EE128
Software Configuration and Simple Parallel I/O

Lab Report #2

Ana Cardenas Beltran
Alison Gonzales

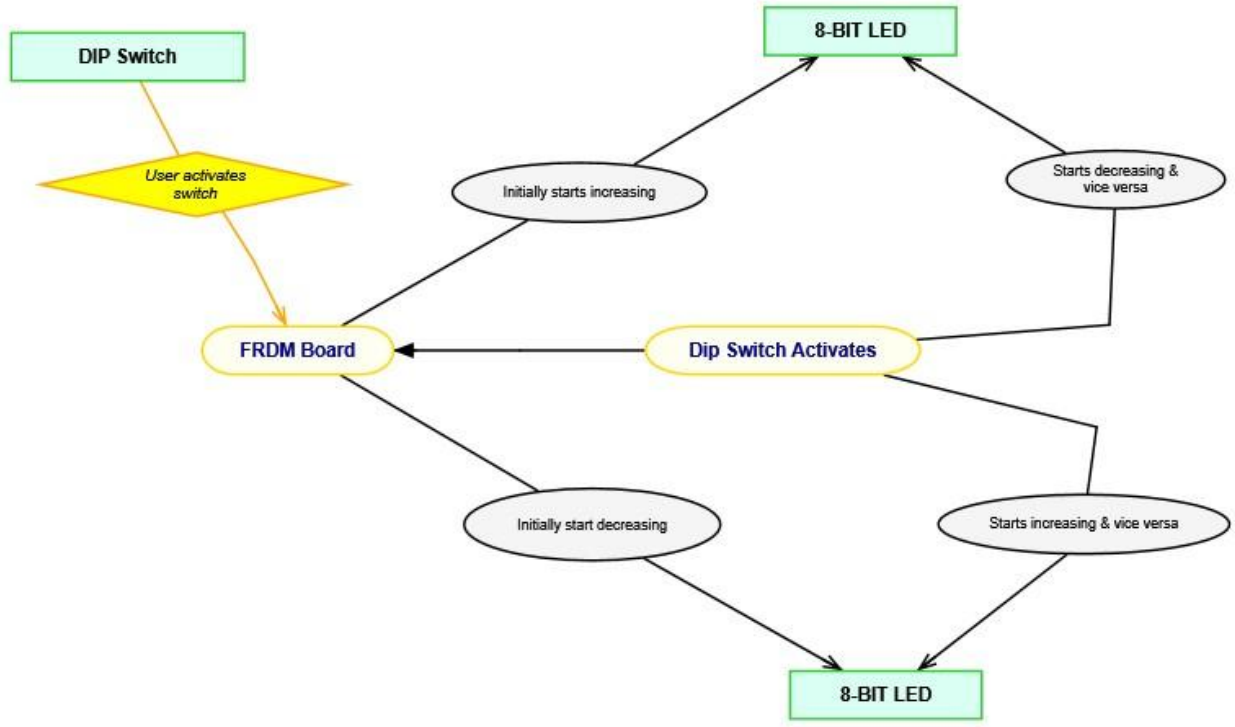
Abstract

The objective of this lab is to gain experience on the Kinetis K64F (Arm Cortex-M4) program. Another objective is to utilize both hardware and software techniques in building microcontroller-based digital applications. Students accomplished the necessary skills to use the FRDM KG4F development board under a Windows work environment and used Digital Multi-Meter to train students for power quality management while building their corresponding circuits.

Experiment System Specification

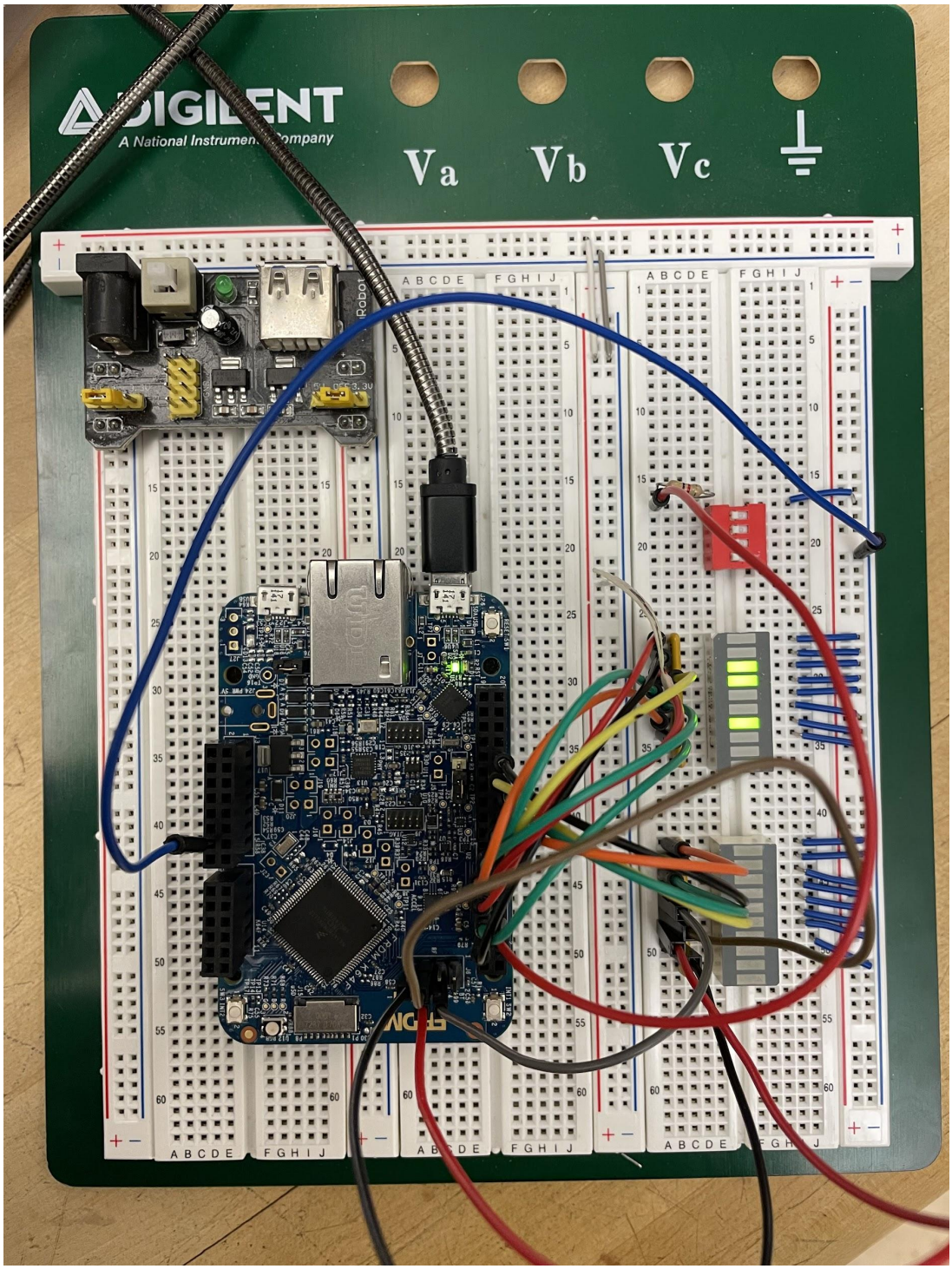
In this lab, Freescale Kinetis Design Studio (KDS) 3.2.0 IDE was used as the main work environment for programming the Kinetis K64F development board to create a digital system. Multiple circuit elements: 8-bit LED bars, 4-bit DIP switch (piano type), and 470 ohms, 10-pin bussed resistor network (SIP) were used to test the FRDM board. Through the development board and circuit elements, students were able to use the Kinetis Design Studio to write a sample program and check if the board is working properly. Overall, students tested and configured their own PCs to run the FRDM K64F board for later technical development.

Flowchart Diagram



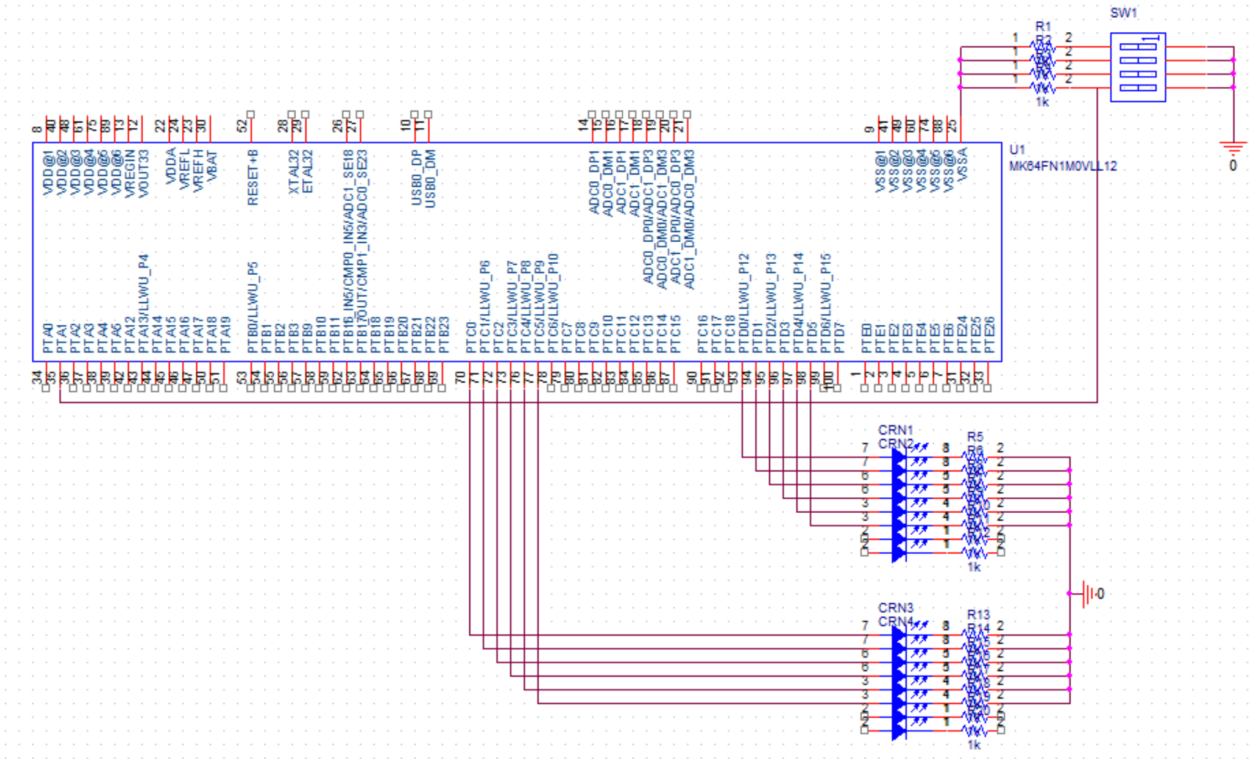
Initially, the 2 8-Bit LEDs start decreasing or increasing. When the user activates the dip switch, the 2 8-bit will switch it's counting ability to either decreasing or increasing.

Boards & Circuits



Power Supply was not used in this circuit.

Hardware Design



Software Design

Software Description: This software counts upward or downward in binary and rotates to the left or to the right depending on whether the Port A Pin 1 reads a 0 or a 1. We used LEDs to display the binary digits. We used Port C to output the binary number increasing and decreasing and used Port D to rotate the binary number right or left.

```
/*
*/
#include "fsl_device_registers.h"
void software_delay(unsigned long delay)
{
	while (delay > 0) delay--;
}

int main(void)
{
	//Schematic of Board on page 19
	//Enable Ports as Clock Gate Control
	//Need to enable clock Gates in order to work in every module in the MCU
	SIM_SCGC5 |= SIM_SCGC5_PORTA_MASK; /*Enable Port A Clock Gate Control*/
	SIM_SCGC5 |= SIM_SCGC5_PORTC_MASK; /*Enable Port C Clock Gate Control*/
	SIM_SCGC5 |= SIM_SCGC5_PORTD_MASK; //Enable Port D Clock Gate Control
```

```
	PORTD_GPCLR = 0x00FF0100;	//Configure Port D Pins 0-7 for GPIO
	PORTC_GPCLR = 0x01BF0100;	//Configure Port C Pins 0-5 and 7-8 for GPIO
	PORTA_GPCLR = 0X00010100;	//Configure Port A Pin 1 GPIO

	GPIOA_PDDR = 0x00000000;	//Configure Pin 1 for Input; Input = 0; Output = 1;
	GPIOD_PDDR = 0x000000FF;	//Configure Port D Pins 0-7 for Output;
	GPIOC_PDDR = 0x000001BF;	//Configure Port C Pins 0-5 and 7-8 for Output;  //0001 1011
1111 <- PINS 0-5, 7-8


	GPIOC_PDOR = 0x00000000;//Initialize Port C to 0;
	GPIOD_PDOR = 0x00000001;//Initialize Port D such that only 1 bit but is ON;



	unsigned long Delay = 0x100000;
	unsigned long val = 0x00;


	while (1) {

				software_delay(Delay);


				long CNT_DIR = GPIOA_PDIR; //Read Port A
				long ROT_DIR = GPIOA_PDIR;

				val = GPIOD_PDOR;

				if (CNT_DIR == 0x00) {
					GPIOC_PDOR = GPIOC_PDOR + 1;//increment PORT C
					if(val == 0x80){
						GPIOD_PDOR = 0x01;
					}else{
						GPIOD_PDOR = GPIOD_PDOR << 1; //left-rotate Port D
					}
				}
				else {
					GPIOC_PDOR = GPIOC_PDOR - 1; //decrement PORT C
					if(val == 0x01){
						GPIOD_PDOR = 0x80;
					}else{
						GPIOD_PDOR = GPIOD_PDOR >> 1; //right-rotate Port D
					}
				}
	}
	return 0;
}
}
```

Technical Problems

The first technical problem we encountered was with the micro-usb we were given. The USB Port wasn't detecting the Freedom Board even though my computer detected the board was connected. We solved this issue by using a different Micro-USB cable and copying a different .bin file to the debugger folder that pops up when we connect the Freedom Board. Another problem we encountered was with wiring. The pins on the Freedom Board are out of order so it was difficult to wire the board without making errors.

Questions

1. *What is the size of your .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 our .elf file is 1,980 KB. We think the actual size of our program code is about 1KB or a number much smaller.
2. *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)*
   The code refers to Port B. We know this because in the MK64F12.h file, the Port B base address is defined as 0x400FF040u.

Conclusion

In this lab, we gained more familiarity with the Kinetis K64F development board and how to program it. We learned and tested different GPIO syntax and how to configure multiple pins for output and input. We also used the following to program the 8-bit LEDs to count upward and downward and rotate left and right. Lastly, we learned how to start designing our own schematic from a high-level description.

Alison contributed in programming the code for the dip switch and 8-bit counter circuit. She also drew the schematics and flowcharts for the planning of the circuit shown in this lab. Ana initialized the hardware set-up, design, and also worked on the code.