Embedded Systems Interfacing Assignment 05

Due: 02 October 2013

Background

Either Timer2 or Timer 3 may be used to provide the timing signals for the PWM waveform. This homework will use Timer 2. Only T2CON need be configured and the register bits are summarized below:

TMR2Register

Upper Byte:

TON	−−	TSIDLE	−−	−−	−−	−−	−−
bit15							bit 8

Lower Byte:

−−	TGATE	TCKPS1	TCKPS0	T32	−−	TCS	−−
bit 7							bit 0

Table 1: TMR2CON Register

bit 15	TON: Timer2 On bit When T2CON<3> = 1: 1 = Starts 32-bit Timer2/3 0 = Stops 32-bit Timer2/3 When T2CON<3> = 0: 1 = Starts 16-bit Timer2 0 = Stops 16-bit Timer2
bit 14	Unimplemented: Read as 0
bit 13	TSIDL: Stop in Idle Mode bit 1 = Discontinue module operation when device enters Idle mode 0 = Continue module operation in Idle mode
bit 12-7	Unimplemented: Read as 0
bit 6	TGATE: Timer2 Gated Time Accumulation Enable bit When TCS = 1: This bit is ignored.

When TCS = 0:
1 = Gated time accumulation enabled
0 = Gated time accumulation disabled

bit 5-4	TCKPS1:TCKPS0: Timer2 Input Clock Prescale Select bits
	11 = 1:256
	10 = 1:64
	01 = 1:8
	00 = 1:1
bit 3	T32: 32-bit Timer Mode Select bit
	1 = Timer2 and Timer3 form a single 32-bit timer
	0 = Timer2 and Timer3 act as two 16-bit timers

Note: In 32-bit mode, T3CON control bits do not affect 32-bit timer operation.

bit 2	Unimplemented: Read as 0
bit 1	TCS: Timer2 Clock Source Select bit
	1 = External clock from pin T2CK (on the rising edge)
	0 = Internal clock (FOSC/2)
bit 0	Unimplemented: Read as 0

OC1CON Register

Upper Byte:

−−	−−	OCSIDL	−−	−−	−−	−−	−−
bit15							bit 8

Lower Byte:

−−	−−	−−	OCFLT	OCTSEL	$OCM2$	OCM1	OCM0
bit 7							bit 0

Table 2: OC1CON Register

bit 15-14	Unimplemented: Read as 0
bit 13	OCSIDL: Stop Output Compare 1 in Idle Mode Control bit
	1 = Output Compare 1 will halt in CPU Idle mode
	0 = Output Compare 1 will continue to operate in CPU Idle mode
bit 12-5	Unimplemented: Read as 0
bit 4	OCFLT: PWM Fault Condition Status bit
	1 = PWM Fault condition has occurred (cleared in HW only)
	0 = No PWM Fault condition has occurred (this bit is only used when OCM¡2:0¿ = 111)
bit 3	OCTSEL: Output Compare 1 Timer Select bit
	1 = Timer3 is the clock source for Output Compare x
	0 = Timer2 is the clock source for Output Compare x

Note: Refer to the device data sheet for specific time bases available to the output compare module.

bit 2-0	OCM2:OCM0: Output Compare x Mode Select bits
	111 = PWM mode on OC1, Fault pin enabled
	110 = PWM mode on OC1, Fault pin disabled
	101 = Initialize OC1 pin low, generate continuous output pulses on OC1 pin
	100 = Initialize OC1 pin low, generate single output pulse on OC1 pin
	011 = Compare event toggles OC1 pin
	010 = Initialize OC1 pin high, compare event forces OC1 pin low
	001 = Initialize OC1 pin low, compare event forces OC1 pin high
	000 = Output compare channel is disabled

PWM Mode

In Pulse Width Modulation (PWM) mode, the OC1 (RD0) pin produces up to a 16-bit resolution PWM output. A PWM waveform has a fixed period and a variable on-time or duty cycle. The PIC24FJ128CA010 in PWM mode produces the waveform illustrated to Figure 1. When the TMR2 is the same as the PR2 register the output is set to a one. At the same time TMR2 is cleared to zero. When OC1R is equal to TMR2 the output is reset to 0. Eventually TMR2 counts to PR2 the output is set to 1 and TMR2 is cleared to zero. This repeats the sequence.

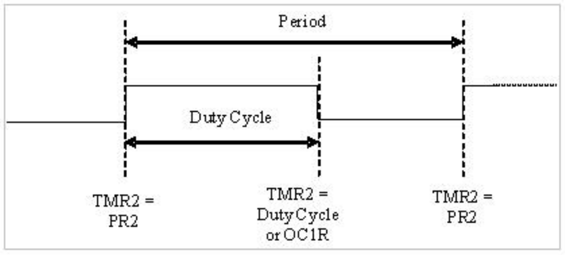

Figure 1: PWM Output.

The value written to the PR2 register specifies the PWM period. The PWM period can be calculated using

$$PWM\_period = (PR2 + 1) \cdot Tcyc \cdot TMR2\_prescale\_value. \tag{1}$$

The PWM frequency is defined as

$$F_{PWM} = \frac{1}{PWM\_period} \tag{2}$$

When TMR2 is equal to PR2, the following three events occur on the next increment cycle:

- TMR2 is cleared to zero

- The OC1/RD0 pin is set to 1 except when the PWM duty cycle is 0%.
- The current OC1RS is copied to OC1R

The last event guaranties that the duty cycle is not changed mid PWM period. Note the postscaler is used in determining the PWM period and duty cycle.

The PWM duty cycle is specified by writing to the OC1R register. Up to 16-bit resolution is available. This 16-bit value is represented by OC1R<15:0>. The following equation is used to calculate the PWM period in time:

$$\begin{aligned} PWM\_duty\_cycle &= \frac{OC1R \cdot Tcyc \cdot (TMR2\_prescale\_value)}{(PR2+1) \cdot Tcyc \cdot TMR2\_prescale\_value} \\ PWM\_duty\_cycle &= \frac{OC1R}{PR2+1} \end{aligned} \tag{3}$$

OC1RS can be written at any time, but the duty cycle value is not latched into OC1R until after a match between PR2 and TMR2 occurs (i.e., the period is complete).

The OC1RS register is used to double buffer the PWM duty cycle. This double buffering is essential for glitchless PWM operation.

When the OC1R matches TMR2 the OC1 pin is cleared.

The maximum PWM resolution, in bits, for a given PWM frequency is given by

$$PWM\_Resolution = \frac{\log_{10}\left(\frac{F_{CY}}{F_{PWM} \cdot (TimerPrescaleValue)}\right)}{\log_{10}(2)} \tag{4}$$

When PWM Resolution is greater than 16, you should round down to 16. If the PWM duty cycle value is longer than the PWM period, than the output is never reset to zero. As a result OC1/RD0 will remain at 1.

Problem 01: PWM Functions

Write a function $initPWM1$ that initializes $OC1CON$, $T2CON$ and $PR2$. $initPWM1$ should also clear timer 2. This function is declared as

void initPWM1(unsigned int period);

where period is the value to be loaded into $PR2$. The following steps should be taken when configuring the OC1 module for PWM operation:

1. Turn off the timer by clearing TON, (T2CON<15>) = 0.
2. Set the PWM period by writing to the selected Timer Period register (PR2).
3. Initialize the PWM duty cycle by writing $0x0000$ to the OC1RS and OC1R registers.

4. Enable interrupts, if required, for the timer and output compare modules.
5. Configure the output compare module for one of two PWM operation modes by writing to the Output Compare mode bits OCM<2:0> (OC1CON<2:0>). We will use without fault pin.
6. Set the TMR2 prescale value and enable the time base by setting TON, (T2CON<15>) = 1.

For this problem, assume 1 : 1 prescaling, and that the user will supply a period that is not in excess of 65, 535. Note that the period has to do with the value to which Timer 2 is counting not a period in terms of seconds.

Write a function called $dutyCyclePWM()$. This function returns nothing and is declared below.

```
void dutyCyclePWM (unsigned int dutyCycle);
```

where dutyCycle is the value loaded into $OC1RS$ and should always be less than or equal to PR2.

Test these functions with a main program. After initialization of PWM1 using $initPWM(0x400)$;, $main()$ is to fall into an endless loop that contains the statement $dutyCyclePWM(0x100)$; What is the duty cycle? Simulate this system, showing $OC1/RD0$.

Problem 02: Input-Controlled PWM

Modify the initialization section of $main()$ so that $TMR3$ causes an interrupt every $100ms$ by initializing $T3CON$ and $PR3$. Clear $\_T3IF$ and enable $\_T3IE$. Use the interrupt routine Timer 3 interrupt shown below. This function is called each time the $TMR3$ reloads $TMR3$ from $PR3$. The routine starts an analog-to-digital sample and clears the Timer 3 interrupt flag.

```
void __attribute__((interrupt,no_auto_psv)) _T3Interrupt (void)
{
    _T3IF=0; //clear interrupt
    AD1CON1bits.SAMP=1; //start sampling
}
```

Modify the initialization section of main( ) to clear _AD1IF and enable _AD1IE. Add a second interrupt service routine that reads the analog-to-digital conversion when done and places the result in a volatile global variable called rawSpeed. Then clear the interrupt flag.

```
volatile unsigned int rawSpeed;

void __attribute__((interrupt,no_auto_psv)) _ADC1Interrupt (void)
{
    rawSpeed = ADC1BUF0 & 0x3FF; //read and mask conversion
    ADC1CON1bits.DONE=0; //Clear DONE
    _AD1IF=0; //clear interrupt
}
```

Use your existing code in PERIPERALS.C to initialize the ADC to channel 5, PWM period to $0x3FF$, and the bargraph. Set the PWM duty cycle to $rawSpeed$ and the upper 8-bits of $rawSpeed$ to the bargraph. Return to the top of the endless loop.