EE2016 : Lab Report

Experiment 6

Putta Shravya EE22B032
Maadhav Patel EE22b033

September 2023

1 Aim

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

2 Code and Results

2.1 Serial Communication

2.1.1 Code

```
#include "LPC23xx.h"

/*********************************************************************************************************
Routine to set processor and pheripheral 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 va
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 TH
//  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') // untill 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;
}
```

2.1.2 Inference and Learning Outcomes

1.Understanding how to initialize hardware settings,clock to set up micro controller.
2.Initializing and configuring peripherals, such as UART, for communication with other devices.
3.Understanding the concepts and implementation of serial communication protocols, including UART.
4.Learning about interrupt handling in serial communication to manage data reception and transmission efficiently.
5.Gaining experience in bit manipulation techniques to configure and interact with individual bits within registers.
6.Implementing routines for both serial transmission and reception, including handling buffers and ensuring efficient communication.
7.Implementing a routine to transmit a string of characters through UART, expanding the understanding of serial data transmission.
8.Developing debugging skills to identify and resolve potential issues or errors in the code to ensure proper functionality.
9.Learning to implement software delays for specific timing requirements within the application.

2.1.3 Observations

The provided code initializes and configures UART communication on an LPC2378 micro controller and continuously receives characters, increments them by 2, and transmits them back.

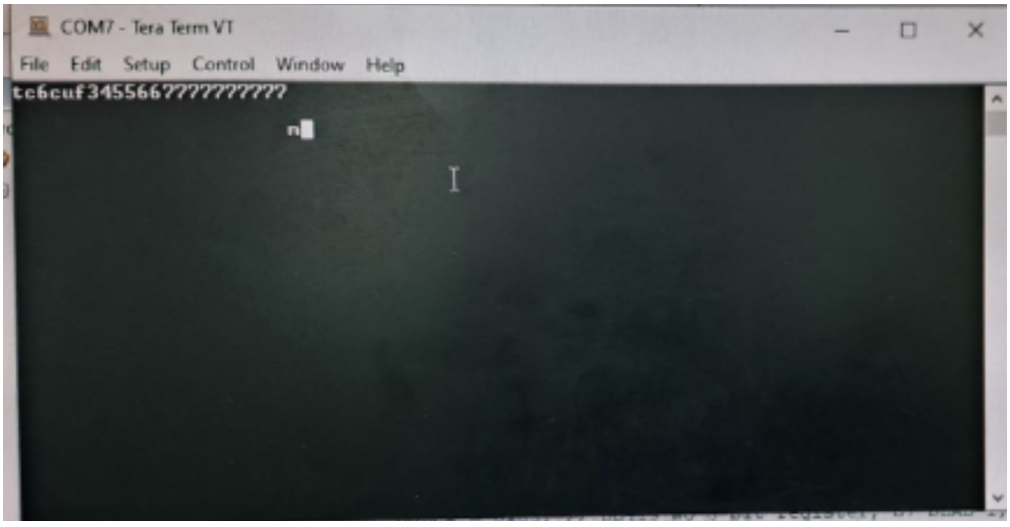

Figure 1: Serial Communication

2.2 ADC :Analog-to-Digital Converter

2.2.1 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 TH
  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;
```

```
}
```

2.2.2 Inference and Learning Outcomes

1.Understanding how to initialize and configure a micro controller.
2.how to configure and use the PLL to achieve a desired clock frequency.
3.how to configure and use UART for serial communication. This includes setting baud rates, configuring data formats, and transmitting data.
4.how to interface with an ADC to read analog signals and convert them to digital data.
5.how to configure peripheral registers and settings to enable specific functionalities, such as enabling and configuring the ADC and UART peripherals.
6.Gaining experience in register-level programming.Understanding how to interact with real-time hardware and peripherals, such as waiting for specific conditions (e.g., PLL stability) before proceeding.
7.Learning how to generate software delays using loops for controlling timing in embedded applications.
8.Developing skills in debugging and troubleshooting embedded code, identifying issues, and fixing them to ensure proper functionality of the application.
9.Understanding how to integrate different components (e.g., ADC, UART) to achieve a specific application goal.
10.Learning about converting analog data (from the ADC) to a suitable format (ASCII) for communication over UART.

2.2.3 Observations

The provided code configures and initializes UART and ADC (Analog-to-Digital Converter) on an LPC2378 microcontroller. It continuously reads ADC values from channel 0 and transmits them over UART.

(a) ADC

(b) ADC

2.3 Results

What is quantization error and how do you reduce it?

Quantization error is the noise introduced by quantization in an ideal ADC. It is a rounding error between the analog input voltage to the ADC and the output digitized value. the Signal-to-quantization-noise ratio (SQNR) can be calculated from

$$\text{SQNR} = 20 \log10(2Q) = 6.02 \cdot Q \text{ dB}$$ [2] where Q is the number of quantization bits.

Ways to reduce error:
Use an ADC with a higher bit resolution.
Increase the sampling rate by oversampling the analog signal and then averaging multiple samples.This reduces error by obtaining a more accurate average value.
Dithering can spread out the error, making it less perceptible and improving overall accuracy.

How step size varies with the number of bits used?
The step size in an Analog-to-Digital Converter (ADC) is determined by the resolution of the ADC, which is typically specified in bits. The resolution of an ADC refers to the number of digital bits that the ADC can represent for a given analog input range.
higher resolution (more bits) in the ADC provides a smaller step size, allowing for more accurate representation of the analog signal. Conversely, a lower resolution (fewer bits) results in a larger step size and less accurate representation of the analog signal.

What is the SQNR in the above case?
The Signal-to-Quantization Noise Ratio (SQNR) for an ADC with a given number of bits (n) of resolution can be calculated using the formula:

SQNR (in dB)=6.02n+1.76 dB

In the provided code, the ADC is configured with 10 bits of resolution, so we can calculate the SQNR in decibels:

SQNR (in dB)=6.02×10+1.76 dB
SQNR (in dB)=60.2+1.76 dB
SQNR (in dB)=61.96 dB

So, in this case, the Signal-to-Quantization Noise Ratio (SQNR) is approximately 61.96 dB.

3 Individual Contribution

The Experiment was done by both of the team members. Implimentation of code of serial communication,results,observations,learning outcomes were done by Maadhav Patel[EE22B033] Implimentation of code of ADC ,results,observations,learning outcomes were done by Shravya[EE22B032] Lab Report was written together by both the team members.