EECS 388

LABORATORY EXERCISE VII

PULSE WIDTH MODULATOR & SERVOS

November 20, 2018

Benjamin Streit

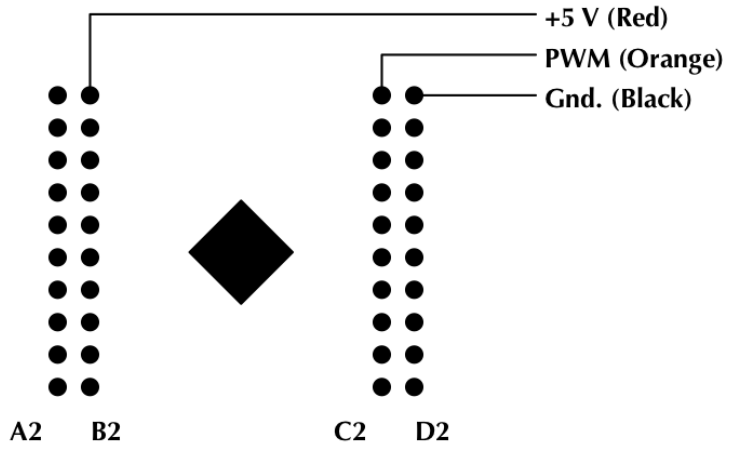
GTA: Ibikunle Oluwanisola

LABORATORY OVERVIEW

This laboratory involves the writing, compiling, and demonstrating of a new task on the Tiva TM4C1294 evaluation board. This new task configures a pulse width modulator (PWM) to drive a small servo.

BACKGROUND

Servos require three connections, ground (often noted `Gnd`), power (`5.0 V` in this case), and a PWM signal. The servo is connected to the evaluation board via three pins in the BoosterPack #2 position. See below schematic for appropriate implementation.



PROCEDURE & RESULTS

Design of the task `Task_PWM.c` follows a simple and efficient procedure. This procedure is as follows:

(i) Defining parameters for PWM.

(ii) Defining and instancing an `enum` for the PWM states. These states include `PWM_Low`, `PWM_Neutral`, and `PWM_High`.

(iii) Initializing required peripherals. This includes `M0PWM5`, and GPIO pin `G<1>`.

(iv) Looping indefinitely, setting PWM pulse width according to current PWM state, with a pause of `1000 ms` each iteration.

The setting of the PWM pulse width in relation to the current PWM state procedure is as follows:

(i) Case `PWM_Neutral`: If `PWM_rising` is `true` then PWM pulse width is set to the high count, and `PWM_CurrentState` is set to `PWM_High`. Else PWM pulse width is set to the low count, and `PWM_CurrentState` is set to `PWM_Low`.

(ii) Case `PWM_High`: PWM pulse width is set to the neutral count, and `PWM_CurrentState` is set to `PWM_Neutral`.

(iii) Case `PWM_Low`: PWM pulse width is set to the neutral count, and `PWM_CurrentState` is set to `PWM_Neutral`.

ANALYSIS

For this laboratory, the PWM period had to be computed during step `(i)` of the procedure. This computation is relatively trivial once each parameter is clearly defined. The system clock period for the evaluation board is `8.33 ns`, and the clock period for the PWM on the evaluation board is 64 times longer than the system clock. The requested period for the PWM signal is `20 ms`. The computations are as follows:

$$T = (8.33 \cdot 10^{-9}) \cdot 64$$

$$T = 5.3312 \cdot 10^{-7}$$

$$P = \frac{20}{5.3312 \cdot 10^{-7}}$$

$$P = 37515\text{ms}$$

With the PWM period computed, it becomes trivial to define `PWM_Low_Count`, `PWM_Neutral_Count`, and `PWM_High_Count`, and therefore alternate between `1.0ms`, `1.5ms`, `2.0ms`, and back to `1.5ms` pulses as seen in the code.

CONCLUSIONS

The logic written for this laboratory exercise is sensible and effective, as all desired outcomes were achieved and in an appropriate manner. However, improvements could be made.

Currently, the cases for `PWM_High` and `PWM_Low` implement the exact same functionality. This repeated code is not ideal and should be avoided if at all possible. Perhaps combining the `PWM_High` and `PWM_Low` states into a singular `PWM_Active` state would avoid this issue and produce cleaner, more efficient, and more readable code.

CODE

```
#include "inc/hw_ints.h"
#include "inc/hw_memmap.h"
#include "inc/hw_sysctl.h"
#include "inc/hw_types.h"

#include <stddef.h>
#include <stdbool.h>
#include <stdint.h>
#include <stdarg.h>

#include "driverlib/sysctl.h"
#include "driverlib/pin_map.h"
#include "driverlib/gpio.h"
#include "driverlib/pwm.h"

#include "Drivers/UARTStdio_Initialization.h"
#include "Drivers/uartstdio.h"

#include "Drivers/Processor_Initialization.h"

#include "Tasks/Task_ReportData.h"
#include "stdio.h"
#include "FreeRTOS.h"
#include "task.h"

//
// Defines
//
#define PWM_Period 1000
#define PWM_State_Low 50
#define PWM_State_High 100
#define PWM_State_Neutral 75

//
```

```
35 // PWM Parameters
36 //
37 #define PWM_0_4_Period 37515
38 #define PWM_0_4_Low_Count ((PWM_0_4_Period * PWM_State_Low) / PWM_Period)
39 #define PWM_0_4_High_Count ((PWM_0_4_Period * PWM_State_High) / PWM_Period)
40 #define PWM_0_4_Neutral_Count ((PWM_0_4_Period * PWM_State_Neutral) / PWM_Period)
41
42 typedef enum { PWM_Low, PWM_Neutral, PWM_High } PWM_States;
43 PWM_States PWM_CurrentState = PWM_Neutral;
44
45 //
46 // Task initialization
47 //
48 extern void Task_PWM( void *pvParameters ) {
49     //
50     // Initialize UART
51     //
52     UARTStdio_Initialization();
53     UARTprintf( "FreeRTOS Starting!\n" );
54
55     //
56     // Constants and Variables
57     //
58     uint32_t DutyCycleDelta = pdMS_TO_TICKS( 1000 );
59
60     bool PWM_rising = true;
61
62     SysCtlPeripheralEnable( SYSCTL_PERIPH_GPIOG );
63     SysCtlPeripheralEnable( SYSCTL_PERIPH_PWM0 );
64
65     GPIOPinConfigure( GPIO_PG1_M0PWM5 );
66     GPIOPinTypePWM( GPIO_PORTG_BASE, GPIO_PIN_1);
67
68     PWMClockSet( PWM0_BASE, PWM_SYSCLK_DIV_64 );
69     PWMGenConfigure( PWM0_BASE, PWM_GEN_2, PWM_GEN_MODE_DOWN |
70     PWM_GEN_MODE_NO_SYNC );
71     PWMGenPeriodSet( PWM0_BASE, PWM_GEN_2, PWM_0_4_Period );
72
73     PWMPulseWidthSet( PWM0_BASE, PWM_OUT_5, PWM_0_4_Neutral_Count );
74
75     PWMOutputState( PWM0_BASE, PWM_OUT_5_BIT, true );
76
```

```
77      PWMGenEnable( PWM0_BASE, PWM_GEN_2 );
78
79      while ( 1 ) {
80          // check current state of PWM and change accordingly
81          switch (PWM_CurrentState) {
82              case PWM_Neutral: {
83                  if (PWM_rising) {
84                      PWM_CurrentState = PWM_High;
85                      PWMPulseWidthSet( PWM0_BASE, PWM_OUT_5, PWM_0_4_High_Count );
86                      PWM_rising = false;
87                  } else {
88                      PWM_CurrentState = PWM_Low;
89                      PWMPulseWidthSet( PWM0_BASE, PWM_OUT_5, PWM_0_4_Low_Count );
90                      PWM_rising = true;
91                  }
92                  break;
93              }
94              case PWM_High: {
95                  PWM_CurrentState = PWM_Neutral;
96                  PWMPulseWidthSet( PWM0_BASE, PWM_OUT_5, PWM_0_4_Neutral_Count );
97                  break;
98              }
99              case PWM_Low: {
100                 PWM_CurrentState = PWM_Neutral;
101                 PWMPulseWidthSet( PWM0_BASE, PWM_OUT_5, PWM_0_4_Neutral_Count );
102                 break;
103             }
104         }
105         //
106         // Task delay.
107         //
108         vTaskDelay( DutyCycleDelta );
109     }
110 }
```