EE445L – Lab 10: DC Motor Control

Michael Park, Jack Zhao, & Corey Cormier
04/25/16

1.0 OBJECTIVES
Requirements document
1. Overview
1.1. Objectives: Why are we doing this project? What is the purpose?
The objectives of this project are to design, build and test a brushed DC motor controller. The motor should spin at a constant speed and the operator can specify the desired set point. Educationally, students are learning how to interface a DC motor, how to measure speed using input capture, and how to implement a digital controller running in the background.

1.2. Process: How will the project be developed?
The project will be developed using the EK-TM4C123GXL or EK-TM4C1294XL LaunchPad. There will be two switches that the operator will use to specify the desired speed of the motor. The system will be built on a solderless breadboard and run on the usual USB power. The system may use the on board switches or off-board switches. A hardware/software interface will be designed that allows software to control the DC motor. There will be at least five hardware/software modules: tachometer input, switch input, motor output, LCD output, and the motor controller. The process will be to design and test each module independently from the other modules. After each module is tested, the system will be built and tested.

1.3. Roles and Responsibilities: Who will do what? Who are the clients?
EE445L students are the engineers and the TA is the client. Michael will build and test the hardware for the sensor system, the actuator, and switch input. Jack will write the software and test the sensor system, the actuator, and switch input. Both Michael and Jack will work on the controller.

1.4. Interactions with Existing Systems: How will it fit in?
The system will use the microcontroller board, a solderless breadboard, and the DC motor shown in Figure 4.1. The wiring connector for the DC motor is described in the PCB Artist file **Lab10.sch**. It will be powered using the USB cable. You may use a +5V power from the lab bench, but please do not power the motor with a voltage above +5V.

1.5. Terminology: Define terms used in the document.
Integral controller – In PID or PI controller, integral controller accounts for past values of an error.
PWM – pulse-width-modulation. PWM creates a digital output wave of fixed frequency, but allow the microcontroller to vary its duty cycle.
Board support package – A set of software routines that abstract the I/O hardware such that the same high-level code can run on multiple computers.
Back emf – refers to the voltage that occurs in electric motors where there is relative motion between the armature of the motor and the magnetic field from the motor's field magnets or windings.
Torque – available force times distance the stepper motor can provide at some speed (rate in rotations per minutes i.e. RPM)
Time constant – The time to reach 63.2% of the final output after the input is instantaneously increased.
Hysteresis – A condition when the output of a system depends not only on the input, but also on the previous outputs, e.g., a transducer that follows a different response curve when the niput is increasing than when the input is decreasing.

1.6. Security: How will intellectual property be managed?

The system may include software from TivaWare and from the book. No software written for this project may be transmitted, viewed, or communicated with any other EE445L student past, present, or future (other than the lab partner of course). It is the responsibility of the team to keep its EE445L lab solutions secure.

2. Function Description

2.1. Functionality: What will the system do precisely?

If all buttons are released, then the motor should spin at a constant speed. If switch 1 is pressed and released, the desired speed should increase by 5 rps, up to a maximum of 40 rps. If switch 2 is pressed and released, the desired speed should decrease by 5 rps, down to a minimum of 0 rps.

Both the desired and actual speeds should be plotted on the color LCD as a function of time similar to Figure 4.4.

2.2. Scope: List the phases and what will be delivered in each phase.

Phase 1 is the preparation; phase 2 is the demonstration; and phase 3 is the lab report. Details can be found in the lab manual.

2.3. Prototypes: How will intermediate progress be demonstrated?

A prototype system running on the EK-TM4C123GXL or EK-TM4C1294XL LaunchPad and solderless breadboard will be demonstrated. Progress will be judged by the preparation, demonstration and lab report.

2.4. Performance: Define the measures and describe how they will be determined.

The system will be judged by three qualitative measures. First, the software modules must be easy to understand and well-organized. Second, the system must employ an integral controller running in the background. There should be a clear and obvious abstraction, separating the state estimator, user interface, the controller and the actuator output. Backward jumps in the ISR are not allowed. Third, all software will be judged according to style guidelines. Software must follow the style described in Section 3.3 of the book. There are three quantitative measures. First, the average speed error at a desired speed of 60 rps will be measured. The average error should be less than 5 rps. Second, the step response is the time it takes for the new speed to hit 60 rps after the set point is changed from 40 to 60 rps. Third, you will measure power supply current to run the system. There is no particular need to minimize controller error, step response, or system current in this system.

2.5. Usability: Describe the interfaces. Be quantitative if possible.

There will be two switch inputs. The tachometer will be used to measure motor speed. The DC motor will operate under no load conditions,

2.6. Safety: Explain any safety requirements and how they will be measured.

Figure 4.2 shows that under a no load condition, the motor current will be less than 100 mA. However, under heavy friction this current could be 5 to 10 times higher. Therefore, please run the motors unloaded. Connecting or disconnecting wires on the protoboard while power is applied will damage the microcontroller. Operating the circuit without a snubber diode will also damage the microcontroller.

3. Deliverables

3.1. Reports: How will the system be described?

A lab report described below is due by the due date listed in the syllabus. This report includes the final requirements document.

3.2. Audits: How will the clients evaluate progress?
The preparation is due at the beginning of the lab period on the date listed in the syllabus.

3.3. Outcomes: What are the deliverables? How do we know when it is done?
There are three deliverables: preparation, demonstration, and report.

2.0 HARDWARE DESIGN

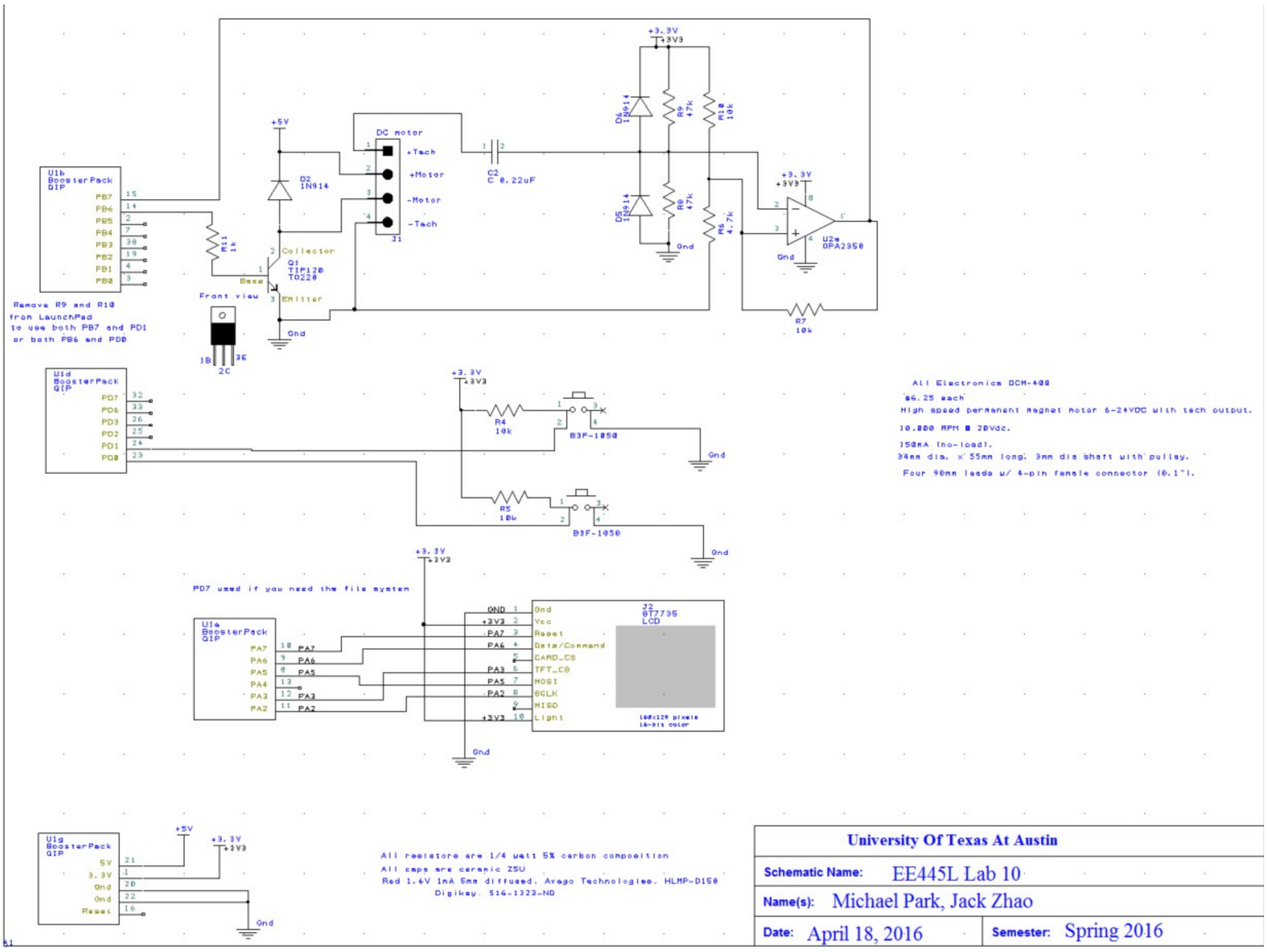


Figure 1: Lab10 Schematic

3.0 SOFTWARE DESIGN

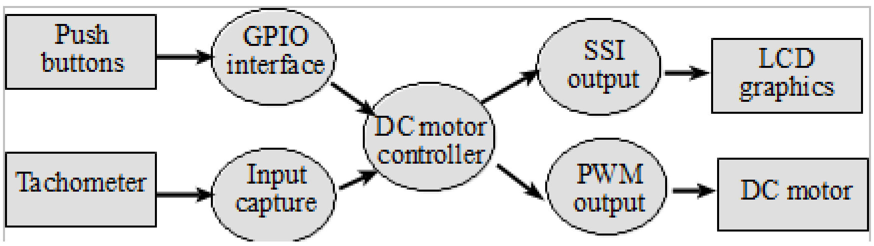


Figure 2: Data Flow

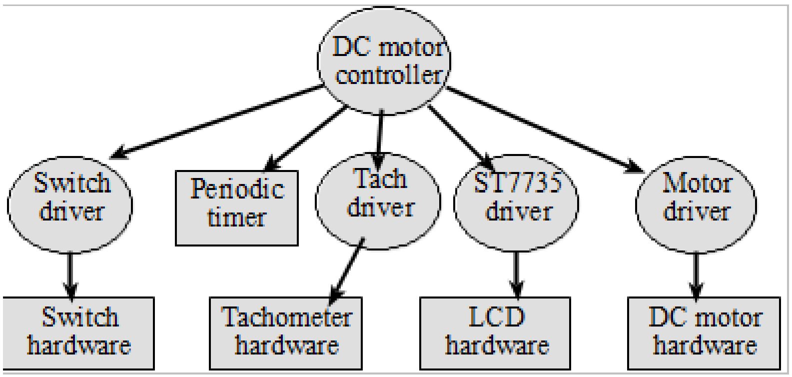


Figure 3: Call Graph

Refer to the source code attached at the end of this document

4.0 MEASUREMENT DATA

Procedure 1) Give the voltage, current, and resistance measurements

1 volt 30 mA
2 volt 71 mA
3 volt 76 mA
4 volt 80 mA
5 volt 83 mA

Internal Resistance is 250 ohm.

Procedure 2) I_{BE} and I_{CE} while spinning.

Ibe = Base emitter current = 0.1 mA
Ice = Collector current = 80 mA

Procedure 3) Two screen shots of the hardware in operation.

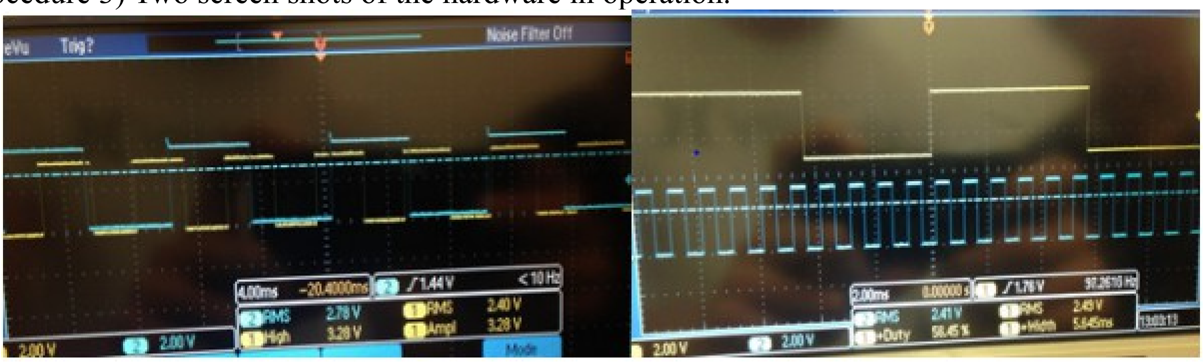


Figure 4: Screen shots of hardware in operation

Procedure 4) Specify the maximum time to execute one instance of the ISR

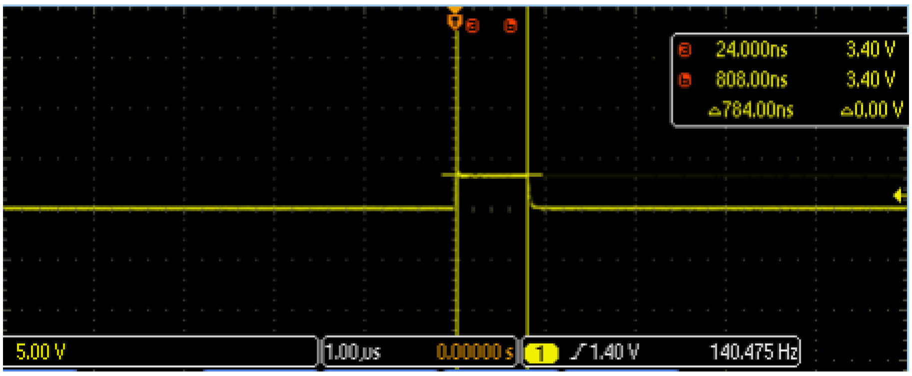


Figure 5: Maximum time to execute one instance of ISR

One instance of ISR takes 784ns.

Procedure 4) Specify the average controller error

During a finite duration when samples were taken in 2.5ms intervals:

Expected value in hex: 0x14
Actual value in hex: 0x12

Total time of this experiment: 20 * (.025) = 0.5s.
Average error: 2 / 20 = 1/10.

Procedure 4) Specify the approximate response time

Target speed :
30 rps
Number of samples taken in debugging module to reach 30rps from 29 rps:
199 samples.
samples were taken in 2.5ms intervals.

Rise time = 0.025*200 = 5s

Procedure 5) Measurements of current required to run the system, with and without the motor spinning

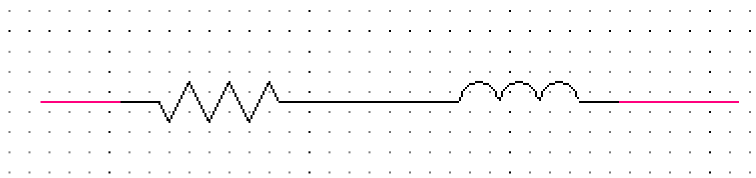
Current with the motor spinning: 100mA
Current without the motor spinning: 500mA

5.0 ANALYSIS AND DISCUSSION

1) What is torque? What are its units?

Torque is the force around an axis. It can be described as the tendency of an object to continue rotating and is calculated by T=rxF Where r is the position vector and F is the force vector. The SI units for torque is N·m

2) Draw an electrical circuit model for the DC motor coil, and explain the components. Use this circuit model to explain why the current goes up when friction is applied to the shaft



A DC motor coil is composed of a resistor and inductor in series. As friction is applied to the shaft, the current goes up because there is a greater torque provided by the motor. The back EMF will decrease and this will result in a greater current through the motor.

3) Explain what parameters were important for choosing a motor drive interface chip (e.g., TIP120)

The parameters necessary were that the interface would have to output enough current to activate the motor coils. In this case we used a base resistor of 1k ohms which created a large enough current and used a transistor which would definitely not be enough current to drive the motor. Therefore, we choose the 2N2222 to output the necessary current for the motor.

4) You implemented an integral controller because it is simple and stable. What other controllers

We implemented the integral controller due to its stability, but we could have used a variety of other controllers under the PID such as the Proportional controller and the Derivative controller. We actually did implement the Proportional controller as well and this is superior to just an integral controller because it allows our motor to reach the desired speed much faster. The issue with this is a proportional controller by itself often has the tendency to overshoot the desired speed of the motor and will need to slow down or speed up to adjust.

5) It the motor is spinning at a constant rate, give a definition of electrical power in terms of parameters of this lab? Research the term "*mechanical power*". Give a definition of mechanical power. Are the electrical power and mechanical power related?

Electrical power = Voltage * current * duty cycle
Mechanical Power = Torque * velocity
According to Newton's 2nd Law, power cannot be created or destroyed.
Thus, Electrical Power = Mechanical Power

```
// PeriodMeasure.c
// Runs on LM4F120/TM4C123
// Use Timer0A in 24-bit edge time mode to request interrupts on the rising
// edge of PB6 (T0CCP0), and measure period between pulses.
// Daniel Valvano
// May 5, 2015
```

// PeriodMeasure.c
// Michael Park, Jack Zhao, Corey Cormier
// Last modified on April 20, 2016

```
// external signal connected to PB6 (T0CCP0) (trigger on rising edge)
#include <stdint.h>
#include "../Shared/tm4c123gh6pm.h"
#include "PLL.h"
#include "PWM.h"
#include "PeriodMeasure.h"
#include "Switch.h"

#define NVIC_EN0_INT19         0x00080000  // Interrupt 19 enable
#define PF2                    (*((volatile uint32_t *)0x40025010))
#define TIMER_TAMR_TACMR       0x00000004  // GPTM TimerA Capture Mode
#define TIMER_TAMR_TAMR_CAP    0x00000003  // Capture mode
#define TIMER_CTL_TAEN         0x00000001  // GPTM TimerA Enable
#define TIMER_CTL_TAEVENT_POS  0x00000000  // Positive edge
#define TIMER_CTL_TAEVENT_NEG  0x00000004  // Negative edge
#define TIMER_CTL_TAEVENT_BOTH 0x0000000C  // Both edges
#define TIMER_IMR_CAEIM        0x00000004  // GPTM CaptureA Event Interrupt
                                           // Mask
#define TIMER_ICR_CAECINT      0x00000004  // GPTM CaptureA Event Interrupt
                                           // Clear
#define TIMER_TAILR_TAILRL_M   0x0000FFFF  // GPTM TimerA Interval Load
                                           // Register Low

void (*PeriodicTask)(void);   // user function


void DisableInterrupts(void); // Disable interrupts
void EnableInterrupts(void);  // Enable interrupts
long StartCritical (void);    // previous I bit, disable interrupts
void EndCritical(long sr);    // restore I bit to previous value
void WaitForInterrupt(void);  // low power mode

volatile int32_t currentDuty;
volatile int32_t newDuty;

volatile uint32_t currentSpeed;
volatile int32_t error;
volatile uint32_t rps;

volatile int32_t Ui=30000;
volatile int32_t Up;
```

```
uint32_t Period;              // (1/clock) units
uint32_t First;              // Timer0A first edge
int32_t Done;               // set each rising
// max period is (2^24-1)*12.5ns = 209.7151ms
// min period determined by time to run ISR, which is about 1us
void PeriodMeasure_Init(void){
  SYSCTL_RCGCTIMER_R |= 0x01;// activate timer0
  SYSCTL_RCGCGPIO_R |= 0x22;       // activate port B and port F
                                 // allow time to finish activating
  First = 0;                    // first will be wrong
  Done = 0;                      // set on subsequent
  GPIO_PORTB_DIR_R &= ~0x40;       // make PB6 in
  GPIO_PORTB_AFSEL_R |= 0x40;      // enable alt funct on PB6/T0CCP0
  GPIO_PORTB_DEN_R |= 0x40;        // enable digital I/O on PB6
                                 // configure PB6 as T0CCP0
  GPIO_PORTB_PCTL_R = (GPIO_PORTB_PCTL_R&0xF0FFFFFF)+0x07000000;
  GPIO_PORTB_AMSEL_R &= ~0x40;     // disable analog functionality on PB6
  GPIO_PORTF_DIR_R |= 0x04;        // make PF2 out (PF2 built-in blue LED)
  GPIO_PORTF_AFSEL_R &= ~0x04;     // disable alt funct on PF2
  GPIO_PORTF_DEN_R |= 0x04;        // enable digital I/O on PF2
                                 // configure PF2 as GPIO
  GPIO_PORTF_PCTL_R = (GPIO_PORTF_PCTL_R&0xFFFFF0FF)+0x00000000;
  GPIO_PORTF_AMSEL_R = 0;          // disable analog functionality on PF
  TIMER0_CTL_R &= ~TIMER_CTL_TAEN; // disable timer0A during setup
  TIMER0_CFG_R = TIMER_CFG_16_BIT; // configure for 16-bit timer mode
                                 // configure for 24-bit capture mode
  TIMER0_TAMR_R = (TIMER_TAMR_TACMR|TIMER_TAMR_TAMR_CAP);
                                 // configure for rising edge event
  TIMER0_CTL_R &= ~(TIMER_CTL_TAEVENT_POS|0xC);
  TIMER0_TAILR_R = TIMER_TAILR_TAILRL_M;// start value
  TIMER0_TAPR_R = 0xFF;            // activate prescale, creating 24-bit
  TIMER0_IMR_R |= TIMER_IMR_CAEIM; // enable capture match interrupt
  TIMER0_ICR_R = TIMER_ICR_CAECINT;// clear timer0A capture match flag
  TIMER0_CTL_R |= TIMER_CTL_TAEN;  // enable timer0A 16-b, +edge timing, interrupts
                                 // Timer0A=priority 2
  NVIC_PRI4_R = (NVIC_PRI4_R&0x00FFFFFF)|0x40000000; // top 3 bits
  NVIC_EN0_R = NVIC_EN0_INT19;     // enable interrupt 19 in NVIC
}

// ***************** Timer2_Init ****************
// Activate Timer2 interrupts to run user task periodically
// Inputs:  task is a pointer to a user function
//          period in units (1/clockfreq)
// Outputs: none
void Timer2_Init(void(*task)(void), unsigned long period){
  SYSCTL_RCGCTIMER_R |= 0x04;   // 0) activate timer2
  PeriodicTask = task;          // user function
  TIMER2_CTL_R = 0x00000000;    // 1) disable timer2A during setup
  TIMER2_CFG_R = 0x00000000;    // 2) configure for 32-bit mode
   TIMER2_TAMR_R = 0x00000002;    // 3) configure for periodic mode, default down-count
      settings
```

```
  TIMER2_TAILR_R = period-1;    // 4) reload value
  TIMER2_TAPR_R = 0;            // 5) bus clock resolution
  TIMER2_ICR_R = 0x00000001;    // 6) clear timer2A timeout flag
  TIMER2_IMR_R = 0x00000001;    // 7) arm timeout interrupt
  NVIC_PRI5_R = (NVIC_PRI5_R&0x00FFFFFF)|0x80000000; // 8) priority 4
// interrupts enabled in the main program after all devices initialized
// vector number 39, interrupt number 23
  NVIC_EN0_R = 1<<23;           // 9) enable IRQ 23 in NVIC
  TIMER2_CTL_R = 0x00000001;    // 10) enable timer2A
}

void Timer2A_Handler(void){
  TIMER2_ICR_R = TIMER_ICR_TATOCINT;// acknowledge TIMER2A timeout
	integralControl();
}


void Timer0A_Handler(void){
  PF2 = PF2^0x04;  // toggle PF2
  PF2 = PF2^0x04;  // toggle PF2
  TIMER0_ICR_R = TIMER_ICR_CAECINT;// acknowledge timer0A capture match
  Period = (First - TIMER0_TAR_R)&0xFFFFFF;// 24 bits, 12.5ns resolution
  First = TIMER0_TAR_R;            // setup for next
  Done = 1;
  PF2 = PF2^0x04;  // toggle PF2
}

void integralControl(void)
{
	currentSpeed = 80000000/(Period*4); // 0.1 Hz, 0.025rps
	//currentSpeed = 800000000/(1.65 * Period); // 0.1 Hz, 0.025rps
  error = desiredRev-currentSpeed;          // 0.1 Hz, 0.025rps
	//rps = currentSpeed/40;
	Ui= Ui +(10000*error)/256;
	if(Ui < 100) Ui = 100;                  // lower/upper bounds
  if(Ui > 39960) Ui = 39960;
	Up=(10000*error)/256;

	newDuty=Ui+Up;

  //newDuty = newDuty+((100*error)/256);        // calculation of duty
  if(newDuty < 23000) newDuty = 23000;                        // lower/upper bounds
  if(newDuty > 39960) newDuty = 39960;
	PWM0B_Duty(newDuty);
}
```

// PeriodMeasure.h
// Michael Park, Jack Zhao, Corey Cormier
// April 20, 2016

```
#include <stdint.h>
```

```
extern volatile int32_t currentDuty;
extern volatile int32_t newDuty;

void PeriodMeasure_Init(void);
uint32_t returnPeriod(void);
void integralControl(void);
void Timer2_Init(void(*task)(void), unsigned long period);
```

```
// PWM.c
// Runs on TM4C123
// Use PWM0A/PB6 and PWM0B/PB7 to generate pulse-width modulated outputs.
// Daniel Valvano
// March 28, 2014
```

// PWM.c
// Michael Park, Jack Zhao, Corey Cormier
// Last modified on April 20, 2016

```
#include <stdint.h>
#include "../Shared/tm4c123gh6pm.h"
#define PWM_0_GENA_ACTCMPAD_ONE 0x000000C0  // Set the output signal to 1
#define PWM_0_GENA_ACTLOAD_ZERO 0x00000008  // Set the output signal to 0
#define PWM_0_GENB_ACTCMPBD_ONE 0x00000C00  // Set the output signal to 1
#define PWM_0_GENB_ACTLOAD_ZERO 0x00000008  // Set the output signal to 0

#define SYSCTL_RCC_USEPWMDIV    0x00100000  // Enable PWM Clock Divisor
#define SYSCTL_RCC_PWMDIV_M     0x000E0000  // PWM Unit Clock Divisor
#define SYSCTL_RCC_PWMDIV_2     0x00000000  // /2

// period is 16-bit number of PWM clock cycles in one period (3<=period)
// period for PB6 and PB7 must be the same
// duty is number of PWM clock cycles output is high  (2<=duty<=period-1)
// PWM clock rate = processor clock rate/SYSCTL_RCC_PWMDIV
//                = BusClock/2
//                = 80 MHz/2 = 40 MHz (in this example)
// Output on PB7/M0PWM1
void PWM0B_Init(uint16_t period, uint16_t duty){
  volatile unsigned long delay;
  SYSCTL_RCGCPWM_R |= 0x01;             // 1) activate PWM0
  SYSCTL_RCGCGPIO_R |= 0x02;            // 2) activate port B
  delay = SYSCTL_RCGCGPIO_R;            // allow time to finish activating
  GPIO_PORTB_AFSEL_R |= 0x80;           // enable alt funct on PB7
  GPIO_PORTB_PCTL_R &= ~0xF0000000;     // configure PB7 as M0PWM1
  GPIO_PORTB_PCTL_R |= 0x40000000;
  GPIO_PORTB_AMSEL_R &= ~0x80;          // disable analog functionality on PB7
  GPIO_PORTB_DEN_R |= 0x80;             // enable digital I/O on PB7
  SYSCTL_RCC_R |= SYSCTL_RCC_USEPWMDIV; // 3) use PWM divider
  SYSCTL_RCC_R &= ~SYSCTL_RCC_PWMDIV_M; //    clear PWM divider field
  SYSCTL_RCC_R += SYSCTL_RCC_PWMDIV_2;  //    configure for /2 divider
  PWM0_0_CTL_R = 0;                     // 4) re-loading down-counting mode
                  PWM0_0_GENB_R           =           (PWM_0_GENB_ACTCMPBD_ONE|
```

```
        PWM_0_GENB_ACTLOAD_ZERO);
  // PB7 goes low on LOAD
  // PB7 goes high on CMPB down
  PWM0_0_LOAD_R = period - 1;        // 5) cycles needed to count down to 0
  PWM0_0_CMPB_R = duty - 1;          // 6) count value when output rises
  PWM0_0_CTL_R |= 0x00000001;        // 7) start PWM0
  PWM0_ENABLE_R |= 0x00000002;       // enable PB7/M0PWM1
}
// change duty cycle of PB7
// duty is number of PWM clock cycles output is high  (2<=duty<=period-1)
void PWM0B_Duty(uint16_t duty){
  PWM0_0_CMPB_R = duty - 1;          // 6) count value when output rises
}

void PWM0B_Off(void){
        PWM0_0_CTL_R &= ~0x00000001;                                   //Stops PWM0
}


// PWM.h
// Runs on TM4C123
// Use PWM0A/PB6 and PWM0B/PB7 to generate pulse-width modulated outputs.
// Daniel Valvano
// March 28, 2014
```

// PWM.h
// Michael Park, Jack Zhao, Corey Cormier
// Last modified on April 20, 2016

```
// period is 16-bit number of PWM clock cycles in one period (3<=period)
// period for PB6 and PB7 must be the same
// duty is number of PWM clock cycles output is high  (2<=duty<=period-1)
// PWM clock rate = processor clock rate/SYSCTL_RCC_PWMDIV
//              = BusClock/2
//              = 80 MHz/2 = 40 MHz (in this example)
// Output on PB7/M0PWM1
void PWM0B_Init(uint16_t period, uint16_t duty);

// change duty cycle of PB7
// duty is number of PWM clock cycles output is high  (2<=duty<=period-1)
void PWM0B_Duty(uint16_t duty);


void PWM0B_Off(void);

// PWMtest.c
// Runs on TM4C123
// Use PWM0/PB6 and PWM1/PB7 to generate pulse-width modulated outputs.
// Daniel Valvano
// March 28, 2014
```

// PWMtest.c
// Michael Park, Jack Zhao, Corey Cormier
// Last modified on April 20, 2016

```
//  PWM0_Duty(4000);    // 10%
//  PWM0_Duty(10000);   // 25%
//  PWM0_Duty(30000);   // 75%

//  PWM0_Init(4000, 2000);         // initialize PWM0, 10000 Hz, 50% duty
//  PWM0_Init(1000, 900);          // initialize PWM0, 40000 Hz, 90% duty
//  PWM0_Init(1000, 100);          // initialize PWM0, 40000 Hz, 10% duty
//  PWM0_Init(40, 20);             // initialize PWM0, 1 MHz, 50% duty


#include <stdint.h>
#include <stdio.h>
#include "PLL.h"
#include "PWM.h"
#include "Switch.h"
#include "PeriodMeasure.h"
#include "ST7735.h"
#include "../Shared/tm4c123gh6pm.h"



#define Timer2Period 1600000


void DisableInterrupts(void); // Disable interrupts
void EnableInterrupts(void);  // Enable interrupts
long StartCritical (void);    // previous I bit, disable interrupts
void EndCritical(long sr);    // restore I bit to previous value
void WaitForInterrupt(void);  // low power mode

volatile uint32_t PWMPeriod=40000;
volatile uint32_t count=0;
extern volatile uint32_t currentSpeed;

int main(void){
	DisableInterrupts();
	currentDuty=30000;
	desiredRev=30;
  PLL_Init(Bus80MHz);              // bus clock at 80 MHz
	ST7735_InitR(INITR_REDTAB);
	ST7735_FillScreen(ST7735_WHITE);
  ST7735_FillRect(0, 0, 128, 50, ST7735_BLACK);
	ST7735_PlotClear(0, 4000);
	Switch_Init();
  PWM0B_Init(PWMPeriod, currentDuty);         // initialize PWM0, 1000 Hz, 25% duty
	//PWM0B_Off();
	PeriodMeasure_Init();
	Timer2_Init(0, Timer2Period);
```

```
	EnableInterrupts();
	newDuty=currentDuty;

  while(1){
		count++;
		if(count>=1000){
		//ST7735_PlotPoint(currentSpeed);
		ST7735_SetCursor(0,0);
		ST7735_OutString("Desired RPS: ");
   ST7735_OutUDec(desiredRev);
		ST7735_SetCursor(0,2);
		ST7735_OutString("Current RPS: ");
		ST7735_OutUDec(currentSpeed);

		ST7735_PlotNextErase();
		ST7735_PlotPoint(currentSpeed*100);
		ST7735_PlotNext();
		ST7735_PlotNextErase();

			count=0;
		}

  }
}
```

```
// Switch.c
// Runs on LM4F120/TM4C123
// Provide functions that initialize a GPIO as an input pin and
// allow reading of two negative logic switches on PF0 and PF4
// and an external switch on PA5.
// Use bit-banded I/O.
// Daniel and Jonathan Valvano
// September 12, 2013
```

// Switch.c
// Michael Park, Jack Zhao, Corey Cormier
// Last modified on April 20, 2016

```
// negative logic switches connected to PF0 and PF4 on the Launchpad
// red LED connected to PF1 on the Launchpad
// blue LED connected to PF2 on the Launchpad
// green LED connected to PF3 on the Launchpad
// NOTE: The NMI (non-maskable interrupt) is on PF0.  That means that
// the Alternate Function Select, Pull-Up Resistor, Pull-Down Resistor,
// and Digital Enable are all locked for PF0 until a value of 0x4C4F434B
// is written to the Port F GPIO Lock Register.  After Port F is
// unlocked, bit 0 of the Port F GPIO Commit Register must be set to
// allow access to PF0's control registers.  On the LM4F120, the other
// bits of the Port F GPIO Commit Register are hard-wired to 1, meaning
// that the rest of Port F can always be freely re-configured at any
// time.  Requiring this procedure makes it unlikely to accidentally
// re-configure the JTAG and NMI pins as GPIO, which can lock the
```

```
// debugger out of the processor and make it permanently unable to be
// debugged or re-programmed.
#include <stdint.h>
#include "../Shared/tm4c123gh6pm.h"
#include "SysTick.h"

void DisableInterrupts(void); // Disable interrupts
void EnableInterrupts(void);  // Enable interrupts
long StartCritical (void);    // previous I bit, disable interrupts
void EndCritical(long sr);    // restore I bit to previous value
void WaitForInterrupt(void);  // low power mode

#define GPIO_LOCK_KEY           0x4C4F434B  // Unlocks the GPIO_CR register
#define PF0                     (*((volatile uint32_t *)0x40025004))
#define PF4                     (*((volatile uint32_t *)0x40025040))
#define PA5                     (*((volatile uint32_t *)0x40004080))
#define SWITCHES                (*((volatile uint32_t *)0x40025044))
#define SW1       0x10                      // on the left side of the Launchpad board
#define SW2       0x01                      // on the right side of the Launchpad board
#define SYSCTL_RCGC2_GPIOF      0x00000020  // port F Clock Gating Control

volatile uint32_t desiredRev;

int last0=1;
int last1=1;
//------------Switch_Init------------
// Initialize GPIO Port D
// Input: none
// Output: none
void Switch_Init(void){
  SYSCTL_RCGCGPIO_R |= 0x08;        // 1) activate clock for Port D
	int i;
	i=SYSCTL_RCGCGPIO_R;
  GPIO_PORTD_DIR_R &= ~0x0C;        // PD0-3 is an input
  GPIO_PORTD_AMSEL_R &= ~0x0C;      // disable analog on PD0-3
  GPIO_PORTD_DEN_R |= 0x0C;         // PD0-3 enabled as a digital port
	GPIO_PORTD_IS_R &= ~0x0C;                                  // PD 0-3 is edge-
	sensitive
	GPIO_PORTD_IBE_R &= ~0x0C;                            // PD 0-3 is not both edges
	GPIO_PORTD_IEV_R &= ~0x0C;                            //  Pd  0-3  falling  edge
	event
	GPIO_PORTD_ICR_R = 0x0C;                                   // clear flag 0-3
	GPIO_PORTD_IM_R |= 0x0C;                                   // arm interrupt on
	PD 0-3
	NVIC_PRI0_R = (NVIC_PRI0_R&0xFF00FFFF)|0x00A00000; // (5) priority 5
	NVIC_EN0_R = 0x00000008;                                   //enable   interrupt
	1(PB) in NVIC
	SYSCTL_RCGCTIMER_R |= 0x02;

}

uint32_t Switch_Input(void){
```

```
  return GPIO_PORTD_RIS_R &0x03; // status of
}

void Timer1_ARM(void){
  TIMER1_CTL_R = 0x00000000;    // 1) disable timer2A during setup
  TIMER1_CFG_R = 0x00000000;    // 2) configure for 32-bit mode
   TIMER1_TAMR_R = 0x00000001;     // 3) configure for periodic mode, default down-count
        settings
  TIMER1_TAILR_R = 20000000;    // 4) reload value
  TIMER1_TAPR_R = 0;            // 5) bus clock resolution
  TIMER1_ICR_R = 0x00000001;    // 6) clear timer2A timeout flag
  TIMER1_IMR_R = 0x00000001;    // 7) arm timeout interrupt
  NVIC_PRI5_R = (NVIC_PRI5_R&0xFFFF00FF)|0x00008000; // 8) priority 4
// interrupts enabled in the main program after all devices initialized
// vector number 39, interrupt number 23
  NVIC_EN0_R = 1<<21;           // 9) enable IRQ 23 in NVIC
  TIMER1_CTL_R = 0x00000001;    // 10) enable timer2A
}

void GPIO_ARM(void){
        GPIO_PORTD_ICR_R = 0x0C;                    // clear flag 0-3
        GPIO_PORTD_IM_R |= 0x0C;                    // arm interrupt on
        PD 0-3
            NVIC_EN0_R = 0x00000008;                //enable    interrupt
        1(PB) in NVIC
}

void Timer1A_Handler(void){
  TIMER1_IMR_R = 0x00000000;    // 7) arm timeout interrupt
        GPIO_ARM();
        last0=GPIO_PORTD_DATA_R & 0x04;
        last1=GPIO_PORTD_DATA_R & 0x08;



}
#define DELAY10MS 160000
#define DELAY10US 160
//------------Switch_Debounce------------
// Read and return the status of the switch
// Input: none
// Output: 0x02 if PB1 is high
//        0x00 if PB1 is low
// debounces switch
uint32_t Switch_Debounce(void){
uint32_t in,old,time;
  time = 1000; // 10 ms
  old = Switch_Input();
  while(time){
    SysTick_Wait(DELAY10US); // 10us
    in = Switch_Input();
    if(in == old){
      time--; // same value
```

```
    }else{
      time = 1000;  // different
      old = in;
    }
  }
  return old;
}


void GPIOPortD_Handler(void)
{

	GPIO_PORTD_IM_R &= ~0x0C;						// arm interrupt on
	PD 0-3
	//switch debounce
	//uint32_t newSwitch=		Switch_Debounce();
	//set functions
	//long sr=StartCritical();
		if(GPIO_PORTD_RIS_R & 0X04 && last0) //poll PD0
		{
			GPIO_PORTD_ICR_R = 0x01; //acknowledge flag1 and clear
			if(desiredRev + 5 < 40)
			{
				desiredRev += 5;
			}
			else
			{
				desiredRev = 40;
			}
		}

		if(GPIO_PORTD_RIS_R & 0X08  &&last1) //poll PD1
		{
			GPIO_PORTD_ICR_R = 0x02; //acknowledge flag1 and clear
			if(desiredRev - 5 > 15)
			{
				desiredRev -= 5;
			}
			else
			{
				desiredRev = 15;
			}

		}
		Timer1_ARM();
		//EndCritical(sr);
}

// Switch.h
// Runs on LM4F120/TM4C123
// Provide functions that initialize a GPIO as an input pin and
// allow reading of two negative logic switches on PF0 and PF4
```

```
// and an external switch on PA5.
// Use bit-banded I/O.
// Daniel and Jonathan Valvano
// September 12, 2013

// Switch.h
// Michael Park, Jack Zhao, Corey Cormier
// Last modified on April 20, 2016

// negative logic switches connected to PF0 and PF4 on the Launchpad
// red LED connected to PF1 on the Launchpad
// blue LED connected to PF2 on the Launchpad
// green LED connected to PF3 on the Launchpad
// NOTE: The NMI (non-maskable interrupt) is on PF0.  That means that
// the Alternate Function Select, Pull-Up Resistor, Pull-Down Resistor,
// and Digital Enable are all locked for PF0 until a value of 0x4C4F434B
// is written to the Port F GPIO Lock Register.  After Port F is
// unlocked, bit 0 of the Port F GPIO Commit Register must be set to
// allow access to PF0's control registers.  On the LM4F120, the other
// bits of the Port F GPIO Commit Register are hard-wired to 1, meaning
// that the rest of Port F can always be freely re-configured at any
// time.  Requiring this procedure makes it unlikely to accidentally
// re-configure the JTAG and NMI pins as GPIO, which can lock the
// debugger out of the processor and make it permanently unable to be
// debugged or re-programmed.
#include <stdint.h>


extern volatile uint32_t desiredRev;


//------------Switch_Init------------
// Initialize GPIO Port A bit 5 for input.
// Input: none
// Output: none
void Switch_Init(void);

//------------Switch_Input------------
// Read and return the status of GPIO Port A bit 5.
// Input: none
// Output: 0x20 if PA5 is high
//         0x00 if PA5 is low
uint32_t Switch_Input(void);

//------------Board_Init------------
// Initialize GPIO Port F for negative logic switches on PF0 and
// PF4 as the Launchpad is wired.  Weak internal pull-up
// resistors are enabled, and the NMI functionality on PF0 is
// disabled.
// Input: none
// Output: none
void Board_Init(void);
```

```
//------------Board_Input------------
// Read and return the status of the switches.
// Input: none
// Output: 0x01 if only Switch 1 is pressed
//         0x10 if only Switch 2 is pressed
//         0x00 if both switches are pressed
//         0x11 if no switches are pressed
uint32_t Board_Input(void);

// Program 2.9 from Volume 2
//------------Switch_Init3------------
// Initialize GPIO Port B bit 1 for input.
// Input: none
// Output: none
void Switch_Init3(void);
//------------Switch_Input3------------
// Read and return the status of GPIO Port B bit 1.
// Input: none
// Output: 0x02 if PB1 is high
//         0x00 if PB1 is low
uint32_t Switch_Input3(void);

//------------Switch_Debounce------------
// Read and return the status of the switch
// Input: none
// Output: 0x02 if PB1 is high
//         0x00 if PB1 is low
// debounces switch
uint32_t Switch_Debounce(void);

//------------Switch_Debounce------------
// wait for the switch to be touched
// Input: none
// Output: none
// debounces switch
void Switch_WaitForTouch(void);

// SysTick.c
// Runs on LM4F120/TM4C123
// 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
// LM4F120 gets its clock from the 16 MHz precision internal oscillator,
// which can vary by +/- 1% at room temperature and +/- 3% across all
// temperature ranges.  This matters for the function
// SysTick_Wait10ms(), which will wait longer than 10 ms if the clock is
// slower.
// Daniel Valvano
// September 11, 2013
```

// SysTick.c
// Michael Park, Jack Zhao, Corey Cormier
// Last modified on April 20, 2016

```
#include <stdint.h>
#include "../Shared/tm4c123gh6pm.h"
#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.
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 62.5 nsec for 16 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 using busy wait.
// This assumes 16 MHz system clock.
void SysTick_Wait10ms(uint32_t delay){
  uint32_t i;
  for(i=0; i<delay; i++){
    SysTick_Wait(160000);  // wait 10ms (assumes 16 MHz clock)
  }
}
```

```
// SysTick.h
// Runs on LM4F120/TM4C123
// 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
// LM4F120 gets its clock from the 16 MHz precision internal oscillator,
// which can vary by +/- 1% at room temperature and +/- 3% across all
// temperature ranges.  This matters for the function
// SysTick_Wait10ms(), which will wait longer than 10 ms if the clock is
// slower.
// Daniel Valvano
// September 11, 2013
```

```
// SysTick.h
// Michael Park, Jack Zhao, Corey Cormier
// Last modified on April 20, 2016

// Initialize SysTick with busy wait running at bus clock.
void SysTick_Init(void);

// Time delay using busy wait.
// The delay parameter is in units of the core clock. (units of 62.5 nsec for 16 MHz clock)
void SysTick_Wait(uint32_t delay);

// Time delay using busy wait.
// This assumes 16 MHz system clock.
void SysTick_Wait10ms(uint32_t delay);
```