The University of Texas at Arlington

Lecture 11
Timers, Capture/Compare/PWM

CSE@UTA

CSE 3442/5442

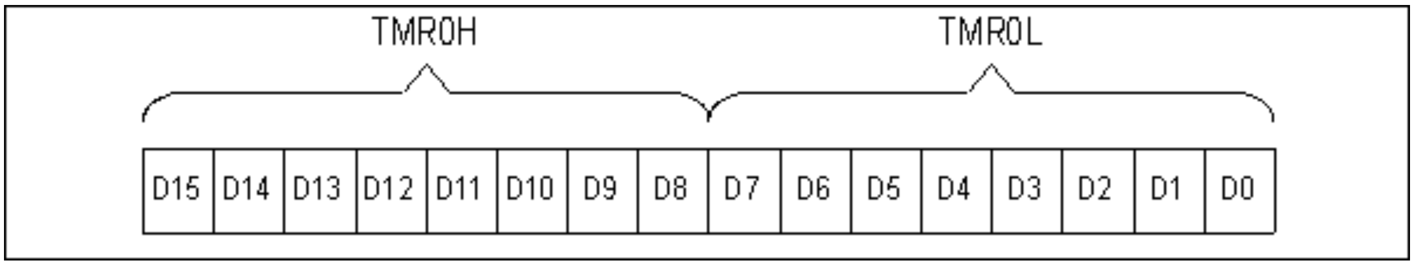
CSE@UTA

PIC Timers

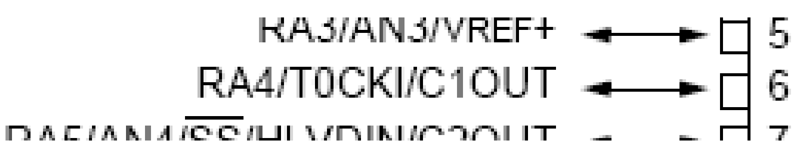
- PIC18 family microcontrollers have two to five timers on board.
- Timers can be used to generate time delays or to count (outside) events happening.
- Some timers can also be used to control timing of other peripherals (the designer needs to pay attention to that).
- Every timer needs a clock that will make it to count.
- PIC18 timers have the option to use at most ¼ of the main clock's frequency or use a separate external signal for clocking.
- Timers can overload several pins on the microcontroller

Timer0

- Timer-0 can be used as an 8-bit or as a 16-bit timer.
- Thus, two SFR are used to contain the count:



- Each timer has a control register: T0CON for timer 0
- Timer SFR-s are read/write registers but do not have immediate access.
- Timer clocks make timers count; the timer clock can be internal or external.



Timer-0 Control Register T0CON

TMR0ON	T08BIT	T0CS	T0SE	PSA	T0PS2	T0PS1	T0PS0

TMR0ON	D7	Timer0 ON and OFF control bit 1 = Enable (start) Timer0 0 = Stop Timer0
T08BIT	D6	Timer0 8-bit/16-bit selector bit 1 = Timer0 is configured as an 8-bit timer/counter. 0 = Timer0 is configured as a 16-bit timer/counter.
T0CS	D5	Timer0 clock source select bit 1 = External clock from RA4/T0CKI pin 0 = Internal clock (Fosc/4 from XTAL oscillator)
T0SE	D4	Timer0 source edge select bit 1 = Increment on H-to-L transition on T0CKI pin 0 = Increment on L-to-H transition on T0CKI pin
PSA	D3	Timer0 prescaler assignment bit 1 = Timer0 clock input bypasses prescaler. 0 = Timer0 clock input comes from prescaler output.

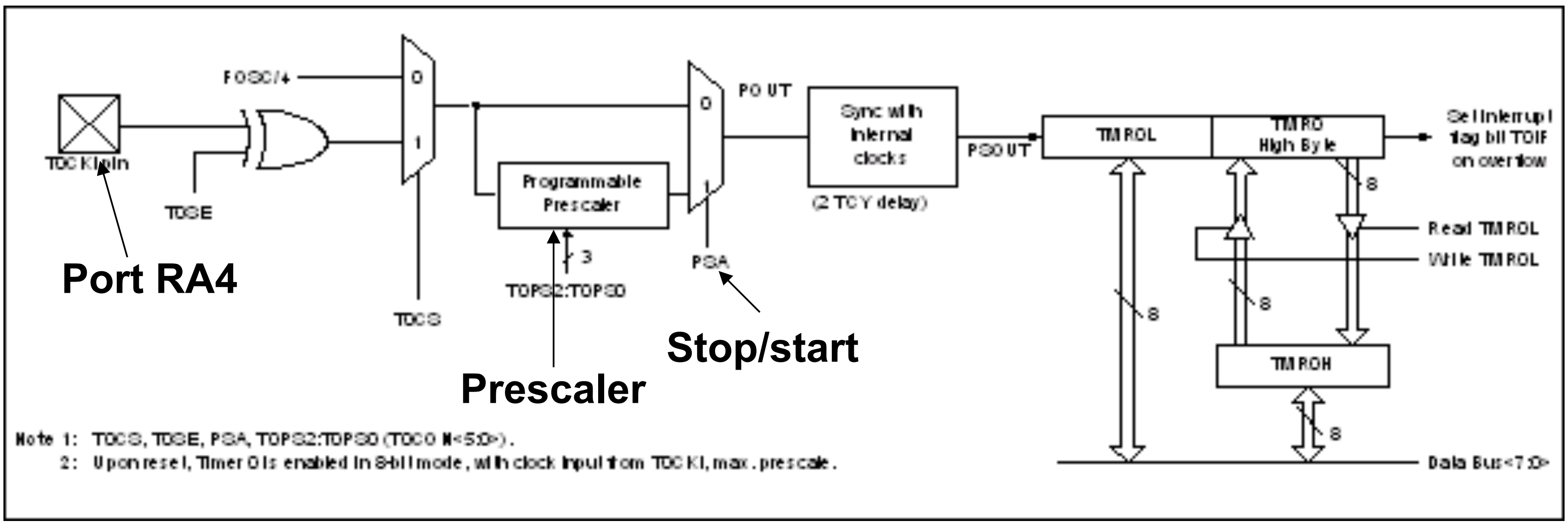
T0PS2:T0PS0	D2 D1 D0	Timer0 prescaler selector
	0 0 0 = 1:2	Prescale value (Fosc / 4 / 2)
	0 0 1 = 1:4	Prescale value (Fosc / 4 / 4)
	0 1 0 = 1:8	Prescale value (Fosc / 4 / 8)
	0 1 1 = 1:16	Prescale value (Fosc / 4 / 16)
	1 0 0 = 1:32	Prescale value (Fosc / 4 / 32)
	1 0 1 = 1:64	Prescale value (Fosc / 4 / 64)
	1 1 0 = 1:128	Prescale value (Fosc / 4 / 128)
	1 1 1 = 1:256	Prescale value (Fosc / 4 / 256)

- Note that timer interrupt enable/flag bits are in registers related to interrupts (e.g., INTCON)
- When the timer overflows, TMR0IF is set.
- 16- vs. 8-bit timer
- Prescalers are useful for large time delays

Timer0 Programming

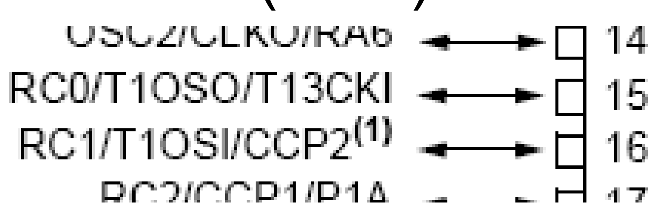
1. Select 16-bit mode
2. Load TMR0H(!) then TMR0L with initial values
3. Start timer
4. Monitor TMR0IF (or set interrupt on it)
5. When TMR0IF is set, stop the timer, reset the flag (and if needed go to step 2)

- Timer-0 can be also used as an 8-bit timer. In this case TIMR0H is ignored and the interrupt flag is set when TMR0H overflows.

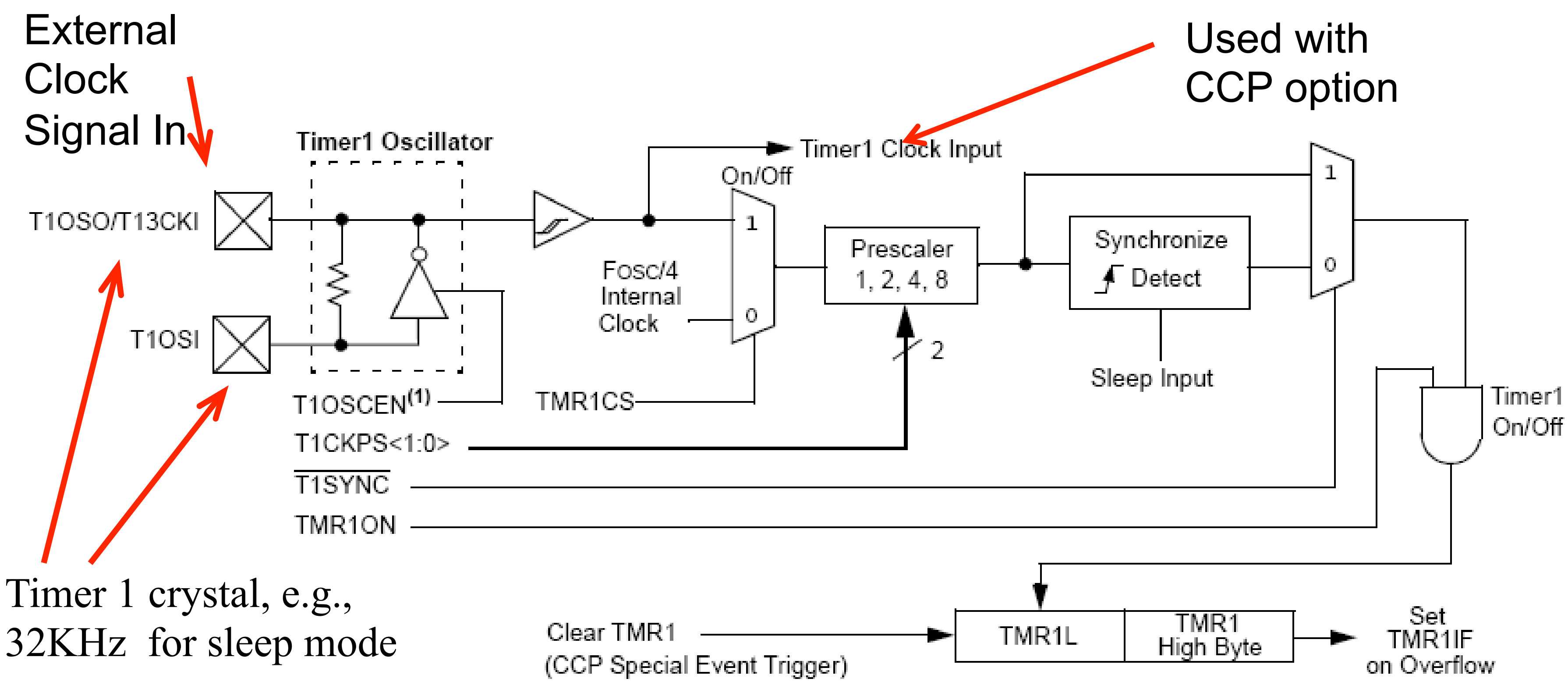


Timer1

- Timer1 is a 16-bit (only) timer (TMR1L, TMR1H)
- T1CON is the control register and TMR1IF is the interrupt flag (PIR1).
- Presacler does not support divisions above 1:8
- Timer1 has two external clock sources
  - Clock fed into T1CK1 pin (RC0)
  - Crystal (typically 32-kHz) connected between the T1CKI and T1OSI PINS (RC0&RC1)– used for saving power during sleep mode. When in sleep mode, Timer1 is not shut down allowing use as real-time clock (RTC) that can be used for waking up



Timer1 Block Diagram



Note 1: When enable bit, T1OSCEN, is cleared, the inverter and feedback resistor are turned off to eliminate power drain.

Timer 1 Control Register T1CON

R/W-0	R-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
RD16	T1RUN	T1CKPS1	T1CKPS0	T1OSCEN	$\overline{\text{T1SYNC}}$	TMR1CS	TMR1ON
bit 7							bit 0

bit 7 **RD16:** 16-Bit Read/Write Mode Enable bit
1 = Enables register read/write of TImer1 in one 16-bit operation
0 = Enables register read/write of Timer1 in two 8-bit operations

bit 6 **T1RUN:** Timer1 System Clock Status bit
1 = Device clock is derived from Timer1 oscillator
0 = Device clock is derived from another source

bit 5-4 **T1CKPS<1:0>:** Timer1 Input Clock Prescale Select bits
11 = 1:8 Prescale value
10 = 1:4 Prescale value
01 = 1:2 Prescale value
00 = 1:1 Prescale value

bit 3 **T1OSCEN:** Timer1 Oscillator Enable bit
1 = Timer1 oscillator is enabled
0 = Timer1 oscillator is shut off
The oscillator inverter and feedback resistor are turned off to eliminate power drain.

bit 2 **$\overline{\text{T1SYNC}}$:** Timer1 External Clock Input Synchronization Select bit
When TMR1CS = 1:
1 = Do not synchronize external clock input
0 = Synchronize external clock input
When TMR1CS = 0:
This bit is ignored. Timer1 uses the internal clock when TMR1CS = 0.

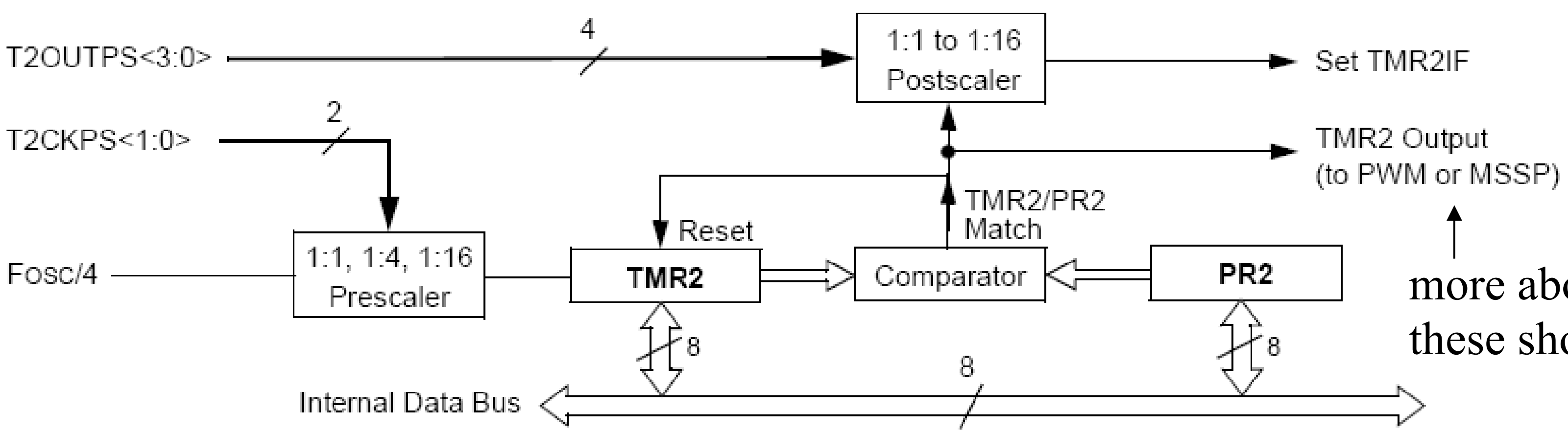
bit 1 **TMR1CS:** Timer1 Clock Source Select bit
1 = External clock from pin RC0/T1OSO/T13CKI (on the rising edge)
0 = Internal clock (FOSC/4)

bit 0 **TMR1ON:** Timer1 On bit
1 = Enables Timer1
0 = Stops Timer1

- RD16=1 is the only option in Timer0 (to avoid changes in Timer1H while Timer1L is read/write a temporary register is used)
- Timer-1 can be used as i) timer, ii) as synchronous counter (T1SYNC), iii) asynchronous counter

Timer2

- Timer2 is an 8-bit (only) timer (TMR2)
- Timer2 has a period register PR2; Timer2 can be set to count only to PR2 and set TMR2IF then.
- Clock source is only Fosc/4 (thus, Timer2 cannot be used as a counter) ; prescalers and postscalers (count on interrupt) are available.



more about these shortly

Timer2 Control Register T2CON

U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	T2OUTPS3	T2OUTPS2	T2OUTPS1	T2OUTPS0	TMR2ON	T2CKPS1	T2CKPS0
bit 7							bit 0

bit 7 **Unimplemented:** Read as '0'

bit 6-3 **T2OUTPS<3:0>:** Timer2 Output Postscale Select bits

0000 = 1:1 Postscale
0001 = 1:2 Postscale
•
•
•
1111 = 1:16 Postscale

bit 2 **TMR2ON:** Timer2 On bit

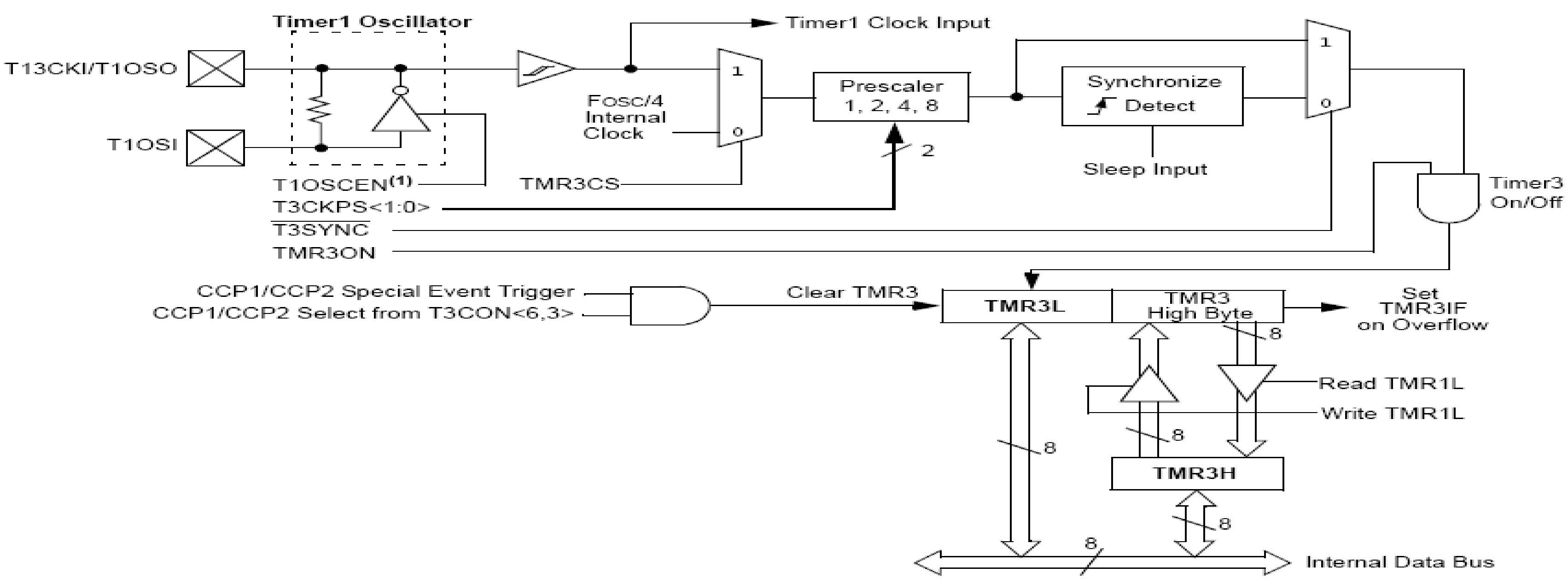
1 = Timer2 is on
0 = Timer2 is off

bit 1-0 **T2CKPS<1:0>:** Timer2 Clock Prescale Select bits

00 = Prescaler is 1
01 = Prescaler is 4
1x = Prescaler is 16

Timer3

- Timer3 is a 16-bit (only) timer (TMR3L, TMR3H)
- Can work with CCP peripheral (later)
- Can use same external source as timer1
- Can be used as timer, ascynchronous, or synchronous counter



Timer3 Control Register T3CON

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
RD16	T3CCP2	T3CKPS1	T3CKPS0	T3CCP1	$\overline{\text{T3SYNC}}$	TMR3CS	TMR3ON
bit 7							bit 0

bit 7 **RD16:** 16-Bit Read/Write Mode Enable bit
1 = Enables register read/write of Timer3 in one 16-bit operation
0 = Enables register read/write of Timer3 in two 8-bit operations

bit 6,3 **T3CCP<2:1>:** Timer3 and Timer1 to CCPx Enable bits
1x = Timer3 is the capture/compare clock source for the CCP modules
01 = Timer3 is the capture/compare clock source for CCP2;
Timer1 is the capture/compare clock source for CCP1
00 = Timer1 is the capture/compare clock source for the CCP modules

← See soon

bit 5-4 **T3CKPS<1:0>**: Timer3 Input Clock Prescale Select bits
11 = 1:8 Prescale value
10 = 1:4 Prescale value
01 = 1:2 Prescale value
00 = 1:1 Prescale value

bit 2 **$\overline{\text{T3SYNC}}$:** Timer3 External Clock Input Synchronization Control bit
(Not usable if the device clock comes from Timer1/Timer3.)
When TMR3CS = 1:
1 = Do not synchronize external clock input
0 = Synchronize external clock input
When TMR3CS = 0:
This bit is ignored. Timer3 uses the internal clock when TMR3CS = 0.

bit 1 **TMR3CS:** Timer3 Clock Source Select bit
1 = External clock input from Timer1 oscillator or T13CKI (on the rising edge after the first falling edge)
0 = Internal clock (FOSC/4)

bit 0 **TMR3ON:** Timer3 On bit
1 = Enables Timer3
0 = Stops Timer3

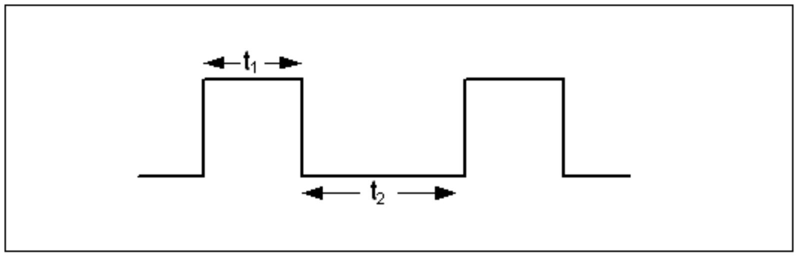
Using PIC18 Timers for Capture, Compare, and PWM

PWM Basics

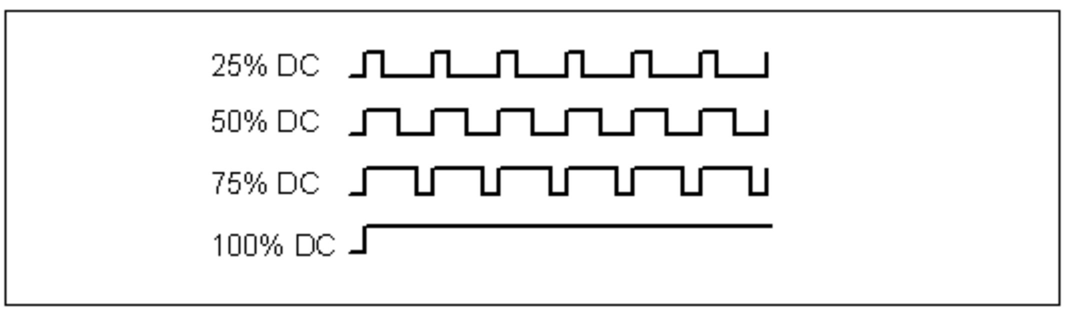
- PWM = Pulse width modulation
- Digital signals have two distinct levels: high and low
- Each of these levels is usually represented by a voltage, e.g., in PIC low is 0V and high is VCC (e.g., 5V).
- A temporal digital signal changes with time from low to high and back.
- Thus we can describe temporal digital signals with a series of values representing the time for which they stay in one state.
- Periodic temporal digital signals have a distinct frequency (the inverse of the time between two consecutive rising edges)

PWM Basics (cont’d)

- If t1+t2 remain constant → frequency remains constant.

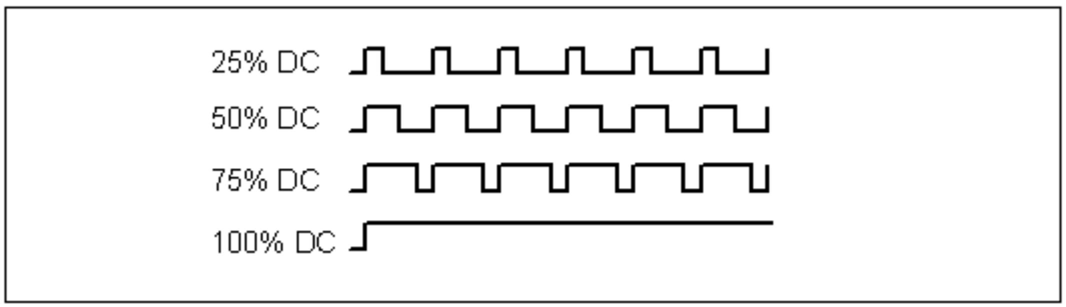


- Such periodic signals can still have varying times they spend in high vs. low state. PWM Duty Cycle is the portion of the pulse that stays HIGH relative to the entire period. DC[%] = 100*t1/(t1+t2)

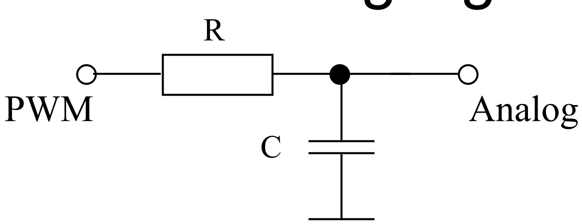


PWM Basics (cont'd)

- There are various sensors that provide their output as PWM signals, where the DC corresponds to the reading.
- There are various actuators that work well with a PWM input.



- Actually, an appropriate RC filter (Integrator) can make an analog signal out of a PWM digital signal



Capture, Compare, PWM Modules

- PIC18 microcontrollers can have up to 5 CCP modules.
- *Compare* enables the counter value of timers to be compared to a 16bit register, and if equal perform an action.
- Capture can use an external input to copy timer values into a 16-bit register. Thus, *Capture* provides the capability of measuring the period of a pulse.
- PWM – pulse width modulation, can be used as a quasi analog output (a timed digital output with duty cycle setting).
- These modules are great for driving motors, or reading encoders.
- For DC motor control some of the CCPs have been enhanced and are called ECCP

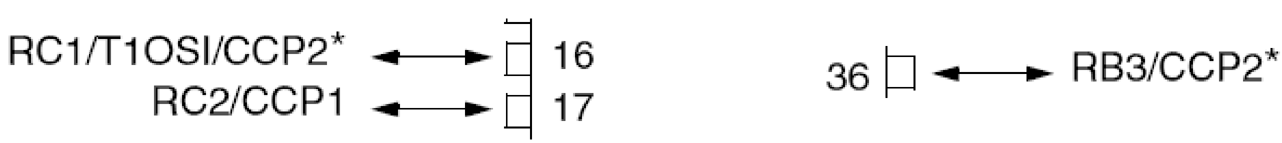
Timers and CCP Features

- Timers 1 and 3 can be used for capture and compare features
- Timer 2 is used for PWM
- As shown before, T3CON is used to chose the timer for C/C

These rules do not always apply – have to check the specific PIC18 datasheet

CCP Module Basics

- Each CCP module has three registers associated with it:
  - CCPxCON controlling the modes
  - CCPxL and CCPxH as a 16-bit compare/ capture/PWM duty cycle register
- Each CCP module has a pin associated with it (input or output)



CCP Module Control CCPxCON

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	DCxB1	DCxB0	CCPxM3	CCPxM2	CCPxM1	CCPxM0
bit 7							bit 0

bit 7-6 **Unimplemented:** Read as '0'

bit 5-4 **DCxB<1:0>:** PWM Duty Cycle bit 1 and bit 0 for CCPx Module

Capture mode:
Unused.

Compare mode:
Unused.

PWM mode:
These bits are the two LSbs (bit 1 and bit 0) of the 10-bit PWM duty cycle. The eight MSbs (DCxB<9:2>) of the duty cycle are found in CCPRxL.

bit 3-0 **CCPxM<3:0>:** CCPx Module Mode Select bits

0000 = Capture/Compare/PWM disabled (resets CCPx module)
0001 = Reserved
0010 = Compare mode, toggle output on match (CCPxIF bit is set)
0011 = Reserved
0100 = Capture mode, every falling edge
0101 = Capture mode, every rising edge
0110 = Capture mode, every 4th rising edge ← prescalers
0111 = Capture mode, every 16th rising edge
1000 = Compare mode, initialize CCPx pin low; on compare match, force CCPx pin high (CCPxIF bit is set)
1001 = Compare mode, initialize CCPx pin high; on compare match, force CCPx pin low (CCPxIF bit is set)
1010 = Compare mode, generate software interrupt on compare match (CCPxIF bit is set, CCPx pin reflects I/O state)
1011 = Compare mode, trigger special event; reset timer; CCP2 match starts A/D conversion (CCPxIF bit is set)
11xx = PWM mode

Compare Mode

- The CCPRxH:CCPRxL is loaded by the user
- If Timer1 TMR1H:TMR1L (or Timer3 – T3CON) count becomes equal to the above set value then the Compare module can:
  - Drive the CCPx pin high (CCPx config'd as out)
  - Drive the CCPx pin low (CCPx config'd as out)
  - Toggle the CCPx pin (CCPx config'd as out)
  - Trigger a CCPxIF interrupt and clear the timer
  - CCP2 can be used to kick off the A/D converter

Compare Mode Programming

0. Set up CCP interrupt if needed
1. Initialize CCPxCON for compare
2. Set timer source (T3CON)
3. Initialize CCPRxH:CCPRxL
4. Make sure CCPx pin is output if used (setting appropriate bits TRISB or TRISC)
5. Initialize Timer1 (or Timer3)
6. Start Timer1 (or Timer3)
7. Monitor CCPxIF or make sure interrupt is handled

Capture Mode

- The CCPx pin is set as input (with the appropriate TRIS)
- When an external event triggers the CCPx pin, then the TMR1H:TMR1L (or Timer3) values will be loaded into CCPRxH:CCPRxL
- Four options for CCPx pin triggering:
  - Every falling edge
  - Every rising edge
  - Every fourth rising edge
  - Every fourth falling edge
- Typical applications are measuring frequency or pulsewidth.

Capture Mode Programming for Frequency Measurement

1. Initialize CCPxCON for capture
2. Make CCPx pin an input pin (TRISB/TRISC)
3. On the first rising edge, Timer1 is loaded into CCPRxH:CCPRxL ; remember values.
4. On the next rising edge, Timer1 is loaded again into CCPRxH:CCPRxL ; subtract previous values from current values.
5. You have now the period of the signal captured by timer ticks. Some basic math will give you frequency

Capture Mode Programming for Measuring PWM Duty Cycle

1. Initialize CCPxCON for capture
2. Make CCPx pin an input pin (TRISB/TRISC)
3. Reset Timer1
4. On the rising edge, Timer1 is started and mode is set to falling edge detction
5. On the falling edge the CCPRxH:CCPRxL should be saved, CCP should be set to rising edge detection
6. On the rising edge CCPRxH:CCPRxL is saved. Now we have measurements for t1 and t2.
7. DC cycle can be calculated while new measurement is prepared (if continuous measuring is needed)

- Hint: interrupts can help

PWM Mode

- PWM output can be created without tedious programming of the compare mode
- The ECCP's PWM mode enables generating temporal digital signals of varying frequencies and varying DC (recall: the width of the pulse indicates some measured quantity).
- Recall, that the PWM Duty Cycle is the portion of the pulse at HIGH relative to the entire period.
- For PWM, Timer2 is used. Recall, that Timer 2 has a period register PR2.
- The period of the PWM signal is then:

  $T_{PWM}=4*N*(PR2+1)/F_{osc}$ where N is the prescaler

PWM Mode - Frequency

- Since most of the time we know what frequency we want to set it we need to set PR2:

$PR2=F_{osc}/(4*N*F_{PWM}) - 1$

- We see thus that the maximum PWM frequency is about a quarter of the clock while the minimum is about $F_{osc}/16382$

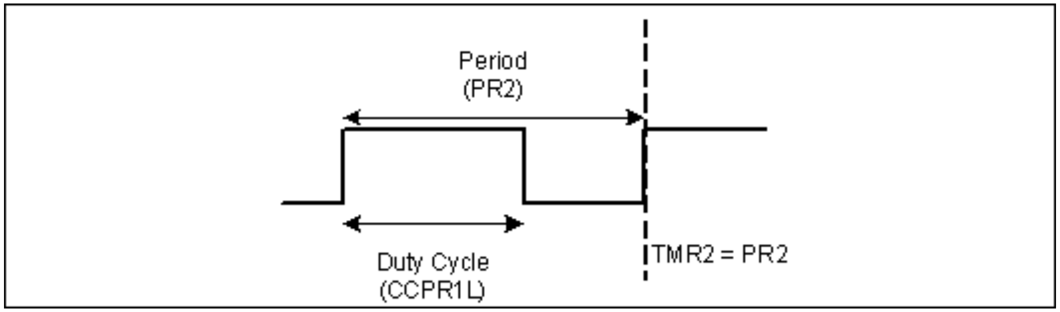
PWM Mode - Duty Cycle

- Assuming that we use CCP1, the duty cycle, is specified in 10-bits: DC1B9:DC1B0 (8 bits of CCPR1L and 2 bits from the CCP1CON register).
- If we denote the desired duty cycle by DCR (in [%]), and then:

  DCE:=PR2*DCR/100

  CCPR1L := ⌊DCE⌋ (i.e., the integer part)

  and DC1B1 and DC1B0 will need to be loaded with the remainder part of DC, where 00 is for 0, 01 is for 0.25, 10 is for 0.5 & 11 is for 0.75
- This then has an obvious precision influence on the PWM signal's duty cycle. Furthermore, the exact value of PR2 also has a strong influence on such precision.



PWM Mode - Programming

1. Set PWM period by setting PR2 and T2CON (prescaler)
2. Set PWM duty cycle by calculating and writing CCPRxL; (remember the remainder)
3. Set the CCPx as output
4. Clear TMR2
5. Load the remainder from step 2 into CCPxCON and set CCPx to PWM mode
6. Start Timer 2.

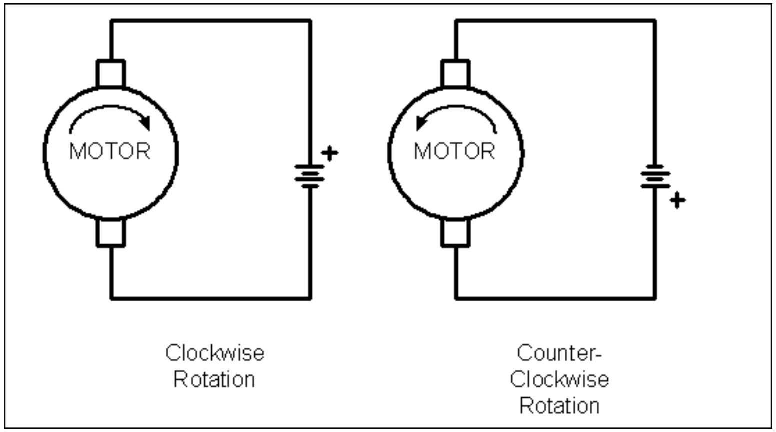
- This will result in a proper periodic temporal digital signal (no need to use goto-s)

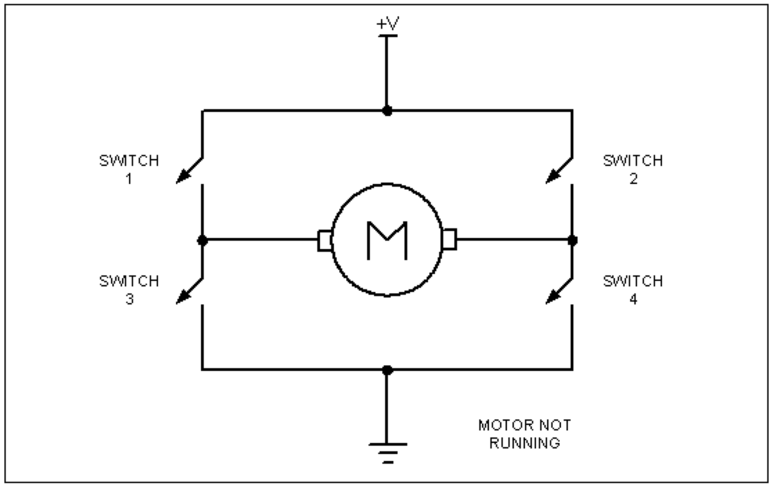
TABLE 15-4: EXAMPLE PWM FREQUENCIES AND RESOLUTIONS AT 40 MHz

PWM Frequency	2.44 kHz	9.77 kHz	39.06 kHz	156.25 kHz	312.50 kHz	416.67 kHz
Timer Prescaler (1, 4, 16)	16	4	1	1	1	1
PR2 Value	FFh	FFh	FFh	3Fh	1Fh	17h
Maximum Resolution (bits)	10	10	10	8	7	6.58

Driving DC Motors

- Brushed DC motors are common
- DC motors can run in both direction based on polarity on the two leads.
- DC motors run continuously when voltage is applied (rpm, voltage, torque, and power are all characteristics that need to be looked up in a catalog).
- How could we create circuitry that lets them rotate in both direction and with controlled speed?

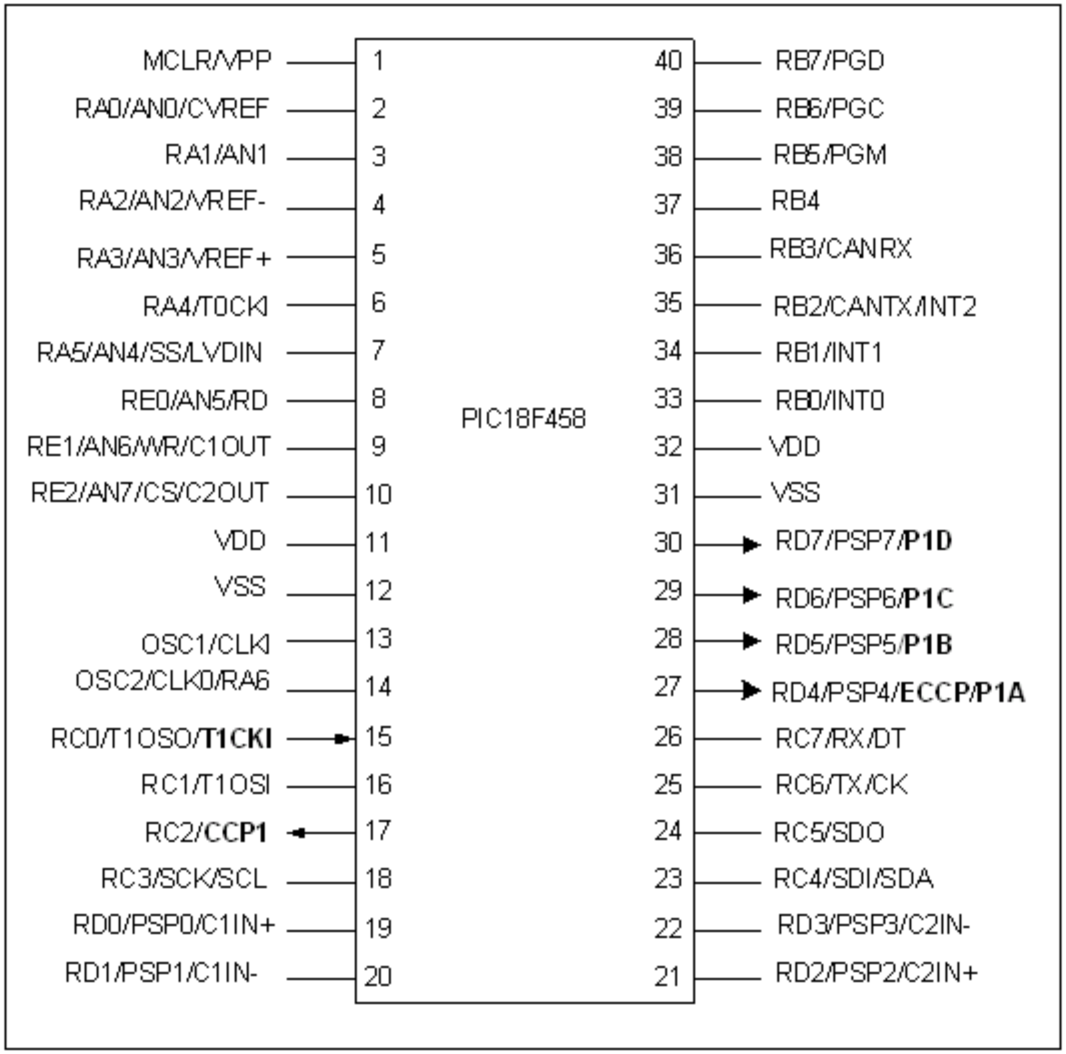




Enhanced CCP

- Many PIC microcontrollers come with an enhanced CCP module - ECCP. The enhanced module is really only enhanced for PWM output.
- Indeed on PIC18F458 the CCP1is enhanced and is ECCP1.
- In the enhanced PWM, active can be set from high to low. In addition, there are four output pins (great for driving DC motors both directions at various speeds)

Enhanced CCP Pins for PWM in PIC18F458/4580



CSE@UTA

Enhanced CCP Control

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
P1M1	P1M0	DC1B1	DC1B0	CCP1M3	CCP1M2	CCP1M1	CCP1M0
bit 7							bit 0

bit 7-6 **P1M<1:0>:** Enhanced PWM Output Configuration bits

If CCP1M3:CCP1M2 = 00, 01, 10:

xx = P1A assigned as capture/compare input/output; P1B, P1C, P1D assigned as port pins

If CCP1M3:CCP1M2 = 11:

00 = Single output, P1A modulated; P1B, P1C, P1D assigned as port pins
01 = Full-bridge output forward, P1D modulated; P1A active; P1B, P1C inactive
10 = Half-bridge output, P1A, P1B modulated with dead-band control; P1C, P1D assigned as port pins
11 = Full-bridge output reverse, P1B modulated; P1C active; P1A, P1D inactive

bit 5-4 **DC1B<1:0>:** PWM Duty Cycle bit 1 and bit 0

Capture mode:
Unused.

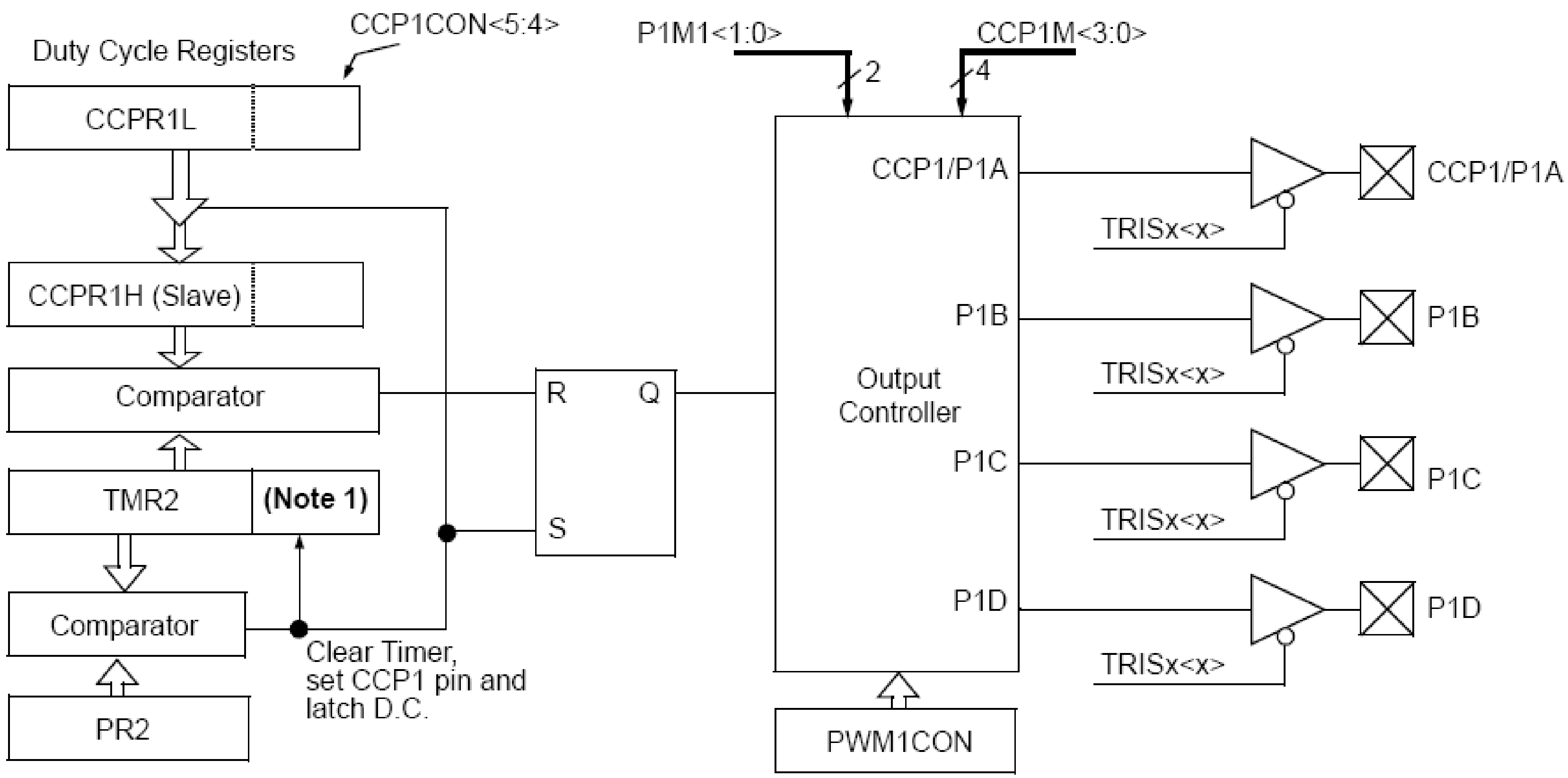
Compare mode:
Unused.

PWM mode:
These bits are the two LSbs of the 10-bit PWM duty cycle. The eight MSbs of the duty cycle are found in CCPR1L.

bit 3-0 **CCP1M<3:0>:** Enhanced CCP Mode Select bits

0000 = Capture/Compare/PWM off (resets ECCP module)
0001 = Reserved
0010 = Compare mode, toggle output on match
0011 = Capture mode
0100 = Capture mode, every falling edge
0101 = Capture mode, every rising edge
0110 = Capture mode, every 4th rising edge
0111 = Capture mode, every 16th rising edge
1000 = Compare mode, initialize CCP1 pin low; set output on compare match (set CCP1IF)
1001 = Compare mode, initialize CCP1 pin high; clear output on compare match (set CCP1IF)
1010 = Compare mode, generate software interrupt only; CCP1 pin reverts to I/O state
1011 = Compare mode, trigger special event (ECCP resets TMR1 or TMR3, sets CCP1IF bit)
1100 = PWM mode, P1A, P1C active-high; P1B, P1D active-high
1101 = PWM mode, P1A, P1C active-high; P1B, P1D active-low
1110 = PWM mode, P1A, P1C active-low; P1B, P1D active-high
1111 = PWM mode, P1A, P1C active-low; P1B, P1D active-low

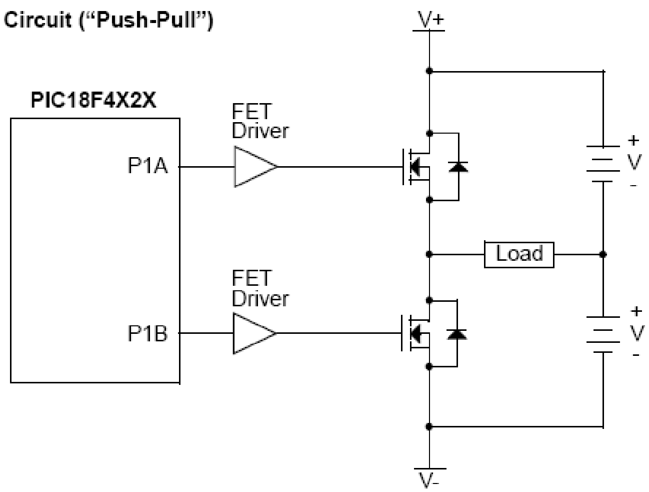
Enhanced CCP Block Diagram



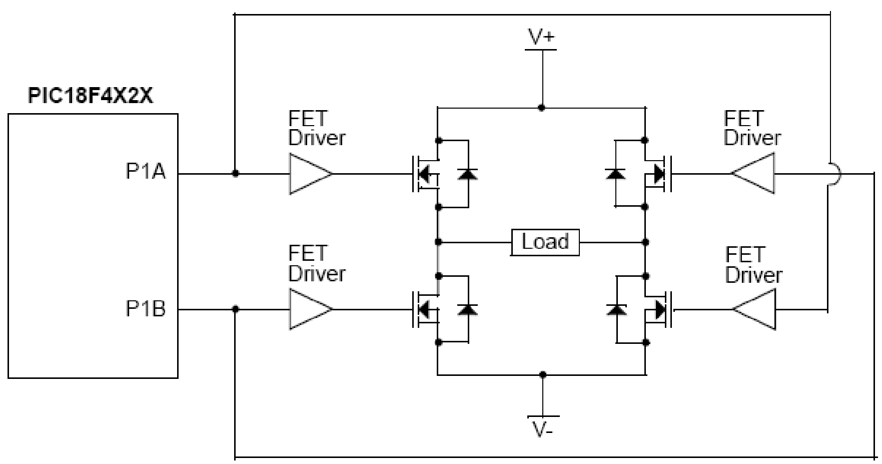
Note: The 8-bit TMR2 register is concatenated with the 2-bit internal Q clock, or 2 bits of the prescaler, to create the 10-bit time base.

DC Motor Drive Half bridge

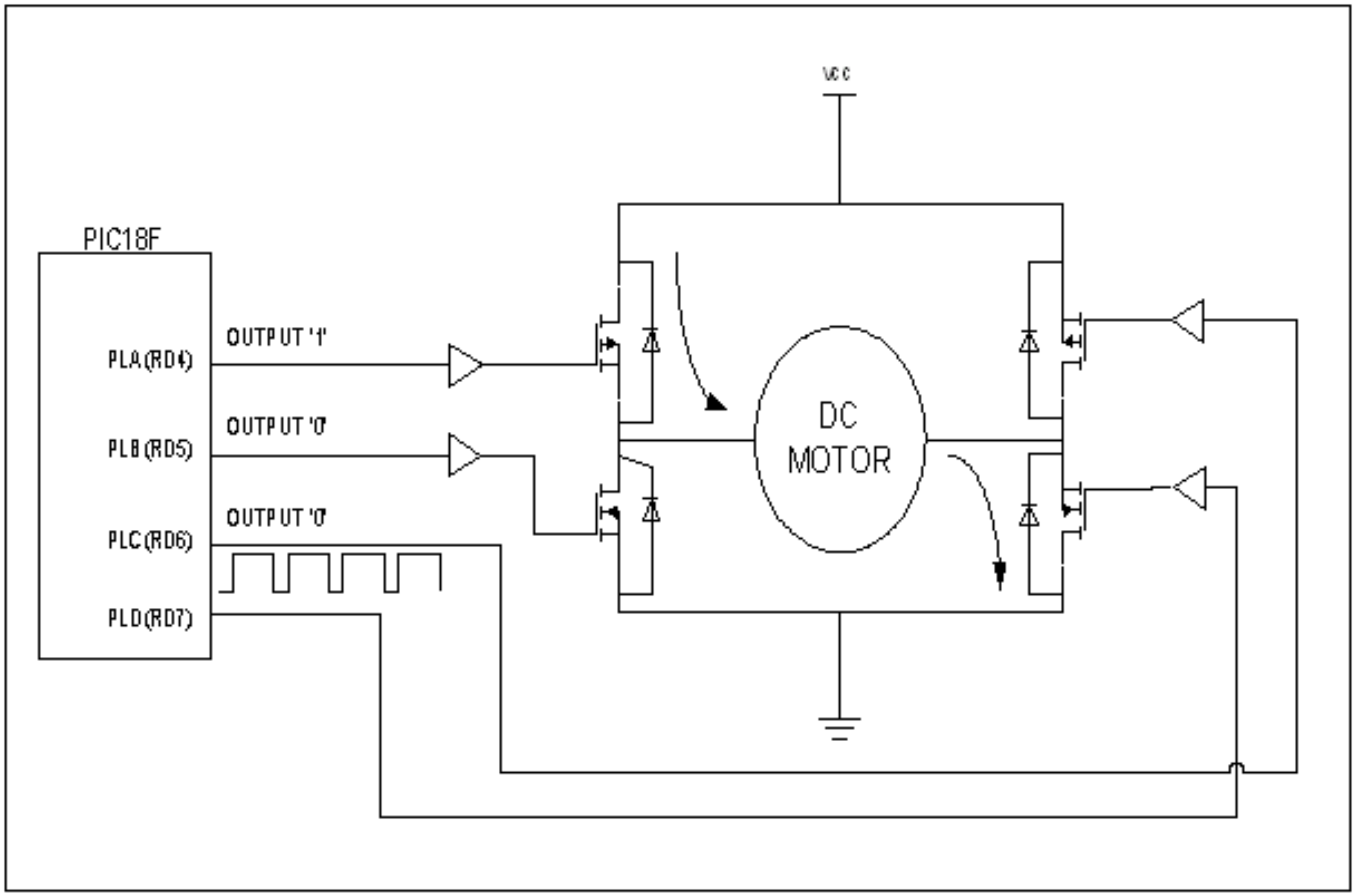
Standard Half-Bridge Circuit ("Push-Pull")



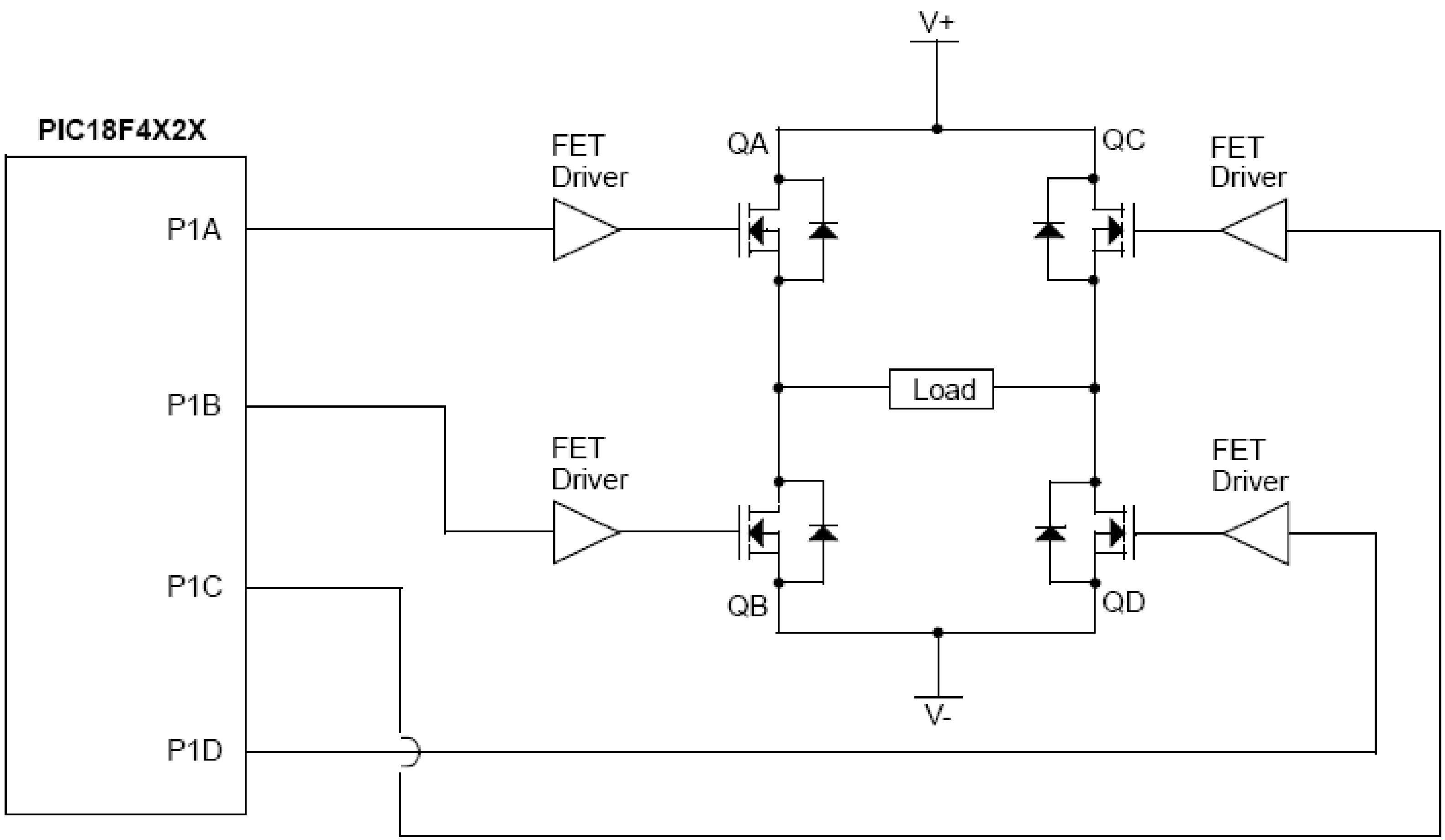
Half-Bridge Output Driving a Full-Bridge Circuit



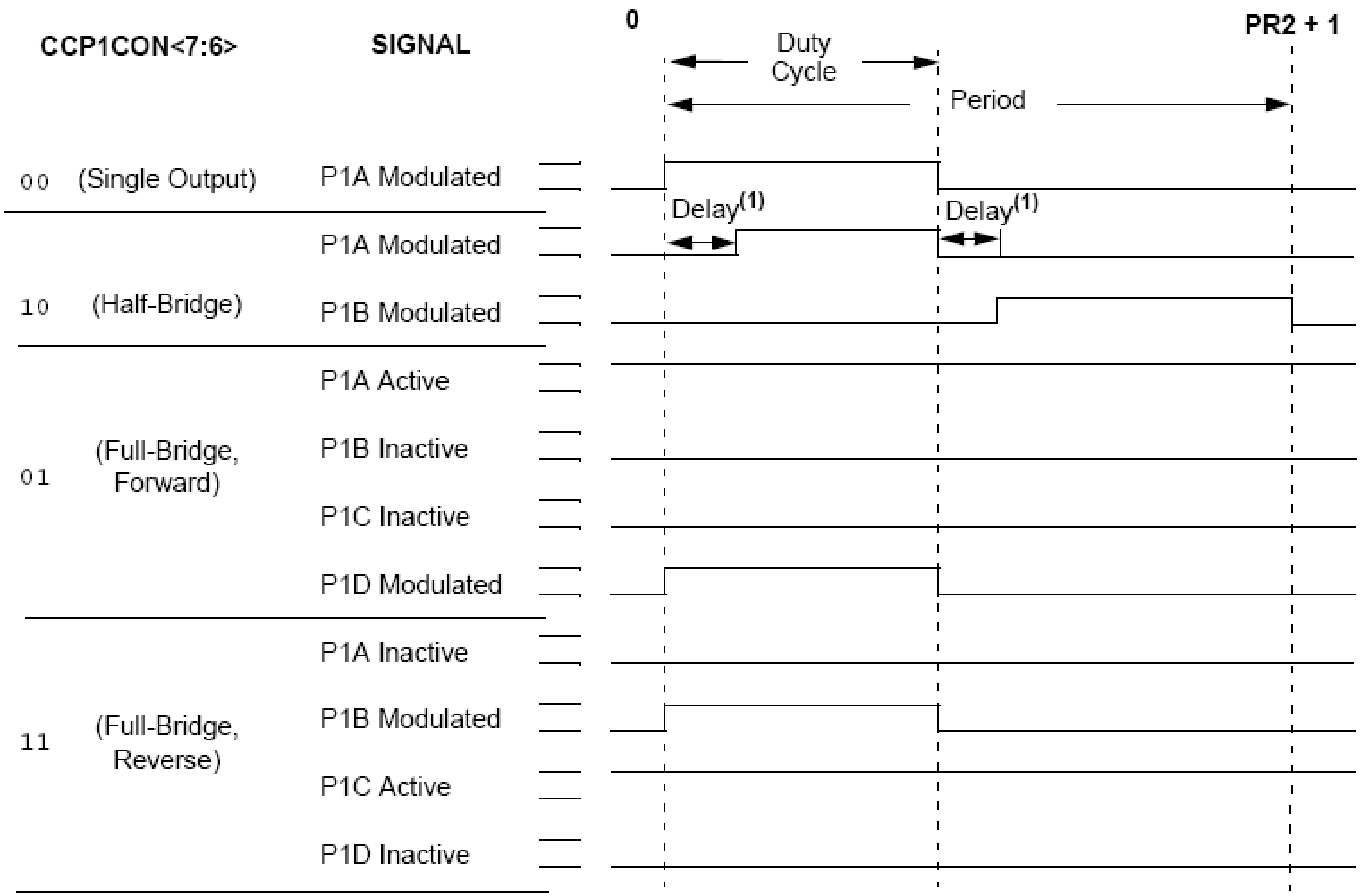
DC Motor Drive Full H-bridge



DC Motor Drive Full H Bridge



Bridge Timing Diagrams



How About Other Types of Motors?

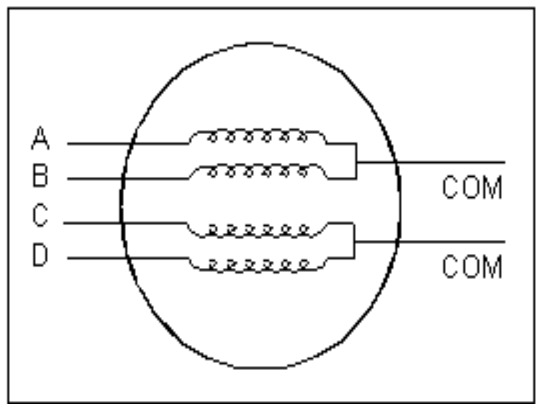
- Common other types of motors that we would like to control are: stepper motors (steppers) and servo motors (servos).
- Steppers are usually used when precise rpms are needed or when precise angles need to be turned to.
- Servos are usually used to make motors turn precisely to an angle with loads on them that could try to move the motor away.

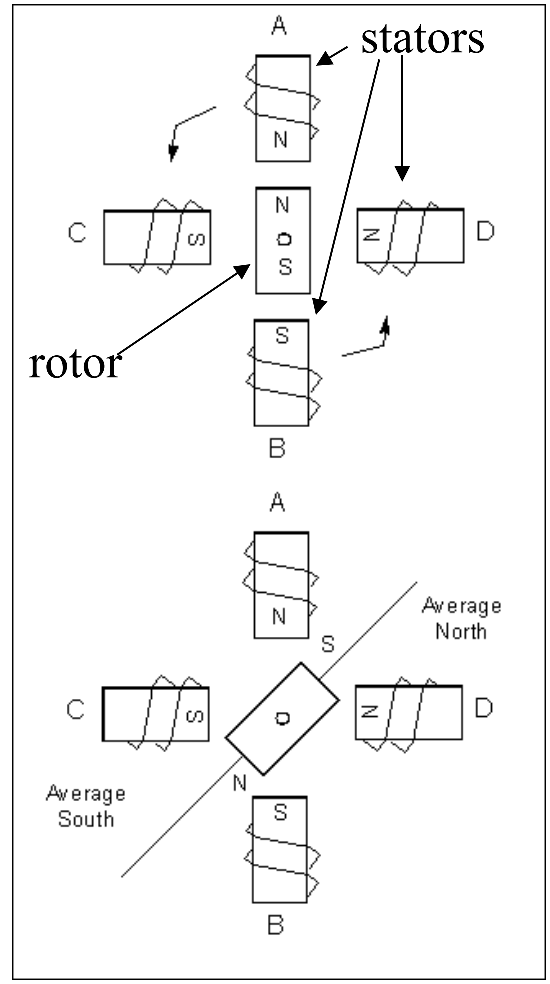
Stepper Motors

- Stepper motors are called that way as the user can turn them in small little precise steps.
- For example a 24 step/revolution (spr) motor has a 15° step angle, while a 48spr motor has a 7.5° angle.
- Maximum speed is usually given in steps per second.
- Holding torque determines how much torque is required to move the motor away from its position when control is applied to it.

Common Stepper Motors

- One of the most common stepper is a unipolar, permanent magnet stepper.
- A permanent magnet rotor is surrounded by four stators whose polarity can be changed by applying voltage to them.
- Thus, making a stepper turn, requires a precise sequence of control voltage applied to the four pins.
- Steps per revolution is controlled by having rotors with multiple N/S poles





CSE@UTA

Making Steppers Turn

- Normal stepper sequence:

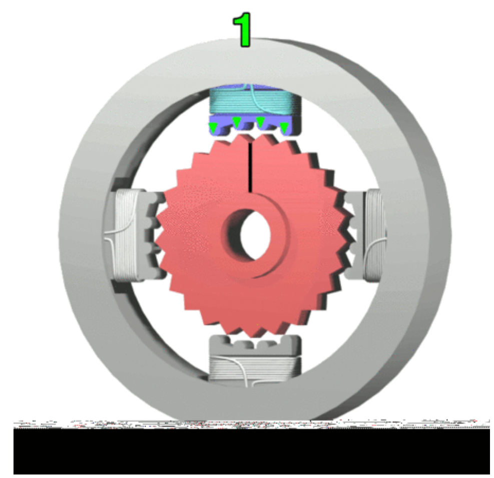
clockwise ↓ / cntr-clockwise ↑

Step#	A	B	C	D
1	1	0	0	1
2	1	1	0	0
3	0	1	1	0
4	0	0	1	1

- Half-step sequence:

clockwise ↓ / cntr-clockwise ↑

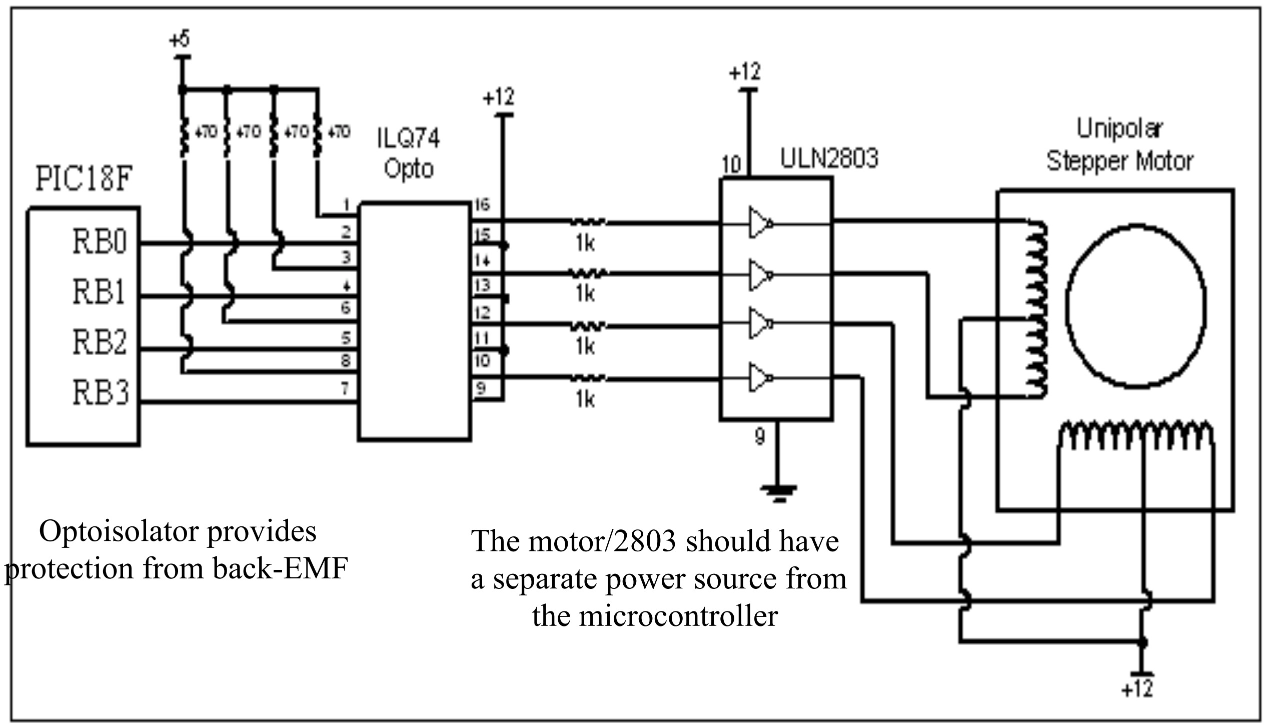
Step#	A	B	C	D
1	1	0	0	1
2	1	0	0	0
3	1	1	0	0
4	0	1	0	0
5	0	1	1	0
6	0	0	1	0
7	0	0	1	1
8	0	0	0	1



Why use one vs. the other? What's wave drive? (40% more torque, twice the power)

Making Steppers Turn

- Connecting steppers to a microcontroller requires back-EMF protection.



Optoisolator provides protection from back-EMF

The motor/2803 should have a separate power source from the microcontroller

```
#include <p18F458.h>
void main()
{
    TRISB = 0;
    while(1)
    {
        PORTB=0x06;  //0110
        MSDelay(100);
        PORTB=0x0C; //1100
        MSDelay(100);
        PORTB=0x09;  //1001
        MSDelay(100);
        PORTB=0x03;  //0011
        MSDelay(100);
    }
}
```

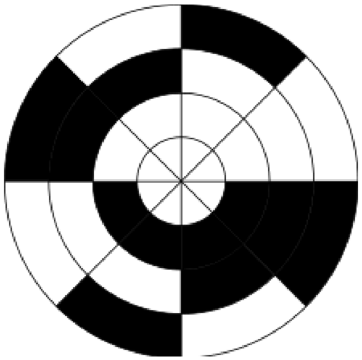
CSE@UTA

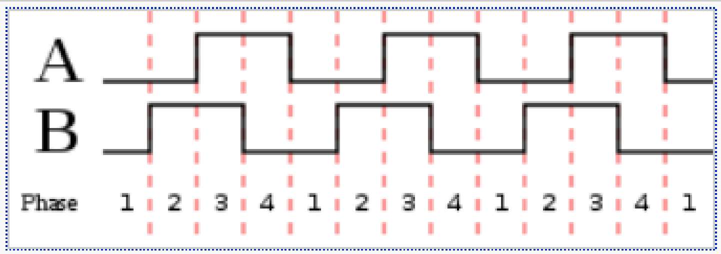
Servo Motors

- Servo motors are usually more expensive than DC motors or steppers.
- Servo motors have a precise control for position and are usually not used for complete rotations (although with a little mod they can be – why?).
- Servo motors have a built in mechanism (close control loop) to keep the position they are set to.
- Servo's usually have three input pins: two for power and one for controlling the angle.
- The control is usually using PWM for position.
- Internally, there is an encoder that measures the location of the shaft, an error signal is produced from the control and current location, and a P, PD, or PID controller is used to reduce this error.
- Length of the pulse dictates location (e.g., 0.6ms = -45 °, 1.5ms = 0 °, 2.4ms = 45 °)

Rotary Encoders

- Rotary encoders are rotational sensors (one component of servos), they can provide precise readings of shafts turning.
- Internally they can be mechanical, magnetic (induction) based or optical.
- Optical encoders are usually of high precision, contain encoder wheels.
- Encoders can be absolute or incremental
- They usually have four to five wires (power+, power-, A, B, 0)
- They can be read using timers but will tie up microcontroller; there are special purpose circuitry to read them, which have parallel or serial interfaces to microcontrollers.





Two square waves in quadrature (clockwise rotation).

Summary

- Timer peripherals can be used to create longer timeouts, to count external events (and act upon), or to create temporal digital signals.
- PWM signals can be used to drive several transducers, some other transducers will output PWM.
- Motor control, and encoder reading are probably two of the most used timer peripheral scenarios.