```
/*****************************************************************************
 Module
     PIC32_AD_Lib.c

 Description
     This is a module implementing  the basic functions to use the A/D
       converter on the PIC32MX170F256B
 Notes


 History
 When           Who     What/Why
 -------------- ---     --------

 10/27/20 16:10 jec     cleaned up the documentation to meet SPDL Standards
 10/20/20 16:38 jec     Began Coding
*****************************************************************************/
/*----------------------------- Include Files -----------------------------*/
#include "PIC32_AD_Lib.h"
#include "bitdefs.h"
#include <xc.h>
#include <p32xxxx.h>
/*----------------------------- External Variables -------------------------*/

/*----------------------------- Module Defines -----------------------------*/

/*----------------------------- Module Types -------------------------------*/

/*----------------------------- Module Functions ---------------------------*/

/*----------------------------- Module Variables ---------------------------*/
uint8_t numChanInSet;   // used to check for bad read requests

/*----------------------------- Module Code --------------------------------*/
/*****************************************************************************
 Function
     ADC_ConfigAutoScan
 Parameters
      uint8_t whichPins spcifies which of the ANx pins will be converted
             a 1 in a bit position indicates that that ANx channel is to be
             converted e.g: to convert on AN0, set bit 0

      uint8_t numPins how many pins in the scan set
 Returns
     nothing
 Description
     configures the A/D converter subsystem for auto-sampling on a set of pins
 Notes
     None.
 Author
     J. Edward Carryer, 10/20/20 15:49
*****************************************************************************/
void ADC_ConfigAutoScan( uint16_t whichPins, uint8_t numPins){

    AD1CON1bits.ON = 0; // disable ADC

    // AD1CON1<2>, ASAM     : 1 = Sampling begins immediately after last conversion
completes
      // AD1CON1<4>, CLRASAM : 0 = buffer contents will be overwritten by the next
conversion sequence
```

```
    // AD1CON1<7:5>, SSRC  : 111 = Internal counter ends sampling and starts conversion
(auto convert)
       // AD1CON1<10:8>, FORM : 000 = unsigned integer data format
    // AD1CON1<13>, SIDL   : 0 = Continue module operation when the device enters Idle
mode
    // AD1CON1<15>, ON     : 0 = ADC remains off

    AD1CON1bits.ASAM = 1; // 1 = Sampling begins immediately after last conversion
completes
       AD1CON1bits.CLRASAM = 0;// 0 = buffer contents will be overwritten by the next
conversion sequence
       AD1CON1bits.SSRC = 0b111;// 111 = Internal counter ends sampling and starts
conversion (auto convert)
    // AD1CON1SET = 0x00e4;     // to set everything above in one fell swoop

    // AD1CON2<0>,  ALTS   : 0 = Always use Sample A input multiplexer settings
       // AD1CON2<1>, BUFM    : 1 = Buffer configured as two 8-word buffers,
ADC1BUF7-ADC1BUF0, ADC1BUFF-ADCBUF8
    // AD1CON2<10>, CSCNA  : 1 = Scan inputs
    // AD1CON2<12>, OFFCAL : 0 = Disable Offset Calibration mode
    // AD1CON2<15:13>,VCFG : 000 = Vrefh = AVDD, Vrefl = AVss

    AD1CON2bits.BUFM = 1; // 1 = Buffer configured as two 8-word buffers
    AD1CON2bits.CSCNA = 1;// 1 = Scan inputs
    // AD1CON2 = 0x0402;    // to set everything above in one fell swoop

    // AD2CON2<5:2>, SMPI  : Interrupt flag set at after numPins completed conversions
    AD1CON2SET = (numPins-1) << 2;

    // AD1CON3<7:0>, ADCS  : 1 = TPB * 2 * (ADCS<7:0> + 1) = 4 * TPB = TAD
    // AD1CON3<12:8>, SAMC : 0x0f = Acquisition time = AD1CON3<12:8> * TAD = 15 * TAD
    // AD1CON3<15>, ADRC   : 0 = Clock derived from Peripheral Bus Clock (PBCLK)

    AD1CON3bits.ADCS = 1; // 1 = TPB * 2 * (ADCS<7:0> + 1) = 4 * TPB = TAD
    AD1CON3bits.SAMC = 0x0f;// 0x0f = Acquisition time = AD1CON3<12:8> * TAD = 15 * TAD
    // AD1CON3 = 0x0f01;    // to set everything above in one fell swoop

    // AD1CHS is ignored in scan mode, but we'll clear it to be sure
    AD1CHS = 0;

    // select which pins to use for scan mode, a 1 indicates that the corresponding ANx
    // input will be converted
    AD1CSSL = whichPins;

    numChanInSet = numPins; // log the number of pins in the set for reading

    AD1CON1bits.ON = 1; // enable ADC

}

/*****************************************************************************
 Function
     ADC_MultiRead
 Parameters
    uint32_t *adcResults pointer to array to hold conversion set results
        this must have room for at least as many results as numPins in
        ADC_ConfigAutoScan
 Returns
    nothing
 Description
```

```
    Reads the most recent conversion results from the channel set and copies
        the reults to the arrya passed as a pointer to this function
        lowest numbered converted channel is in adcResults[0]
 Notes
    None.
 Author
    J. Edward Carryer, 10/20/20 16:39
*****************************************************************************/
void ADC_MultiRead(uint32_t *adcResults){
uint8_t i;
uint32_t LocalResult;
volatile uint32_t *resultSet;

// stop automatic sampling during the read to be sure we get a coherent set
AD1CON1bits.ASAM = 0;

if( AD1CON2bits.BUFS == 1){
// 1 = ADC is currently filling buffer 0x8-0xF, user should access data in 0x0-0x7
    resultSet = &ADC1BUF0;
}else{
// 0 = ADC is currently filling buffer 0x0-0x7, user should access data in 0x8-0xF
    resultSet = &ADC1BUF8;
}
for (i=0; i < numChanInSet; i++)
{
    // read the results from the ADC1BUFx registers they are 16 bytes apart in
    // the memory map, hence *4
    LocalResult= adcResults[i] = *(resultSet+(4*i)); // read the results from the
ADC1BUFx registers

}
AD1CON1bits.ASAM = 1;  // restart automatic sampling
IFS0CLR = BIT28HI;     // clear ADC interrupt flag, see table 7-1, pg 68
}
```