Microprocessors B (17.384)

Spring 2011

Lecture Outline

Class # 09

March 29, 2011

Dohn Bowden

Today's Lecture

- Administrative
- Microcontroller Hardware and/or Interface
  - Pulse Width Modulation
- Programming/Software
- Lab
- Homework

Course Admin

Administrative

- Admin for tonight …
  - Syllabus Highlights
    - Lab #3 is now due – April 5th
  - We shall continue Lab #3 and / or start Lab #4 this week
  - Course Project … did you submit your topic via email?

Syllabus Review

Week	Date	Topics	Lab	Lab Report Due
1	01/25/11	PIC pin out, C programming, Watchdog Timer, Sleep		
2	02/01/11	General-purpose IO, LED/switch IO, FSM	1	
3	02/08/11	Lab	1 con't	
4	02/15/11	Interrupts, Timers, interrupt-driven IO	2	
5	02/22/11	Lab	2 con't	**1**
6	03/01/11	Asynchronous and Synchronous Serial IO (UART, I^2C, SPI)	3	
7	03/08/11	**Examination 1**		
X	03/15/11	**No Class – Spring Break**		
8	03/22/11	Lab	3 con't	2
9	03/29/11	**Pulse Width Modulation and DC motor control**	**3,** 4	
10	04/05/11	Lab	4 con't	**3**
11	04/12/11	Advanced Hardware Topics	Project	
12	04/19/11	**Examination 2**		
13	04/26/11	Work on Course Project	Project	**4**
14	05/03/11	Final Exam/Course Project Brief and Demonstration	Demo	

Microcontroller Hardware and / or Interfaces

Timers – Revisited …

General Terms

- F_{OSC} = Clocks signal frequency
- T_{OSC} = Clocks period
- F_{CY} = Instruction execution frequency
- T_{CY} = 1/ F_{CY} Instruction execution period
- Ttmr = Clock period = 1/F_{CY}

General Information

Recall that a Timer is just a counter. Time can be converted from elapsed Timer Ticks (*Ticks*) by multiplying by the clock period (*Ttmr*) of the timer:

Time = Ticks x Ttmr

If a timer is a 16-bit timer, and it is clocked at the FCY = 40 MHz, then will count from 0x0000 to 0xFFFF (65536 ticks) in:

Time = 65536 x (1/40 MHz)

= 65536 x 25 ns = 1638400 ns = 1638.4 us = 1.6384 ms

Timers

- Recall that a **timer** on a μC is simply a counter

Basic equations that we have used:

General:

Time = Ticks * Clock period of counter
Ticks = Time / Clock period of counter

PIC24 specific:

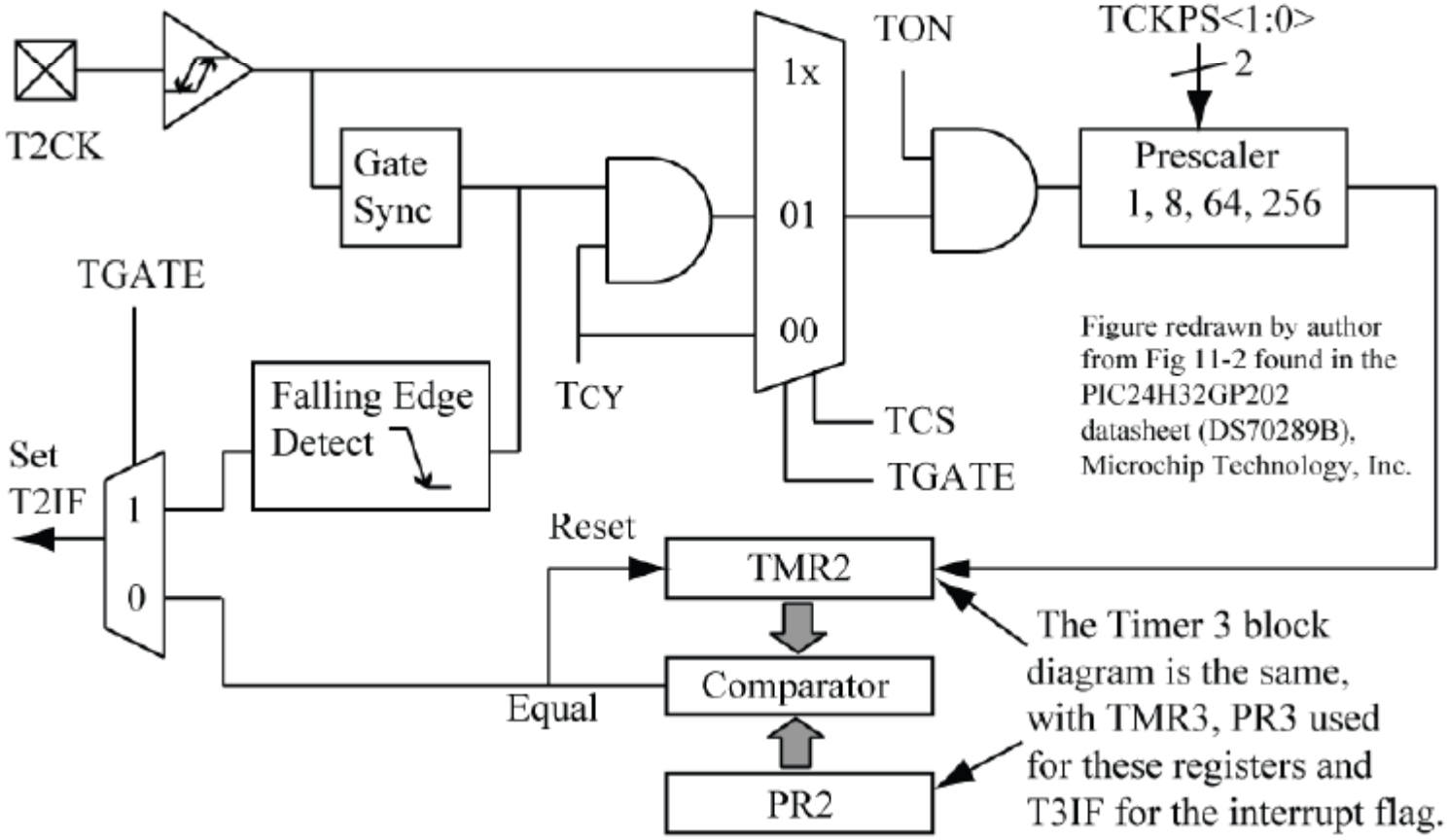
Time = Ticks * timer_prescale / FCY
Ticks = Time * FCY / timer_prescale

Period Register

Recall that the timer **period register** controls the amount of time for setting the TxIF flag (controls the Timer roll-over time):

TxIF period = (PRx + 1) * Prescale / FCY

Timer 2 Block Diagram



Timer Module Registers

- Each timer module is a 16-bit timer/counter consisting of the following readable/writable registers …
  - TMRx: 16-bit timer count register
  - PRx: 16-bit period register associated with the timer
  - TxCON: 16-bit control register associated with the timer

Timer Interrupt Control

- Each timer module has interrupt control bits…
  - Interrupt Enable Control bit (TxIE)
  - Interrupt Flag Status bit (TxIF)
  - Interrupt Priority Control bits (TxIP<2:0>)

Prescaler

- PRE = prescaler
  - Values can be …
    - 1:256
    - 1:64
    - 1:8
    - 1:1

T2IF Period

The T2IF flag is set at the following period (T_{t2if}):

$T_{t2if} = (PR2+1) \times PRE \times Tcy = (PR2+1) \times PRE/Fcy$

Observe that because Timer2 is a 16-bit timer, if PR2 is its maximum value of 0xFFFF (65535), and the prescaler is '1', this is just:

$T_{t2if} = 65536 \times 1/Fcy$

We typically want to solve for Tt2if, given a PRE value:

$PR2 = (T_{t2if} \times Fcy / PRE) - 1$

Example T2IF Periods

PR2/PRE Values for T_{t2if} = 15 ms, Fcy = 40 MHz

	PRE=1	**PRE=8**	**PRE=64**	**PRE=256**
PR2	600000	75000	9375	2344
	(invalid)	(invalid)		

The PR2 for PRE=1, PRE=8 are invalid because they are greater than 65535 (PR2 is a 16-bit register).

Configuring Timer2 to interrupt every T_{t2if} period is called a **PERIODIC INTERRUPT**.

Pulse Width Modulation …

Pulse Width Modulation

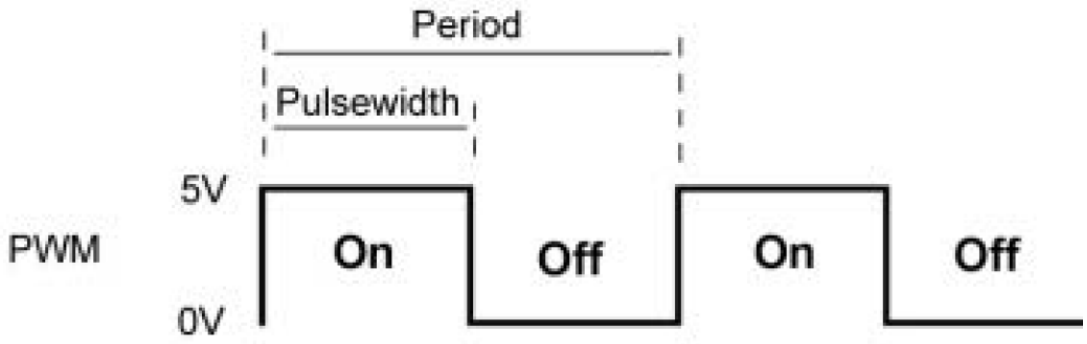
- Pulse Width Modulation (PWM) is a …
  - technique … used in a variety of applications … which …
    - Allows dynamic changing of the DC voltage level to a load

Pulse Width Modulation

- Pulse Width Modulation (PWM) is …
  - Varies the duty cycle of a square wave … while …
    - Keeping the period constant
- Duty cycle = pulse width/period …
    - Where the period remains constant

Duty Cycle

- Duty cycle = [pulse width/period] x 100%



Pulse Width Modulation

- Pulse Width Modulation will enable us to …
  - Generate a modulated signal of …
    - Varying frequency … and …
    - A duty-cycle
  - Which will allow us to vary the voltage across a motor, for example …
    - Thus varying the motor's speed

Pulse Width Modulation

- The voltage across the motor … or … average voltage is …
  - Equal to …

    The voltage supply multiplied by the duty cycle of the PWM signal

Pulse Width Modulation

- The DC voltage is …
  - The average voltage value of the DC pulse across the waveform period
- **For example** … if we have the pulse high for 50% of the period consistently and the pulses are +3.3V … then ….
  - We are supplying (on the average) … 50% of 5 Volts to the load … or …

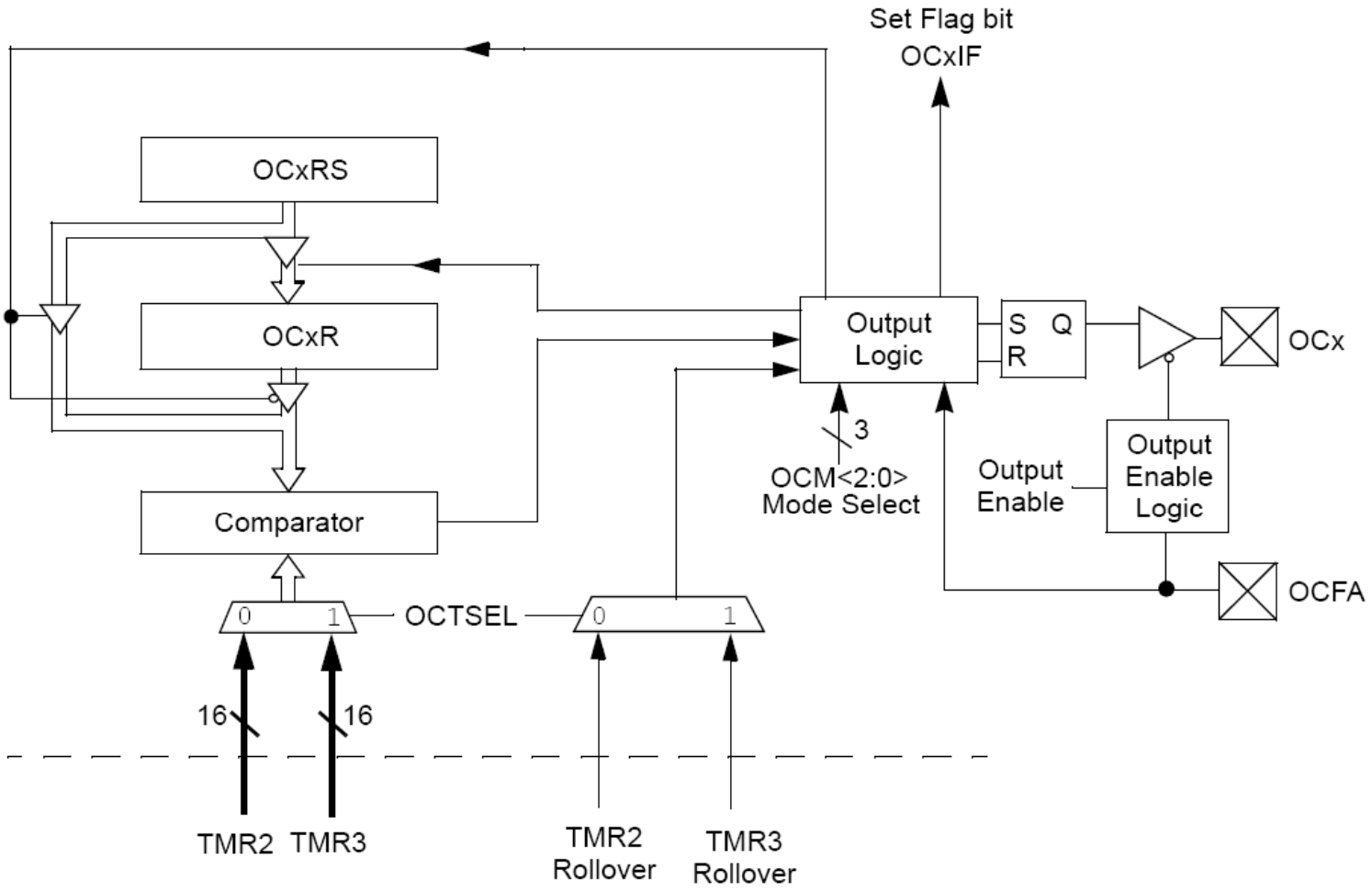
    2.5 Volts

Output Compare Module …

Output Compare Module

- Output Compare module compares …
  - The value of the timer with …
  - The value of one or two compare registers
    - Depending on the operating mode selected

Output Compare Module



Output Compare Module

- The state of the output pin changes when …
  - The timer value matches the compare register value
- The Output Compare module generates either …
  - A single output pulse … or …
  - A sequence of output pulses
    - By changing the state of the output pin on the compare match events
- It can also generate interrupts on compare match events

Output Compare Module

- The Output Compare module has multiple operating modes …
  - Active Low One-Shot mode
  - Active High One-Shot mode
  - Toggle mode
  - Delayed One-Shot mode
  - Continuous Pulse mode
  - PWM mode without fault protection
  - PWM mode with fault protection

Output Compare Module

- We are concerned with the following Output Compare module operating modes …
  - PWM mode without fault protection

PWM mode with fault protection

- Just in case you are wondering …
  - PWM mode *with* fault protection
    - Will not be used this semester … however … let briefing explain how it can be used …
      - Fault protection … is a feedback error signal of the inverter related to a possible hazardous state of operation of the inverter

PWM mode with fault protection

- PWM mode with fault protection …
  - The signal at input pin OCFA is a feedback error signal of the inverter related to a possible hazardous state of operation of the inverter
  - If the input pin OCFA is low … the inverter is considered to be in a hazardous (error) state
  - Then the output OCx pin is disabled automatically and the pin is driven to the high impedance state
  - The respective interrupt flag is asserted and in the register OCxCON the OCFLT bit (OCxCON<4>) is set

Output Compare Module Associated Pins

- The Pins we are concerned with on the PIC24 …
  - OC1
    - Output Compare Module – Channel 1
  - OC2
    - Output Compare Module – Channel 2
  - OCFA
    - Compare Fault A input (for Compare Channels 1 and 2)
- Are remappable pins

Output Compare Module

- PWM mode function … is used to generate variable duty cycle output
- The PWM duty cycle is specified through Secondary Output Compare (OCxRS) register, and it acts as a shadow register for the Output Compare (OCxR) register
  - This prevents glitches in the PWM output
- In PWM mode … the … Output Compare (OCxR) register is a read-only compare register

Output Compare Module

- When the PWM mode is enabled … the … OCx pin is …
  - Driven high … if … the OCxR register value is non-zero
  - Driven low … if … the OCxR register value is zero

Output Compare Module

- When a selected timer is enabled …
  - It starts incrementing until it reaches the value in the period register
    - The Compare Register (OCxR) value is constantly compared with the timer value
  - When a match occurs …
    - The OCx pin is driven low

Output Compare Module

- On a timer rollover …
  - The OCxRS value is loaded into the OCxR register and the OCx pin is …
    - Driven high if the OCxR register value is non-zero
    - Driven low if the OCxR register value is zero

PWM Period

- The PWM period is specified by writing to PRy, the TMRy Period register

- The PWM period …

$$\text{PWM Period} = [(\text{PRy}) + 1] \bullet T_{CY} \bullet (\text{TMRy Prescale Value})$$

$$\text{PWM Frequency} = 1/[\text{PWM Period}]$$

Note: A PRy value of N produces a PWM period of N + 1 time base count cycles. For example, a value of 7 written into the PRy register yields a period consisting of 8 time base cycles.

PWM Duty Cycle

- The PWM duty cycle is specified by writing to the …
  - OCxRS register
- The duty cycle value can be written at any time … but …
  - The duty cycle value is not latched onto the OCxR register until timer Reset on a period match
    - This provides a double buffer for the PWM duty cycle and is essential for glitch-free PWM operation
- In PWM mode … the … OCxR is a read-only register

Boundary Parameters of the PWM duty cycle

- Some important boundary parameters of the PWM duty cycle include …
  - If the duty cycle register … OCxR … is loaded with …
    - 0000h … the OCx pin remains low (0% duty cycle)
  - If the OCxR register is greater than PRy (Timer Period register) …
    - The pin remains high (100% duty cycle)
  - If the OCxR register is equal to PRy … the OCx pin is low for one time base count value … and … high for all other count values

Boundary Parameters of the PWM duty cycle

- In order to change the pulse width …
  - While the system is running …
    - We simply write into OCxRS … and …
      - It automatically loads into OCXR on the next cycle

PWM Resolution

- The PWM resolution depends on …
  - PWM frequency … and …
  - The timer clock frequency
    - The timer clock is derived from the internal clock (FCY) divided by a programmable prescaler

Calculation for PWM Resolution

- Calculation for PWM Resolution

$$\text{PWM Resolution (bits)} = \log_2 \left(\frac{\text{PWM Frequency}}{\text{Timer Clock Frequency}} \right) \text{ bits}$$

Setup for Output Compare Module in PWM Mode

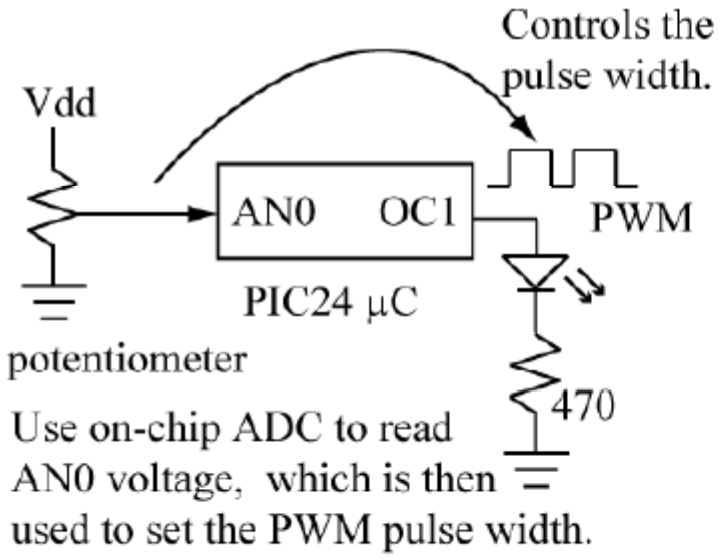
```
// Initialize Output Compare Module
OC1CONbits.OCM = 0b000; // Disable Output Compare Module
OC1R = 100;             // Write the duty cycle for the first PWM pulse
OC1RS = 200;            // Write the duty cycle for the second PWM pulse
OC1CONbits.OCTSEL = 0;  // Select Timer 2 as output compare time base
OC1R = 100;             // Load the Compare Register Value
OC1CONbits.OCM = 0b110; // Select the Output Compare mode
```

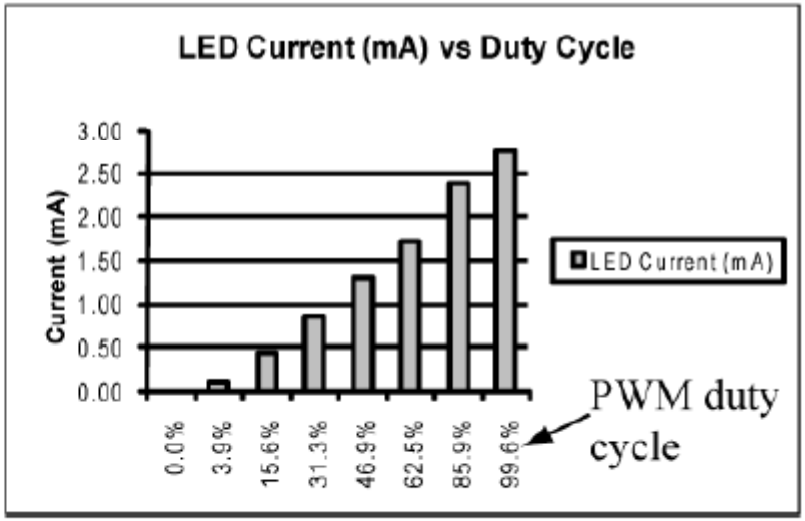
Setup for Output Compare Module in PWM Mode
continued …

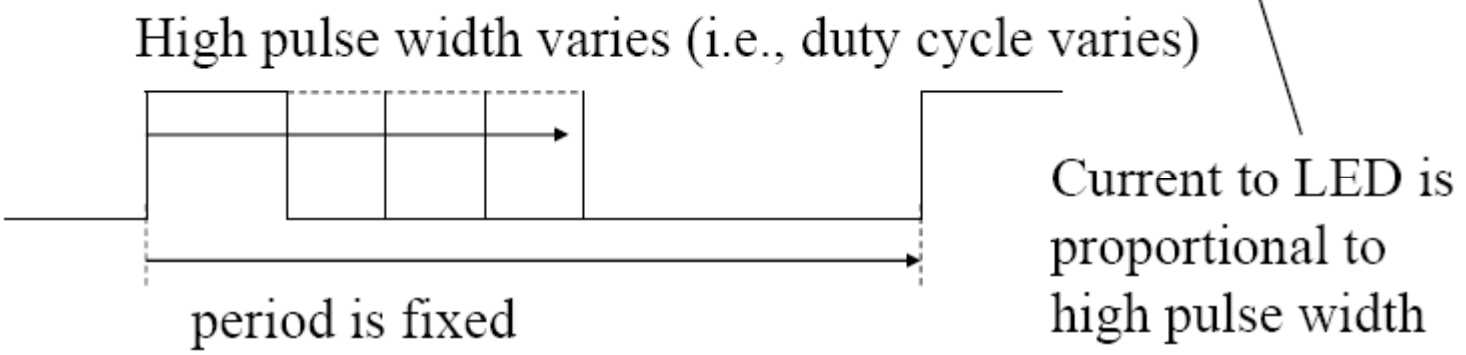
```
// Initialize and enable Timer2
T2CONbits.TON = 0;              // Disable Timer
T2CONbits.TCS = 0;              // Select internal instruction cycle clock
T2CONbits.TGATE = 0;            // Disable Gated Timer mode
T2CONbits.TCKPS = 0b00;         // Select 1:1 Prescaler
TMR2 = 0x00;                    // Clear timer register
PR2 = 500;                      // Load the period value

IPC1bits.T2IP = 0x01;           // Set Timer 2 Interrupt Priority Level
IFS0bits.T2IF = 0;              // Clear Timer 2 Interrupt Flag
IEC0bits.T2IE = 1;              // Enable Timer 2 interrupt
T2CONbits.TON = 1;              // Start Timer

/* Example code for Timer 2 ISR*/
void __attribute__((__interrupt__)) _T2Interrupt( void )
{
/* Interrupt Service Routine code goes here */
OC1RS = 300;                    // Write Duty Cycle value for next PWM cycle
IFS0bits.T2IF = 0;              // Clear Timer 2 interrupt flag
}
```

Pulse Width Modulation Applications …

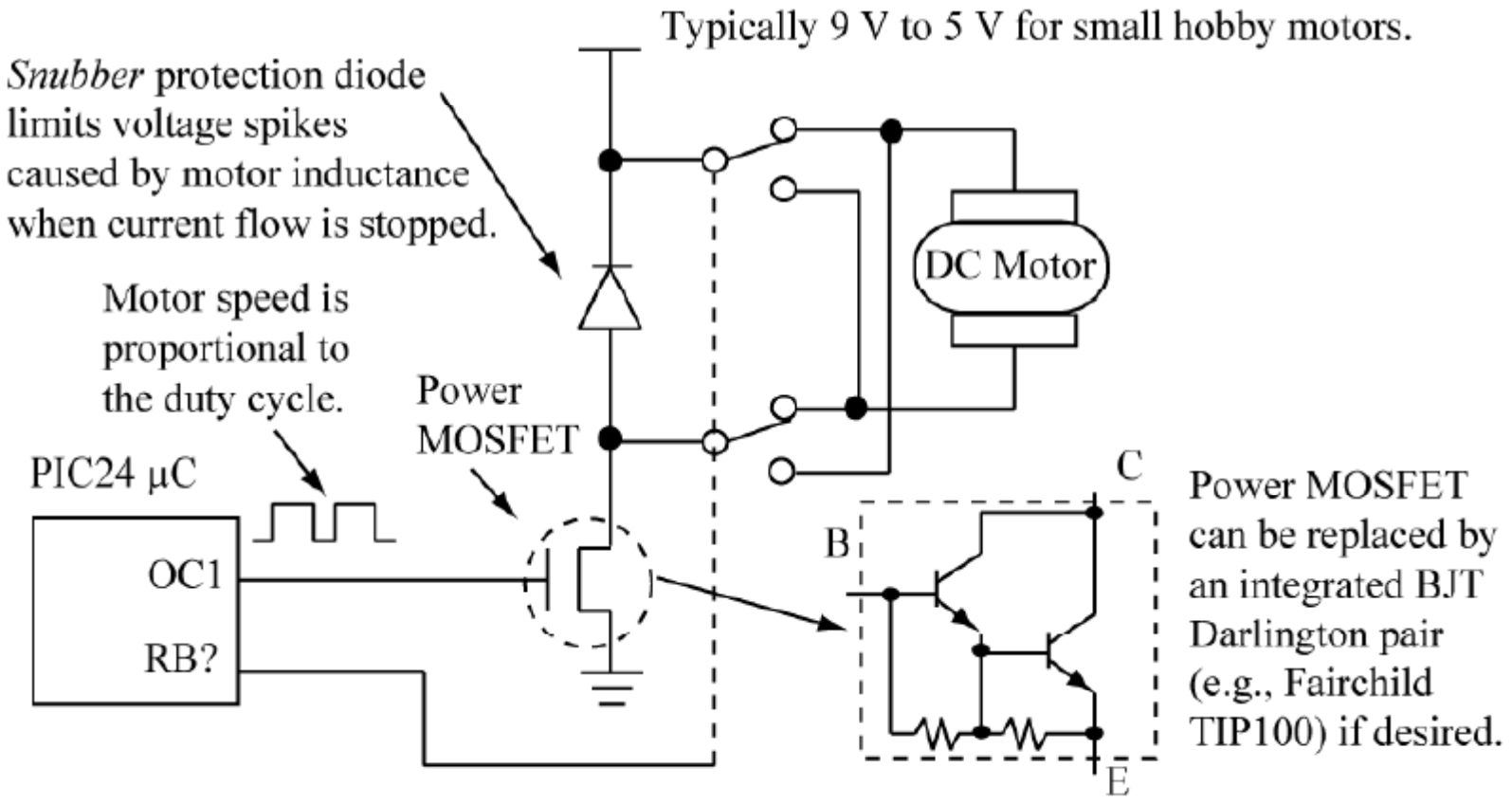
Pulse Width Modulation (PWM)



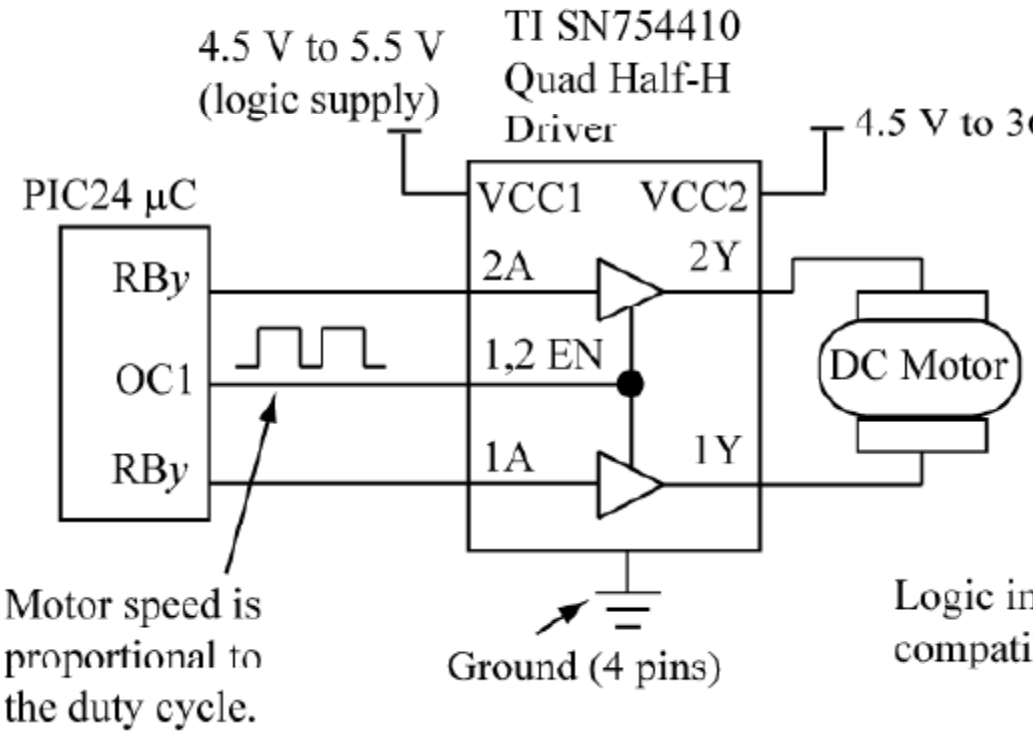




DC Motor Speed Control



Half-bridge Driver



1A	2A	EN	Direction
0	1	1	Forward
1	0	1	Reverse
1	1	1	Brake
x	x	0	Coast

Logic inputs to the SN754410 are TTL compatible (V_{IH} minimum is 2 V).

Integrates MOSFET/BJT drivers, protection diodes, switches.

Programming/Software

Programming/Software

- As discussed during the lecture

Lab

Peer Review of Software …

Peer Review of Software Developed

- How did you write your code?
- What problems did you encountered?
- Any questions that you need resolved?

Lab #4 …

Lab #4

- Will use PWM to control Motor speed (a fan)
  - Use …
    1. Pre-determined values and a pushbutton
       - » High
       - » Medium
       - » Low
  - To control the speed of a motor (fan)

Lab #4

- We will need to build a 5 volt rectifier circuit
  - For the motor … which is a 5 volt fan
- Use a half H-driver
- PWM to vary the speed
- We could … replace the pushbutton with a variable resistor
  - Use the analog input to the PIC24 to control the fan speed

Next Class

Next Class Topics

- April 5th … continue with Lab #4, no lecture
    - REMINDER … I expect to see you working in the lab

Homework

Homework

- Labs …
  - Code development for Lab #4
  - Lab #3 Report … due April 5th

Time to start the lab …

Lab

- Continue Lab #3, if needed
- Start Lab #4

White Board …

References

References

1. None