# LAB REPORT– EXPERIMENT 7

Introduction to the LPC2378 Microcontroller

BY

AJOE GEORGE (EE21B009)

AND

MANDLA SIVA MANOJ (EE21B083)

GROUP 3

NOVEMBER 5, 2022

# EE2016 - Microprocessor Theory + Lab

# Experiment-7(part-1): Introduction to the LPC2378 Microcontroller

## 1.The LPC2378 Microcontroller:

The LPC2378 microcontroller to be used in this experiment is from NXP semiconductors. LPC stands for Low Power Consumption. The LPC2378 is based on the ARM7TDMI-S CPU so we will discuss aspects of ARM below.

### 1.1 The ARM and ARM7TDMI-S:

In this experiment, we will use an ARM7-based processor, namely LPC2378. The ARM core here is called the ARM7TDMI-S (where T stands for Thumb Instructions, D for on-chip debugging support, M for multiplier, I for embedded In circuit emulation hardware and S for the synthesizable option based on RTL provided).

ARM7TDMI was the first of a range of processors introduced in 1995 by ARM. ARM7TDMI is the first core to include the Thumb instruction set. The Thumb instruction set contains 16-bit instructions. These serve as a 'compact shorthand' for a subset of the 32-bit instructions of the ARM. For 32-bit systems, ARM instructions are recommended for higher performance. Thumb instructions are appropriate for memory constrained systems.

## 2 Tasks for the Experiment:

In this experiment, we will write C programs for various tasks.

**Task 1:** Complete the following program to cause the LEDs on the ARM-board to blink.

```
#include "LPC23xx.h"
int main ()
{

while(1)
{
FIO3DIR=0x--; // LEDs are connected to lower 8 bits of
// Fast IO port 3 (FIO3PIN). Hence set the lower 8 bits
// as output in the corresponding direction
// register (FIO3DIR)

FIO3PIN=0x--; Port to which actual data is to be written
delay_code ; to be written
FIO3PIN=0x--
delay_code; to be written



}
return 0;
}
```

**Task 2**: In the previous task, we focussed on output on the LEDs. In this task, we will take input. In particular, read the settings of an 8-way DIP (Dual Inline Package) switch and display it on the LEDs. Use FIO4DIR AND FIO4PIN for data input.

**Task 3**: Write a C program to read a DIP switch, split into two nibbles (4 bits), multiply them and display the product on the LEDs.

## 3 Codes for Tasks:

**Task-1:** LED blink on ARM board

```
#include "LPC23xx.h"
int main ()
{ int i=1;
while(1)
{
FIO3DIR=0x00FF; // LEDs are connected to lower 8 bits of
// Fast IO port 3 (FIO3PIN). Hence set the lower 8 bits
// as output in the corresponding direction
// register (FIO3DIR)
FIO3PIN=0x00FF; //Port to which actual data is to be written
for ( i ; i<=10000 ; i++)
{
}
FIO3PIN=0x0000;
for ( i ; i<=20000 ; i++)
{
}
return 0;
}
}
```

**Task-2:** Reading data from input and showing output

```
#include "LPC23xx.h"
int main()
{
int a;
FIO3DIR = 0xFF;
FIO4DIR = 0x00;
while(1)
{
a = FIO4PIN;
FIO3PIN = a;
}
return 0;
}
```

**Task-3:** Showing product of two binary numbers using LED’s

```
#include "LPC23xx.h"
```

```
int main()
{
int a;
int highByte;
int lowByte;
FIO3DIR = 0xFF:
FIO4DIR = 0x00;
while(1)
{
a = FIO4PIN;
highByte = a & 0xF0;
highByte = highByte >> 4;
lowByte = a & 0x0F;
FIO3PIN = highByte * lowByte;
}
return 0;
}
```

## 4 Explanations:

**Task-1:**

- To change the LEDs' status, we use a for loop.
- FIO3PIN's value is high in the first FOR loop and low in the following one.
- The LEDs are turned on and off using this. The LEDs start to blink as a result.
- The LEDs can be seen blinking in the results video.

**Task-2:**

- We should receive the appropriate output for the input in the ViARM kit. To keep things simple, we merely provided the output as an input.
- The while loop in the code transfers data from the input to the CPU and then to the LEDs, determining whether to turn them on or off.
- First, we set the input on the FIR4DIR pin, which then sets the value on the LED-connected FIR3DIR pin.
- The processor processes the input value in this manner, and LEDs display the result.

**Task-3:**

- The while loop in the code is what allows us to multiply the given integers successively.
- The ViARM kit's first four switches function as a single 4-bit number. The ViARM kit's subsequent four switches function as the following four bit number.
- The numbers are assigned to the FIO3DIR and FIO4DIR, multiplied, and the results are displayed in the LEDs above them.

## 5 Results:

https://drive.google.com/drive/folders/12w4XtRM9fV9dfBe2b4lkf74nT0gR15VN?usp=share_link

# EE2016 - Microprocessor Theory + Lab

## Experiment-7(part-2): Introduction to the LPC2378 Microcontroller

In this experiment, we will interface a stepper motor to the LPC2378 microcontroller. We will begin by describing how a stepper motor works.

### 1 How does a stepper motor work ?

A stepper motor is a brushless electric motor that creates discrete rotation from the electrical input pulses. Such a motor is also called a fractional horsepower motor and the absence of a commutator (therefore wear and tear) adds to the robustness. While there are various types of stepper motors, the ones used in the lab experiment are called Permanent Magnet Stepper motors. These have multiple toothed electromagnets (called stator) arranged around a central gear-shaped permanent magnet (termed as rotor) as shown in Figure 1. The electromagnets are energized through the contact pins of the microcontroller. To make the motor shaft rotate, the stator is given an excitation, which attracts the rotor teeth (magnetically) to the stator electromagnet. When the rotor teeth are aligned with stator 1 (as shown in Figure 1), they are slightly offset from stator 2. When the next excitation is given, the rotor aligns with the stator 2. This process continues until the rotor makes one complete revolution. Each of these rotations is known as a “Step” with an integer number of steps making one full revolution.

### 2 Waveforms that can drive a stepper motor

The following are the possible drive control methods for a stepper motor.

• Wave Drive control
• Full Step Drive control
• Half Step Drive control

The lab experiment will be performed using the Wave Drive control of Stepper motor.

### 2.1 Wave Drive Control

In Wave Drive control as shown in Figure 2, only one winding is energized per sequence (if the sequence is denoted by ABA B and A is excited, the other three remain unexcited). The stepper motor has input pins termed as contact pins that allow current for excitation from the LPC2378 microcontroller of the stator coil windings. Pulsed waveforms in the correct sequence are used to create the electromagnetic fields required to drive the motor. The stepper motor driver circuit has two major tasks:

• To change the current and flux direction in the phase windings. This is achieved by taking a A center-tapped wire connection between each pair of phase windings is termed a Unipolar connection as shown in Figure 3 ( A, B, A and B correspond respectively to Black, Orange, Red and Brown leads on the right in Figure 3 ).

• To drive a controllable amount of current and adjust the drive sequence according to the speed Requirements.

### 3 The tasks to do in this experiment

**Task 1:** The first task in this experiment involves completion of the program given below to make the motor rotate in a specific direction at a fixed speed.

**Task 2:** Modify the program given in Task 1 to cause rotation of the stepper motor in both clockwise and anti-clockwise directions. That is, the motor should make a few rotations in clockwise direction, stop and then make a few rotations in the anti-clockwise direction.

**Task 3:** The program in Task 1 causes the motor to rotate at approximately 90 rpm. Write a program which will allow the motor to rotate at four different speeds. That is, it should rotate at (say) 30 rpm for one complete revolution, then at (say) 50 rpm for another revolution, then at 70 rpm and finally at 90 rpm for the last revolution.

## 4 Codes for Tasks:

**Task-1:** Rotating a motor

```
#include "LPC23xx.h"
void delay()
{
int i;
for(i=0;i<0xFFFF;i++)
{}
}
int main ()
{
IODIR0 = 0xFFFFFFFF;
while(1)
{
IOPIN0 = 0x00000280;
delay();
IOPIN0 = 0x00000240;
delay();
IOPIN0 = 0x00000140;
delay();
IOPIN0 = 0x00000180;
delay();
}
return 0;
}
```

**Task-2:** Motor rotates counterclockwise for fixed duration and changes directions and rotates in clockwise direction for the same duration.

```
#include "LPC23xx.h"
void delay()
{
int i;
for(i=0;i<0x0FFF;i++)
{}
}
int main ()
{
IODIR0 = 0xFFFFFFFF;
while(1)
{
for(int j=0;j<0x28;j++)
{ IOPIN0 = 0x00000280;
delay();
IOPIN0 = 0x00000240;
delay();
IOPIN0 = 0x00000140;
delay();
IOPIN0 = 0x00000180;
delay();
}
for(int j=0;j<0x12;j++)
{
delay();
}
for(int j=0;j<0x28;j++)
{
IOPIN0 = 0x00000180;
delay();
IOPIN0 = 0x00000140;
delay();
IOPIN0 = 0x00000240;
delay();
IOPIN0 = 0x00000280;
delay();
}
for(int j=0;j<0x12;j++)
{
delay();
}
}
return 0;
}
```

**Task-3:** Motor changing speeds

```
#include "LPC23xx.h"
void delay4()
{
int i;
for(i=0;i<0x02FF;i++)
{}
}
void delay3()
{
int i;
for(i=0;i<0x05FF;i++)
{}
}
void delay2()
{
```

```
int i;
for(i=0;i<0x0AFF;i++)
{}
}
void delay1()
{
int i;
for(i=0;i<0x0FFF;i++)
{}
}
int main ()
{
IODIR0 = 0xFFFFFFFF;
while(1)
{
for(int j=0;j<0x35;j++)
{ IOPIN0 = 0x00000280;
delay1();
IOPIN0 = 0x00000240;
delay1();
IOPIN0 = 0x00000140;
delay1();
IOPIN0 = 0x00000180;
delay1();
}
for(int j=0;j<0x10;j++)
{
delay1();
}
for(int j=0;j<0x35;j++)
{ IOPIN0 = 0x00000280;
delay2();
IOPIN0 = 0x00000240;
delay2();
IOPIN0 = 0x00000140;
delay2();
IOPIN0 = 0x00000180;
delay2();
}
for(int j=0;j<0x10;j++)
{
delay1();
}
for(int j=0;j<0x35;j++)
{ IOPIN0 = 0x00000280;
delay3();
IOPIN0 = 0x00000240;
delay3();
IOPIN0 = 0x00000140;
delay3();
IOPIN0 = 0x00000180;
delay3();
}
for(int j=0;j<0x10;j++)
{
delay1();
}
for(int j=0;j<0x35;j++)
{ IOPIN0 = 0x00000280;
delay4();
IOPIN0 = 0x00000240;
delay4();
IOPIN0 = 0x00000140;
delay4();
IOPIN0 = 0x00000180;
delay4();
}
for(int j=0;j<0x10;j++)
{
delay1();
}
}
return 0;
}
```

## 5 Explanations:

**Task-1:**

- Since permanent magnet stepper motors are what we use in the lab. By placing all of the magnets in the same direction, we can rotate the motor.
- In the code above, we replicate this action. To rotate the shaft, we simply assign all of the magnets in the motor to the same phase.
- As you can see from the code, we accomplished this using a while loop.

**Task-2:**

- By aligning each magnet in the motor in a single direction, we rotated the shaft in the previous task.
- By flipping the magnets in the motor around, we can change the shaft's direction.
- In the code, we have implemented this.
- After allowing the motor to rotate in one direction for a while, we used the microcontroller to shift the magnets' orientation.
- To rotate the magnets in this operation, FOR loops were employed inside of a while loop.
- Please take note that we also added a pause after each direction shift.Note we also included a delay between every directional change

**Task-3:**

- By adjusting the length of time the motor's magnets are driven, the speed of the shaft can be adjusted.
- Therefore, the speed is high if the magnets are turned off for a little period of time. The speed is low if they are turned off for a long time.
- We have used the code in this way.
- We have written it so that there are delays between each magnet in every FOR loop. The shaft's speed is now determined by the delay.
- The first FOR loop has a high initial delay of 0FF, which reduces speed when compared to subsequent loops.
- By reducing the delay in each loop from 0FF to 0AFF to 05FF to 03FF, we were able to boost the speed from 30 rpm to 50 rpm to 70 rpm to 90 rpm.
- Noting that the loop begins again whenever the shaft reaches the highest speed, or 90 rpm, the speed once more drops to 30 rpm.

## 6 Results:

https://drive.google.com/drive/folders/1r-vHytfFnP4WIAgp7IZmVj99mcuzrAN8?usp=share_link