Peripheral Interfacing on LPC2148 Dev Board using ARM C Programming

EE2016 Microprocessors

Sanjeev Subrahmaniyan S B

EE23B102

Abstract

ARM processors are the buzzword in the microprocessor industry now, with most devices having processors built on the ARM RISC architecture. They are particulary common in embedded and handheld devices like microcontrollers and smartphones. This experiment uses the LPC ARM chip, which is suited for low power applications. Using the C programming language to program ARM based microcontrollers is a very useful skill to be picked up from this course. This experiment involves interfacing peripherals such as LEDs, DIP Switches and a DC Stepper motor with the microcontroller. The programs for the tasks are written with the C programming language. The tasks include blinking LEDs, mirroring DIP switches on LEDs, addition and multiplication of 4 bit numbers on the 32 bit processor and controlling a stepper motor.

1 Introduction

The LPC2148 ARM microcontroller is an ARM7 TDMI microcontroller produced by NXP semiconductors and is largely suited for low power applications. It is a versatile microcontroller in providing support for 16 bit instruction through the thumb instruction set. For this experiment, a development kit centred around this microcontroller is provided, which has support for a number of peripherals. Specifically, the LEDs and the DIP switches of the development board will be used in this experiment, alongside the J16 pin for the DC stepper motor. The tasks are performed by using Keil Microvision to write and debug the appropriate C programs and flashing the hex files on to the microcontroller. This experiment helps on gain experience in working with a fully built microcontroller development kit.

2 Objectives

1. Blink the LEDs in the development board.

2. Mirror the DIP switch inputs onto the LEDs of the board.
3. Add 2 4-bit numbers on the 32 bit processor and display the output on LEDs.
4. Multiply 4-bit numbers and display the output on the LEDs.
5. Control a stepper motor by making it rotate in both directions and varying the speed of the motor.

3 Procedure, Programs and Results

3.1 Blinking LEDs

The objective of this task is to be able to blink all the LEDs to test their working and the correctness of the port configurations.

3.1.1 Code

```
#include "LPC214x.h"                               /* LPC21xx
   definitions */

 #define LED_IOPIN       IO0PIN
#define BIT(x)  (1 << x)

#define LED_D0  (1 << 10)       // P0.10
#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
   )))
    #ifndef LED_DRIVER_OUTPUT_EN
#define LED_DRIVER_OUTPUT_EN (1 << 5)   // P0.5
#endif
#define LED1_ON     LED_IOPIN |= (unsigned long)(LED_D0);
        // LED1 ON
```

```c
#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

int main (void)
{

  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)
    {
        int value=0x86;

    if(value & BIT(0)) LED8_ON;
    if(value & BIT(1)) LED7_ON;
    if(value & BIT(2)) LED6_ON;
    if(value & BIT(3)) LED5_ON;

    if(value & BIT(4)) LED4_ON;
    if(value & BIT(5)) LED3_ON;
    if(value & BIT(6)) LED2_ON;
    if(value & BIT(7)) LED1_ON;
    }

  return 0;
}
```

3.1.2 Result

The result of the code is that all the LEDs will be turned on and off in succession in a smooth fashion. The attached screenshot shows a still from the video, showing all the LEDs active on the bottom right corner of the image on the LPC development kit.

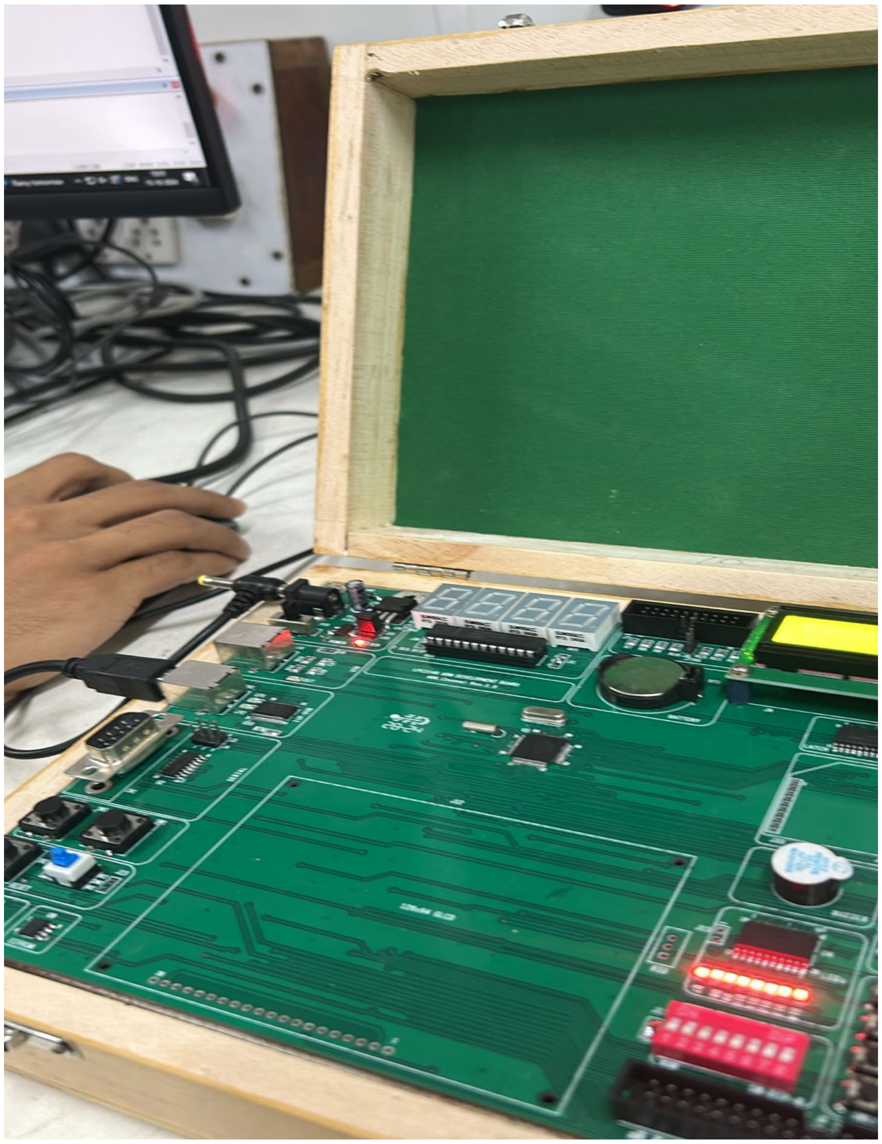

Figure 1: Snapshot of the LEDs blinking showing all the LEDs turned on

3.2 Mirroring DIP Switches onto LEDs

This task required us to show the inputs on the DIP switches on the LED lights. This means the LEDs corresponding to each of the DIP switch will be turned on/off as per the status of the switch itself.

3.2.1 Code

```c
#include "LPC214x.h"                                /* LPC21xx definitions */

 #define LED_IOPIN      IO0PIN
#define BIT(x)  (1 << x)

#define LED_D0  (1 << 10)       // P0.10
#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)))
    #ifndef LED_DRIVER_OUTPUT_EN
#define LED_DRIVER_OUTPUT_EN (1 << 5)   // P0.5
#endif
#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

int main (void)
{

  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)
    {
        int value=0x86;

    if(value & BIT(0)) LED8_ON;
    if(value & BIT(1)) LED7_ON;
    if(value & BIT(2)) LED6_ON;
    if(value & BIT(3)) LED5_ON;

    if(value & BIT(4)) LED4_ON;
    if(value & BIT(5)) LED3_ON;
    if(value & BIT(6)) LED2_ON;
    if(value & BIT(7)) LED1_ON;
    }

  return 0;
}
```

3.2.2 Result

The result of the program is that whatever is input in the DIP switch will be shown on the LED array after a specified time delay as given in the program. In the attached image it can be seen that the DIP switch inputs are 0b00111010 which is also the value that is seen on the LED array.

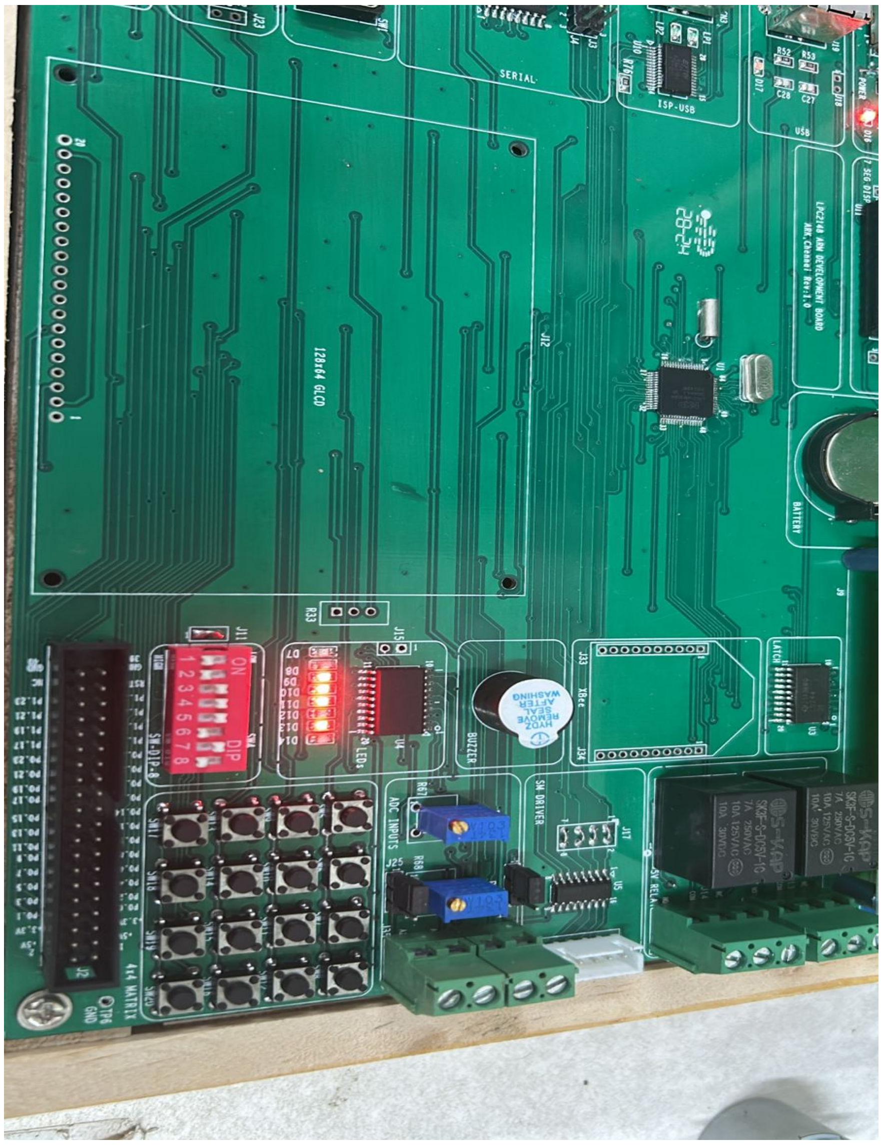

Figure 2: Snippet of the DIP inputs mirrored onto the LED array

3.3 Addition of 4-bit numbers

This task requires the user to feed into two 4 bit numbers on the DIP switch array, and would display an output which is equal to the addition of the two 4 bit numbers on the LED array. This means, the 8 bit DIP switch is divided into two 4 bit nibbles which are the numbers to be added.

3.3.1 Code

```
#include "LPC214x.H"                             /* LPC214x
   definitions */
#include "led.h"
#include "delay.h"


#define DIP_SW_D0 (1 << 16)      // P0.16
#define DIP_SW_D1 (1 << 17)      // P0.17
#define DIP_SW_D2 (1 << 18)      // P0.18
#define DIP_SW_D3 (1 << 19)      // P0.19

#define DIP_SW_D4 (1 << 22)      // P0.22
#define DIP_SW_D5 (1 << 23)      // P0.23
#define DIP_SW_D6 (1 << 24)      // P0.24
#define DIP_SW_D7 (1 << 25)      // P0.25

#define DIP_SW_DIR       IO1DIR
#define DIP_SW_PIN       IO1PIN

#define DIP_SW_DATA_MASK    (DIP_SW_D7 | DIP_SW_D6 | DIP_SW_D5
   | DIP_SW_D4 | DIP_SW_D3 | DIP_SW_D2 | DIP_SW_D1 | DIP_SW_D0)

void set_dipswitch_port_input( void )
{
    DIP_SW_DIR &= ~(DIP_SW_DATA_MASK);
}

unsigned long read_dip_switch( void )
{
    return DIP_SW_PIN;
}

int main (void)
{
    unsigned long sw_status;

    set_led_port_output();
    set_dipswitch_port_input();

    while(1)
    {
        sw_status = read_dip_switch();
        unsigned long input1 = (~(sw_status >> 16))&0x0F;
        unsigned long input2 = (~(sw_status >> 22))&0x0F;

        unsigned long sum = input1 * input2;


        sw_status = sum << 16;
        sw_status = ~sw_status;


        if(sw_status & DIP_SW_D0){ LED1_OFF;} else{ LED1_ON;}
                delay_mSec(10);
        if(sw_status & DIP_SW_D1){ LED2_OFF;} else{ LED2_ON;}z
                delay_mSec(10);
        if(sw_status & DIP_SW_D2){ LED3_OFF;} else{ LED3_ON;}
                delay_mSec(10);
        if(sw_status & DIP_SW_D3){ LED4_OFF;} else{ LED4_ON;}
                delay_mSec(10);

        if(sw_status & DIP_SW_D4){ LED5_OFF;} else{ LED5_ON;}
                delay_mSec(10);
        if(sw_status & DIP_SW_D5){ LED6_OFF;} else{ LED6_ON;}
                delay_mSec(10);
        if(sw_status & DIP_SW_D6){ LED7_OFF;} else{ LED7_ON;}
                delay_mSec(10);
        if(sw_status & DIP_SW_D7){ LED8_OFF;} else{ LED8_ON;}
                delay_mSec(10);

        delay_mSec(100);
    }

//      return 0;
}
```

3.3.2 Result

As expected from the objective of the task, the output is simply the addition of the two 4 bit numbers. The following images show the results of the addition, which also demonstrate the carry capacity of the adder that has been constructed. **Note that here the bit order is inverted. This means the least significant bit is on the left in both the input and the output.**

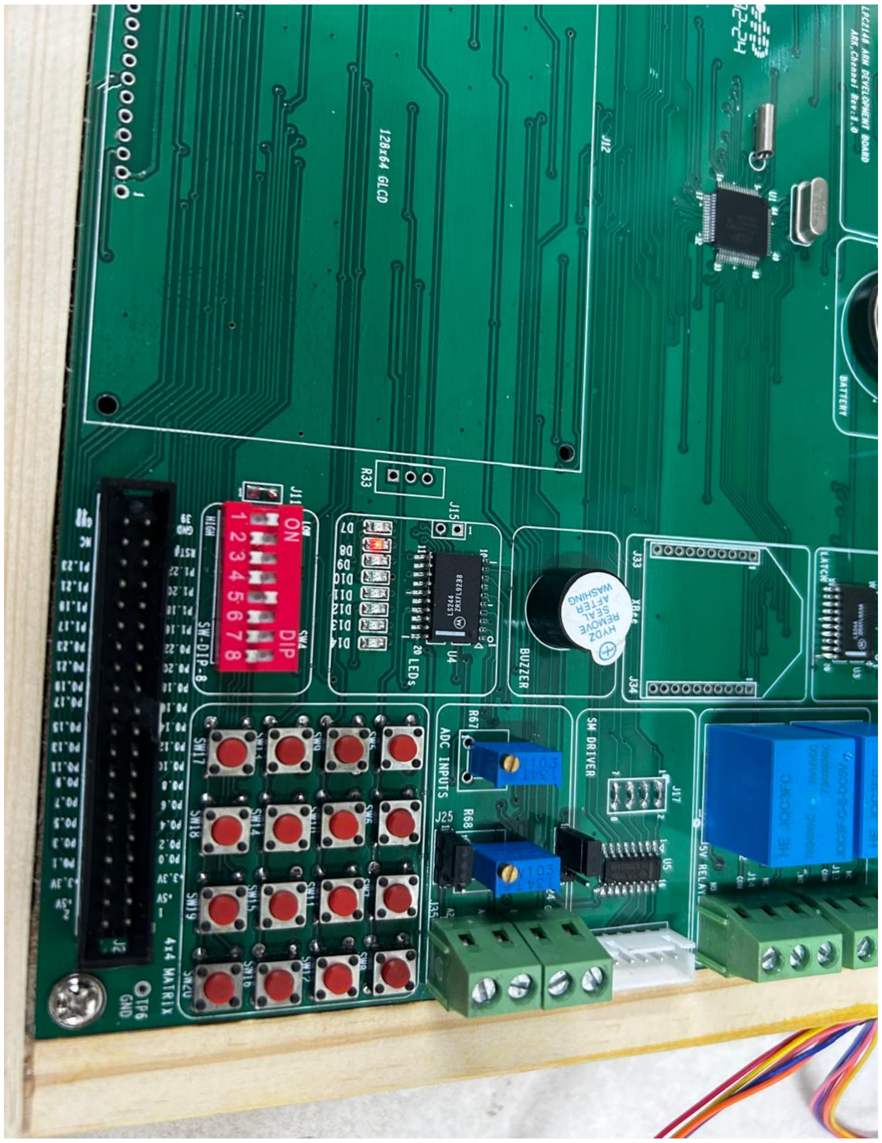

Figure 3: Addition result of 0b0001 and 0b0001 resulting in 0b0010

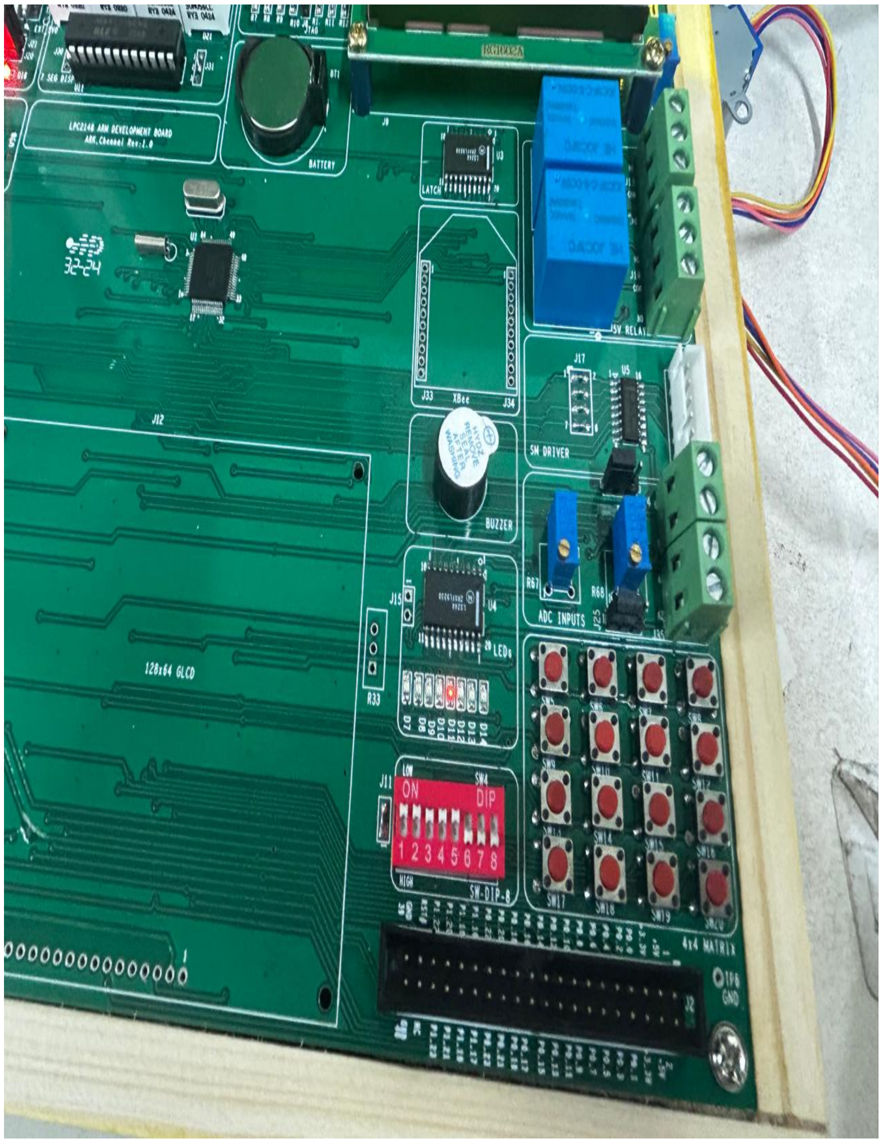

Figure 4: Addition result of 0b1111 and 0b0001 resulting in 0b00010000 showing the capacity for carry in addition

3.4 Multiplication of 4-bit numbers

This task requires us to take in two 4 bit numbers similar to the task for addition and display the result which is the multiplication of the two numbers. The result is usually a 8-bit number which is shown in the led array. **Note here also that the least significant bit is on the left side in both the input and output elements.**

3.4.1 Code

```
#include "LPC214x.H"                                /* LPC214x
   definitions */
#include "led.h"
#include "delay.h"


#define DIP_SW_D0 (1 << 16)      // P0.16
#define DIP_SW_D1 (1 << 17)      // P0.17
#define DIP_SW_D2 (1 << 18)      // P0.18
#define DIP_SW_D3 (1 << 19)      // P0.19

#define DIP_SW_D4 (1 << 22)      // P0.22
#define DIP_SW_D5 (1 << 23)      // P0.23
#define DIP_SW_D6 (1 << 24)      // P0.24
#define DIP_SW_D7 (1 << 25)      // P0.25

#define DIP_SW_DIR       IO1DIR
#define DIP_SW_PIN       IO1PIN

#define DIP_SW_DATA_MASK    (DIP_SW_D7 | DIP_SW_D6 | DIP_SW_D5
   | DIP_SW_D4 | DIP_SW_D3 | DIP_SW_D2 | DIP_SW_D1 | DIP_SW_D0)

void set_dipswitch_port_input( void )
{
    DIP_SW_DIR &= ~(DIP_SW_DATA_MASK);
}

unsigned long read_dip_switch( void )
{
    return DIP_SW_PIN;
}

int main (void)
{
    unsigned long sw_status;

    set_led_port_output();
    set_dipswitch_port_input();

    while(1)
    {
        sw_status = read_dip_switch();
        unsigned long input1 = (~(sw_status >> 16))&0x0F;
        unsigned long input2 = (~(sw_status >> 22))&0x0F;

        unsigned long sum = input1 * input2;

        unsigned long temp = (sum & 0xF0);
        temp = temp << 2;
        sum |= temp;

        sw_status = sum << 16;
        sw_status = ~sw_status;


        if(sw_status & DIP_SW_D0){ LED1_OFF;} else{ LED1_ON;}
                delay_mSec(10);
        if(sw_status & DIP_SW_D1){ LED2_OFF;} else{ LED2_ON;}z
                delay_mSec(10);
        if(sw_status & DIP_SW_D2){ LED3_OFF;} else{ LED3_ON;}
                delay_mSec(10);
        if(sw_status & DIP_SW_D3){ LED4_OFF;} else{ LED4_ON;}
                delay_mSec(10);

        if(sw_status & DIP_SW_D4){ LED5_OFF;} else{ LED5_ON;}
                delay_mSec(10);
        if(sw_status & DIP_SW_D5){ LED6_OFF;} else{ LED6_ON;}
                delay_mSec(10);
        if(sw_status & DIP_SW_D6){ LED7_OFF;} else{ LED7_ON;}
                delay_mSec(10);
        if(sw_status & DIP_SW_D7){ LED8_OFF;} else{ LED8_ON;}
                delay_mSec(10);

        delay_mSec(100);
    }

//    return 0;
}
```

3.4.2 Result

In the following image it can be seen that the output is the multiplication of the two 4 bit inputs.

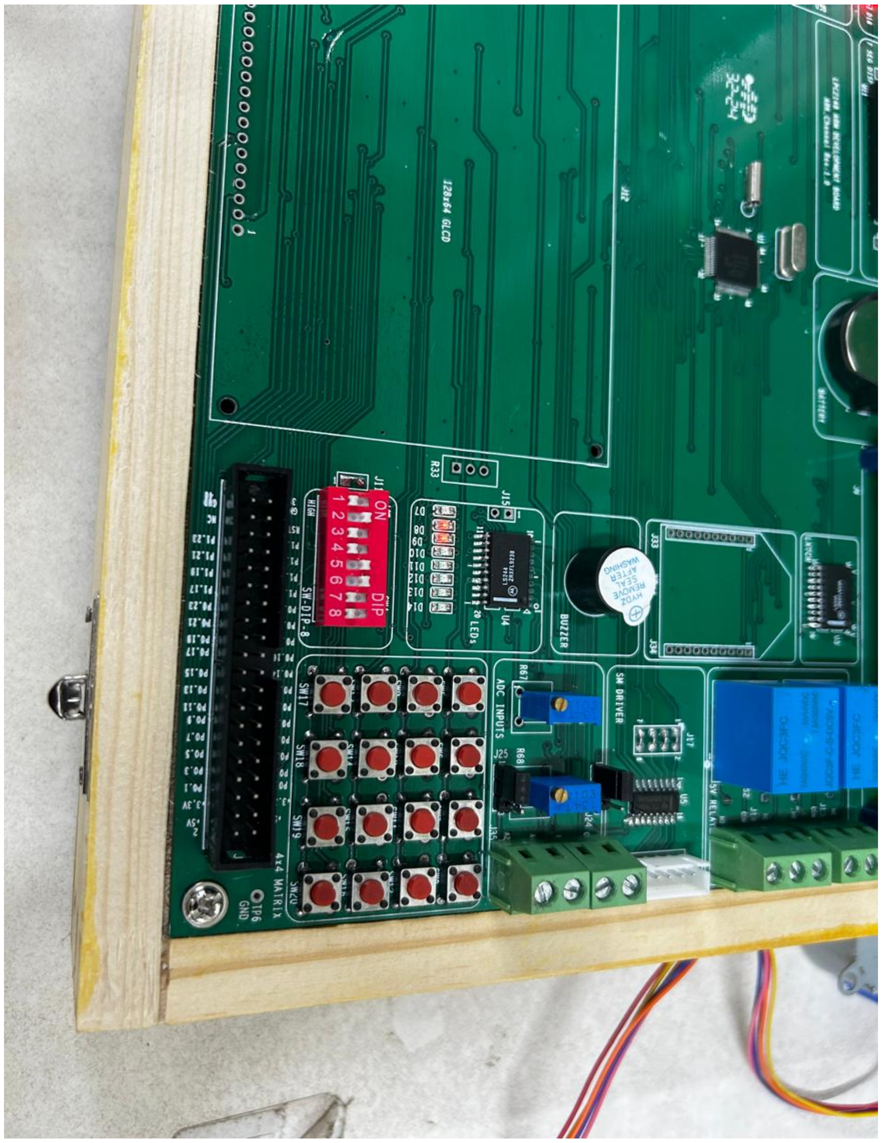

Figure 5: Multiplication of 0b0010(2) and 0b0011(3) resulting in 0b0110(6)

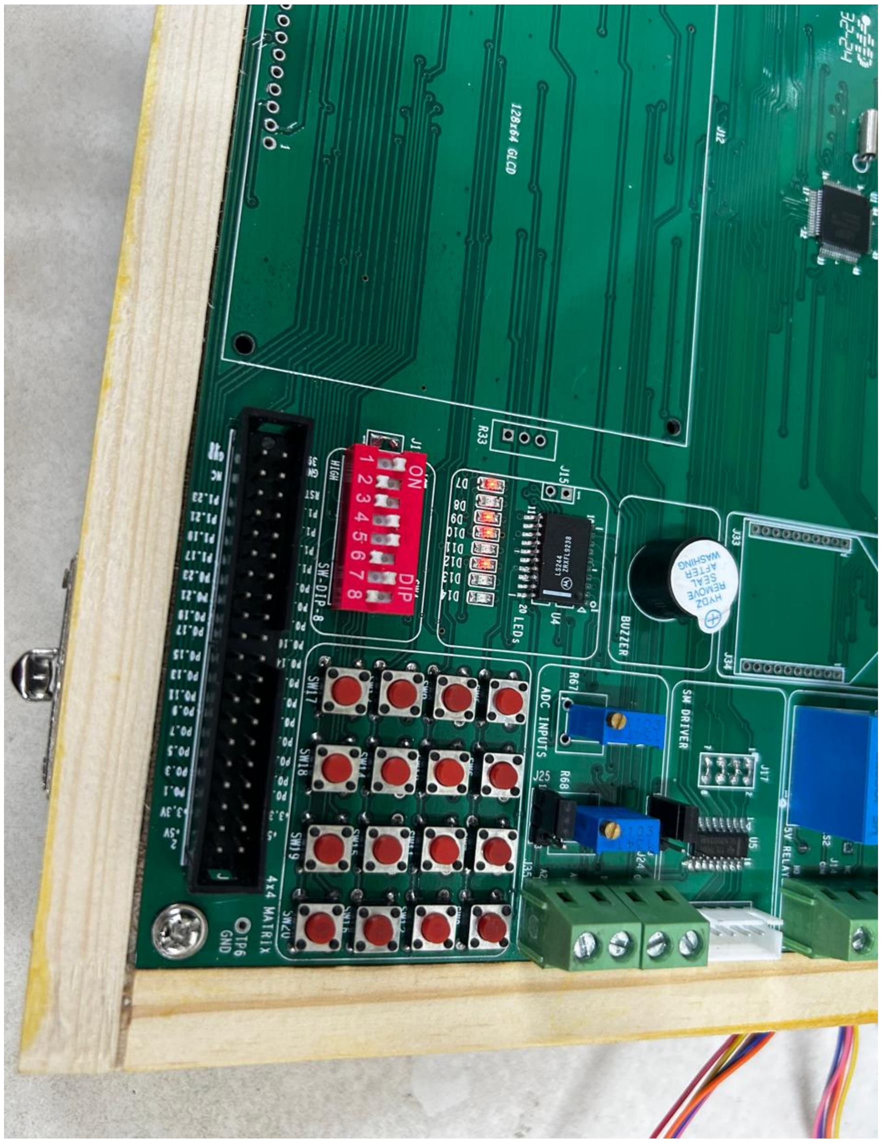

Figure 6: Multiplication of 0b1010(9) and 0b0101(5) resulting in 0b00101101(45)

3.5 Control of Stepper motor

This task of the experiment requires one to control the speed and direction of a stepper motor using C programming language. **The direction of rotation can be inverted by reversing the order of the standard step sequence in the program given. The speed of rotation of the stepper motor can be varied by changing the delay between the sucessive step commands given to the motor driver. This means that decresing the delay between two commands will result in the motor rotating faster while increasing the delay would reduce the speed of rotation of the motors.**

3.5.1 Code

```
#include <LPC214x.h>                        /* LPC21xx definitions */

void delay_mSec(int);

int main (void)
{
     int i;
//unsigned char steps[4] = {0x09, 0x0c, 0x06, 0x03};  // standard step sequence for stepper motor
unsigned char steps[4] = {0x03, 0x06, 0x0c, 0x09}; //Modified step sequence for reverse rotation of stepper motor
signed char x = 0;


    PINSEL0 = 0x0;                // Pin function Select -> P0.0 to P0.15 -> GPIO Port
    IO0DIR |= 0xF0;                         // Set stepper motor pins as output in IO0 port
    delay_mSec(10);

    while(1)
    {
    for(i=0;i<2500;i++)
    {
        IO0PIN = (steps[x++] << 4); //send the 4 bit step value to stepper motor lines connected to IO0 port
        if(x > 3)
            x = 0;

        delay_mSec(2);
    }
    }
    return 0;
}

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++);
         }
   }
}
```

3.5.2 Results

While it is not possible to show the motor spinning and its speed with images, the outcome of the experiment was demonstrated to the TA during the lab hours.

4 Conclusion

From this experiment, I gained familiarity with interfacing simple peripherals like leds and DIP switches with an ARM microcontroller based development board with C languagage programming. I also gained insights on the operational principles of a stepper motor.