Apéndice B

Biblioteca Analog.h

En este apartado se desglosa el código de la biblioteca *Analog.h*, la cual configura los módulos ADC y DAC utilizando las configuraciones propuestas en ControlSuite, encapsulando dichos procedimientos en métodos simples.

```
/*
 * Analog.h
 *
 * Esta libreria contiene funciones para la
 * configuración del subsistema analógico del
 * DSP TMS320F28377S que se encuentra en el
 * kit de desarrollo LAUNCHXL-F28377S. Este
 * subsistema cuenta con dos ADC (A y B), con
 * 6 canales cada uno (0-5) y dos canales
 * compartidos (14 y 15), los cuales pueden ser
 * usados en modo de 12 y 16 bits, la configuración
 * empleada usa 12 bits.
 *
 * Así mismo, tiene tres DAC de 12 bits (A,B y C),
 * los cuales emplean los voltajes de referencia
 * internos.
 *
 * Se emplea la configuración de reloj básica:
 * InitSysPll(XTAL_OSC,IMULT_20,FMULT_0,PLLCLK_BY_2),
 * con el cristal externo del kit de desarrollo
 * (f_xtal = 10 MHz).
 *
 *
 * IMPORTANTE: Esta librería emplea los ePWM1 y
 * ePWM2 para iniciar la conversión del ADC-A y
 * ADC-B, respectivamente, por lo que se recomienda
 * no emplear estos ePWM en otras aplicaciones, esto
 * con el fin de no modificar la frecuencia de muestreo.
 */
```

```
#define XTAL_FREQ 10000000  //frecuencia del cristal

/*   Definiciones para el ADC          */
#define ADCA 0
#define ADCB 1
//#define DEBUG



/*   Definiciones para el DAC          */
#define DACA 0
#define DACB 1
#define DACC 2
#define DAC_ENABLE 1
#define DAC_DISBLE 0
#define REFERENCE_VDAC              0
#define REFERENCE_VREF              1



volatile bool intrA = false;        //variable que indica que el
                                //ADC-A generó una interrupción
volatile bool intrB = false;        //variable que indica que el
                                //ADC-A generó una interrupción

volatile      struct ADC_RESULT_REGS*     ADC_RESULT_PTR[2] =
      {&AdcaResultRegs, &AdcbResultRegs};
volatile      struct ADC_REGS*      ADC_PTR[2] =
      {&AdcaRegs,&AdcbRegs};
volatile      struct DAC_REGS*      DAC_PTR[3] =
      {&DacaRegs,&DacbRegs,&DaccRegs};

void    ADCA_Process(void);
void    ADCB_Process(void);


interrupt     void    adca1_isr(void);
interrupt     void    adcb2_isr(void);

void    EPWM_Configure(Uint16 adc_num,  Uint32 Freq);

void    ADC_Configure(Uint16 adc_num,  Uint32 Freq);
void    ADC_Init(Uint16 adc_num, Uint16 channel);
void    ADC_Int(Uint16 adc_num, Uint16 channel);
void    ADC_Start(Uint16 adc_num);
Uint16 ADC_Read(Uint16 adc_num, Uint16 channel);

void    DAC_Configure(Uint16 dac_num);
```

```
void    DAC_Send(Uint16 dac_num, int dacval);


/*************************************************************
* Configura el ADC                                           *
*                                                            *
* adc_num: número de ADC a configurar (ADCA O ADCB)          *
* Freq: frecuencia de muestreo (Hz)                          *
*                                                            *
* IMPORTANTE: la frecuencia máxima de conversión está acotada *
* entre 382 Hz y 3.40 MHz (290 ns). Estos valores se pueden  *
* modificar si se emplea otra configuración de reloj.        *
*************************************************************/
void    ADC_Configure(Uint16 adc_num,  Uint32 Freq){

        if(Freq<382 || Freq>3400000){
        //No se puede muestrear tan rápido o tan lento
                __asm(" ESTOP0");
                return;                 //Error en las frecuencias
        }

        EALLOW;
        //Preescalador  /4
        ADC_PTR[adc_num]->ADCCTL2.bit.PRESCALE = 6;
        AdcSetMode(adc_num, ADC_RESOLUTION_12BIT, ADC_SIGNALMODE_SINGLE);
        //La interrupción se genera al terminar la conversión
        ADC_PTR[adc_num]->ADCCTL1.bit.INTPULSEPOS = 1;
        //Enciende el ADC
        ADC_PTR[adc_num]->ADCCTL1.bit.ADCPWDNZ = 1;
        //Tiempo de espera para que se encienda el ADC
        DELAY_US(1000);

        EDIS;

        EPWM_Configure(adc_num, Freq);
}       //end ADC_Configure



/*************************************************************
* Configura el el canal y el SOC  del ADC a emplear          *
*                                                            *
* adc_num: número de ADC a configurar (ADCA O ADCB)          *
* channel: canal a emplear (0,1,2,3,4,5,14 o 15)             *
* los canales 14 y 15 son compartidos en los ADC             *
*                                                            *
* IMPORTANTE: los SOC se configuran en orden respecto a los  *
* canales, de tal manera que en el caso de emplear varios    *
* canales por cada ADC, estos se adquieran de forma          *
```

```
* consecutiva.                                                    *
*****************************************************************/
void    ADC_Init(Uint16 adc_num, Uint16 channel){

        Uint16 acqps = 14;
        Uint16 Trigsel;

        EALLOW;

        if(adc_num == ADCA){
                Trigsel = 0x05; //EPWM1
#ifdef DEBUG
        /* Configuración extra para debuggear Fs  */
        /* GPIO 2 como salida                     */
        GpioCtrlRegs.GPAMUX1.bit.GPIO2 = 0;
        GpioCtrlRegs.GPADIR.bit.GPIO2 =  1;
#endif
        }
        else{
                Trigsel = 0x08;              //EPWM2
#ifdef DEBUG
        /* Configuración extra para debuggear Fs */
        /*    GPIO 3 como salida                 */
        GpioCtrlRegs.GPAMUX1.bit.GPIO3 = 0;
        GpioCtrlRegs.GPADIR.bit.GPIO3 =  1;
#endif
        }

        switch(channel){
        case 1:
                //Configura el canal 1
                ADC_PTR[adc_num]->ADCSOC1CTL.bit.CHSEL  = 1;
                //Tiempo de espera para el
                //S+H acqps + 1 SYSCLK ciclos
                ADC_PTR[adc_num]->ADCSOC1CTL.bit.ACQPS = acqps;
                //Conversion por EPWM
                ADC_PTR[adc_num]->ADCSOC1CTL.bit.TRIGSEL=Trigsel;
                break;
        case 2:
                //Configura el canal 2
                ADC_PTR[adc_num]->ADCSOC2CTL.bit.CHSEL  = 2;
                //Tiempo de espera para el
                //S+H acqps + 1 SYSCLK ciclos
                ADC_PTR[adc_num]->ADCSOC2CTL.bit.ACQPS = acqps;
                //Conversión por EPWM
                ADC_PTR[adc_num]->ADCSOC2CTL.bit.TRIGSEL=Trigsel;
                break;
        case 3:
                //Configura el canal 3
```

```
        ADC_PTR[adc_num]->ADCSOC3CTL.bit.CHSEL  = 3;
        //Tiempo de espera para el
        //S+H acqps + 1 SYSCLK ciclos
        ADC_PTR[adc_num]->ADCSOC3CTL.bit.ACQPS = acqps;
        //Conversión por EPWM
        ADC_PTR[adc_num]->ADCSOC3CTL.bit.TRIGSEL=Trigsel;
        break;
case 4:
        //Configura el canal 4
        ADC_PTR[adc_num]->ADCSOC4CTL.bit.CHSEL  = 4;
        //Tiempo de espera para el
        //S+H acqps + 1 SYSCLK ciclos
        ADC_PTR[adc_num]->ADCSOC4CTL.bit.ACQPS = acqps;
        //Conversión por EPWM
        ADC_PTR[adc_num]->ADCSOC4CTL.bit.TRIGSEL=Trigsel;
        break;
case 5:
        //Configura el canal 5
        ADC_PTR[adc_num]->ADCSOC5CTL.bit.CHSEL  = 5;
        //Tiempo de espera para el
        //S+H acqps + 1 SYSCLK ciclos
        ADC_PTR[adc_num]->ADCSOC5CTL.bit.ACQPS = acqps;
        //Conversión por EPWM
        ADC_PTR[adc_num]->ADCSOC5CTL.bit.TRIGSEL=Trigsel;
        break;
case 14:
        //Configura el canal 14
        ADC_PTR[adc_num]->ADCSOC14CTL.bit.CHSEL  = 14;
        //Tiempo de espera para el
        //S+H acqps + 1 SYSCLK ciclos
        ADC_PTR[adc_num]->ADCSOC14CTL.bit.ACQPS = acqps;
        //Conversión por EPWM
        ADC_PTR[adc_num]->ADCSOC14CTL.bit.TRIGSEL=Trigsel;
        break;
case 15:
        //Configura el canal 15
        ADC_PTR[adc_num]->ADCSOC15CTL.bit.CHSEL  = 15;
        //Tiempo de espera para el
        //S+H acqps + 1 SYSCLK ciclos
        ADC_PTR[adc_num]->ADCSOC15CTL.bit.ACQPS = acqps;
        //Conversión por EPWM
        ADC_PTR[adc_num]->ADCSOC15CTL.bit.TRIGSEL=Trigsel;
        break;
default:
        //Configura el canal 0
        ADC_PTR[adc_num]->ADCSOC0CTL.bit.CHSEL  = 0;
        //Tiempo de espera para el
        //S+H acqps + 1 SYSCLK ciclos
        ADC_PTR[adc_num]->ADCSOC0CTL.bit.ACQPS = acqps;
```

```
            //Conversión por EPWM
            ADC_PTR[adc_num]->ADCSOC0CTL.bit.TRIGSEL=Trigsel;
            break;

        }

    CpuSysRegs.PCLKCR0.bit.TBCLKSYNC = 1;
    EDIS;
}       //end ADC_Init



/*************************************************************
* Configura el canal que genera la interrupción del ADC      *
*                                                            *
* adc_num: número de ADC a configurar (ADCA O ADCB)          *
* channel: canal y soc a emplear (0-15)                      *
*                                                            *
* IMPORTANTE: para evitar problemas con las interrupciones   *
* se configuro el ADC-A con la interrupción 1, con el PIE    *
* PIEIER1_1.1, mientras que el ADC-B fue con la interrupción *
* 2, con el PIE PIEIER1_10.6                                 *
*************************************************************/
void ADC_Int(Uint16 adc_num, Uint16 channel){


        if( (channel>5 && channel <14) || channel>15){
            __asm(" ESTOP0");
            //El canal elegido no existe en el ADC
            return;
        }

        EALLOW;
        if(adc_num == ADCA){
            //nombre de la función de interrupción
            PieVectTable.ADCA1_INT = &adca1_isr;
            //Habilita la interrupción del PIE INT1.1
            PieCtrlRegs.PIEIER1.bit.INTx1 = 1;
            //canal que genera la interrupción INT1
            ADC_PTR[adc_num]->ADCINTSEL1N2.bit.INT1SEL = channel;
            //habilita INT1
            ADC_PTR[adc_num]->ADCINTSEL1N2.bit.INT1E = 1;
            //Habilita el grupo 1 de interrupciones
            IER |= M_INT1;

        }else{
            //nombre de la función de interrupción
            PieVectTable.ADCB2_INT = &adcb2_isr;
            //Habilita la interrupción del PIE INT10.6
```

```
            PieCtrlRegs.PIEIER10.bit.INTx6 = 1;
            //canal que genera la interrupción INT1
            ADC_PTR[adc_num]->ADCINTSEL1N2.bit.INT2SEL = channel;
            //habilita INT2
            ADC_PTR[adc_num]->ADCINTSEL1N2.bit.INT2E = 1;
            //Habilita el grupo 10 de interrupciones
            IER |= M_INT10;
        }


        ADC_PTR[adc_num]->ADCINTFLGCLR.all = 0x000F;

        EINT;           //Habilita interrupciones globales
        //Inicia el conteo de los EPWM
        CpuSysRegs.PCLKCR0.bit.TBCLKSYNC = 1;


    EDIS;
}       //end ADC_Int


/*************************************************************
* Configura el ePWM como trigger del ADC                     *
*                                                            *
* adc_num: número de ePWM a configurar                       *
*                                                            *
* epwm1->SCOA->ADCA                                          *
* epwm2->SOCB->ADCB                                          *
*************************************************************/
void EPWM_Configure(Uint16 adc_num, Uint32 Freq){

        Uint32 IMult, FMult, DivSel;
        Uint32 T;
        float f;

        //Obtenemos la configuración actual del reloj
        IMult = ClkCfgRegs.SYSPLLMULT.bit.IMULT;
        FMult = ClkCfgRegs.SYSPLLMULT.bit.FMULT;
        DivSel = ClkCfgRegs.SYSCLKDIVSEL.bit.PLLSYSCLKDIV;

        //calcula la frecuencia del reloj
        f = XTAL_FREQ*(float)(IMult+FMult)/(DivSel<<1);
        f = f/Freq;     //calcula el periodo del contador
        f = f/4;        //por el divisor /4 del ADC
        T = (int)f+1; //periodo del EPWM

        EALLOW;

        if(adc_num == ADCA){
```

```
            //Deshabilita el SOC-A
            EPwm1Regs.ETSEL.bit.SOCAEN = 0;
            //SOC cuenta up
            EPwm1Regs.ETSEL.bit.SOCASEL = 4;
            //Genera un pulso al primer evento
            EPwm1Regs.ETPS.bit.SOCAPRD = 1;
            //Frecuencia de muestreo
            EPwm1Regs.TBPRD = T; //0x0C36;
            //Detiene el contador
            EPwm1Regs.TBCTL.bit.CTRMODE = 3;
        }else{
            //Deshabilita el SOC-B
            EPwm2Regs.ETSEL.bit.SOCBEN = 0;
            //SOC cuenta up
            EPwm2Regs.ETSEL.bit.SOCBSEL = 4;
            //Genera un pulso al primer evento
            EPwm2Regs.ETPS.bit.SOCBPRD = 1;
            //Frecuencia de muestreo
            EPwm2Regs.TBPRD = T;//0x0209;
            //Detiene el contador
            EPwm2Regs.TBCTL.bit.CTRMODE = 3;
        }

        EDIS;

        return;
}       //end EPWM_Configure


/*************************************************************
* Inicia la operanción del ADC                               *
*                                                            *
* adc_num: número de ADC a iniciar (ADCA O ADCB)             *
*************************************************************/
void ADC_Start(Uint16 adc_num){
        if(adc_num == ADCA){
            //Habilita el SOC-A
            EPwm1Regs.ETSEL.bit.SOCAEN = 1;
            //Inicia el conteo del EPWM
            EPwm1Regs.TBCTL.bit.CTRMODE = 0;
        }else{
            //Habilita el SOC-B
            EPwm2Regs.ETSEL.bit.SOCBEN = 1;
            //Inicia el conteo del EPWM
            EPwm2Regs.TBCTL.bit.CTRMODE = 0;
        }

}       //end ADC_Run
```

```
/*************************************************************
* Lee el último valor del ADC                                *
*                                                            *
* adc_num: número de ADC a leer (ADCA O ADCB)                *
* channel: número de canal a leer                            *
*************************************************************/
Uint16 ADC_Read(Uint16 adc_num, Uint16 channel){

        Uint16 val = 0;
        //espera hasta que se presente la interrupción
        while(!intrA);
        //baja la bandera de la interrupción
        intrA = false;

        switch(channel){
        case 1:
                val = ADC_RESULT_PTR[adc_num]->ADCRESULT1;
                break;
        case 2:
                val = ADC_RESULT_PTR[adc_num]->ADCRESULT2;
                break;
        case 3:
                val = ADC_RESULT_PTR[adc_num]->ADCRESULT3;
                break;
        case 4:
                val = ADC_RESULT_PTR[adc_num]->ADCRESULT4;
                break;
        case 5:
                val = ADC_RESULT_PTR[adc_num]->ADCRESULT5;
                break;
        case 6:
                val = ADC_RESULT_PTR[adc_num]->ADCRESULT6;
                break;
        case 7:
                val = ADC_RESULT_PTR[adc_num]->ADCRESULT7;
                break;
        case 8:
                val = ADC_RESULT_PTR[adc_num]->ADCRESULT8;
                break;
        case 9:
                val = ADC_RESULT_PTR[adc_num]->ADCRESULT9;
                break;
        case 10:
                val = ADC_RESULT_PTR[adc_num]->ADCRESULT10;
                break;
        case 11:
                val = ADC_RESULT_PTR[adc_num]->ADCRESULT11;
                break;
        case 12:
```

```
            val = ADC_RESULT_PTR[adc_num]->ADCRESULT12;
            break;
        case 13:
            val = ADC_RESULT_PTR[adc_num]->ADCRESULT13;
            break;
        case 14:
            val = ADC_RESULT_PTR[adc_num]->ADCRESULT14;
            break;
        case 15:
            val = ADC_RESULT_PTR[adc_num]->ADCRESULT15;
            break;
        default:
            val = ADC_RESULT_PTR[adc_num]->ADCRESULT0;
            break;
        }

        return val;
}       //end ADC_Read




/************************************************************
* Configura el DAC-C                                        *
************************************************************/
void DAC_Configure(Uint16 dac_num){
        EALLOW;
        //Voltaje de referencia
        DAC_PTR[dac_num]->DACCTL.bit.DACREFSEL = REFERENCE_VREF;
        //Habilita el DAC
        DAC_PTR[dac_num]->DACOUTEN.bit.DACOUTEN = DAC_ENABLE;
        //Pone el 0V la salida
        DAC_PTR[dac_num]->DACVALS.all = 0;
        //Retraso para que encienda el DAC
        DELAY_US(10);
        EDIS;
}

/************************************************************
* Envia información al DAC-C                                *
************************************************************/
void DAC_Send(Uint16 dac_num, int dacval){
        // Envia el valor al DAC
        DAC_PTR[dac_num]->DACVALS.all = dacval;
}
```

```
/*************************************************************
* Rutina de interrupción del ADC-A                           *
*************************************************************/
interrupt void adca1_isr(void){
        //bandera para indicar que ya se presentó la interrupción
        intrA = true;
        //saltamos al proceso
        ADCA_Process();
        //borra la bandera INT1
        AdcaRegs.ADCINTFLGCLR.bit.ADCINT1 = 1;
        //PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;
        PieCtrlRegs.PIEACK.bit.ACK1 = 1;

#ifdef DEBUG
        /* Configuración extra para debuggear Fs  */
        /*     GPIO 2 como salida                 */
        GpioDataRegs.GPATOGGLE.bit.GPIO2 = 1;
#endif

}//end adca1_isr

/*************************************************************
* Rutina de interrupción del ADC-B                           *
*************************************************************/
interrupt void adcb2_isr(void){
        //bandera para indicar que ya se presentó la interrupción
        intrB = true;

        //saltamos al proceso
        ADCB_Process();

        //borra la bandera INT2
        AdcbRegs.ADCINTFLGCLR.bit.ADCINT2 = 1;
        //PieCtrlRegs.PIEACK.all = PIEACK_GROUP10;
        PieCtrlRegs.PIEACK.bit.ACK10 = 1;

#ifdef DEBUG
        /* Configuración extra para debuggear Fs   */
        /*     GPIO 3 como salida                  */
        GpioDataRegs.GPATOGGLE.bit.GPIO3 = 1;
#endif

}       //end adcB1_isr
```