,

EE2016: Microprocessors Lab
Experiment # 8: Serial Communication and ADC/DAC Implementation in ViARM 2378 Development Board (through C - Interface)

Amizhthni PRK, EE21B015
Nachiket Dighe, EE21B093

November 16, 2022

Contents

List of Figures

1 Equipments used

- ARM ViARM 2378 Development board and accessories
- RS-232 cable
- Keil microvision 5 (C - interface)
- flash magic
- Burn o-mat
- DSO (Digital Storage Oscilloscope)
- Sample programs for generating digital inputs. (For analog inputs, potentiometer is used)

Figure 1: ViARM - 2378 Development Board

2 Serial Communication

2.1 Aim in this part of the experiment:

Write a program (in C) to display the ASCII code in LEDs, corresponding to the key pressed in the key board of the PC interfaced to ViARM-2378. Use the RS232 serial cable interfaced to the Vi Microsystem's ViARM 2378 development board.

2.2 Approach and Code Explanation

- Here the input is taken from keyboard and the output is displayed on the LEDs which are present on LPC2378.
- The FIO3DIR is assigned as the output which causes the output to be displayed on LEDs.
- The serial input is stored in the variable "value" and assigned to FIO3PIN for display.
- In the infinite while loop, the programme takes serial input from the user continuously and computes its ASCII value and hence displays the output on LEDs.
- The ASCII value is output in binary using the 8 LEDs on LPC2378 development board.

2.3 Code

The code for displaying the ASCII code in LEDS is given in listing 1.

```
#include "LPC23xx.h"

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

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           */
```

```
21    SCS      |=  0x20;                          /* OSCEN = 1, Enable the main
      ↪  oscillator                    */
22    while ((SCS &  0x40) == 0);
23    CLKSRCSEL = 0x01;                          /* Select main OSC, 12MHz, as
      ↪  the PLL clock source          */
24    PLLCFG    = (24 << 0) | (1 << 16);         /* Configure the PLL
      ↪  multiplier and divider                 */
25    PLLFEED   = 0xAA;                          /* PLL register update
      ↪  sequence, 0xAA, 0x55                  */
26    PLLFEED   = 0x55;
27    PLLCON    |= 0x01;                         /* Enable the PLL
      ↪  */
28    PLLFEED   = 0xAA;                          /* PLL register update sequence,
      ↪  0xAA, 0x55                 */
29    PLLFEED   = 0x55;
30    CCLKCFG   = 3;                           /* Configure the ARM Core Processor
      ↪  clock divider           */
31    USBCLKCFG = 5;                       /* Configure the USB clock divider
      ↪  */
32    while ((PLLSTAT & 0x04000000) == 0);
33    PCLKSEL0  = 0xAAAAAAAA;                     /* Set peripheral clocks to
      ↪  be half of main clock          */
34    PCLKSEL1  = 0x22AAA8AA;
35    PLLCON    |= 0x02;                         /* Connect the PLL. The PLL is
      ↪  now the active clock source  */
36    PLLFEED   = 0xAA;                          /* PLL register update
      ↪  sequence, 0xAA, 0x55                  */
37    PLLFEED   = 0x55;
38    while ((PLLSTAT & 0x02000000) == 0);
39    PCLKSEL0 = 0x55555555;                      /* PCLK is the same as CCLK
      ↪  */
40    PCLKSEL1 = 0x55555555;
41  }
42
43
44  // serial Reception routine
45  int serial_rx(void)
46  {
47          while (!(U0LSR & 0x01));
48          return (U0RBR);
49  }
50  //serial transmission routine
51  void serial_tx(int ch)
52  {
53  //  while ((U0LSR & 0x20)!=0x20);
54    while ((U0LSR & 0x20)==0);
55    U0THR = ch;
56  }
57  // serial transmission routine for string of characters
58  void string_tx(char *a)
```

```
{
      while(*a!='\0')
      {
        while((U0LSR&0X20)!=0X20);
        U0THR=*a;
        a++;
      }
}
/************************ main routine
 ↪  ************************************************************/
int  main ()
{
        unsigned int Fdiv;
        char value;
        TargetResetInit();
        FIO3DIR = 0xFF;

        /*************************** uart1 initialization
 ↪  ***********************************/
        PINSEL0 = 0x00000050;

        U0LCR = 0x83;                // 8 bits, no Parity, 1 Stop bit
        Fdiv = ( 72000000 / 16 ) / 19200 ;  //baud rate
        U0DLM = Fdiv / 256;
        U0DLL = Fdiv % 256;
  U0LCR = 0x03;               //DLAB = 0


  while(1)
        {
                value=serial_rx();
                FIO3PIN = value;
                serial_tx(value);
        }
        return 0;
}
```

Listing 1: Code for factorial finding

2.4 Input and Output

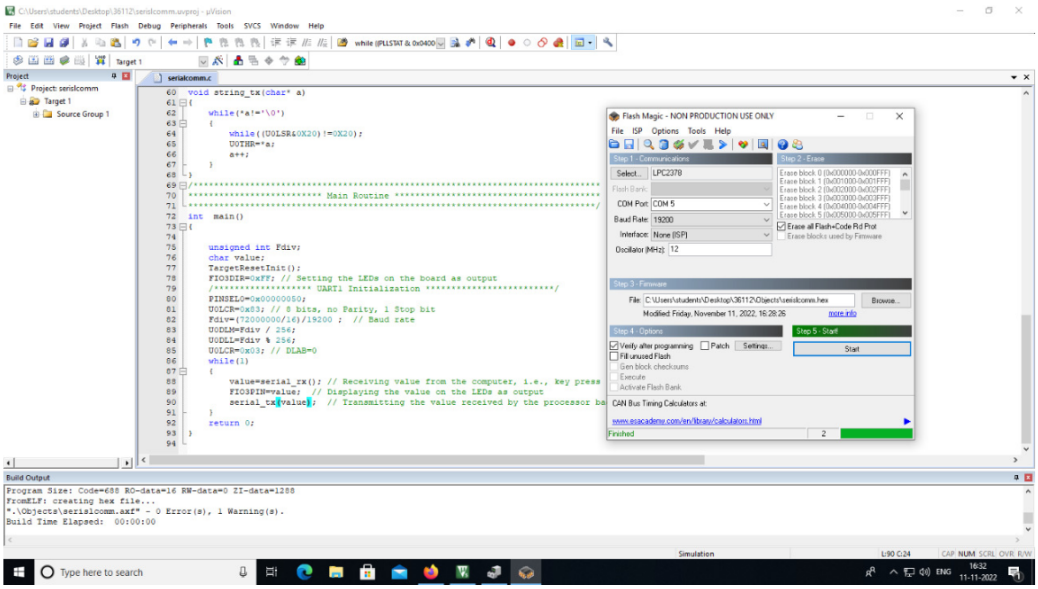

Figure 2: Burning of Code for LED displaying

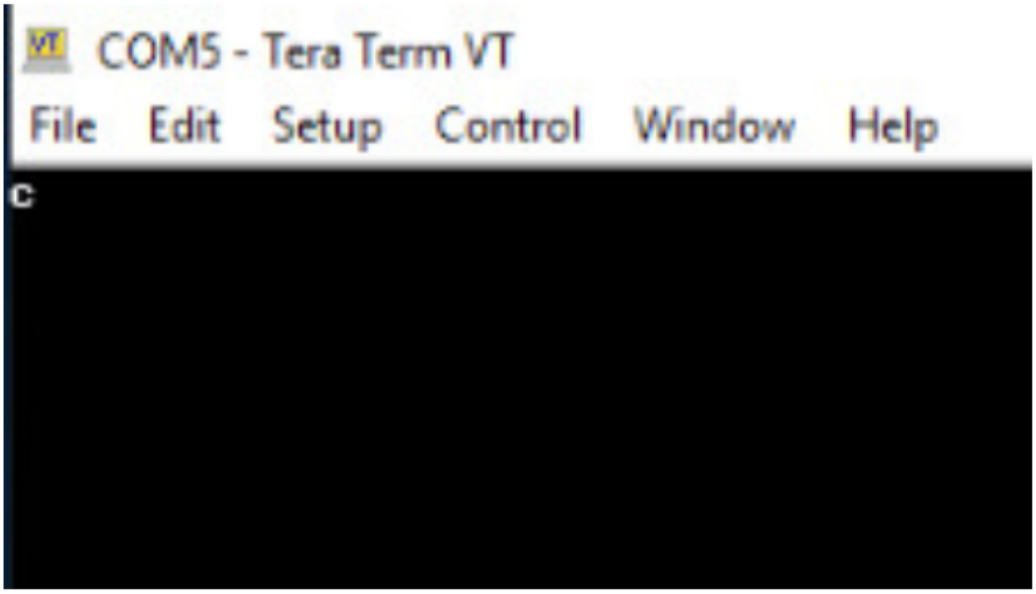

Figure 3: Input

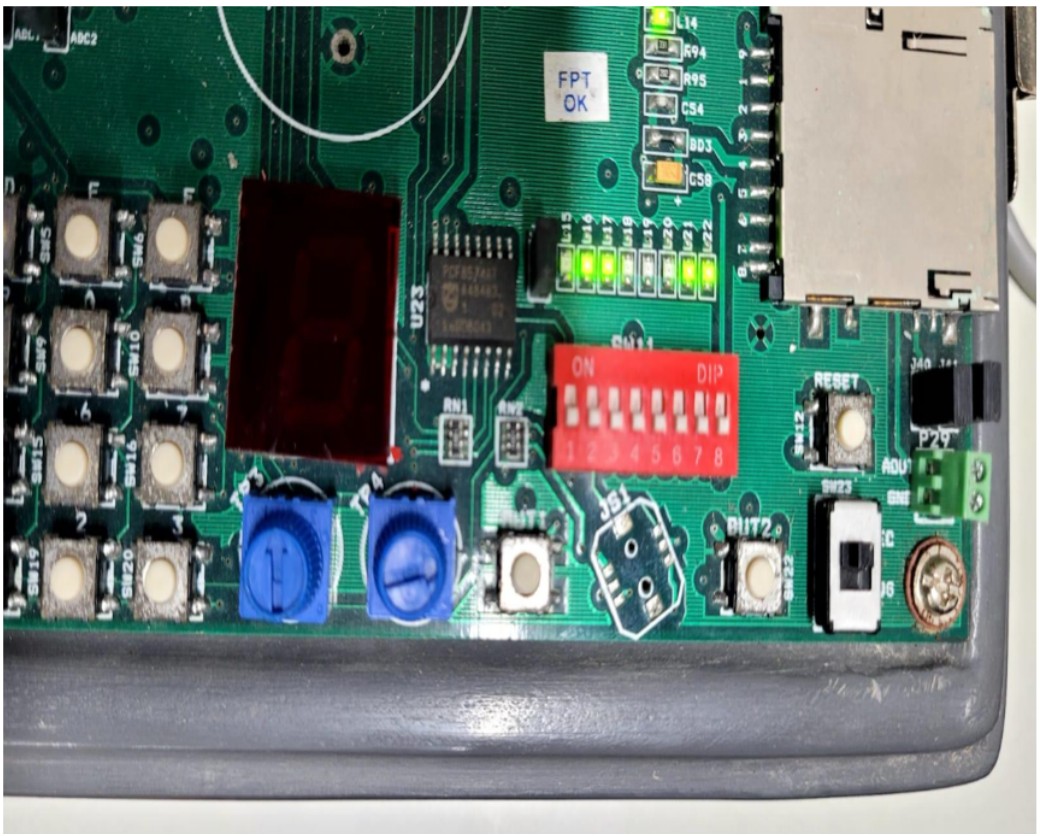

Figure 4: Output

2.5 Inference

Here we have input the value ‘c’ into the Tera Term software. We can see leds glowing. In binary form we can represent glowing of leds as 01100011, which in binary is equivalent to 99. ASCII value of ‘c’ = 99. Hence the task was accomplished successfully.

3 ADC

3.1 Aim in this part 2 of experiment:

Given a real-time (analog) signal from a sensor, convert it into a digital signal (Implement an ADC). Decrease the step size? Do you see any change in the bits used to represent the whole range? What is the quantization error?

3.2 Approach and Explanantion

- We are to convert the given input of analog signal into digital signal.
- Thus, we quantise the signal and assign the step size according to our quantisation size o For example, an 8-bit DAC that generates a maximum output voltage of 5 volts has a step size or resolution of

$$\frac{5V}{28} = 19.5mV$$

.

- Here we have taken 10-bit which ranges from 0 to 0x3FF.We have a voltage signal of 3V which is varied from 0 to 3V. Hence the step size or resolution is given by

$$\frac{3V}{22} * 28 = 2.92mV$$

.

- The second part is to reduce the step size of the signal. As an example, if we use 11-bit which ranges from 0 to 0x7FF. The step size or resolution is given by

$$\frac{3V}{23} * 28 = 1.46mV$$

.

3.3 Code

The code used for ADC is given below in listing 2 and for Decreasing Step size is in listing 3.

```
#include "LPC23xx.h"

/*************************
****** Routine to set processor and peripheral clock ******
*************************/
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)
{
    while ((U0LSR&0x20)!=0x20);
    U0THR=ch;
}
/***************************
********* Hex to ASCII Routine ***********
***************************/
int atoh(int ch)
{
    if(ch<=0x09)
        ch=ch+0x30;
    else
        ch=ch+0x37;
    return(ch);
}
/***************************
********* Main Routine **************
***************************/
int main()
{
```

```
    unsigned int Fdiv,value,i,j;
    TargetResetInit();

    PCONP|=0X00001000; // switch ADC from disable state to enable state
    PINSEL0=0x00000050; // Pinselection for UART TX and RX lines
    PINSEL1=0X01554000; // Pinselection for ADC0.0
    /******* UART Initialization ***********/
    U0LCR=0x83; // 8 bits, no Parity, 1 Stop bit
    Fdiv=(72000000/16)/19200 ; // Baud rate
    U0DLM=Fdiv/256;
    U0DLL=Fdiv%256;
    U0LCR=0x03; // DLAB=0
    AD0CR=0X01210F01; // ADC initialization
    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('\t');
        serial_tx(atoh((value&0x300)>>8));
        serial_tx(atoh((value&0xf0)>>4));
        serial_tx(atoh(value&0x0f));
        serial_tx(0x0d);
        serial_tx(0x0a);
        for(i=0;i<=0xFF;i++)
            for(j=0;j<=0xFF;j++);
    }
    return 0;
}
```

Listing 2: ADC part 1

```
#include "LPC23xx.h"

/*************************
****** Routine to set processor and peripheral clock ******
*************************/
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)
{
    while ((U0LSR&0x20)!=0x20);
    U0THR=ch;
}
/*************************
********* Hex to ASCII Routine ***********
*************************/
int atoh(int ch)
{
    if(ch<=0x09)
        ch=ch+0x30;
    else
        ch=ch+0x37;
    return(ch);
}
/*************************
********* Main Routine ***************
*************************/
int main()
{
    unsigned int Fdiv,value,i,j;
```

```
    TargetResetInit();

    PCONP|=0X00001000; // switch ADC from disable state to enable state
    PINSEL0=0x00000050; // Pinselection for UART TX and RX lines
    PINSEL1=0X01554000; // Pinselection for ADC0.0
    /******* UART Initialization ***********/
    U0LCR=0x83; // 8 bits, no Parity, 1 Stop bit
    Fdiv=(72000000/16)/19200 ; // Baud rate
    U0DLM=Fdiv/256;
    U0DLL=Fdiv%256;
    U0LCR=0x03; // DLAB=0
    AD0CR=0X01210F01; // ADC initialization
    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('\t');
        serial_tx(atoh((value&0x300)>>8));
        serial_tx(atoh((value&0xf0)>>4));
        serial_tx(atoh(value&0x0f));
        serial_tx(0x0d);
        serial_tx(0x0a);
        for(i=0;i<=0xFF;i++)
            for(j=0;j<=0xFF;j++);
    }
    return 0;
}
```

Listing 3: ADC part 2

3.4 Explanation

- Here input is supposed to be an analog signal and corresponding output must be digital signal. We are giving input through potentiometer varies from **(0-3V). Our digital output for the input range (0-3V) will be in the range of (0-3FF).**
- From the above output image, we can see some value (339) as our output.
- This is the output corresponding to some voltage in the potentiometer.
- We also observed that the output bits increase as the step size decreases. For example, the desired step size of **1.46 mV could be obtained by increasing the number of bits to 11.**
- We also observed that the output bits decrease as the step size increases. For example, the desired step size of **5.85 mV could be obtained by decreasing the number of bits to 9.**
- **Quantisation Error**: We have expressed the quantisation error in terms of the **Signal-to quantization-noise ratio (SQNR)**

- **For Q = 10 bits** $SNQR = 20\log 2^Q \approx 6.0210Db = 60.2Db$
- **For Q = 9 bits:** $SNQR = 20\log 2^Q \approx 6.029Db = 54.18Db$
- **For Q = 11 bits:** $SNQR = 20\log 2^Q \approx 6.0211Db = 66.22Db$
- Hence the task was accomplished successfully.

3.5 Output and Code burn

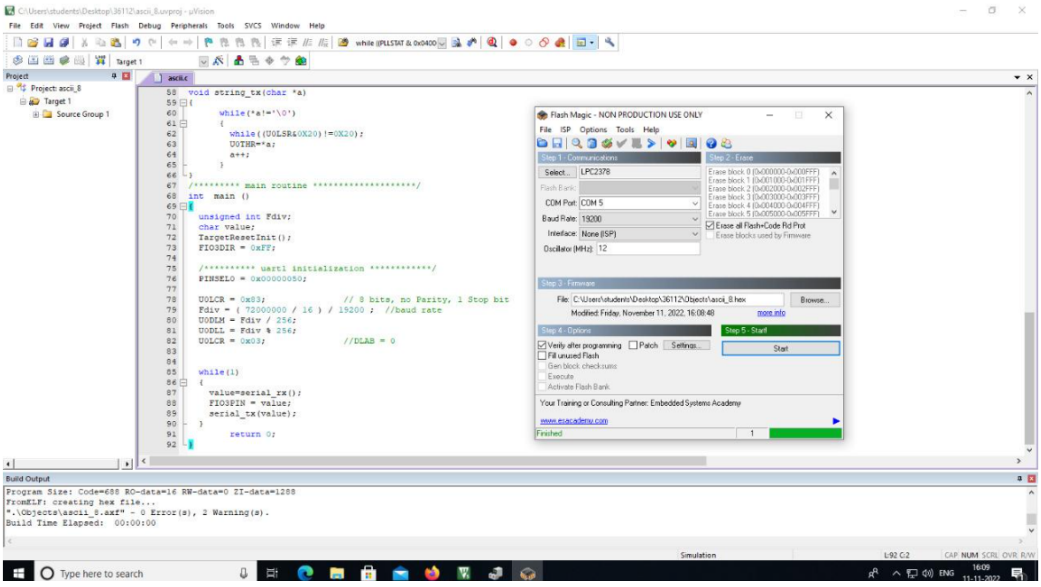

Figure 5: Burning of Code

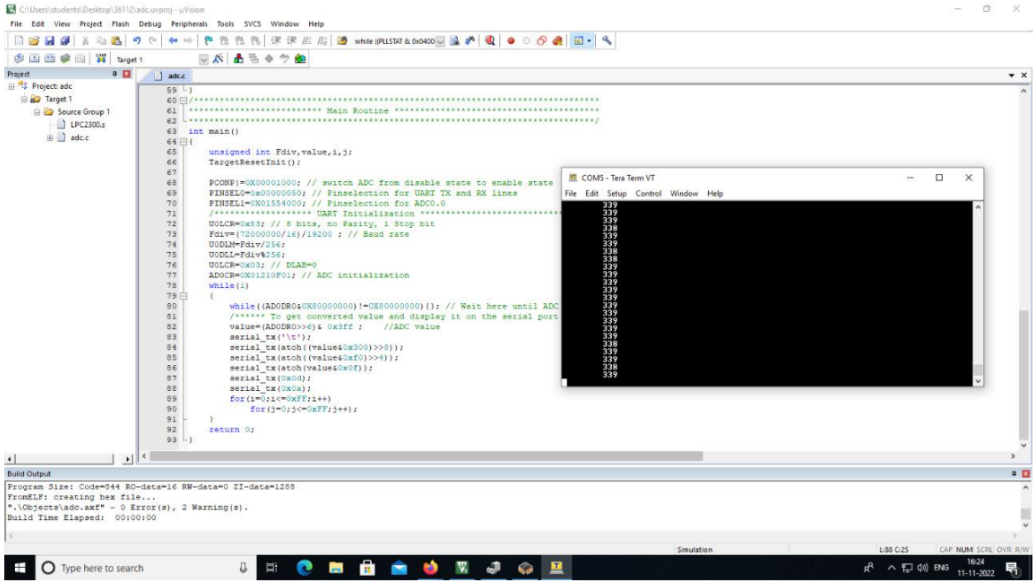

Figure 6: Output

4 DAC

4.1 Aim in this part 3 of experiment:

Given the ViARM2378 ARM development board, generate :

- Square wave
- Triangular wave
- Sine wave (using lookup table)

4.2 Square Wave

4.2.1 Explanation for Square wave

- Here the square wave is obtained by the input we give to the variable "Value"
- Execute an infinite while loop
- Assign the value as High (1023)
- Give the delay
- Assign the value as low (0)
- Give the delay
- Since we change the input to value drastically, we get a square wave as desired

4.2.2 Code

The code used for combining Elements of an array is given below in listing 4.

```
#include "LPC23xx.h"
void delay(int n)
{
    int i,j;
    for(i=0;i<n;i++)
        for(j=0;j<0x0F;j++);
}

int main (void)
{
    PCLKSEL0=0x00C00000;
    PINMODE1=0x00300000;
    PINSEL1=0x00200000;
    int value;
    int i=0;

    while(1)
    {
        value=1023;
        DACR=(value<<6);
        delay(100);
```

```
        value=0;
        DACR=(value<<6);
        delay(100);
    }
    return 0;
}
```

Listing 4: Square Wave

4.3 Output and Code burn

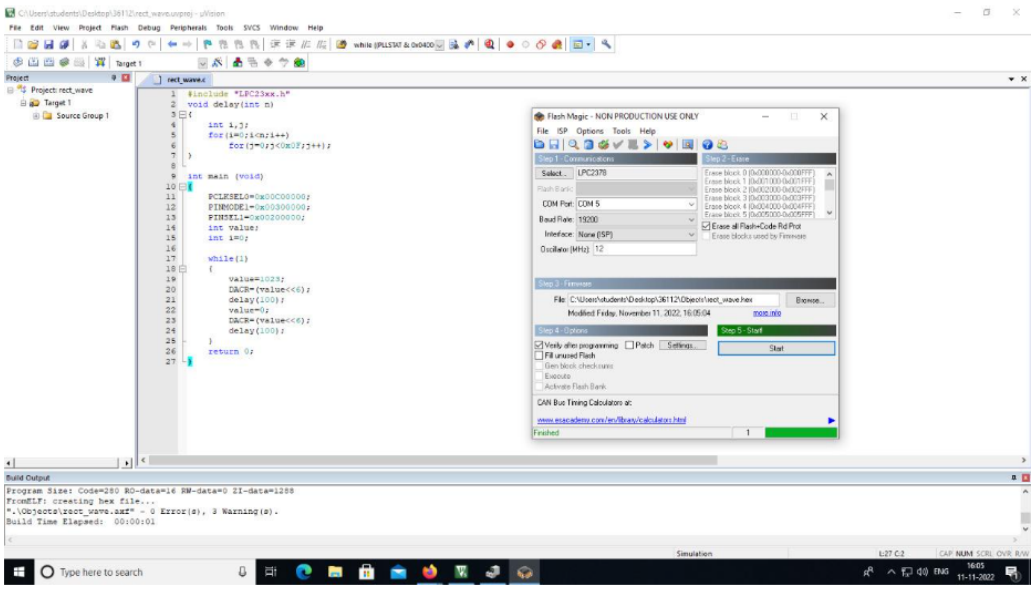

Figure 7: Burning of Code for Square wave

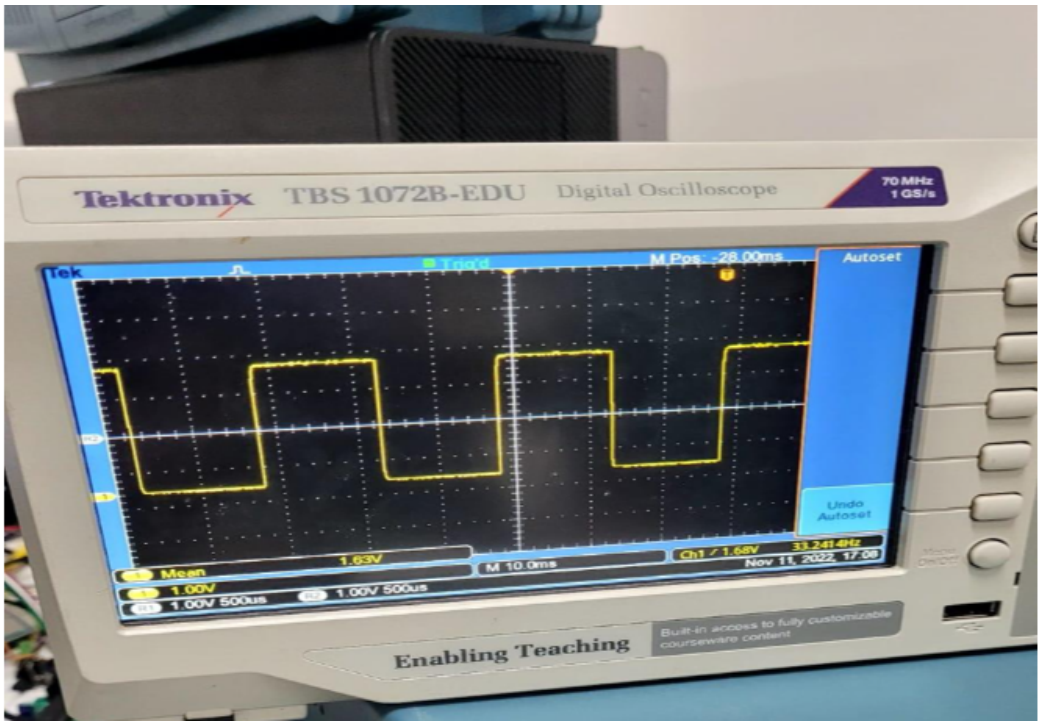

Figure 8: Output for Square wave

4.4 Triangular Wave

4.4.1 Explanation for Trianngular wave

- Here the triangular wave is obtained by the input we give to the variable "Value"
- Execute an infinite while loop
- Assign the value as low (0)
- Increment the value continuously until the value = 1023.
- Here we increase the value linearly by sampling, hence, we get the straight line with +ve slope as the desired AC output.
- Now decrement the value from 1023 to 0 in the steps of 1 .
- Here we decrease the value linearly by sampling, hence, we get the straight line with -ve slope as the desired output.
- These two outputs combine to form a period of triangular wave.
- Hence the desired triangular wave is obtained by changing the value continuously in linear fashion

4.4.2 Code

The code used for Triangular wave is given below in listing 5.

```
1 #include "LPC23xx.h"
2
3 void delay(unsigned int k)
```

```
{
unsigned int i,j;
for(i=0;i<=k;i++);
for(j=0;j<=0xFF;j++);
}

int  main ()
{
        unsigned int value=0;

        PINSEL1 = 0x00200000;                   //Pinselection for uart tx
 ↪  and rx lines

        /************** Uart initialization
 ↪  *********************************/
    PCLKSEL0 = 0x00C00000;
    PINMODE1=0x00300000;
        FIO4DIR=0x00;

 while(1)
 {
  value=0;
  while(value!=1023)
        {
                DACR = ((1<<16)|(value<<6));
                value++;
        }
        while(value!=0){
                DACR = ((1<<16)|(value<<6));
                value--;
        }
return 0;
}
 }
```

Listing 5: Triangular Wave

4.5 Output and Code burn

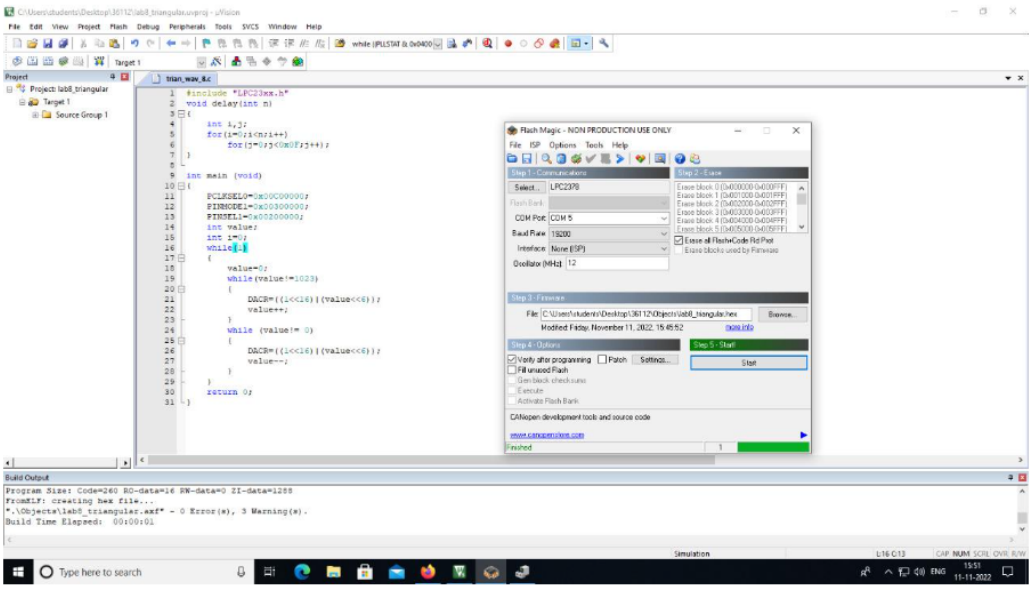

Figure 9: Burning of Code for Triangular wave

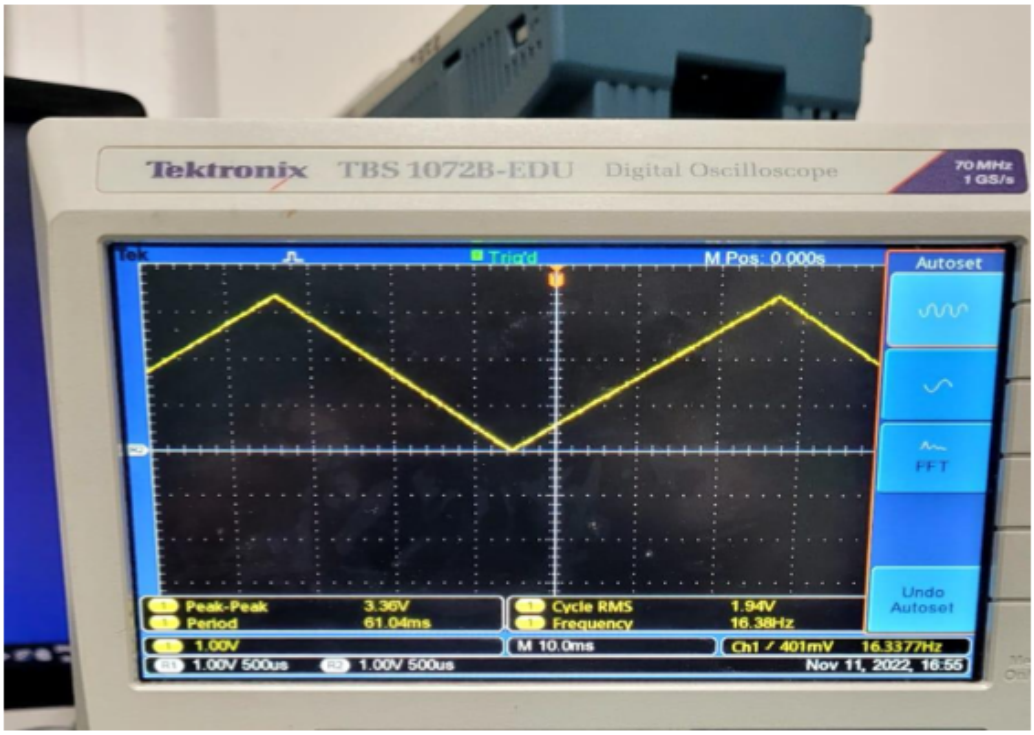

Figure 10: Output for triangular wave

4.6 Sine Wave

4.6.1 Explanation for Sine wave

- Here the sinusoidal wave is obtained by the input we give to the variable "Value"
- The sine wave is modeled in DC as an array with the series of 101 values which vary sinusoidally with each step
- The value is assigned as sinewave[i] where "i" ranges from 0 to 101
- A for loop is executed in which the values are given such that a complete cycle of sine wave is executed by LPC2378.
- The loop is put inside an infinite while loop to produce the required signal
- Thus the ARM board generates the required sinusoidal signal as shown below

4.6.2 Code

The code used for Sine wave is given below in listing 6.

```
#include "LPC23xx.h"
int sin_wave[101]={0x200,0x220,0x240,0x25f,0x27f,
    0x29e,0x2bc,0x2d9,0x2f6,0x312,0x32c,0x346,0x35e,0x374,
    0x38a,0x39d,0x3af,0x3c0,0x3ce,0x3db,0x3e6,0x3ef,0x3f6,
    0x3fb,0x3fe,0x3ff,0x3fe,0x3fb,0x3f6,0x3ef,0x3e6,0x3db,
    0x3ce,0x3c0,0x3af,0x39d,0x38a,0x374,0x35e,0x346,0x32c,
    0x312,0x2f6,0x2d9,0x2bc,0x29e,0x27f,0x25f,0x240,0x220,
    0x200,0x1df,0x1bf,0x1a0,0x180,0x161,0x143,0x126,0x109,
    0xed,0xd3,0xb9,0xa1,0x8b,0x75,0x62,0x50,0x3f,0x31,0x24,
    0x19,0x10,0x9,0x4,0x1,0x0,0x1,0x4,0x9,0x10,0x19,0x24,
    0x31,0x3f,0x50,0x62,0x75,0x8b,0xa1,0xb9,0xd3,0xed,0x109,
    0x126,0x143,0x161,0x180,0x1a0,0x1bf,0x1df,0x200};

void delay(unsigned int k)
{
unsigned int i,j;
for(i=0;i<=k;i++);
for(j=0;j<=0x0F;j++);
}

int  main ()
{
        unsigned int value=0;

        PINSEL1 = 0x00200000;                   //Pinselection for uart tx
↪   and rx lines

        /** Uart initialization ******/
    PCLKSEL0 = 0x00C00000;
    PINMODE1=0x00300000;
        FIO4DIR=0x00;
    //  Lookup Table for sine values
```

```
32
33
34 while(1){
35         for (int i=0; i<101; i++){
36                 value = sin_wave[i];
37                 DACR = (value<<6);
38                 delay(100);
39         }
40 }
41
42 return 0;
43  }
```

Listing 6: Sine Wave

4.7 Output and Code burn

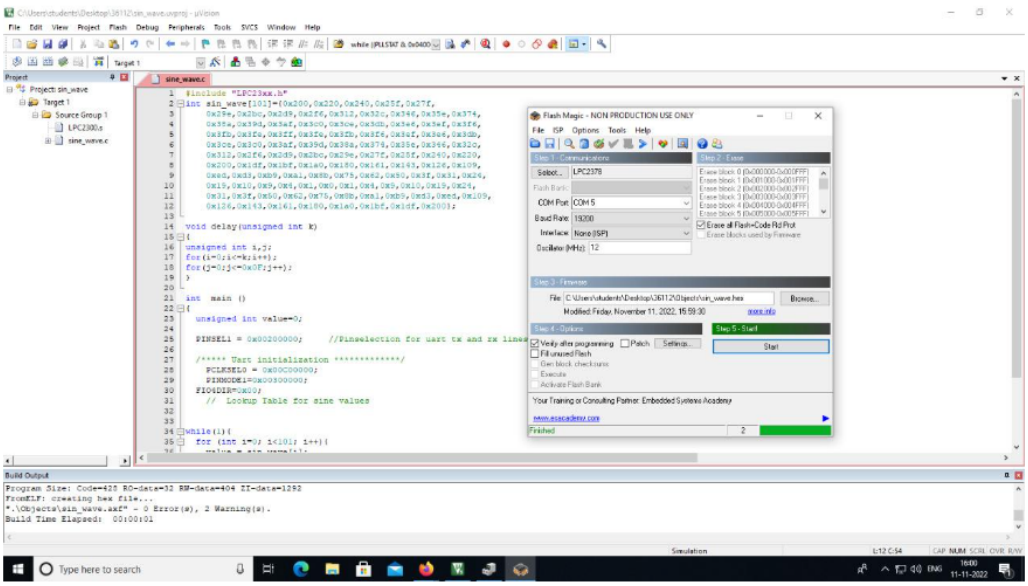

Figure 11: Burning of Code for Sine Wave

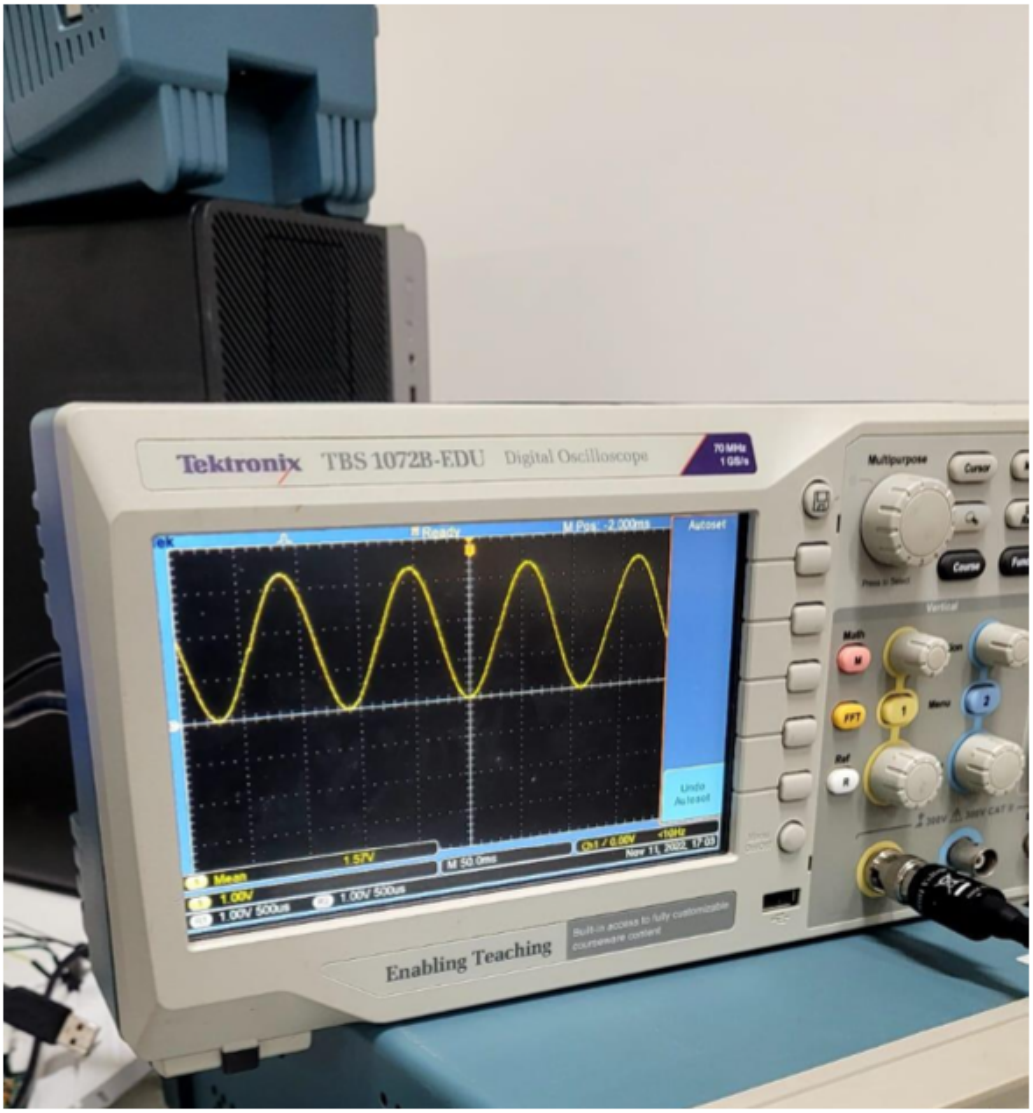

Figure 12: Output for Sine wave

5 Videos and Photos of the Experiment

https://drive.google.com/drive/folders/1FuoAFY4LjPoV6gwBKa7EYNsK3tXG9paF?usp=sharing