Exercise 3: Timer and Pulse Width Modulation (TPM) Module

EG-252 Group Design Exercise – Microcontroller Laboratory

Dr K. S. (Joseph) Kim Dr Chris P. Jobling

September 2020

This exercise is designed for understanding of the TPM modules of MC9S08AW60 MCU and the practice of the control of TPM registers in C programming. For this exercise you will be provided with two sample C programs, one for generating a delay by the timer function and the other for pulse-width modulation (PWM) signals for driving motor, respectively. Electronic versions of the programs can be downloaded from the BlackBoard. You are to carry out the following tasks with this exercise:

- Use the sample program "tpm_timer.c" to understand configurations of and programming with the TPM module 1 (TPM1) registers for timer function.
- Modify the sample program "tpm_timer.c" to generate a delay by configuring the TPM module 2 (TPM2) registers.
- Use the sample program "tpm_motor.c" to understand how to generate PWM signals to drive two DC motors on the demonstration board with speed and direction control.
- Modify the sample program "tpm_motor.c" to set PWM duty cycle and implement independent motor speed control for the two DC motors.

This exercise is worth 14 Marks. You should keep careful records of your solutions as you will need to upload your completed programmes and answer questions to gain the credits.

You can view this document as a web page HTML, PDF or as a Word Document .docx

I. Task 1: Experiment with Sample Program "tpm_timer.c"

This sample C program is designed to display 5 digits included in the student number 12345 over the LED bar. Each digit is displayed for a configured time, which is determined by the timer modulo registers `TPM1MODH:TPM1MODL` of the TPM1 module (lines 19 and 20). An interrupt is generated upon the timer

overflow, which is configured through the TPM1 status and control register `TPM1SC` (line 18). More details on the registers and their configurations are referred to the Freescale MC9S08AW60 datasheet document, Chapter 10 for Timer/PWM.

Once the timer overflow interrupt is generated, the corresponding interrupt service routine, i.e., `TPM1_overflow()` in this example, will be executed. This interrupt service routine is associated to the timer overflow interrupt by preceding the designated word "interrupt" and followed by the timer overflow interrupt vector "11" in line 28. The interrupt vectors associated with various interrupt sources can be obtained from the Freescale MC9S08AW60 datasheet.

The lines 32 and 33 are a two-step procedure to clear the timer overflow flag (TOF) in the `TPM1SC` register. Note that once the TOF is set, it will not be automatically cleared. If the TOF is not cleared, the timer overflow routine `TPM1_overflow` (line 28) will be continuously called. The following method introduced in the MC9S08AW60 datasheet should be used to clear the overflow flag:

- Line 32: Read the TOF (`TPM1SC_TOF` as we are using the TPM1 module). Note that you can address a register or a specific bit in a register using the name of that register or that bit in a register. Their names can be obtained from the MC9S08AW60 datasheet. For sample, if you want to read the content of TPM1SC register, you can use "`varTOF = TPM1SC;` if you want to read the content of THE TOF flag in the `TMP1SC` register, you can simply use"`varTOF = TPM1SC_TOF`". The registers and flags that can be recognized by CodeWarrior are highlighted by blue colour in the editor window.
- Line 33: Write a zero to the overflow flag TOF by "`TPM1SC_TOF = 0`"

```
1  #include <hidef.h>        // for EnableInterrupts macro
2  #include "derivative.h" // include peripheral declarations
3
4  #define VNtpm1ovf 11 // Interrupt vector for timer 1 overflow
5
6  byte tof_cnt, period;
7  byte student_num[6] = {1, 2, 3, 4, 5, 6};
8  byte digit_cnt = 6;
9
10 void main(void)
11 {
12     SOPT = 0x00; // disable COP (watchtimer)
13
14     ICGC1 = 0x74; // select external quartz crystal
15
16     /*The AW60 has an internal oscillator which runs at approx. 8 MHz, resulting in a syste
17   If the line above is included in your programme, the AW60 uses an external 4 MHz quartz cr
```

```
 instead, resulting in a system clock of 2 MHz. This is more accurate than the internal osc
 any time intervals defined in your programme will be correct to a fraction of a percent.*/

    // Init_GPIO init code
    PTFDD = 0xFF; // configure port F as outputs for LEDs

    // configure TPM module 1
    TPM1SC = 0x4F; // format: TOF(0) TOIE(1) CPWMS(0) CLKSB(0) CLKSA(1) PS2(1) PS1(1) PS0(1
                   // Turn on the overflow interrupt and set the prescaler to 128.

    TPM1MOD = 0x8000; // set the counter modulo registers to hex 8000 = 32,768 decimal.
        //You do not need to make separate writes to the low and high bytes of
        //a 16-bit register such as TPM1MOD. The file "derivative.h" includes
        //macros so that 16 bit variables are split and are written separately.*/

    tof_cnt = 0; // initialize the number of timer overflow to 0.

    EnableInterrupts; // enable interrupts; from now on the interrupts are active

    for (;;)
    {
    } // loop forever
}

interrupt VNtpm1ovf void TPM1_overflow()
{
    TPM1SC_TOF = 0; //Clear the overflow interrupt flag.

    if (tof_cnt >= digit_cnt)
    {
        tof_cnt = 0; // reset tof_cnt if larger than digit_cnt
    }

    PTFD = student_num[tof_cnt++];
}
```

View on GitHub"

Listing 1. Sample program for TPM timer function.

II. Task 2: Modify the Sample Program to Display your Student Number

For this task you are required to display all the digits in your student number.

- Each digit needs to be displayed for approximately one second. You can control the modulo registers to generate the desired delay.

- Instead of using the TPM1 module, you are required to use the TPM2 module to generate the delay. Your job is to find correct TPM registers to use and configure them accordingly. This task is worth 5 marks.

III. Task 3: Experiment with Sample Program "tpm_motor.c

This sample C program is designed to generate PWM signals to drive two motors. The motor speeds are controlled by the PWM signal duty cycle (i.e., 'on' time with respect to a total period), while the motor directions (brake, forward and reverse etc) can be controlled by the logic levels of output signals to the motors.

With this sample program the motor direction is configured by the setting of the rocker switches. The motor speed is controlled by configuring the modulo registers and the channel value registers. The modulo registers determine the PWM signal period. Upon the timer overflow interrupt the motors are turned on and direction is set by writing to port G where the two motors are connected. Upon the output compare interrupt (when the free-running counter value matches that stored in the channel value registers, an output compare interrupt is generated if an interrupt is enabled for that channel) both motors are turned off.

Note that to drive the motor, an additional power supply is needed, which is available in the laboratory. Connect a 6-volt output to the demonstration board. Compile and run the sample program, you will be able to try different motor directions by setting the rocker switches accordingly.

In this sample program there is a main program (Listing 2: Lines 27–51) and two interrupt service routines (Listing 2: Lines 53–73). Use the MC9S08AW60 datasheet to find out the interrupt vectors for the TPM1 timer interrupt and TPM1 channel 1 interrupt. Again check the MC9S08AW60 datasheet to understand the configuration of the modulo register and IOC register.

```
1  /* TPM_motor.C
2
3  Modified test Programme for new AW60 teaching board thanks to contributions from
4  Dr Tim Davies.
5
6  This version gives direction control of the two motors but with interrupts to
7  give PWM speed control.
8
9  There are two interrupts: the timer overflow for TPM1 timer, which sets the
10 overflow frequency of 100 Hz with bus clock rate 2MHz, and TPM1 channel 1 to
11 turn off the motor. The motors are turned off individually to give individual
12 speed control if required.
13
14 The lower four rocker switches 1-4 on Port A determine the direction of the two
```

```
15  motors, e.g. 00001010 is reverse both motors, 00000101 is forward both motors,
16  00000000 is braked.
17
18  */
19
20  #include <hidef.h>      // for EnableInterrupts macro
21  #include "derivative.h" // include peripheral declarations
22
23  #define T1ovf 11 // Interrupt vector for timer 1 overflow
24  #define T1C1 6   // Interrupt vector for timer 1 channel 1
25
26  //Define a 16 bit value to write directly to the modulus
27  #define REPEAT 0x4E20 // Timer repeat rate 0x4E20 (20,000 decimal)
28  //Similarly, lets define the pulse width as a 16 bit value.
29  #define INITPW 0x3000 // Initial pulse width 0x3000 (12,000 approx)
30
31  byte drive;
32
33  void main(void)
34  {
35      SOPT = 0x00; // disable COP (watchtimer)
36
37      ICGC1 = 0x74; // Select external 4 MHz quartz crystal.
38
39      // Init_GPIO init code
40      PTADD = 0x00; // set port A as inputs for the rocker switches.
41      PTAPE = 0xFF; // turn on the pullups for port A
42      PTFDD = 0xFF; // set port F as outputs for LEDs
43      PTGDD = 0xFF; // set port G as outputs for motor drive where motors are connected.
44
45      // configure TPM module 1
46      TPM1SC = 0x48;    // format: TOF(0) TOIE(1) CPWMS(0) CLKSB(0) CLKSA(1) PS2(0) PS1(0) PS
47      TPM1MOD = REPEAT; // write the 16-bit value REPEAT to the modulus register
48
49      // configure TPM1 channel 1
50      TPM1C1SC = 0x50;  // TPM1 Channel 1 interrupt enabled, output compare, no external outpu
51      TPM1C1V = INITPW; //write 16-bit value INTPW to the channel 1 register
52
53      EnableInterrupts; // enable interrupts
54
55      for (;;)
56      {
57          drive = PTAD & 0x0F; // read the motor direction settings from the rocker switches
58          PTFD = drive;
59      } // loop forever
60  }
```

```
61
62  interrupt T1ovf void TPM1SC_overflow()
63  { // interrupt vector: Vtpm1
64
65      TPM1SC_TOF = 0; // clear the overflow interrupt flag
66
67      PTGD = drive; // turn on motors as configured by drive (port A switches).
68  }
69
70  interrupt T1C1 void TPM1C1SC_int()
71  { // interrupt vector: Vtpm1ch1
72
73      TPM1C1SC_CH1F = 0; // clear the channel 1 interrupt flag
74
75      PTGD = PTGD | 0x0F; // set free-wheel mode for both motors instead of turn off
76  }
```

View on GitHub

IV. Task 4: PWM Duty Cycle and Motor Speed Control

For Task 4 you are required to modify the sample program in order to:

- Set the duty cycle of the two motors to xx%, *where* xx *is the leftmost two digits of your student number. For example, if your student number is 567890, you should set the duty cycle to 56%. In the sample program, only TPM1 channel 1 is used to control the duty cycle (and motor speed) for both motors, resulting identical speed for the two motors.* This part is worth 4 marks*.
- Enable independent speed control of the two motors*: Use the leftmost two digits of your student number set the duty cycle of motor 1 and the rightmost two digits to set the speed of the second motor. For example, if your student number is 123450, you should set the duty cycle of two motors to 12% and 50%, respectively. You are expected to utilize another TPM1 channel (e.g. channel 0). Therefore two TPM1 channels are available for speed control, one channel per each motor. You should introduce another interrupt service routine for the new TPM1 channel and make changes to the interrupt service routine for TPM1 channel 1. *This part is worth 5 marks.*

Listing 2 Interrupt service routines for timer overflow interrupt and output compare interrupt.

Appendix

Theory of DC Motor Speed Control

The speed of a DC motor is directly proportional to the supply voltage, so if we reduce the supply voltage from 12 volts to 6 volts, the motor will run at half the speed. How can this be achieved when the battery is fixed at 12 volts? The speed controller works by varying the average voltage sent to the motor. It could do this by simply adjusting the voltage sent to the motor, but this is quite inefficient to do. A better way is to switch the motor's supply on and off very quickly. If the switching is fast enough, the motor doesn't notice it, it only notices the average effect.

When you watch a film in the cinema, or the television, what you are actually seeing is a series of fixed pictures, which change rapidly enough that your eyes just see the average effect - movement. Your brain fills in the gaps to give an average effect. Now imagine a light bulb with a switch. When you close the switch, the bulb goes on and is at full brightness, say 100 watts. When you open the switch it goes off (0 watt). Now if you close the switch for a fraction of a second, then open it for the same amount of time, the filament won't have time to cool down and heat up, and you will just get an average glow of 50 watts. This is how lamp dimmers work, and the same principle is used by speed controllers to drive a motor. When the switch is closed, the motor sees 12 volts, and when it is open it sees 0 volt. If the switch is open for the same amount of time as it is closed, the motor will see an average of 6 volts, and will run more slowly accordingly.

As the amount of time that the voltage is on increases compared with the amount of time that it is off, the average speed of the motor increases. This is the principle of switch mode speed control. Thus the speed is set by PWM.[1]

[1] Refer to Speed Controllers by Paul Hills for more details.