB06 | PORT_PB07;
                }
                else
                porB->OUTCLR.reg = PORT_PB00 | PORT_PB01 | PORT_PB02 |
PORT_PB05 | PORT_PB06;
                break;
```

```
                case 0:
                if (decimal_place == position){
                        porB->OUTCLR.reg = PORT_PB00 | PORT_PB01 | PORT_PB02 |
PORT_PB03 | PORT_PB04 | PORT_PB05 | PORT_PB07;
                }
                else
                porB->OUTCLR.reg = PORT_PB00 | PORT_PB01 | PORT_PB02 |
PORT_PB03 | PORT_PB04 | PORT_PB05;
                break;
        }
        wait(500);
}

void displayInit(PortGroup *porA, PortGroup *porB){
        porA->DIRSET.reg = PORT_PA04|PORT_PA05|PORT_PA06|PORT_PA07;
//Transistor outputs, ACTIVE LOW
        porB->DIRSET.reg =
PORT_PB00|PORT_PB01|PORT_PB02|PORT_PB03|PORT_PB04|PORT_PB05|PORT_PB0
6|PORT_PB07|PORT_PB09; //7 Segment Display Pins, ACTIVE LOW

        porB->OUTSET.reg =
PORT_PB00|PORT_PB01|PORT_PB02|PORT_PB03|PORT_PB04|PORT_PB05|PORT_PB0
6|PORT_PB07|PORT_PB09; //Reseting Pins to all 0
        porA->OUTSET.reg = PORT_PA04|PORT_PA05|PORT_PA06|PORT_PA07;

        for(int i=0; i<7; i++){               //Toggle drive strength high for LED output
                porB->PINCFG[i].reg=PORT_PINCFG_DRVSTR;
        }
        for(int i=16; i<20; i++){             //Configure keypad as input
                porA->PINCFG[i].reg=PORT_PINCFG_INEN | PORT_PINCFG_PULLEN;
        }
}

void wait(int t)                        //Wait function: Simple wait function
{
        int count = 0;
        while (count < t)
        {
                count++;
        }
}

unsigned int read_adc()
```

```
{

	// start the conversion
	ADC_Ptr -> SWTRIG.reg = 0x2; // starts adc conversion but does not flush pipeline

	while(!(ADC_Ptr->INTFLAG.bit.RESRDY));  //wait for conversion to be available

	return(ADC_Ptr-> RESULT.reg ); //insert register where ADC store value

}

int main (void)
{
	system_clock_init();

	Port *ports = PORT_INSTS;
	PortGroup *porA = &(ports->Group[0]);
	PortGroup *porB = &(ports->Group[1]);

	enable_adc_clocks();
	init_adc(porA);
	displayInit(porA,porB);

	int g=0;
	int x=0;

	int duty1=62;
	int duty2=62;

	while(1)
	{
		x = read_adc();			//Read in voltage across POT

		x = x*3285;			//Scale incoming ADC value
		g = x>>12;

		if(g>3273){			//Upper bound for motor control
			g=3273;
		}
		if(g<27){			//Lower bound for motor control
			g=27;
		}
```

```
                display(g,1,porA,porB); //(integer value, decimal place, port group, port group);

                if(g>=1500 && g<=1800){             //Dead zone in middle to stop motor spin
                        duty2= 62;
                        duty1= 62;
                }

                else if(g<1500 || g>1800){          //Scaling factor for both duty cycles
                        duty2 = abs(3285-g)*0.0375;
                        duty1 = g* 0.0375;
                }

                enable_tc(duty1,duty2,porA,porB);   //imports the raw values from the array and
inserts it into the TC register as a duty cycle
                wait(10000);
        }

}
```