MICROPROCESSOR LAB (EE2016)

EXPERIMENT 6

Serial Communication & ADC / DAC Implementation in ViARM 2378 Development Board (through C - Interface)

Submitted by:

Ajeet ES, EE22B086

Amogh Agrawal, EE22B087

CONTENTS

ADC (Analog to Digital Converter)

AIM

1. To Study and Implement ADC in an ARM Platform (ViARM 2378) using C, a high-level programming language.
2. From the code and its working, deduce the dependence of number of bits on the step size, and hence, the change in amount of data that can be captured by the ADC
3. Calculate the value of SQNR(Signal-to-Quantisation-Noise Ratio) for this Analog to Digital Converter using the number of bits in the Digital Output.

CODE

```
#include "LPC23xx.h"

void  TargetResetInit(void)
{
  // 72 Mhz Frequency
  if ((PLLSTAT & 0x02000000) > 0)
  {
      /* If the PLL is already running   */
      PLLCON  &= ~0x02;          /* Disconnect the PLL
*/
      PLLFEED  =  0xAA;          /* PLL register update sequence, 0xAA, 0x55
*/
      PLLFEED  =  0x55;
  }
  PLLCON   &= ~0x01;             /* Disable the PLL
*/
  PLLFEED   =  0xAA;             /* PLL register update sequence, 0xAA, 0x55
*/
  PLLFEED   =  0x55;
  SCS      &= ~0x10;             /* OSCRANGE = 0, Main OSC is between 1 and 20
Mhz                  */
  SCS      |=  0x20;             /* OSCEN = 1, Enable the main oscillator
*/
  while ((SCS &  0x40) == 0);
  CLKSRCSEL = 0x01;                      /* Select main OSC, 12MHz, as the PLL
clock source  */
  PLLCFG    = (24 << 0) | (1 << 16);     /* Configure the PLL multiplier and
divider                  */
```

```
  PLLFEED    = 0xAA;                         /* PLL register update sequence, 0xAA,
0x55                 */
  PLLFEED    = 0x55;
  PLLCON    |= 0x01;                        /* Enable the PLL
*/
  PLLFEED    = 0xAA;                        /* PLL register update sequence, 0xAA,
0x55                 */
  PLLFEED    = 0x55;
  CCLKCFG    = 3;                           /* Configure the ARM Core Processor
clock divider           */
  USBCLKCFG = 5;                             /* Configure the USB clock divider
*/
  while ((PLLSTAT & 0x04000000) == 0);
  PCLKSEL0  = 0xAAAAAAAA;                /* Set peripheral clocks to be half of main
clock              */
  PCLKSEL1  = 0x22AAA8AA;
  PLLCON    |= 0x02;                         /* Connect the PLL. The PLL is now the
active clock source  */
  PLLFEED    = 0xAA;                     /* PLL register update sequence, 0xAA, 0x55
*/
  PLLFEED    = 0x55;
  while ((PLLSTAT & 0x02000000) == 0);
  PCLKSEL0 = 0x55555555;          /* PCLK is the same as CCLK */
  PCLKSEL1 = 0x55555555;
}


/******************** *serial Transmission routine* **********************/
void serial_tx(int ch)
{ // pp421 LSR is RO 8-bit register in which b5 Transmit Holding Register Empty
(THRE) is set to 1 if THR is empty
  while ((U0LSR & 0x20)!=0x20); /* UARTn Line status register. If (THRE!=1)
wait for tx to be over */
  U0THR = ch;   /* Transmit holding register. Once THR is empty, then send the
next ch   */
}
/**************************** Routine for converting hex value to ascii value
*****************/
int htoa(int ch)
{       /* refer ASCII table */
    if(ch<=0x09)
        ch = ch + 0x30; /* For numerals hex 0 to 9 ASCII adds 0x30  */
    else
        ch = ch + 0x37; /* For hex numerals A to F, characters only? */
    return(ch);
}
/*********************************** main routine
*****************************************************/
int  main ()
{
    unsigned int Fdiv,value,i,j;
```

```
//  char value;
    TargetResetInit();
//  init_timer( ((72000000/100) - 1) );
    /* power control for peripherals (PCONP) 32 bit WO register. pp 68 */
    PCONP |=0X00001000; // b12 of PCONP PCAD=1 to ENABLE A/D converter module
    PINSEL0 = 0x00000050; // Pin selection for uart tx (5:4) & rx (7:6) lines.
Table 106 pp157.
                          // for pin names, refer pp11 - block diagram
    PINSEL1 = 0X01554000; // Pin selection for AD0.0 (15:14) [nibble 0100],
AD0.1 (17:16), AD0.2
                  // (19:18), [nibble 0101], AD0.3 (21:20), SDA0 (23:22)
[nibble 0101],
                  // SCL0 (25:24) [nibble 0001] concatenated to 0x01554000 - 8
bytes
                          // 4 channels in AD0, SDA0 (I2C serial data port),
SCL0 I2C Serial Clk
                          // Why I2C ports are invoked?


    /************** Uart initialization **********************************/
                      // Line control register. To access Divisor latches (DLM
& DLL), DLAB = 1
    U0LCR = 0x83; // pp419 WO 8 bit rgstr: 7 (DLAB)--> 1, 6-->0, 5:4-->00 (odd
parity), b3--> 0 (no parity),
                  // b2-->0 (1 Stop bit), b1:b0 --> 11 (8 bit character lngth),
    Fdiv = ( 72000000 / 16 ) / 19200; // Given baud rate being 19200, dividers
DL_est computation
    //Fdiv = ( 72000000 / 16 ) / 2400 ; // If DL_est is an integer (see
flowchart pp427), then
                                        // quotient is U0DLM & remainder is
U0DLL. For fraction?????
    U0DLM = Fdiv / 256; // pp414 Division Latch MSB register - quotient
    U0DLL = Fdiv % 256; // pp428 Exmpls Division Latch LSB register - remainder
    U0LCR = 0x03; // pp419 DLAB = 0 to disable access to divisor latches -

    AD0CR = 0X01210F01; // pp 598. ADC settngs: AD0.1 other AD0.* disabled,
4clks/3 bits?, start now
    while(1)
    {
        while((AD0DR0 & 0X80000000)!=0X80000000){}; // Wait here until adc make
conversion complete

        /************* To get converted value and display it on the serial
port***************/
                value = (AD0DR0>>6)& 0x3ff ;    //ADC value
            //serial_tx(value);
        serial_tx('\t');
        serial_tx(htoa((value&0x300)>>8));
        serial_tx(htoa((value&0xf0)>>4));
        serial_tx(htoa(value&0x0f));
        serial_tx(0x0d);
        serial_tx(0x0a);

            for(i=0;i<=0xFF;i++) /* delay?? */
            {
                    for(j=0;j<=0xFF;j++);
            }
    }
  return 0;
}
```

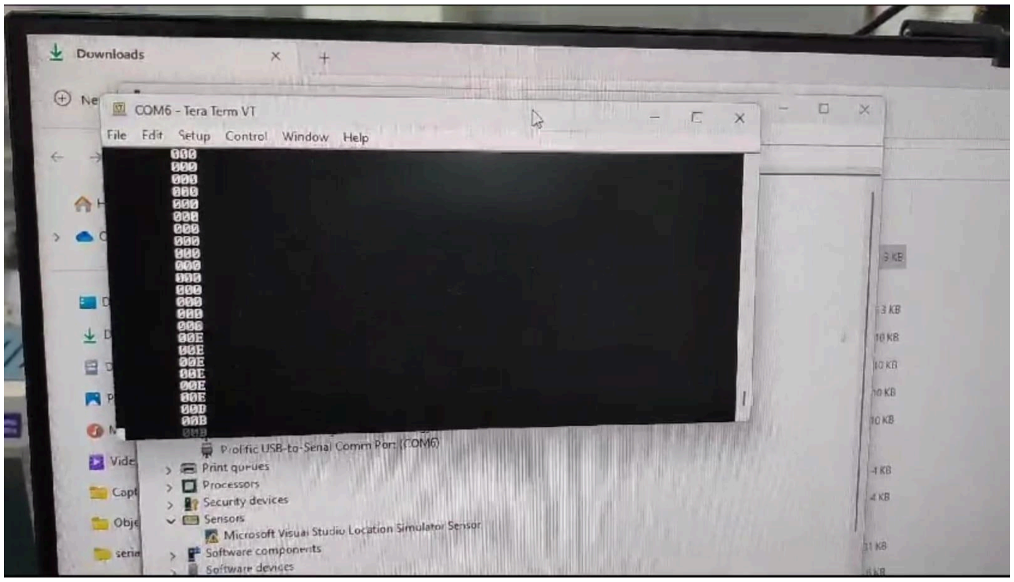
RESULTS

The code gave the following results:

1. The ADC code runs in an infinite loop, taking value from the Analog Input Trimmer and sending it to the connected device (computer screen) using UART, in ASCII Format.
2. As the Analog Input Trimmer is rotated, the value transmitted, and hence, the value printed on the computer screen changes from 0x000 to 0x3FF, making the number of bits in the maximum value of the output 10.

IMAGES AND VIDEOS

Images taken during the experiment and a video showing the demonstration are given below:


Downloads
COM6 - Tera Term VT
File Edit Setup Control Window Help
Prolific USB-to-Serial Comm Port (COM6)
Print queues
Processors
Security devices
Sensors
Microsoft Visual Studio Location Simulator Sensor
Software components
Software devices

```
000
000
000
000
000
000
000
000
000
000
000
000
000
000
000
00E
00E
00E
00E
00E
00E
00D
00B
00B
```

Link to Video Demonstration: https://youtu.be/3aMqf8ZcXeg

OBSERVATIONS

As the Analog Trimmer was turned from the minimum value to the maximum value, data was being continuously transmitted on the screen (going from 0x000 to 0x3FF)

INFERENCES

1. Using a high-level language such as C in writing ARM code helps make the code easier to write, while being able to use all of ARM's capability.
2. As the number of bits increases, the step size decreases, hence making the converter capable of converting smaller changes in the value of analog input to digital. Also, the maximum value of the digital value changes, as is trivial.
3. **SQNR** is given by the formula:

   $SQNR = 20\, log_{10}(2Q) \approx 6.02 \cdot Q$, where Q is the number of quantization bits.

   Here, the maximum value of the output is 0x3FF, which, when written in binary, is 11 1111 1111. This has 10 bits and hence, SQNR can is calculated to be:

   $$SQNR = 20\, log_{10}(2Q) \approx 6.02 \cdot Q = 6.02 \cdot 10 = 60.2$$

   The SQNR in this case, is therefore, **60.2**

LEARNING OUTCOMES

This implementation of ADC gave us the chance to actually understand the internal working of an ADC, which is used in a lot of applications to take values from the analog domain to the digital domain for various purposes such as processing data.

SERIAL COMMUNICATION

AIM

1. To display the ASCII code in LEDs, corresponding to the key pressed in the keyboard of the PC.
2. To receive characters serially from the pc and give a shifted output.

CODE

```c
#include "LPC23xx.h"

/*******************************************************************************
****************************
      Routine to set processor and peripheral clock
*******************************************************************************
****************************/

void  TargetResetInit(void)
{
  // 72 Mhz Frequency. Similar to ADC
  if ((PLLSTAT & 0x02000000) > 0)
  {
      /* If the PLL is already running */
      PLLCON  &= ~0x02;     /* Disconnect the PLL                           */
      PLLFEED  =  0xAA;                 /* PLL register update sequence, 0xAA,
0x55          */
      PLLFEED  =  0x55;
  }
  PLLCON   &= ~0x01;            /* Disable the PLL                           */
  PLLFEED   =  0xAA;            /* PLL register update sequence, 0xAA, 0x55
*/
  PLLFEED   =  0x55;
  SCS      &= ~0x10;            /* OSC RANGE = 0, Main OSC is between 1 and 20
Mhz      */
  SCS      |=  0x20;                /* OS CEN = 1, Enable the main oscillator
*/
  while ((SCS &  0x40) == 0);
  CLKSRCSEL = 0x01;                     /* Select main OSC, 12MHz, as the PLL
clock source        */
  PLLCFG    = (24 << 0) | (1 << 16);  /* Configure the PLL multiplier and
divider                   */
  PLLFEED   = 0xAA;                   /* PLL register update sequence, 0xAA,
0x55           */
  PLLFEED   = 0x55;
  PLLCON   |= 0x01;                     /* Enable the PLL
*/
  PLLFEED   = 0xAA;                     /* PLL register update sequence, 0xAA,
0x55                   */
  PLLFEED   = 0x55;
  CCLKCFG   = 3;                        /* Configure the ARM Core Processor
clock divider             */
  USBCLKCFG = 5;                      /* Configure the USB clock divider
*/
  while ((PLLSTAT & 0x04000000) == 0);
  PCLKSEL0  = 0xAAAAAAAA;               /* Set peripheral clocks to be half of
main clock        */
  PCLKSEL1  = 0x22AAA8AA;
  PLLCON    |= 0x02;                    /* Connect the PLL. The PLL is now the
active clock source    */
  PLLFEED   = 0xAA;                     /* PLL register update sequence, 0xAA,
0x55                   */
  PLLFEED   = 0x55;
  while ((PLLSTAT & 0x02000000) == 0);
  PCLKSEL0 = 0x55555555;      /* PCLK is the same as CCLK */
  PCLKSEL1 = 0x55555555;
}

// serial Reception routine
int serial_rx(void)
{ // pp421 LSR is RO 8-bit register in which b0 is set only if Receive Buffer
Register (RBR) contains valid data
  while (!(U0LSR & 0x01)); /* UARTn Line status register. If (RDR!=1) wait for
Rx to be over */
  return (U0RBR); /* Once RBR contains valid received data, read it */
}

//serial transmission routine
void serial_tx(int ch)
{ // pp421 LSR is RO 8-bit register in which b5 Transmit Holding Register Empty
(THRE) is set to 1 if THR is empty
//  while ((U0LSR & 0x20)!=0x20); THR - transmit holding register. == checks a
bit or 0x**?
  while ((U0LSR & 0x20)==0); /* UARTn Line status register. If (THRE!=1) wait
for tx to be over */
  U0THR = ch; // Once THR is empty, send the next ch to transmit
}

// serial transmission routine for string of characters
void string_tx(char *a)
{
      while(*a!='\0') // until end, keep transmitting
      {
        while((U0LSR&0X20)!=0X20); /* As long as (THRE!=1), wait (for tx to be
over) */
        U0THR=*a;  // once previous ch tx is over, take the next ch
        a++; // increment the address of ptr a to get the next ch
      }
}
/************************* main routine
************************************************************/
```

```
int  main ()
{
  unsigned int Fdiv;
  char value;
  TargetResetInit(); // initialization

  /**************************** uart1 initialization
************************************/
  PINSEL0 = 0x00000050; // Pin selection for uart tx (5:4) & rx (7:6) lines.
Table 106 pp157.
                        // for pin names, refer pp11 - block diagram
  U0LCR = 0x83; // pp419 WO 8 bit register, b7 DLAB=1, no Parity, 1 Stop bit
  Fdiv = ( 72000000 / 16 ) / 19200 ; // Given baud rate being 19200, dividers
DL_est computation
  //Fdiv = ( 72000000 / 16 ) / 2400 ; // If DL_est is an integer (see flowchart
pp427), then
                                      // quotient is U0DLM & remainder is
U0DLL. For fraction?????
  U0DLM = Fdiv / 256; // pp414 Division Latch MSB register - quotient
  U0DLL = Fdiv % 256; // pp428 Exmpls Division Latch LSB register - remainder
    U0LCR = 0x03;               // pp419 DLAB = 0 to disable access to divisor
latches

    while(1)
  {
    value=serial_rx(); // task given is to Rx & add 2 to it & Tx
    serial_tx(value+2);
  }
        return 0;
}
```

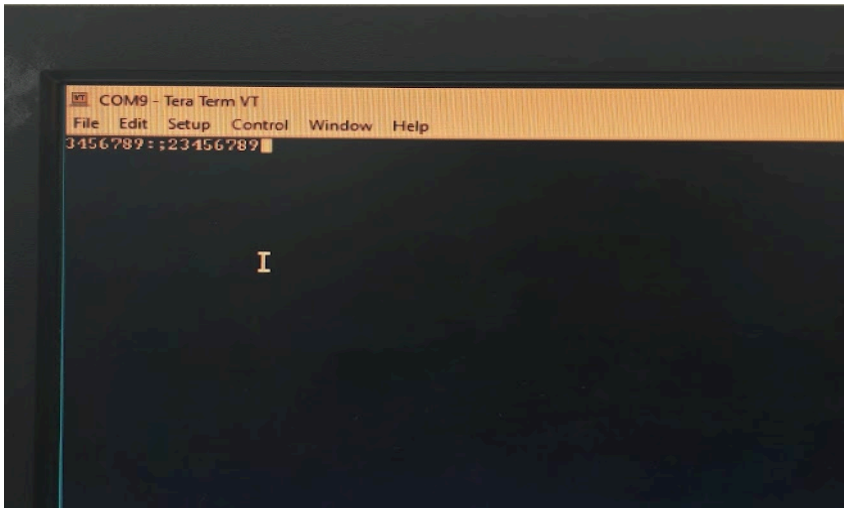
RESULTS

The code gave the following results:

1. The Serial Code takes the character inputted on the keyboard and displays a ASCII character 2 places shifted from the original input.
2. If we input ASCII character “A” ,we get a shifted output “C”.

IMAGES AND VIDEOS

Images taken during the experiment and a video showing the demonstration are given below:



```
3456789:;23456789
```

Link to Video Demonstration:https://youtube.com/shorts/UceuerOY758

OBSERVATIONS

The ASCII character that is typed on the keyboard is taken as input and an ASCII character shifted by 2 places is shown.
Eg - If ‘1’ is typed ‘3’ is displayed on the screen.

INFERENCES

The part of the code that shifts the ASCII character to be displayed by 2 is given below:

```
while(1)
{
value=serial_rx(); // task given is to Rx & add 2 to it & Tx
serial_tx(value+2);
}
```

The number that is added to value (in the argument for serial_tx function) can be changed to change the offset that the code produces in the character displayed.

Eg: Changing serial_tx(value+2) to serial_tx(value+5) will change the offset by 5. That is, instead of ‘1’, the value displayed will be ‘6’ and so on.

LEARNING OUTCOMES

This implementation of Serial code helps us in understanding how a computer takes in an input serially from a keyboard, converts it into ASCII and then displays the input on PC.

INDIVIDUAL CONTRIBUTIONS

1. Ajeet ES(EE22B086)
    - ADC Implementation Experiment
    - Performed the Experiment and took video and photo while Amogh was performing the other experiment
    - Wrote the report on ADC

2. Amogh Agrawal(EE22B087)
    - Serial Communication Experiment
    - Performed the experiment and took videos for ADC Implementation while Ajeet was performing the other experiment
    - Wrote the report on Serial Communication

Both of us worked together in writing the report in Google Docs