Experiment 8: ADC / DAC Implementation in ARM based LPC2148 Development Board (through C - Interface)

Deepak Charan, EE23B022

November 2024

1 Objective

1. To understand C-interfacing (use C-programming) in an ARM platform
2. To study and implement ADC / DAC in ARM platform

2 Equipments/Software Required:

1. ARK's LPC2148 embedded ARM development board and accessories
2. RS-232 cable
3. Windows PC with Keil microvision 5, Flash magic, and Burn o-mat.
4. USB - serial converter
5. DSO (Digital Storage Oscilloscope)
6. Sample programs for generating digital inputs. (For analog inputs, potentiometer is used)

3 General Procedure:

1. Go through the handout to understand the workings of LPC2148 board.
2. Download KEIL Microvision 5 and Flash Magic on a Windows PC.

3. build a new project in KEIL Microvision 5 with the right configurations and create a new C file in the Source File section
4. Write the proper C code to accomplish all the desired objectives.
5. Burn the code into the board using Flash Magic (By using the .hex file that would have been automatically generated when we built the project)
6. Verify if the code works by checking for multiple test cases.
7. Repeat this procedure for all the tasks

4 ADC Tasks:

4.1 Convert analog signal from a sensor into a digital signal

4.1.1 Code:

```

#include <LPC214x.H>                              /* LPC214x
   definitions */

//#include "delay.h"
//#include "led.c"

unsigned long Read_ADC0(unsigned char);
void Init_ADC0(unsigned char);
void delay_mSec(int dCnt);

#ifndef __ADC_H
#define __ADC_H


#define CHANNEL_0   0
#define CHANNEL_1   1
#define CHANNEL_2   2
#define CHANNEL_3   3
#define CHANNEL_4   4
#define CHANNEL_5   5
#define CHANNEL_6   6
#define CHANNEL_7   7
```

```


/* Crystal frequency,10MHz~25MHz should be the same as
   actual status. */
#define Fosc        12000000    /* 12 MHz is the
   operational frequency of o/p dgtlClk */
#define ADC_CLK     1000000     /* set to 1Mhz */

/* A/D Converter 0 (AD0) */
#define AD0_BASE_ADDR       0xE0034000
#define ADC_INDEX           4

#define ADC_DONE            0x80000000
#define ADC_OVERRUN         0x40000000


#define ADC_FullScale_Volt  3.3 // 3.3V - ADC Referance
   Voltage
#define ADC_FullScale_Count 1024    // 2^10 - 10 bit ADC
#define LED_IOPIN       IO0PIN
#define BIT(x)  (1 << x)

#define LED_D0  (1 << 10)       // P0.10 mapping same as
    in Exp7 switch LED
#define LED_D1  (1 << 11)       // P0.11
#define LED_D2  (1 << 12)       // P0.12
#define LED_D3  (1 << 13)       // P0.13

#define LED_D4  (1 << 15)       // P0.15
#define LED_D5  (1 << 16)       // P0.16
#define LED_D6  (1 << 17)       // P0.17
#define LED_D7  (1 << 18)       // P0.18
#define LED_DATA_MASK           ((unsigned long)((LED_D7
    | LED_D6 | LED_D5 | LED_D4 | LED_D3 | LED_D2 |
   LED_D1 | LED_D0)))

#define LED1_ON     LED_IOPIN |= (unsigned long)(LED_D0)
   ;        // LED1 ON
#define LED2_ON     LED_IOPIN |= (unsigned long)(LED_D1)
   ;        // LED2 ON
#define LED3_ON     LED_IOPIN |= (unsigned long)(LED_D2)
   ;        // LED3 ON
#define LED4_ON     LED_IOPIN |= (unsigned long)(LED_D3)
```

```
    ;        // LED4 ON
#define LED5_ON     LED_IOPIN |= (unsigned long)(LED_D4);        // LED5 ON
#define LED6_ON     LED_IOPIN |= (unsigned long)(LED_D5);        // LED6 ON
#define LED7_ON     LED_IOPIN |= (unsigned long)(LED_D6);        // LED7 ON
#define LED8_ON     LED_IOPIN |= (unsigned long)(LED_D7);        // LED8 ON

#define LED1_OFF        LED_IOPIN &= (unsigned long)~(LED_D0);       // LED1 ON
#define LED2_OFF        LED_IOPIN &= (unsigned long)~(LED_D1);       // LED2 ON
#define LED3_OFF        LED_IOPIN &= (unsigned long)~(LED_D2);       // LED3 ON
#define LED4_OFF        LED_IOPIN &= (unsigned long)~(LED_D3);       // LED4 ON
#define LED5_OFF        LED_IOPIN &= (unsigned long)~(LED_D4);       // LED5 ON
#define LED6_OFF        LED_IOPIN &= (unsigned long)~(LED_D5);       // LED6 ON
#define LED7_OFF        LED_IOPIN &= (unsigned long)~(LED_D6);       // LED7 ON
#define LED8_OFF        LED_IOPIN &= (unsigned long)~(LED_D7);       // LED8 ON

#endif
    #ifndef LED_DRIVER_OUTPUT_EN
#define LED_DRIVER_OUTPUT_EN (1 << 5)   // P0.5
#endif
//LED definitions





int main (void)
{

    unsigned long ADC_val;

    Init_ADC0(CHANNEL_1);
```

```
    Init_ADC0(CHANNEL_2);

    delay_mSec(100);

    IO0DIR |= LED_DATA_MASK;              // GPIO Direction control -> pin is output
    IO0DIR |= LED_DRIVER_OUTPUT_EN;       // GPIO Direction control -> pin is output
    IO0CLR |= LED_DRIVER_OUTPUT_EN;


    while(1)
    {
        //ADC_val = Read_ADC0(CHANNEL_1);
        ADC_val = Read_ADC0(CHANNEL_2);
        ADC_val=(ADC_val>>2);
        delay_mSec(5);



    if(ADC_val & BIT(0)){ LED8_ON; }
    else {LED8_OFF;}
    if(ADC_val & BIT(1)) {LED7_ON;}
    else {LED7_OFF;}
    if(ADC_val & BIT(2)) {LED6_ON;}
    else {LED6_OFF;}
    if(ADC_val & BIT(3)) {LED5_ON;}
    else {LED5_OFF;}
    if(ADC_val & BIT(4)) {LED4_ON;}
    else {LED4_OFF;}
    if(ADC_val & BIT(5)) {LED3_ON;}
    else {LED3_OFF;}
    if(ADC_val & BIT(6)){ LED2_ON;}
    else {LED2_OFF;}
    if(ADC_val & BIT(7)) {LED1_ON;}
    else {LED1_OFF;}
    }



//     return 0;
}

```

```
void Init_ADC0(unsigned char channelNum)
{
    if(channelNum == CHANNEL_1)
        PINSEL1 = (PINSEL1 & ~(3 << 24)) | (1 << 24);       // P0.28 -> AD0.1

    if(channelNum == CHANNEL_2)
        PINSEL1 = (PINSEL1 & ~(3 << 26)) | (1 << 26);       // P0.29 -> AD0.2

    if(channelNum == CHANNEL_3)
        PINSEL1 = (PINSEL1 & ~(3 << 28)) | (1 << 28);       // P0.30 -> AD0.3


    AD0CR = ( 0x01 << 1 ) |                             // SEL=1, select channel 0, 1 to 4 on ADC0
            (( Fosc / ADC_CLK - 1 ) << 8 ) |     // CLKDIV = Fpclk / 1000000 - 1
            ( 0 << 16 ) |                               // BURST = 0, no BURST, software controlled
            ( 0 << 17 ) |                               // CLKS = 0, 11 clocks/10 bits
            ( 1 << 21 ) |                               // PDN = 1, normal operation
            ( 0 << 22 ) |                               // TEST1:0 = 00
            ( 0 << 24 ) |                               // START = 0 A/D conversion stops
            ( 0 << 27 );                                /* EDGE = 0 (CAP/MAT singal falling,trigger A/D conversion) */
}
unsigned long Read_ADC0( unsigned char channelNum )
{
    unsigned long regVal, ADC_Data;

    /* Clear all SEL bits */
    AD0CR &= 0xFFFFFF00;
    /* switch channel, start A/D convert */
    AD0CR |= (1 << 24) | (1 << channelNum);


```

```
    /* wait until end of A/D convert */
    while ( 1 ) {

//        regVal = *(volatile unsigned long *)(
   AD0_BASE_ADDR + ADC_INDEX);
        regVal = AD0GDR;

        if ( regVal & ADC_DONE ){
            break;
        }
    }

    /* stop ADC now */
    AD0CR &= 0xF8FFFFFF;
    /* save data when it's not overru otherwise, return
       zero */
    if ( regVal & ADC_OVERRUN ) {
        return ( 0 );
    }
    ADC_Data = ( regVal >> 6 ) & 0x3FF;
    /* return A/D conversion value */
    return ( ADC_Data );

}

void delay_mSec(int dCnt)       // pr_note:~dCnt mSec
{
  int j=0,i=0;

  while(dCnt--)
  {
      for(j=0;j<1000;j++)
      {
        /* At 60Mhz, the below loop introduces
        delay of 10 us */
        for(i=0;i<10;i++);
      }
  }
}
```

4.1.2 Observations:

- Decreasing step size increases number of bits, and also quantization error increases
- Since q=10 for our ADC, Quantization error is 60.2.

Video for ADC is provided here

5 DAC Tasks:

5.1 Saw Tooth Wave

5.1.1 Implementation:

- First set up the DAC outputs
- For constructing a Saw Tooth Wave, we write a for loop in an infinite while loop to send values in an increasing order (and ORing it with the DAC_BIAS) until a certain threshold (and for loop would start from beginning).
- the while loop ensures we continuously observe a Saw tooth Wave

5.1.2 Code:

```
//DAC program for LPC2148 SUNTECH KIT


#include "LPC214x.H"                       /* LPC214x
   definitions */

#define DAC_BIAS            0x00010000
void mydelay(int);

void DACInit( void )
{
    /* setup the related pin to DAC output */
    PINSEL1 &= 0xFFF3FFFF;
    PINSEL1 |= 0x00080000;  /* set p0.25 to DAC output
       */
    return;
}
```

```
int main (void)
{
     DACInit();
     int cnt=0;
//SQUARE WAVE
while(1)
{
     cnt=0;
    while(cnt<=0x3ff)
    {
     DACR=(cnt<<6)|DAC_BIAS;
          cnt=cnt+1;
     if(cnt==0x03FF)
          {cnt=0;
               mydelay(0xFF);
          }
}*/
     return 0;

}
void mydelay(int x)
{
int j,k;
for(j=0;j<=x;j++)
{
     for(k=0;k<=0xFF;k++);
}
}
```

5.2 Triangle Wave

5.2.1 Implementation:

- Similar to the Saw tooth Wave, we write a reverse for loop (which starts from 0x3FF and goes till 0) after the first for loop to obtain a triangle wave

5.2.2 Code:

```
//DAC program for LPC2148 SUNTECH KIT


```

```c
#include "LPC214x.H"                        /* LPC214x definitions */

#define DAC_BIAS            0x00010000
void mydelay(int);

void DACInit( void )
{
    /* setup the related pin to DAC output */
    PINSEL1 &= 0xFFF3FFFF;
    PINSEL1 |= 0x00080000;  /* set p0.25 to DAC output */
    return;
}

int main (void)
{
    DACInit();


while(1){
    for(int cnt=0;cnt<1000;cnt++){
        DACR=((cnt)<<6)|DAC_BIAS;
    }


    for(int dnt=1000;dnt>=0;dnt--){
        DACR=((dnt)<<6)|DAC_BIAS;
    }

}

    return 0;

}
void mydelay(int x)
{
int j,k;
for(j=0;j<=x;j++)
{
    for(k=0;k<=0xFF;k++);
}
}
```

5.3 Sine Wave

5.3.1 Implementation:

- For a sine wave, we use *'math.h'* library to get the sine of the value
- We use a for loop to pass the sine of a number (which increases till a certain limit) and send it to the DACR (we accordingly scale and round off the sine value)

5.3.2 Code:

```
#include "LPC214x.h"
#include <math.h>
#define DAC_BIAS 0x00010000

// configure the DAC
void DACInit(void)
{
PINSEL1 &= 0xFFF3FFFF;
 PINSEL1 |= 0x00080000;
}
// main routine


void delay_ms( int j)
{
     int x,i;
for(i=0;i<j;i++)
{
    for(x=0; x<6000; x++);    /* loop to generate 1
       milisecond delay with Cclk = 60MHz */
}
}

int main (void)
{

   int i=0,value=0;
DACInit();
while(1)
{

  while(i!=32)
```

```
  {
   value = (int)((sin((2*3.141/32)*i)+1)*500);


DACR = ( (1<<16) | (value<<6) );
delay_ms(1);
i++;
}

   i=0;

}
}
```

5.4 Photos

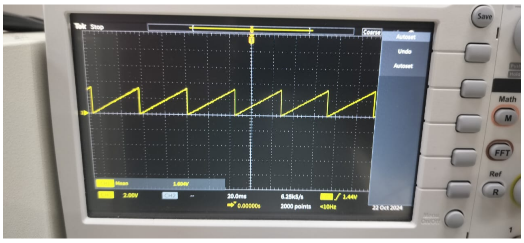

Figure 1: Saw Tooth Wave

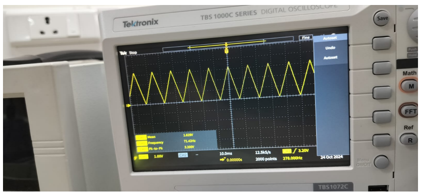

Figure 2: Triangle Wave

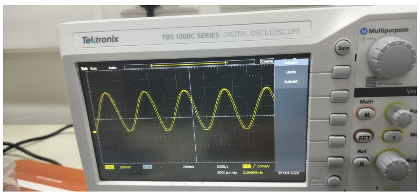

Figure 3: Sine Wave

Note: I constructed the Sine Wave

6 References:

1. Handouts and the Code provided in Moodle