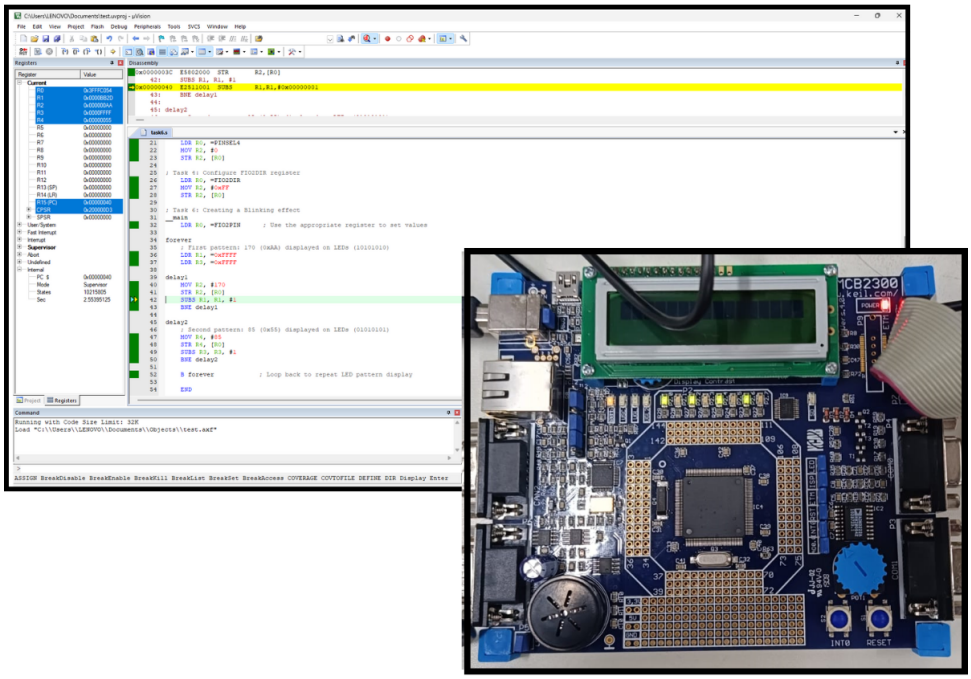

EE2016: Assignment-8

By Siva(EE23B151),
Bhavesh(EE23B017)
and Naveen(EE23B113)

Group 28

Date: 1/11/2024

Indian Institute of Technology Madras

Contents

1 Introduction

This experiment involves writing assembly language programs to interface light-emitting diodes (LEDs) with an ARM-based microcontroller. Using the MCB2300 Keil-ARM board, which is equipped with an NXP LPC2378 microcontroller, we will program LED signals directly from the microcontroller.

1.1 MCB2300 Keil-ARM Board

The **MCB2300** development board by Keil supports the NXP LPC23xx ARM microcontrollers, ideal for embedded application prototyping. Key features of the MCB2300 board include:

- **I/O Interfaces:** Multiple GPIO ports enable sensor, display, and peripheral integration.
- **Communication Support:** Ethernet, USB, CAN, and RS232 support makes the board suitable for networked applications.
- **Debugging:** Integrated JTAG for real-time debugging with Keil uVision.
- **LCD Display:** Onboard LCD offers real-time data display for immediate feedback.

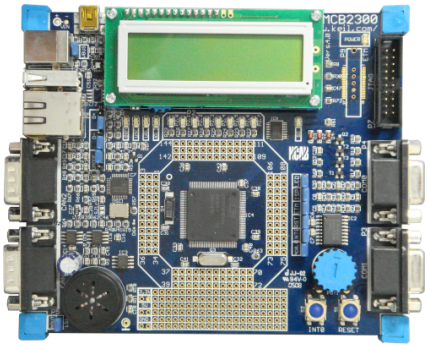

Figure 1: MCB2300 Keil-ARM Board

1.2 LPC2378 Microcontroller

The **LPC2378** microcontroller by NXP is a 32-bit ARM7TDMI-S-based MCU offering balanced power efficiency and performance, suited for control and communication-intensive applications. Key features of the LPC2378:

- **Core and Clock Speed:** ARM7TDMI-S core, up to 72 MHz, for real-time processing.
- **Memory:** 512 KB Flash and 58 KB SRAM for substantial data and program storage.
- **Peripherals and Interfaces:** USB 2.0, Ethernet MAC, CAN controllers, multiple UARTs, SPIs, and I2C interfaces for broad connectivity.
- **Timers and ADC:** Equipped with timers, PWM, and a 10-bit ADC for precision control.

Figure 2: LPC2378 Microcontroller

LPC23XX User Manuel provides essential information on pin addresses, register configurations, memory mapping, and peripheral functionalities necessary for programming and interfacing with the LPC23xx microcontroller series.

2 Tasks

2.1 Task 1: Specify the Port Register Addresses Using the LPC2378 Manual

The purpose of this task is to define specific register addresses based on the LPC2378 manual. Using the `EQU` directive, we assign addresses to the registers needed for interfacing with the LEDs. Each register address is used to set or control the function of specific pins on the microcontroller.

```
; Define register addresses based on LPC2378 manual
PINSEL10 EQU 0xE002C028  ; Pin Function Select Register 10
FIO2DIR  EQU 0x3FFFC040  ; Fast GPIO Port 2 Direction Register
PINSEL4  EQU 0xE002C010  ; Pin Function Select Register 4
FIO2PIN  EQU 0x3FFFC054  ; Fast GPIO Port 2 Pin Register
```

2.2 Task 2: Setting the PINSEL10 Register

To disable the ETM function on the FIO2 port, we use the `PINSEL10` register. The goal is to set the 3rd bit of the register to 0, which disables the ETM function while keeping the remaining bits unchanged.

```
ldr r0, =PINSEL10     ; Load the address of PINSEL10 into r0
ldr r1, [r0]          ; Load current value of PINSEL10 into r1
mov r2, #0xfffffffb   ; Prepare a mask with the 3rd bit as 0
and r1, r2            ; Clear the 3rd bit of r1 to disable ETM function
str r1, [r0]          ; Store the modified value back into PINSEL10
```

2.3 Task 3: Setting the PINSEL4 Register

We set the `PINSEL4` register to configure the pins as GPIO (General Purpose Input/Output) by setting all bits to 0. This ensures the direction control bit in the `FIO2DIR` register is effective.

```
ldr r0, =PINSEL4      ; Load the address of PINSEL4 into r0
mov r2, #0            ; Move 0 into r2 to configure pins as GPIO
str r2, [r0]          ; Store 0 in PINSEL4 to enable GPIO function
```

2.4 Task 4: Setting the FIO2DIR Register

In this task, we set the direction of the pins in the `FIO2DIR` register as output. By setting each bit in `FIO2DIR` to 1, we ensure all pins are configured for output.

```
ldr r0, =FIO2DIR      ; Load the address of FIO2DIR into r0
mov r2, #0xff         ; Set all bits to 1 for output direction
str r2, [r0]          ; Store value in FIO2DIR to set all pins as output
```

2.5 Task 5: Display a Number Using LEDs on the Board

To display a number on the LEDs, we use the `FIO2PIN` register, where setting specific values turns on corresponding LEDs. Here, we set up a sequence to display a chosen value (170) on the LEDs.

```
ldr r0, =FIO2PIN      ; Load the address of FIO2PIN into r0
mov r2, #170          ; Load the number to display on LEDs into r2
str r2, [r0]          ; Store this value in FIO2PIN to turn on specific LEDs
```

2.6 Task 6: Modify Task 5 to Make the LEDs Blink with a Delay

This task builds on Task 5 by making the LEDs blink. We alternate between two LED states with a delay loop between each state to create a blinking effect. Here, we display 170 for one delay period, then switch to 85 (the 2's complement of 170) for another delay period.

```
ldr r0, =FIO2PIN      ; Load the address of FIO2PIN into r0
forever
    ; First LED pattern
    mov r2, #170      ; Set the first LED pattern (170)
    str r2, [r0]      ; Display this pattern on LEDs
    ldr r1, =0xffff   ; Load delay counter value into r1
delay1
    subs r1, r1, #1   ; Decrement delay counter
    bne delay1        ; Repeat delay loop until counter is 0

    ; Second LED pattern
    mov r2, #85       ; Set the second LED pattern (85)
    str r2, [r0]      ; Display this pattern on LEDs
    ldr r1, =0xffff   ; Reload delay counter value into r1
delay2
    subs r1, r1, #1   ; Decrement delay counter
    bne delay2        ; Repeat delay loop until counter is 0

    b forever         ; Loop back to alternate LED patterns indefinitely
```

3 Program Files and Outputs

Code for Tasks 1-5: Contains the ARM code for the tasks 1 to 5.
Code for Task 6: Contains the ARM code for task 6.
Video for Task 6: Output for the modified code which adds delay loop to create a blinking effect.

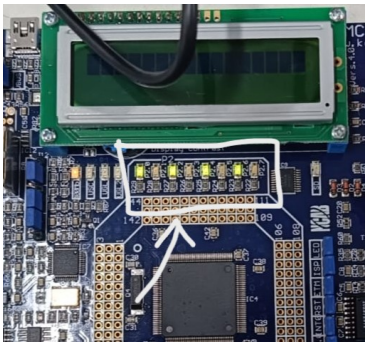

Figure 3: LED display $(170)_{10}$ in binary $(10101010)_2$