```
/*-----------------------------------------------------------*/
// dac_ad5620_H.c
/*-----------------------------------------------------------*/
//Description :Utiilisation DAC AD5620
//
//Auteur : LDD
//Version:V1.0
//Compilateur:XC32 V2.50
//
/*-----------------------------------------------------------*/



#include "dac_ad5620.h"
#include "peripheral\SPI\plib_spi.h"
#include "app.h"

// prototypes des fonctions
void Dac_Init(void)
{
    SYNCHRO_DATA_DACOn();
    /* Disable the SPI module to configure it*/
    PLIB_SPI_Disable ( SPI_ID_1 );

    /* Set up Master or Slave Mode*/
    PLIB_SPI_MasterEnable ( SPI_ID_1 );
    PLIB_SPI_PinDisable(SPI_ID_1, SPI_PIN_SLAVE_SELECT);

    /* Set up if the SPI is allowed to run while the rest
     * of the CPU is in idle mode*/
    PLIB_SPI_StopInIdleEnable( SPI_ID_1 );

    /* Set up clock Polarity and output data phase*/
    PLIB_SPI_ClockPolaritySelect( SPI_ID_1, SPI_CLOCK_POLARITY_IDLE_LOW );
    PLIB_SPI_OutputDataPhaseSelect
            ( SPI_ID_1, SPI_OUTPUT_DATA_PHASE_ON_ACTIVE_TO_IDLE_CLOCK );

    /* Set up the Input Sample Phase*/
    PLIB_SPI_InputSamplePhaseSelect
            ( SPI_ID_1, SPI_INPUT_SAMPLING_PHASE_IN_MIDDLE);

    /* Communication Width Selection */
    PLIB_SPI_CommunicationWidthSelect
            ( SPI_ID_1, SPI_COMMUNICATION_WIDTH_8BITS );

    /* Baud rate selection */ // modification de la fréquence
    PLIB_SPI_BaudRateSet( SPI_ID_1 , SYS_CLK_PeripheralFrequencyGet
            (CLK_BUS_PERIPHERAL_1), 300000);

    /* Protocol selection */
    PLIB_SPI_FramedCommunicationDisable( SPI_ID_1  );
    #if defined (PLIB_SPI_ExistsAudioProtocolControl)
```

```
            if (PLIB_SPI_ExistsAudioProtocolControl(SPI_ID_1))
            {
                PLIB_SPI_AudioProtocolDisable(SPI_ID_1);
            }
    #endif

    /* Buffer type selection */
    #if defined (PLIB_SPI_ExistsFIFOControl)
        if (PLIB_SPI_ExistsFIFOControl( SPI_ID_1 ))
        {
            PLIB_SPI_FIFOEnable( SPI_ID_1 );
            PLIB_SPI_FIFOInterruptModeSelect
                (SPI_ID_1,
                 SPI_FIFO_INTERRUPT_WHEN_TRANSMIT_BUFFER_IS_COMPLETELY_EMPTY);
            PLIB_SPI_FIFOInterruptModeSelect
                (SPI_ID_1, SPI_FIFO_INTERRUPT_WHEN_RECEIVE_BUFFER_IS_NOT_EMPTY);
        }
    #else
        {
            SYS_ASSERT(false, "\r\nInvalid SPI Configuration.");
            return SYS_MODULE_OBJ_INVALID;
        }
    #endif

    PLIB_SPI_BufferClear( SPI_ID_1 );
    PLIB_SPI_ReceiverOverflowClear ( SPI_ID_1 );

    SYS_INT_SourceDisable(INT_SOURCE_SPI_2_TRANSMIT);
    SYS_INT_SourceDisable(INT_SOURCE_SPI_2_RECEIVE);
    SYS_INT_SourceDisable(INT_SOURCE_SPI_2_ERROR);

    /* Clear all interrupt sources */
    SYS_INT_SourceStatusClear(INT_SOURCE_SPI_1_TRANSMIT);
    SYS_INT_SourceStatusClear(INT_SOURCE_SPI_1_RECEIVE);
    SYS_INT_SourceStatusClear(INT_SOURCE_SPI_1_ERROR);

    /* Enable the Module */
    PLIB_SPI_Enable(SPI_ID_1);
}



void Dac_Write( uint16_t ech)
{
    //préparation valeur 16 bits à envoyer

    ech = ech <<2 ;
    ech &= 0x3FFC;

    SYNCHRO_DATA_DACOff();
    spi_write(ech >> 8);
    spi_write(ech);
    SYNCHRO_DATA_DACOn();
```

```
}
void spi_write( uint8_t Val){
   int SpiBusy;

   PLIB_SPI_BufferWrite(SPI_ID_1, Val);
   do {
     SpiBusy =  PLIB_SPI_IsBusy(SPI_ID_1) ;
   } while (SpiBusy == 1);
}
```