EE2016: Microprocessers Lab - Assignment 8 Report

Aadarsh Ramachandiran (EE23B001), Aditya Kartik (EE23B003), Rohita Datta (EE23B065)

October 31, 2024

Abstract

This is a detailed report on an assignment to write assembly language programs for interfacing light emitting diodes to an ARM based microcontroller from NXP semiconductors. In particular, we have used the MCB2300 Keil-ARM board that is equipped with an LPC2378 microcontroller from NXP. The programs send signals from the microcontroller to the LEDs on the board. The report contains the details of the solution, observation and comments on the programming and debugging experience.

1 Introduction

This assignment involves designing and implementing programs for: (i) displaying a number using the LEDs on the ARM board and (ii) blinking an LED on the ARM board.

1.1 Ports and Register Settings

In LPC2378 (as shown in 1), there are a few registers whose settings determine the configuration of a port. These are (i) PINSEL10 (ii) FIO2DIR (iii) FIO2PIN (iv) PINSEL4. Further, these registers will have an address assciated with them.

Note- Just as we have addresses for external memory, there are also addresses associated with GPIO ports. In other words, we have what is called memory mapped input-output. General purpose registers such as r0, r1 are referred to by their names and do not have an address associated. The purpose of each port register is explained below.

In general, port pins have multiple functions (input/output, serial receive, serial transmit etc.) and hence we use PINSELx registers to specify the function. For example, when specific register bits are 00, a certain function is performed by the corresponding port pin while the same pins will perform a different function when the register bits are 01. In the MCB2300 board, eight LEDs are connected to the lowest 8 pins of Fast Input/Output Port 2 (FIO2). The pins of FIO2 are shared between I/O and the Embedded Trace Module (ETM) interface. So in order to use the LEDs, the ETM interface has to be disabled. This is done by setting bit 3 of PINSEL10 register to 0. Further, we have to select FIO2 lower 8-bits as I/O. Each pair of bits in PINSEL4 sets the function of one pin of FIO2. Setting the pair to 00 selects the pin for I/O function. To make all the eight pins as I/O, we have to set 8 pairs of bits in PINSEL4 to 0.

Every port has a direction register (to set the direction of each pin of the port). A setting of 1 in the direction register sets the corresponding pin of the port as output while a setting of 0 makes the pin as input. The specific register of interest to us is FIO2DIR. Once the port pins are selected and the direction is set, we have to specify the port register which is directly related to the (numerical) value to be displayed. This is accomplished with the FIO2PIN register. FIO2PIN is chosen (instead of FIO1PIN or FIO3PIN for instance) since we have chosen to use fast input/output port 2 above for the LED interfacing task. We refer to the LPC 2378 User Manual for additional details, if required.

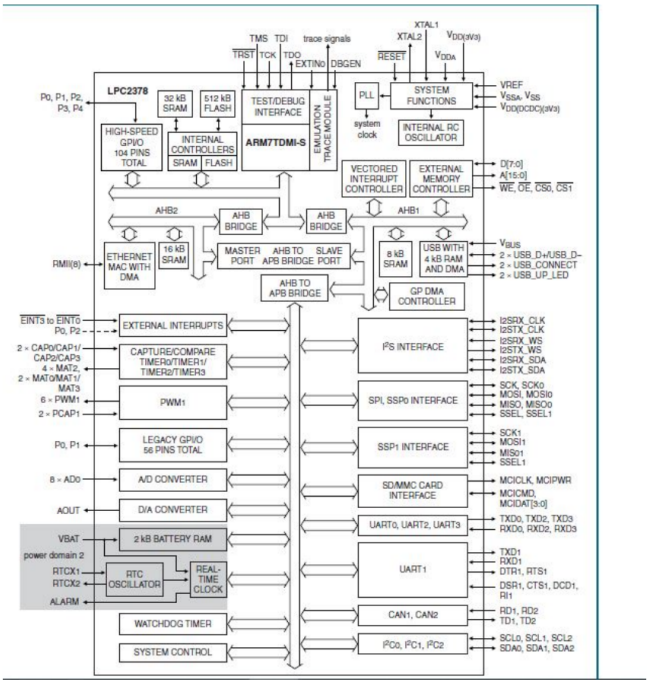

Figure 1: LPC2378

2 Design

We first attempt to write a program to display a number using LEDs on the ARM board. The task is divided into smaller subtasks for ease.

2.1 Displaying a number on LEDs in ARM Board

2.1.1 Specifying addresses for various registers

Description
We refer to the user manual to fetch the addresses for the registers to be used. As said before, we use the addresses to refer to them. For ease, we define Symbols as equivalent for the address values.
Code Section

```
    ; ARM Assebly code to interface LED with ARM microcontroller to
       display a number

; Defining the PORT addresses

    AREA LED, CODE, READONLY
    ENTRY
    EXPORT SystemInit
    EXPORT __main
; Fill corresponding addresses based on the LPC2378 manual
PINSEL10 EQU 0xE002C028
FIO2DIR EQU 0x3FFFC040
PINSEL4 EQU 0xE002C010
FIO2PIN EQU 0x3FFFC054
```

- AREA indicates the start of a new area (or segment) of code. LED is the name of the area (arbitrary choice), which can be referenced later. CODE specifies that the area contains executable code. READONLY indicates that the code in this area is not meant to be modified at runtime.

- ENTRY marks the entry point of the program.
- The EXPORT SystemInit line makes the SystemInit function accessible to other modules or files that may be linked during the build process. By exporting this function, it can be called from other parts of the program or from external libraries. Similarly for EXPORT __main.
- We fill in the addresses for the registers based on the LPC2378 manual using EQU (the manual can be found here)

2.1.2 Setting up the PINSEL10 register

Description
In the MCB2300 board, eight LEDs are connected to the lowest 8 pins of Fast Input/Output Port 2 (FIO2). The pins of FIO2 are shared between I/O and the Embedded Trace Module (ETM) interface. So in order to use the LEDs, the ETM interface has to be disabled. This is done by setting bit 3 of PINSEL10 register to 0. The initial setting for PINSEL10 can be seen by opening *Peripherals* and then *Pin Connect Block*.
Code Section

```
; Set up PINSEL10 by disabling ETM function in FIO2
SystemInit
LDR R0, =PINSEL10
LDR R2, [R0]
BIC R2, R2, #0x00000008
STR R2, [R0]
```

- The address of PINSEL10 register is loaded into register R0.
- LDR R2, [R0] loads the current value stored at the address in R0 (which now holds the address of PINSEL10) into register R2. Essentially, R2 now contains the configuration settings for the pins associated with PINSEL10.
- We use BIC to clear the bit 3 in R2 (to disable the ETM function). This performs a bitwise AND operation with 0x00000008 and thereby sets bit 3 to 0 in R2.
- The modified value from R2 is stored in address given by R0, which is the PINSEL10 register.

2.1.3 Setting up the PINSEL4 register

Description
We have to select FIO2 lower 8-bits as I/O. Each pair of bits in PINSEL4 sets the function of one pin of FIO2. Setting the pair to 00 selects the pin for I/O function. To make all the eight pins as I/O, we have to set 8 pairs of bits in PINSEL4 to 0.
Code Section

```
; Set lower 8-bits of FIO2 as I/O by making 8 -pairs of PINSEL4 to 00
   since we are using only 8 LEDs
LDR R0, =PINSEL4
LDR R2, [R0]
BIC R2, R2, #0x0000FFFF;
STR R2, [R0]
```

- The address of PINSEL4 register is loaded into register R0.
- LDR R2, [R0] loads the current value stored at the address in R0 (which now holds the address of PINSEL4) into register R2.
- We use BIC to set the last 16 bits in R2 to 0. This performs a bitwise AND operation of the existing value with 0x0000FFFF and stores it in R2.
- The modified value from R2 is stored in address given by R0, which is the PINSEL4 register.

2.1.4 Setting up the FIO2DIR register

Description
Every port has a direction register (to set the direction of each pin of the port). A setting of 1 in the direction register sets the corresponding pin of the port as output while a setting of 0 makes the pin as input. The specific register of interest to us is FIO2DIR. We set the last 8 bits of FIO2DIR as 1 as we are using 8 LEDs for output function.
Code Section

```
; Set up FIO2DIR for the output pins
LDR R0, =FIO2DIR
MOV R2, #0x000000FF
STR R2, [R0]
```

- The address of FIO2DIR register is loaded into register R0.
- The value in R2 register is set to the immediate value 0x000000FF by using MOV.
- The value from R2 is stored in address given by R0, which is the FIO2DIR register.

2.1.5 Displaying the number

Description
The FIO2PIN will dictate the functionality for the LEDs. For a given number desired to be displayed, we find its binary equivalent and set it to the FIO2PIN. A value of 1 indicates ON, 0 indicates OFF.
Code Section

```
__main
; Display a number on LED
    LDR R0, =FIO2PIN
    MOV R2, #0x0000000A ; Display number 10
    STR R2, [R0]
forever b forever
    END
```

- The address of FIO2PIN register is loaded into register R0.
- The value in R2 register is set to the immediate value 0x0000000A (can be set to any arbitrary number that can be stored using an 8 bit number (0-255) and rotation). This is the number we wish to display.
- The value from R2 is stored in address given by R0, which is the FIO2PIN register.
- forever and b forever effectively create an infinite loop as there are no conditions to exit the loop. This is to ensure the program keeps running.

2.2 Blinking an LED on an ARM board

2.2.1 Description

The `PINSEL10`, `PINSEL4`, and `FIO2DIR` registers are set up as before to allow output on 8 LEDs. To keep an LED indefinitely blinking, it is turned on and off repeatedly with a branch to a delay subroutine in between to create the blinking effect.

1. `LDR R0, =FIO2PIN`: Load the address of `FIO2PIN` into register `R0`.
2. To turn on the LED:
   - `MOV R2, #0x01`
     `STR R2, [R0]`
     (a) Load the value `0x01` into register `R2`.

(b) Store `R2` into the memory address in `R0` to activate the LED.

3. Create a delay:

   - `MOV R3, #0xF000`
     `BL delay1`
     (a) Load a delay count (e.g., `0xF000`) into register `R3`.
     (b) Branch to the `delay1` subroutine.

4. To turn off the LED:

   - `MOV R2, #0x00`
     `STR R2, [R0]`
     (a) Load the value `0x00` into `R2`.
     (b) Store `R2` into the address in `R0` to turn off the LED.

5. `BL delay1`: Create another delay by repeating the previous delay steps.

6. `B blinkloop`: Branch back to the start of the loop to repeat the blink cycle.

The Delay Subroutine works as follows:

```
delay1
SUBS R3, R3, #0x01
BNE delay1
BX LR
```

1. Decrement the delay counter in `R3` by `1`.
2. Check if the counter has reached zero:
   (a) If `R3` is not zero, branch back to repeat the delay loop.
3. Once `R3` reaches zero, return from the subroutine to resume blinking.

2.2.2 Code Section

```
; ARM Assebly code to interface LED with ARM microcontroller and make
   it blink

; Defining the PORT addresses

    AREA LED, CODE, READONLY
    ENTRY
    EXPORT SystemInit
    EXPORT __main
; Fill corresponding addresses based on the LPC2378 manual
PINSEL10 EQU 0xE002C028
FIO2DIR EQU 0x3FFFC040
PINSEL4 EQU 0xE002C010
FIO2PIN EQU 0x3FFFC054

; Set up PINSEL10 by disabling ETM function in FIO2
SystemInit

        LDR R0, =PINSEL10
        LDR R2, [R0]
        BIC R2, R2, #0x08
        STR R2, [R0]
```

```
        ; Set lower 8-bits of FIO2 as I/O by making 8 -pairs of
           PINSEL4 to 00 since we r using only 8 leds
        LDR R0, =PINSEL4
        LDR R2, [R0]
        MOV R2, #0x00
        STR R2, [R0]

        ; Set up FIO2DIR for the output pins
        LDR R0, =FIO2DIR
        MOV R2, #0xFF ; Set everything as output to allow any number
           input
        STR R2, [R0]

__main
blinkloop
; Make the LED blink
    LDR R0, =FIO2PIN
    MOV R2, #0x01
    STR R2, [R0]
        MOV R3, #0xF000
        BL delay1
        MOV R2, #0X00
        STR R2,[R0]
        MOV R3, #0xF000
        BL delay1
        B blinkloop

delay1
        SUBS R3, R3, #0x01
        BNE delay1
        BX LR

forever B forever
    END
```

3 Implementation

1. **Install Keil µVision**: Ensure Keil µVision is installed. If not, download it from the Keil website and install it.
2. **Open Keil µVision**: Launch the Keil µVision IDE.
3. **Create a New Project**: Go to **Project** → **New µVision Project**, choose a folder to save your project, and give it a name.
4. **Select the Target Device (LPC 2378)**: From the list of microcontrollers, choose **LPC 2378** from NXP.
5. **Copy the Startup File**: When asked, choose **Yes** to copy the **LPC2300.s** startup file, which is needed for the LPC2378.
6. **Configure the Target Device**: µVision automatically includes the correct startup code and headers after selecting the microcontroller, setting up the required compiler settings for ARM assembly.
7. **Add Source Files**: Go to **File** → **New** and write your ARM assembly or C code. Save it and add it to the project by right-clicking the project in the **Project window** → **Add Files to Group**.

8. **Set Microlib Option**: Right-click **Target 1**, select **Options for Target 'Target 1'**, and enable **Use Microlib** under the **Target** tab.

9. **Compile the Code**: Click **Project** → **Build Target** or press **F7** to compile the code.

10. **Configure Debugging Settings**: Go to **Debug** → **Settings** in the **ULINK2/ME ARM Debugger**, and set the **Max JTAG Clock** to 200 kHz for stable communication.

11. **Connect the ULINK2 Debugger**: Attach the ULINK2 debugger to the Keil-ARM board, and go to **Flash** → **Download** to load the code onto the microcontroller.

12. **Debug the Program**: Start debugging by going to **Debug** → **Start/Stop Debug Session**. Use **F10** (Step Over), **F11** (Step Into), or **F5** (Run) to execute the program. Use **View** → **Registers Window** to inspect the ARM CPU registers.

4 Observation

4.1 Displaying a number using the LEDs on the ARM Board

We observe the second and fourth LED lit on. We had input the number 10 and its binary equivalent is 1010. Thus, we observe the correct number 10 displayed through the LEDs as can be seen in 2

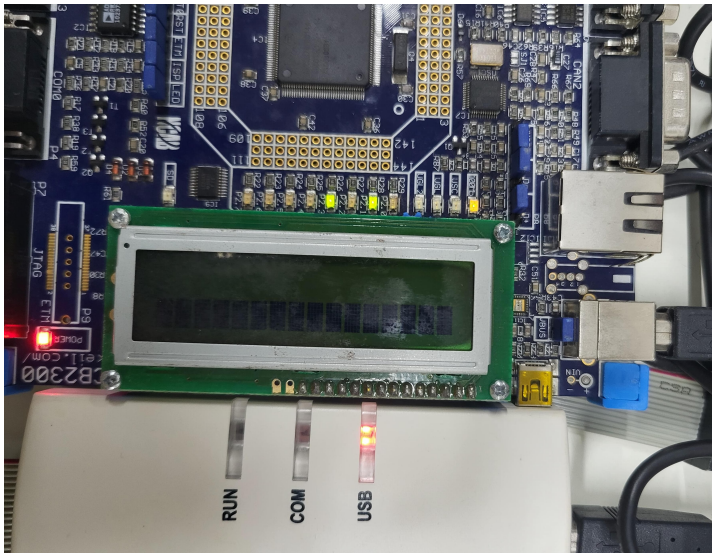

Figure 2: Displaying number 10 on ARM boar

4.2 Blinking an LED on the ARM Board

- We observe the LED that we had set blinking on the ARM board. The rate at which it blinks depends on the Counter Loop in the program. We modify the counter accordingly to increase or decrease speed for easy perception.

5 Conclusion

The experiment successfully demonstrated the application of ARM assembly language for interfacing LEDs with the MCB2300 Keil-ARM board, which is equipped with the LPC2378 microcontroller from NXP. This was achieved by effectively displaying numerical values and implementing a blinking LED function. All the code written for this assignment can also be found here.