E C E 315: Introductory Microprocessor Laboratory

LAB #2

Motor Control Using PWM

1 Design Objectives

- Configure GPIO pin(s) as a PWM peripheral
- Configure a PWM peripheral
- Develop a driver that enables software to control the direction and speed of the robot

1.1 Background

The ECE315 robot has two brushed DC motors that drive the treads when a voltage is applied. The motors are controlled by a Texas Instruments DRV8833 motor driver. The DRV8833 provides two independent H-Bridge circuits that allow us to toggle which motor terminal is connected to ground and VCC. The H-Bridge allows us to drive the robot both forwards and backwards.

We will also use the DRV8833 to set the speed of the robot. If the input signals to the DRV8833 have a pulse width modulated (PWM) inputs, we are able to increase and decrease the speed of the robot. So how can we drive the inputs of the DRV8833 with a PWM signal? We could use regular GPIO pins in combination with a timer to PWM the required pins. While this might be the simplest approach, it does have some draw backs. If software directly controls the GPIO Pins, then we are consuming a major portion of the CPU's time to generate the PWM signals. We also have to use a timer resource to determine when to toggle the signal.

So instead of using software to generate the PWM signals, we are going to use one of the dedicated hardware PWM peripherals in TM4C123. Using the PWM peripheral will offload some of the computation from the application itself. It also removes the need for a timer resource.

1.2 GPIO Configuration

There are 4 different inputs to the DRV8833 that we need to control the PWM signals: AIN1, AIN2, BIN1, and BIN2. These signals are connected to PB4, PB5, PE4, and PE5 of the Tiva Launchpad. These GPIO pins must be configured as the PWM peripheral in the alternate function register.

In addition to setting the alternate function, we also have to configure the port control register for each GPIO pin. In order to determine which PWM module that each of our 4 PWM pins is connected to, we need to look at chapter 21 of the data sheet. This table helps us to determine the following port control mappings for our GPIO pins. There is a second possible mapping, but we're going to use the ones below.

PB4 ← M0PWM2
PB5 ← M0PWM3
PE4 ← M1PWM2
PE5 ← M1PWM3

There are two additional GPIO pins that must be configured as digital GPIO pins. PF2 should be a digital input and PF3 must be a digital output. PF3 is connected to the nSLEEP pin. **<u>You have to set this pin to a '1' to make the motors move.</u>**

Complete the function drv8833_gpioInit () in drv8833.c to properly initialize these all 6 pins used to control the DRV8833. You will then add the necessary code to boardUtil.c that calls drv8833_gpioInit().

1.3 PWM Configuration

In addition to configuring the GPIO pins, you will write a routine that configures the PWM peripheral. When completing this function, make sure to read chapter 20 of the TM4C123 data sheet. Pay close attention to section 20.4. *[**Important note:** Use **RCGCPWM** register instead of **RCGC0** mentioned in Step 1. This change has been documented in pg. 354 of the datasheet]*

Also, be sure to look at line 383 of TM4C123GH6PM.h to see the struct that defines how to access the registers for the PWM peripheral.

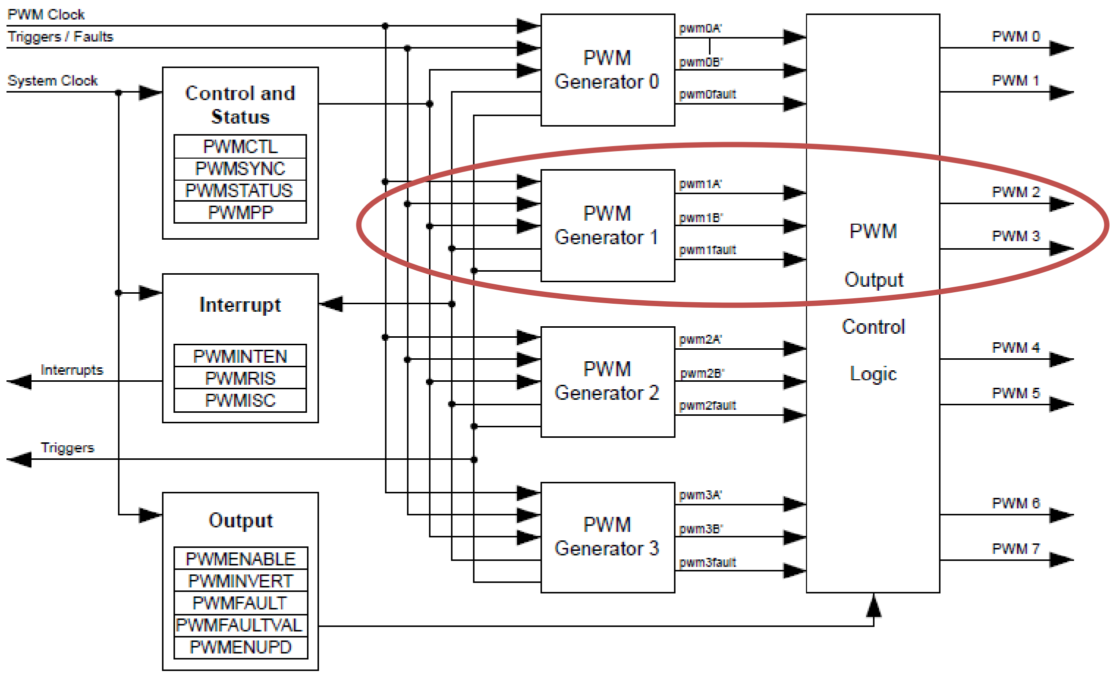
The configuration routine with have the following interface:

```
uint8_t pwmConfig(
  uint32_t base,
  uint8_t pwm_generator,
  uint32_t load,
  uint32_t cmpa,
  uint32_t cmpb,
  uint32_t gena,
  uint32_t genb
);
```

Parameter	Description
`Base`	The base address of the PWM peripheral. `PWM0_BASE` or `PWM1_BASE`.
`pwm_generator`	Which PWM generator to is being configured (0, 1, 2, or 3).
`Load`	Value written to `_x_LOAD` (where x is 0, 1, 2, or 3)
`Cmpa`	Value written to `_x_CMPA` (where x is 0, 1, 2, or 3)
`Cmpb`	Value written to `_x_CMPB` (where x is 0, 1, 2, or 3)
`Gena`	Value written to `_x_GENA` (where x is 0, 1, 2, or 3)
`Genb`	Value written to `_x_GENB` (where x is 0, 1, 2, or 3)

Make sure to check that `base` and `pwm_generator` are valid parameters!

Each PWM module has 4 PWM generators. Each of those generators has two PWM outputs that it can control.



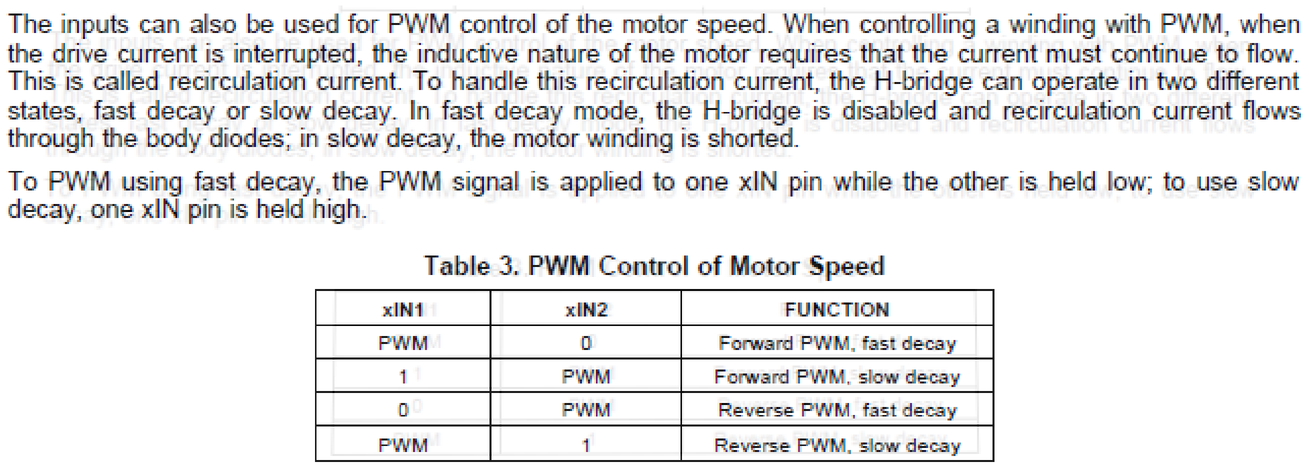
Let’s examine how we can set the PWM values for PB4 and PB5. If you look at chapter 21, we can see that PB4 maps to M0PWM2 and PB5 maps to M0PWM3. In order to set the PWM for each of these pins, we would need to call `pwmConfig` using `PWM0_BASE` and ‘1’ for the first two parameters . The other 4 parameters will be determined by the action you want the robot to take (Forward, Backwards, etc…)

1.4 Driver Development

After the GPIO and PWM drivers are completed, you will need to complete the following functions in drv8833.c. Each of these functions takes a single argument that indicates what the duty cycle output to one of the motors should be.

```
void  drv8833_leftForward(uint8_t dutyCycle);
void  drv8833_leftReverse(uint8_t dutyCycle);
void  drv8833_rightForward(uint8_t dutyCycle);
void  drv8833_rightReverse(uint8_t dutyCycle);
void  drv8833_turnLeft(uint8_t dutyCycle);
void  drv8833_turnRight(uint8_t dutyCycle);
```

Each of these functions will call pwmConfig(…) with the appropriate parameters. The following is a snippet of the DRV8833 data sheet.



The inputs can also be used for PWM control of the motor speed. When controlling a winding with PWM, when the drive current is interrupted, the inductive nature of the motor requires that the current must continue to flow. This is called recirculation current. To handle this recirculation current, the H-bridge can operate in two different states, fast decay or slow decay. In fast decay mode, the H-bridge is disabled and recirculation current flows through the body diodes; in slow decay, the motor winding is shorted.

To PWM using fast decay, the PWM signal is applied to one xIN pin while the other is held low; to use slow decay, one xIN pin is held high.

Table 3. PWM Control of Motor Speed

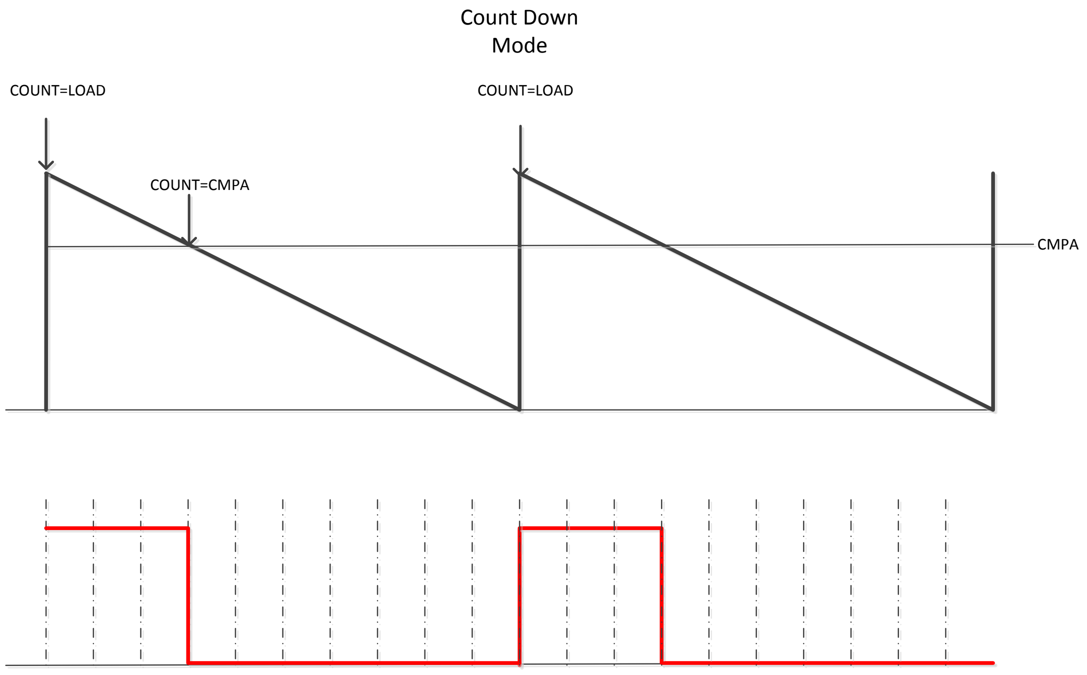
xIN1	xIN2	FUNCTION
PWM	0	Forward PWM, fast decay
1	PWM	Forward PWM, slow decay
0	PWM	Reverse PWM, fast decay
PWM	1	Reverse PWM, slow decay

The direction of the motor can be set to forward by configuring a PWM signal to output on PB4 and setting PB5 to be 0. You can reverse the direction of the motor simply by swapping the behavior of PB4 and PB5. For each of the functions above, you will determine the final 4 parameters based on the desired behavior of the function. Be aware that the left and right motors are physically flipped by 180 degrees. In order to drive in a single direction, one motor will run forwards and the other backwards!

Now, let's figure out how to generate the PWM signal. As an example, let's say that we wanted to drive PB4 with a 70% duty cycle. The registers needed to configure to set the duty cycle are

```
PWM0->_0_LOAD
PWM0->_0_CMPA
PWM0->_0_GENA
```

When configured in Count Down mode, we want the signal to look like this



If the load value was set to 10,000, then we would want CMPA to be 7,000. The GENA register is used to determine how the output of PB4 reacts to one of three important situations: the counter gets loaded, the counter reaches CMPA, and the counter reaches 0. The GENA register allows you to specify the value of the PWM output for each of these situations.

PB5 will need to be programmed similarly. `cmpa` and `gena` are used to set the behavior of PB4. `cmpb` and `genb` are used to set the behavior of PB5. The important thing to recognize that both signals for a PWM generator shares the LOAD register but that GENA is independent of GENB and CMPA is independent of CMPB!

Finally, make sure to set the proper bits in the following registers to enable the PWM signals.

```
PWM0->_0_CTL

PWM0->ENABLE
```

1.5 What to Demo

You should write a basic program that allows your robot to drive forward for two seconds, drive backwards for 2 seconds, turn left for 5 seconds, and then turn right for 5 seconds.

1.6 General Advice

1. Examine the schematic and understand how the motor drivers are connected to the Launchpad.
2. Use the functions in gpio.h to configure the 6 gpio pins.
3. Use the debugger to verify the GPIO pins are configured correctly.
4. For each motor, determine what each signals should do to drive forward / backwards / left turn / right turn.
5. Use the oscilloscope to verify the PWM signals are outputting the value you would expect.