2DX4: Microprocessor System
Lab 4

Instructor: Drs.Boursalie, Doyle, Haddara

Lab TAs: Fatemeh Derakshani (derakhsf), Han Zhou (zhouh115)

Junbo Wang – L01 – wangj430

Yichen Lu – L01 – luy191

As a future member of the engineering profession, the student is responsible for performing the required work in an honest manner, without plagiarism and cheating. Submitting this work with my name and student number is a statement and understanding that this work is our own and adheres to the Academic Integrity Policy of McMaster University and the Code of Conduct of the Professional Engineers of Ontario. Submitted by **[Junbo Wang wangj430 400249823]**

As a future member of the engineering profession, the student is responsible for performing the required work in an honest manner, without plagiarism and cheating. Submitting this work with my name and student number is a statement and understanding that this work is our own and adheres to the Academic Integrity Policy of McMaster University and the Code of Conduct of the Professional Engineers of Ontario. Submitted by **[Yichen Lu luy191 400247938]**

1. Purpose

The purpose of this lab is to use embedded properties and timing using C language to cross-assembled for ARM. We will use LEDs and stepper motors in order to illustrate the timing.

2. BackGround

There are two milestones in this lab. In milestone 1, we will create a C program that varies the duty-cycle of your square wave and perform it on the RGB LED. Studio 4A introduces how to change the duty cycle and how the pulse width modulation changes the brightness of an LED. In milestone 2, we want to control the rotation of the stepper motor by writing a C program. We learn three different stepping methods on the stepper motor and the calculation of full rotation steps through studio 4B.

3. Method

Milestone 1

We need to write a C program to show this milestone. We create a function called *IntensitySteps* and use the for loop to execute the duty-cycle 10 times which steps from 0% to 100%. Also, we need to write another for loop for duty cycle decrease from 100% to 0%. Inside these two for loops, we still create a for loop to repeat each duty cycle 10 times in order to observe the effect of duty cycle on LED. After we finished the code and ran it , the brightness of the RGB LED changed from weak to strong and strong to weak.

Milestone 2

We write a C program to control the spinning of the stepper motor. First, we calculated the steps for the motor to rotate 360 degrees. We used gear ratio 64 times 32 rounds for 360 degrees rotor rotation and divided 4 due to four input ports from PortM, so we got 512 steps. From the studio codes, we know that the stepping method is full step which is two phases at a time and the motor performs a counterclockwise rotation. We created two functions for clockwise and counterclockwise. For the clockwise function, we switched the sequence of GPIO_PORTM_DATA of counterclockwise codes, so the motor rotated clockwise first. Then we put counterclockwise code in its function. Therefore, we show one full revolution clockwise followed by one full revolution counterclockwise.

4. Observation and conclusion

In milestone 1, we write for loops in C for the new function *IntensitySteps*. After we finished the code and ran it, the RGB LED on the breadboard started shining and the brightness went from weak to strong and strong to weak. Also, the scope on Waveform generated square waves for each duty cycle. The greater duty cycle led to a wider width on square waves, vice versa.

In milestone 2, we created two functions to control the rotation of the stepper motor. When we ran the code, the motor started spinning in one full clockwise direction, and then rotated in one full counterclockwise direction.

Code Appendix

Lab4_Milestone1

```
// 2DX4_Knowledge_Thread_3_Session 0
// This program illustrates the use of SysTick in the C language.
// Note the library headers asscoaited are PLL.h and SysTick.h,
// which define functions and variables used in PLL.c and SysTick.c.


// Name: Yichen Lu Junbo Wang
// Student id: 400247938
// Date: Feb, 14th, 2022


#include <stdint.h>
#include "tm4c1294ncpdt.h"
#include "PLL.h"
#include "SysTick.h"


void PortN_Init(void){
    //Use PortN onboard LED
    SYSCTL_RCGCGPIO_R |= SYSCTL_RCGCGPIO_R12;               // activate clock
for Port N
    while((SYSCTL_PRGPIO_R&SYSCTL_PRGPIO_R12) == 0){};  // allow time for clock
to stabilize
    GPIO_PORTN_DIR_R |= 0x05;                                       // make PN0
out (PN0 built-in LED1)
  GPIO_PORTN_AFSEL_R &= ~0x05;                                  // disable alt
funct on PN0
  GPIO_PORTN_DEN_R |= 0x05;                                     // enable
digital I/O on PN0

// configure PN1 as GPIO
  //GPIO_PORTN_PCTL_R = (GPIO_PORTN_PCTL_R&0xFFFFFF0F)+0x00000000;
  GPIO_PORTN_AMSEL_R &= ~0x05;                                  // disable
analog functionality on PN0

    GPIO_PORTN_DATA_R ^= 0b00000001;                                //hello
world!
    SysTick_Wait10ms(10);
//.1s delay
    GPIO_PORTN_DATA_R ^= 0b00000001;
    return;
}
```

```c
void DutyCycle_Percent(uint8_t duty){
        float percent;
        percent = ((float)duty*1000)/(255);
        int percent_int;
        percent_int = (int)percent;
        GPIO_PORTN_DATA_R ^= 0b00000100;
        SysTick_Wait10ms(percent_int);  //SysTick was changed to 0.01 ms in
order for this to work
        GPIO_PORTN_DATA_R ^= 0b00000100;
        SysTick_Wait10ms(1000-percent_int);
}


void IntensitySteps(){
      int i;
      int j;
      uint8_t duty_int = 0; // starting at 0 and finally reaching 255 to
approximately represent the requirement of 0% ~ 100%
        for (i=0; i<10; i++){
              duty_int += 25; // increase 10%
                for (j=0; j<10; j++){
                        DutyCycle_Percent(duty_int);
                }
        }
        for (i=10; i>0; i--){
              duty_int -= 25; // decrease 10%
                for (j=0; j<10; j++){
                        DutyCycle_Percent(duty_int);
                }
        }
}

int main(void){

    PLL_Init();
// Default Set System Clock to 120MHz
    SysTick_Init();
// Initialize SysTick configuration
    PortN_Init();
// Initialize Port N

    while(1){
        IntensitySteps();
    }
}
```

Systick

```c
// SysTick.c
// Runs on TM4C1294
// Provide functions that initialize the SysTick module, wait at least a
// designated number of clock cycles, and wait approximately a multiple
// of 10 milliseconds using busy wait.  After a power-on-reset, the
// TM4C1294 gets its clock from the 16 MHz precision internal oscillator,
// which can vary by +/- 1% at room temperature and +/- 3% across all
// temperature ranges.  If you are using this module, you may need more
// precise timing, so it is assumed that you are using the PLL to set
// the system clock to 120 MHz.  This matters for the function
// SysTick_Wait10ms(), which will wait longer than 10 ms if the clock is
// slower.
// Daniel Valvano
// April 3, 2014

/* This example accompanies the books
   "Embedded Systems: Introduction to ARM Cortex M Microcontrollers",
   ISBN: 978-1469998749, Jonathan Valvano, copyright (c) 2014
   Volume 1, Program 4.7

   "Embedded Systems: Real Time Interfacing to ARM Cortex M Microcontrollers",
   ISBN: 978-1463590154, Jonathan Valvano, copyright (c) 2014
   Program 2.11, Section 2.6

 Copyright 2014 by Jonathan W. Valvano, valvano@mail.utexas.edu
    You may use, edit, run or distribute this file
    as long as the above copyright notice remains
 THIS SOFTWARE IS PROVIDED "AS IS".  NO WARRANTIES, WHETHER EXPRESS, IMPLIED
 OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF
 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE.
 VALVANO SHALL NOT, IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL,
 OR CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.
 For more information about my classes, my research, and my books, see
 http://users.ece.utexas.edu/~valvano/
 */

#include <stdint.h>
#include "SysTick.h"
#define NVIC_ST_CTRL_R          (*((volatile uint32_t *)0xE000E010))
#define NVIC_ST_RELOAD_R        (*((volatile uint32_t *)0xE000E014))
#define NVIC_ST_CURRENT_R       (*((volatile uint32_t *)0xE000E018))
#define NVIC_ST_CTRL_COUNT      0x00010000  // Count flag
#define NVIC_ST_CTRL_CLK_SRC    0x00000004  // Clock Source
#define NVIC_ST_CTRL_INTEN      0x00000002  // Interrupt enable
#define NVIC_ST_CTRL_ENABLE     0x00000001  // Counter mode
#define NVIC_ST_RELOAD_M        0x00FFFFFF  // Counter load value

// Initialize SysTick with busy wait running at bus clock.
```

```c
void SysTick_Init(void){
  NVIC_ST_CTRL_R = 0;                   // disable SysTick during setup
  NVIC_ST_RELOAD_R = NVIC_ST_RELOAD_M;  // maximum reload value
  NVIC_ST_CURRENT_R = 0;                // any write to current clears it
                                        // enable SysTick with core clock
  NVIC_ST_CTRL_R = NVIC_ST_CTRL_ENABLE+NVIC_ST_CTRL_CLK_SRC;
}
// Time delay using busy wait.
// The delay parameter is in units of the core clock. (units of 8.333 nsec for
120 MHz clock)
void SysTick_Wait(uint32_t delay){
  volatile uint32_t elapsedTime;
  uint32_t startTime = NVIC_ST_CURRENT_R;
  do{
    elapsedTime = (startTime-NVIC_ST_CURRENT_R)&0x00FFFFFF;
  }
  while(elapsedTime <= delay);
}
// Time delay 10ms using busy wait.
// This assumes 120 MHz system clock.
void SysTick_Wait10ms(uint32_t delay){
  uint32_t i;
  for(i=0; i<delay; i++){
    SysTick_Wait(120000);  // wait 10ms (assumes 120 MHz clock)
  }
}

// Time delay 10ns using busy wait.
// This assumes 120 MHz system clock.
void SysTick_Wait10us(uint32_t delay){
  uint32_t i;
  for(i=0; i<delay; i++){
    SysTick_Wait(1200);  // change this to wait 10us (assumes 120 MHz clock)
  }
}
```

Lab4_Milestone2

```c
// 2DX4_Knowledge_Thread_3_Session_1
// This program illustrates the use of SysTick in the C language.
// Note the library headers asscoaited are PLL.h and SysTick.h,
// which define functions and variables used in PLL.c and SysTick.c.
// This program uses code directly from your course textbook.

//  Written by Yichen Lu Junbo Wang
//  Feb 14, 2022
//  Student Number:400247938


#include <stdint.h>
#include "tm4c1294ncpdt.h"
#include "PLL.h"
#include "SysTick.h"



void PortM_Init(void){
    //Use PortM pins for output
    SYSCTL_RCGCGPIO_R |= SYSCTL_RCGCGPIO_R11;                // activate clock
for Port N
    while((SYSCTL_PRGPIO_R&SYSCTL_PRGPIO_R11) == 0){};  // allow time for clock
to stabilize
    GPIO_PORTM_DIR_R |= 0xFF;                                        // make PN0
out (PN0 built-in LED1)
  GPIO_PORTM_AFSEL_R &= ~0xFF;                                     // disable alt
funct on PN0
  GPIO_PORTM_DEN_R |= 0xFF;                                        // enable
digital I/O on PN0

// configure PN1 as GPIO
  //GPIO_PORTM_PCTL_R = (GPIO_PORTM_PCTL_R&0xFFFFFF0F)+0x00000000;
  GPIO_PORTM_AMSEL_R &= ~0xFF;                                     // disable
analog functionality on PN0
    return;
}
void spin_clock(){
    for(int i=0; i<512; i++){
        GPIO_PORTM_DATA_R = 0b00001001;
        SysTick_Wait10ms(1);
        GPIO_PORTM_DATA_R = 0b00000011;
        SysTick_Wait10ms(1);
        GPIO_PORTM_DATA_R = 0b00000110;
        SysTick_Wait10ms(1);
        GPIO_PORTM_DATA_R = 0b00001100;
        SysTick_Wait10ms(1);
    }
```

```
}
void spin_counterclock(){
    for(int i=0; i<512; i++){
        GPIO_PORTM_DATA_R = 0b00001100;
        SysTick_Wait10ms(1);
        GPIO_PORTM_DATA_R = 0b00000110;
        SysTick_Wait10ms(1);
        GPIO_PORTM_DATA_R = 0b00000011;
        SysTick_Wait10ms(1);
        GPIO_PORTM_DATA_R = 0b00001001;
        SysTick_Wait10ms(1);
    }
}


int main(void){
    PLL_Init();
// Default Set System Clock to 120MHz
    SysTick_Init();
// Initialize SysTick configuration
    PortM_Init();
    spin_clock();
    spin_counterclock();
    return 0;
}
```