Pulse Width Modulation on the TM4C123GH6PM Microcontroller

Joshua Lamb

11/13/2015

Contents

Introduction

This document walks the user through enabling pulse width modulation on the Texas Instruments TM4C123GH6PM microcontroller. The methods to enable pins and features used to enable pulse width modulation (PWM) are unique to this microcontroller, however, the general method is portable to other Texas Instruments microcontrollers. This document makes the assumption that the user is familiar with computer programming, is familiar with the concepts of pulse with modulation and already has an integrated development environment, such as Code Composer Studio, installed. It also makes the assumption that the user has enabled the microcontroller properly within the environment which installs the necessary files for this specific microcontroller.

There are two ways to enable pins and functionality for this microcontroller. The first is to access the hardware registers within the microcontroller directly by changing the bits within each register. This document makes use of the second method which uses the libraries that are installed while enabling the functionality for the microcontroller within the integrated development environment. A library in this context is a set of pre-defined values and functions that take a more natural language approach to access the functionality within the microcontroller. This approach masks the task of enabling pins and hardware functions in an attempt to make the process easier on the user.

Libraries

The first step in enabling the pins for use and the PWM function is to include the correct libraries. These files end with an .h to denote that they are header files, files which are used to define functions and values for reuse. These libraries are enabled by the “#include” before the library for the compiler to know to include the files when building the software.

```
#include "inc/hw_memmap.h"
```
hw_memmap defines the base port addresses for the general purpose input output (GPIO) ports and the base memory address for the PWM output. This allows the microcontroller to know which register to use.

```
#include "inc/hw_GPIO.h"
```
hw_gpio has the set of values needed to unlock the pins that are locked to specific functions on the microcontroller.

```
#include "driverlib/gpio.h"
```
gpio has the functional values for the GPIO pins as well as the functions needed to do the enabling of the PWM on the pins.

```
#include "driverlib/pin_map.h"
```
pinmap has the functions available per pin for many Texas Instruments microcontrollers, include the TM4C123.

```
#include "driverlib/pwm.h"
```
Pwm has the functions needed to enable the PWM module within the chip.

```
#include "driverlib/sysctl.h"
```
Sysctl, or system control, has the drivers needed by the chip to access the modules, such as the GPIO modules or the pwm modules.

Enabling Functionality

Next is enabling the modules themselves. Each function used takes 1 or more unsigned 32 bit integers, this type of value is denoted uint32_t. This document uses the natural language strings rather than the integer values themselves for ease of use and readability. These strings are then converted to the integer values defined within the library. All of the string values used can be found within each functions library.

```
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
SysCtlPeripheralEnable(SYSCTL_PERIPH_PWM1);
```
This function enables the peripherals. It takes a value in the form of a pre-defined string found within sysctl.h. For the purposes of enabling PWM on the pins, SYSCTL_PERIPH_GPIOF is used to enable the F set of GPIO pins and SYSCTL_PERIPH_PWM1 is used to enable PWM1, which is the PWM module that can be used by the F GPIO pins.

```
SysCtlPWMClockSet(SYSCTL_PWMDIV_1);
```
This function allows the default clock tied to the PWM to be divided. SYSCTL_PWMDIV_2 would divide the clock by 2, SYSCTL_PWMDIV_4 would divide the clock by 4, etc.

```
HWREG(GPIO_PORTF_BASE + GPIO_O_LOCK) = GPIO_LOCK_KEY;
HWREG(GPIO_PORTF_BASE + GPIO_O_CR)   |= 0x01;
```
This function allows the lock placed on the pins of the GPIO module to be removed so that additional functionality may be used, in this case PWM. Without this, the pin will not be available for use by any other module besides the module it is linked with.

```
GPIOPinConfigure(GPIO_PF0_M1PWM4);
GPIOPinConfigure(GPIO_PF1_M1PWM5);
GPIOPinConfigure(GPIO_PF2_M1PWM6);
GPIOPinConfigure(GPIO_PF3_M1PWM7);
```
This function configures to use a specific pin on the PWM module. GPIO_PF0_M1PWM4 reads as GPIO pin F0, PWM module 1 pin 4. This ties the pin 4 output of the PWM module to the GPIO pin 4.

```
GPIOPinTypePWM(GPIO_PORTF_BASE, GPIO_PIN_0 | GPIO_PIN_1 | GPIO_PIN_2 |
GPIO_PIN_3);
```
This function sets the functionality of the GPIO pin within the GPIO module. The “|” character is a logical OR. The pins relate to binary values. An example would be the values 1001 and 1010. The OR will choose a 1 if either value has a 1 in that position or a 0 if both are 0. For this example the output would be 1011. As this function only takes 1 value for multiple pins, the OR is required.

Configuring PWM

Next is to enable the pulse width modulation functionality within the microcontroller.

```
PWMGenConfigure(PWM1_BASE, PWM_GEN_2, PWM_GEN_MODE_DOWN |
PWM_GEN_MODE_NO_SYNC);
PWMGenConfigure(PWM1_BASE, PWM_GEN_3, PWM_GEN_MODE_DOWN |
PWM_GEN_MODE_NO_SYNC);
```

This function configures the PWM generator on the module. The generator creates the pulses needed for PWM output. PWM1_BASE is the base of the module being used. PWM_GEN_2 is the generator for the module tied to PMW pins 4 and 5, PWM_GEN_3 is for pins 6 and 7. PWM_GEN_MODE_DOWN counts the values for the generator down, creating a saw-tooth wave whereas PWM_GEN_MODE_UP_DOWN would create a triangle wave. PWM_GEN_MODE_NO_SYNC is used so that the generator gives immediate updates rather than synchronizing with the clock to update.

```
PWMGenPeriodSet(PWM1_BASE, PWM_GEN_2, 400);
PWMGenPeriodSet(PWM1_BASE, PWM_GEN_3, 400);
```

This function sets the period of the PWM output. PWM1_BASE is the base of the module being used. PWM_GEN_2 is the generator for the module tied to PMW pins 4 and 5, PWM_GEN_3 is for pins 6 and 7. 400 in this example is the period, measured in clock ticks. 400 would be 400 clock ticks as a period.

```
PWMPulseWidthSet(PWM1_BASE, PWM_OUT_4, 300);
PWMPulseWidthSet(PWM1_BASE, PWM_OUT_5, 300);
PWMPulseWidthSet(PWM1_BASE, PWM_OUT_6, 300);
PWMPulseWidthSet(PWM1_BASE, PWM_OUT_7, 300);
```

This function sets the pulse width, or duty cycle of the output to the PWM pins. PWM1_BASE is the base of the module being used. PWM_OUT_4 is the PWM output to modify, which is tied to a pin. 300 is the clock ticks of the width of the pulse. 300 would be a 75% duty cycle for the 400 clock tick period.

```
PWMGenEnable(PWM1_BASE, PWM_GEN_2);
PWMGenEnable(PWM1_BASE, PWM_GEN_3);
```

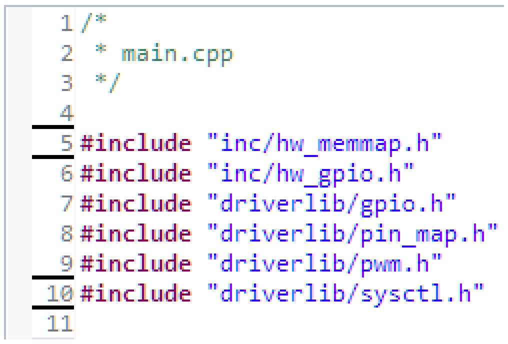
This function enables the PWM generator. Now that the previous values have been set, the generator may now be turned on. PWM1_BASE is the base of the module being used. PWM_GEN_2 is the generator for the module tied to PMW pins 4 and 5, PWM_GEN_3 is for pins 6 and 7.

```
PWMOutputState(PWM1_BASE, (PWM_OUT_4_BIT | PWM_OUT_5_BIT | PWM_OUT_6_BIT |
PWM_OUT_7_BIT  ), true);
```

This function enables the PWM module to start modify the pins set for PWM output. With all of the previous values set, the output to the physical pins of the microcontroller can be turned on. PWM1_BASE is the base of the module being used. PWM_OUT_4_BIT | PWM_OUT_5_BIT | PWM_OUT_6_BIT | PWM_OUT_7_BIT are the bits needed to OR together for this single value. true or false in the last field determines if the signal is enabled or disabled.

Example

This example shows the locations of the library includes and example locations for enabling functionality and configuring the PWM.

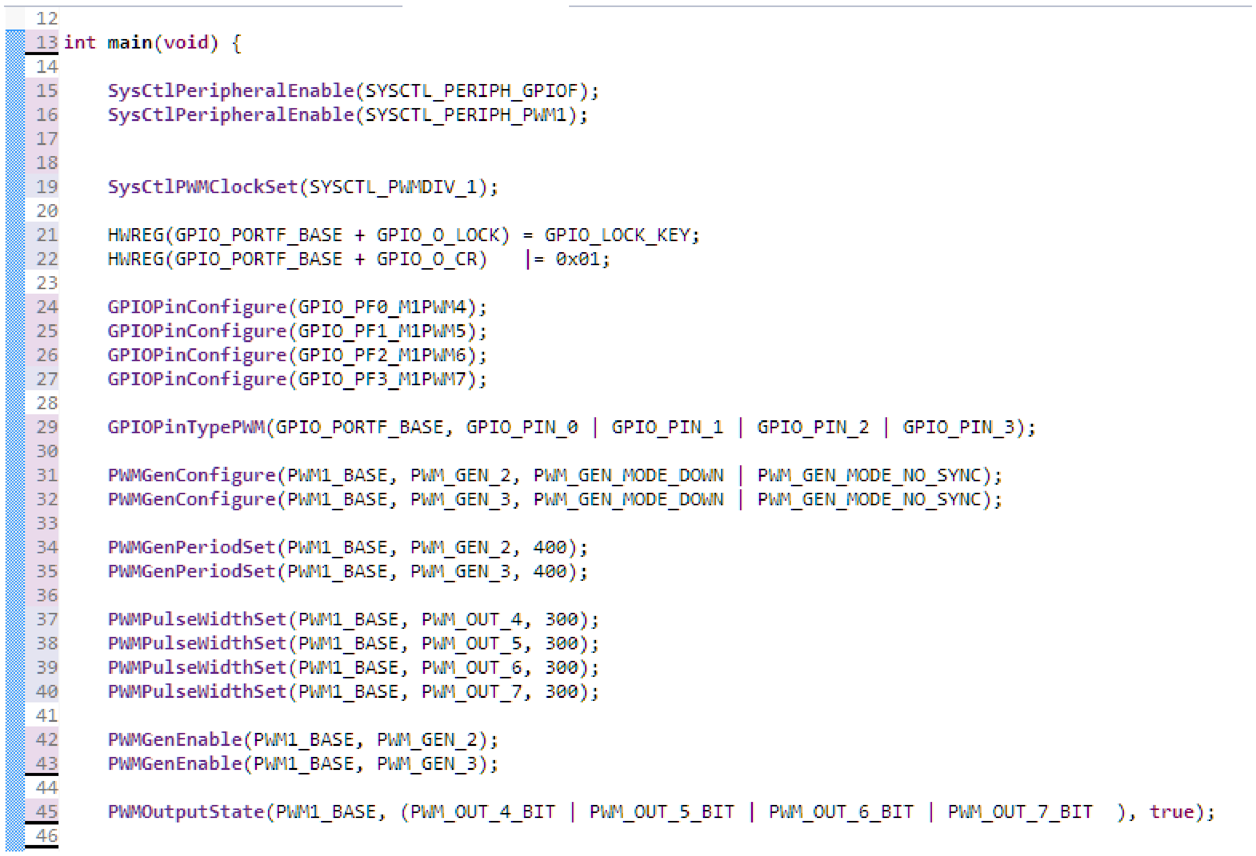


```
/*
 * main.cpp
 */

#include "inc/hw_memmap.h"
#include "inc/hw_gpio.h"
#include "driverlib/gpio.h"
#include "driverlib/pin_map.h"
#include "driverlib/pwm.h"
#include "driverlib/sysctl.h"

```

Figure 1: The includes at the top of the main program



```

int main(void) {

    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_PWM1);


    SysCtlPWMClockSet(SYSCTL_PWMDIV_1);

    HWREG(GPIO_PORTF_BASE + GPIO_O_LOCK)  = GPIO_LOCK_KEY;
    HWREG(GPIO_PORTF_BASE + GPIO_O_CR)    |= 0x01;

    GPIOPinConfigure(GPIO_PF0_M1PWM4);
    GPIOPinConfigure(GPIO_PF1_M1PWM5);
    GPIOPinConfigure(GPIO_PF2_M1PWM6);
    GPIOPinConfigure(GPIO_PF3_M1PWM7);

    GPIOPinTypePWM(GPIO_PORTF_BASE, GPIO_PIN_0 | GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3);

    PWMGenConfigure(PWM1_BASE, PWM_GEN_2, PWM_GEN_MODE_DOWN | PWM_GEN_MODE_NO_SYNC);
    PWMGenConfigure(PWM1_BASE, PWM_GEN_3, PWM_GEN_MODE_DOWN | PWM_GEN_MODE_NO_SYNC);

    PWMGenPeriodSet(PWM1_BASE, PWM_GEN_2, 400);
    PWMGenPeriodSet(PWM1_BASE, PWM_GEN_3, 400);

    PWMPulseWidthSet(PWM1_BASE, PWM_OUT_4, 300);
    PWMPulseWidthSet(PWM1_BASE, PWM_OUT_5, 300);
    PWMPulseWidthSet(PWM1_BASE, PWM_OUT_6, 300);
    PWMPulseWidthSet(PWM1_BASE, PWM_OUT_7, 300);

    PWMGenEnable(PWM1_BASE, PWM_GEN_2);
    PWMGenEnable(PWM1_BASE, PWM_GEN_3);

    PWMOutputState(PWM1_BASE, (PWM_OUT_4_BIT | PWM_OUT_5_BIT | PWM_OUT_6_BIT | PWM_OUT_7_BIT  ), true);

```

Figure 2: Enabling the functions within the main program

With PWM enabled, the output can be toggled on and off or the width and period can be dynamically adjusted. In this example, the duty cycle is being adjusted in a loop. SysCtlDelay adds a delay in clock ticks to the loop.

```
while(1){
  PWMPulseWidthSet(PWM1_BASE, PWM_OUT_4,300);
  PWMPulseWidthSet(PWM1_BASE, PWM_OUT_5,300);
  PWMPulseWidthSet(PWM1_BASE, PWM_OUT_6,300);
  PWMPulseWidthSet(PWM1_BASE, PWM_OUT_7,300);
  SysCtlDelay(10000);

  PWMPulseWidthSet(PWM1_BASE, PWM_OUT_4,100);
  PWMPulseWidthSet(PWM1_BASE, PWM_OUT_5,100);
  PWMPulseWidthSet(PWM1_BASE, PWM_OUT_6,100);
  PWMPulseWidthSet(PWM1_BASE, PWM_OUT_7,100);
  SysCtlDelay(10000);
}
```

Resources

Texas Instruments TivaWare Peripheral Driver Library:

http://www.ti.com/lit/ug/spmu298a/spmu298a.pdf

Texas Instruments Tiva TM4C123GH6PM Microcontroller Data Sheet:

http://www.ti.com/lit/ds/spms376e/spms376e.pdf

Texas Instruments Code Composer Studio:

http://www.ti.com/tool/ccstudio