University of California, Riverside

Lab 2 Report

Schematic Drawings and Software Installation

EE 128 Section 21

Name: Muneer Al Jufout

Student ID: 862410258

Lab Partners: Sameer Anjum

Lab Partner ID: 862422332

TA: Johnson Zhang

Introduction

The objective of this lab is to develop an understanding using the FRDM-K64F development board and to install the required software. We worked on a sample program and flashed the device. We also designed and implemented an 8-bit bidirectional binary counter and a one-hot rotator. The system also used DIP switches to control the counting and rotation direction.

Experiment System Specification

For part four of this lab, we used Port C for the 8-bit bidirectional counter (excluding pin 6), Port D for the 8-bit one-hot rotator, and Port A pins 1 and 2 as digital inputs for CNT_DIR and ROT_DIR from DIP switches.

Flowchart

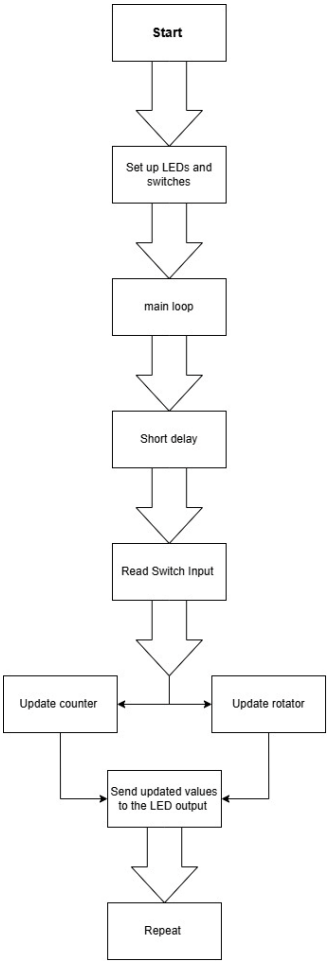
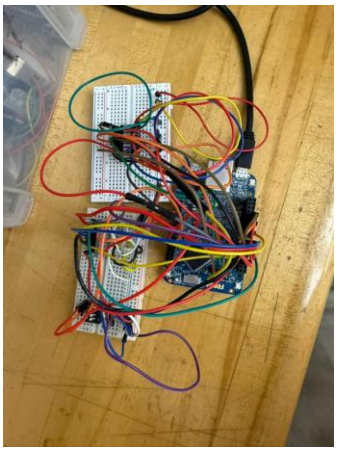
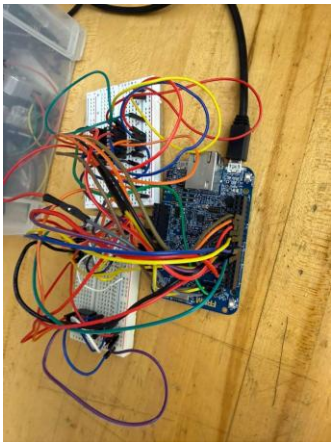


Photo of the setup

Two 8-bit LED bars connected to Port C (counter) and Port D (rotator).

DIP switch connected to Port A pins 1 and 2 for direction control.

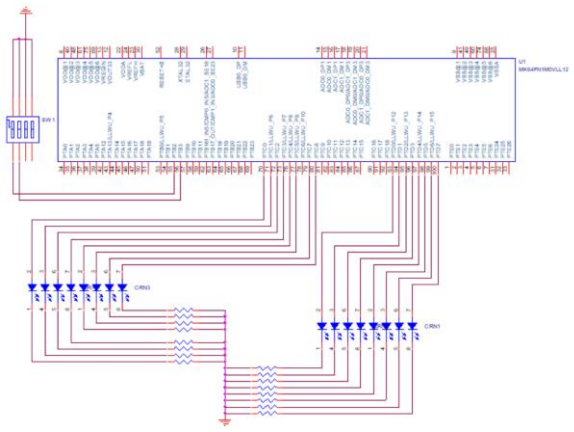




Schematic Diagram

This schematic included:

- MK64FN1M0VLL12 microcontroller
- DIP switch (SW_DIP_4)
- Two 8-bit LED bars
- Resistors (R)
- Ground (GND)



High-Level Description

The software uses Port C for the first 8-bit LED bars and Port D for the second 8-bit LED bar. It sets Port C and D pins as outputs, and Port A pins as inputs. It reads the DIP switch values, updates the counter and rotator, and lastly delays using a loop

Program Listing (Main Sections)

```
#include "fsl_device_registers.h"


void software_delay(unsigned long delay)
{
   while (delay > 0) delay--;
}


int main(void)
{
   SIM_SCGC5 |= SIM_SCGC5_PORTD_MASK;  /* Enable Port D Clock Gate Control */
   SIM_SCGC5 |= SIM_SCGC5_PORTB_MASK;  /* Enable Port B Clock Gate Control */
```

```
SIM_SCGC5 |= SIM_SCGC5_PORTC_MASK;  /* Enable Port C Clock Gate Control */

PORTD_GPCLR = 0x00FF0100;
PORTB_GPCLR = 0x000C0100;
PORTC_GPCLR = 0x00BF0100;          // PTC0–5,7
PORTC_PCR8 = PORT_PCR_MUX(1);      // Enable PTC8 as GPIO

GPIOB_PDDR = 0x00000000;
GPIOC_PDDR = 0x000001BF;           // Add bit 8 as output (0x100 + 0xBF)
GPIOD_PDDR = 0x000000FF;

unsigned long cycleDelay = 0x060000;

GPIOC_PDOR = 0x000001BF;           // Turn ON PC0–5,7,8
GPIOD_PDOR = 0x000000FF;

uint32_t CNT_DIR = 0;
uint32_t ROT_DIR = 1;
uint32_t base = 1;
uint32_t power = 1;
uint8_t rotatorVal = 0x01;

while (1) {
   uint32_t CNT_SWITCH = GPIOB_PDIR & 0x04;
   uint32_t ROT_SWITCH = GPIOB_PDIR & 0x08;

   if (CNT_SWITCH == 0x04) {
```

```
            if (CNT_DIR < 255) {
                CNT_DIR++;
            }
        } else {
            if (CNT_DIR > 1) {
                CNT_DIR--;
            } else {
                CNT_DIR = 1;
            }
        }

        if (ROT_SWITCH != 0) {              // rotate left
            if (rotatorVal != 0x80) {       // stop at last bit
                rotatorVal <<= 1;
            }
        } else {                            // rotate right
            if (rotatorVal != 0x01) {       // stop at first bit
                rotatorVal >>= 1;
            }
        }

        GPIOD_PDOR = CNT_DIR;
        GPIOC_PDOR = (rotatorVal & 0x3F) | ((rotatorVal & 0xC0) << 1);
    }

    return 0;
}
```

Technical Problems Encountered and Solutions

During the lab, we initially had trouble with the LED bar. At first, we assumed the issue was in our code because the LEDs were not lighting up as expected. However, after testing the connections carefully, we discovered that two of the LED pins were physically broken. Since the lab required only eight working LEDs, we decided to skip the broken first and last LEDs.

Questions

1. 1) What is the size of yo

ur .elf file for PART 4? 2) What do you think is the actual size of your
program code? (hint: check log messages in the Console tab when you build the project)

The size of the .elf file for Part 4 is 4252 bytes (decimal), which includes code and data sections. The actual size of the program code itself (text section) is 2068 bytes.

```
arm-none-eabi-size --format=berkeley "Lab_2.elf"
   text    data     bss     dec     hex filename
   2068     108    2076    4252    109c Lab_2.elf
Finished building: Lab_2.siz
```

2. When the software delay is removed, how fast can your program run -- in terms of the frequency in which the counter and rotator are updated? Justify your answer with experimental evidence –you may wish to use an oscilloscope to find it out. (e.g., take pictures of the oscilloscope output)

When we removed the delay from the program, the LEDs started updating extremely fast. It was not even noticeable with human eyes. It was hard for us to use the oscilloscope since we had never used but we got a number around 78 kHz, and that means the program was updating the counter and rotator about 78 times every second.

3. Consider the following C source code:
#define PTX_BASE (0x400FF040u)
#define PTX ((GPIO_Type *)PTX_BASE)
#define GPIO_PDOR_REG(PTX_BASE) ((base)->PDOR)
#define GPIO_PDDR_REG(PTX_BASE) ((base)->PDDR)
#define GPIOX_PDOR GPIO_PDOR_REG(PTX)

#define GPIOX_PDDR GPIO_PDDR_REG(PTX)

From this code, which I/O port of K64F is referred to by GPIOX_PDOR and GPIOX_PDDR? Why? (Hint: must be A, B, C, D, or E. Also, check MK64F12.h)

It refers to Port B of the K64F microcontroller, because the base address 0x400FF040 belongs to Port B.

Conclusion

In the laboratory, we were able to gain experience in using the FRDM-K64F microcontroller. We correctly installed and configured the software tools, created and constructed a basic I/O program, and performed circuit connections via LED bars and DIP switches. We implemented an 8-bit counter and a one-hot rotator, switched their direction through switches, and observed their functionality through the LED outputs.