```
/**

 */
#include <asf.h>
#include <clock.h>
#include <conf_clocks.h>

Tc *TCptr2 = (Tc*)0x42002800UL;
Tc *TCptr4 = (Tc*)0x42003000UL;
Dac *DAC_Ptr = (Dac *)0x42004800UL;

Adc *ADC_Ptr = (Adc*)0x42004000UL;

float input_value;
float output_value;
float y,ya,yb,yc,yd,ye;
float u,u1,u2,u3,u4,u5;

int state = 1;                  //60Hz Second Order Notch Filter
//int state = 2;    //100Hz First Order Low Pass Filter

void enable_port(void);
void enable_adc_clocks(void);
void init_adc(void);
unsigned int read_adc(void);
void enable_tc_clocks(void);
void enable_tc(void);
void configure_dac(void);
void configure_dac_clock(void);


void enable_port(void)                  //setup pins
{
        Port *ports = PORT_INSTS;
        PortGroup *porA = &(ports->Group[0]);
//      PortGroup *porB = &(ports->Group[1]);

        porA->DIRSET.reg = PORT_PA13|PORT_PA11|PORT_PA22;
        //porA->DIRCLR.reg = PORT_PA16|PORT_PA17|PORT_PA18|PORT_PA19; //Keypad Inputs

}

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);

        system_gclk_chan_enable(ADC_GCLK_ID );
}

void init_adc(void)
{
        Port *ports = PORT_INSTS;
        PortGroup *porA = &(ports->Group[0]);

        ADC_Ptr -> CTRLA.reg = 0x0;  //adc disabled + reset operation ongoing

        ADC_Ptr -> REFCTRL.reg = 0x2;
        ADC_Ptr -> AVGCTRL.reg = 0x0;
        ADC_Ptr -> SAMPCTRL.reg =0x1F;
        ADC_Ptr -> CTRLB.reg = ADC_CTRLB_RESSEL_10BIT|ADC_CTRLB_PRESCALER_DIV4; //(12 bit resolution running with differential mode)
        ADC_Ptr -> INPUTCTRL.reg = ADC_INPUTCTRL_GAIN_DIV2|ADC_INPUTCTRL_MUXNEG_GND|ADC_INPUTCTRL_MUXPOS_PIN19 ;
        //(gain , muxneg, muxpos)

        porA -> PMUX[5].bit.PMUXO = 0x1;
        porA -> PINCFG[11].bit.PMUXEN =0x1;


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

```
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

}

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->APBAMASK.reg |=0x1 << 6;	// PM_APBAMASK for the EIC
	PM->APBCMASK.reg |=0x1 << 10;		// PM_APBCMASK for TC2
	PM->APBCMASK.reg |=0x1 << 12;		// PM_APBC MASK for TC4

	uint32_t temp=0x14;					// 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.

	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.

	temp=0x03;							// 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(void)
{
	enable_port();
	enable_tc_clocks();

	TCptr4 ->COUNT8.CTRLA.bit.MODE=0x1; //normal frequency operation
	TCptr4 ->COUNT8.CTRLA.bit.PRESCALER=0x5;
	TCptr4 ->COUNT8.CTRLA.bit.PRESCSYNC=0x1;
	//TCptr4 ->COUNT8.CTRLA.bit.WAVEGEN=0x2;
	//TCptr4 ->COUNT8.CC[1].reg = 87;
	//TCptr4 ->COUNT8.CC[0].reg = 87;
	TCptr4 ->COUNT8.PER.reg = 0x7C;   //174 period for 0x6 prescaler for 500 Hz
	TCptr4 ->COUNT8.INTENSET.bit.OVF = 0x1;

	while(TCptr4->COUNT8.STATUS.reg & TC_STATUS_SYNCBUSY) {}
	TCptr4 ->COUNT8.CTRLA.reg |= 0x2;
	//NVIC_EnableIRQ(TC2_IRQn);
	NVIC_EnableIRQ(TC4_IRQn);
	//NVIC -> IP[3] = 0xC0000000;
	//NVIC -> IP[4] = 0x00008000;
}
```

```
void TC4_Handler(void){
	Port *ports = PORT_INSTS;
	PortGroup *porA = &(ports->Group[0]);
//	PortGroup *porB = &(ports->Group[1]);

	porA->OUTTGL.reg = PORT_PA13;

	input_value = read_adc();

	u = input_value*(3.3)/2047;

	if(state==1){  //notch filter (needs to fix)
		porA->OUTSET.reg = PORT_PA22;
		y = u - (1.862*u1) + (0.9986*u2) + (1.803*ya) - (0.9391*yb);
		porA->OUTCLR.reg = PORT_PA22;
		output_value = y*(2047)/3.3;
		yb = ya;
		ya = y;
		u2 = u1;
		u1 = u;
	}

	if(state==2){    //lowpass filter 100Hz

		y = (0.5335*ya + 0.4665*u1);

		u1 = u;
		ya = y;
		output_value = y*(2047)/3.3;
	}


	while (DAC_Ptr->STATUS.reg & DAC_STATUS_SYNCBUSY) {}


	DAC_Ptr->DATA.reg = output_value;

	TCptr4->COUNT8.INTFLAG.bit.OVF = 0x1; //clears the interrupt flag

}

void configure_dac_clock(void)
{
	/* Turn on the digital interface clock */
	system_apb_clock_set_mask(SYSTEM_CLOCK_APB_APBC, PM_APBCMASK_DAC);

	/* Configure GCLK channel and enable clock */
	struct system_gclk_chan_config gclk_chan_conf;
	system_gclk_chan_get_config_defaults(&gclk_chan_conf);
	gclk_chan_conf.source_generator = GCLK_GENERATOR_0;
	system_gclk_chan_set_config(DAC_GCLK_ID, &gclk_chan_conf);
	system_gclk_chan_enable(DAC_GCLK_ID);
}

void configure_dac(void)
{
	//set pin as output for the dac
	Port *ports = PORT_INSTS;
	PortGroup *porA = &(ports->Group[0]);
//	PortGroup *porB = &(ports->Group[1]);

	porA->PINCFG[2].bit.PMUXEN = 0x1;		//set to correct pin configuration
	porA->PMUX[2].bit.PMUXE = 0x1;			//set to correct peripheral

	while (DAC_Ptr->STATUS.reg & DAC_STATUS_SYNCBUSY) {}		/* Wait until the synchronization is complete */

	DAC_Ptr->CTRLB.reg = 0b01000000;		/* Set reference voltage with CTRLB */

	while (DAC_Ptr->STATUS.reg & DAC_STATUS_SYNCBUSY) {}		/* Wait until the synchronization is complete */

	DAC_Ptr->CTRLA.reg = 0b00000010;		/* Enable the module with CTRLA */
	DAC_Ptr->CTRLB.reg = 0b01000001;

}
```

```
int main (void){

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

		//y = ya = yb = u = u1 = u2 = 0;

		system_clock_init();
		enable_tc();
		enable_adc_clocks();
		init_adc();
		configure_dac_clock();
		configure_dac();

		while(1){
			state=2;
		}
}
```