MICROPROCESSORS A (17.383)

Fall 2010

Lecture Outline

Class # 11

November 23, 2010

Dohn Bowden

Today's Lecture

- Syllabus review
- Microcontroller Hardware and/or Interface
  - Last week's lecture (worked on Labs the entire class)
    - Comparators
    - Timers
    - Watchdog Timer
    - Pulse Width Modulation (PWM)
- Programming/Software
  - Mixed C and Assembly Programming
- Lab
  - Finish Lab #5 (Optional Lab … however, recommend looking at)
  - Course Project
- Homework

Course Admin

Administrative

- Admin for tonight …
  - Syllabus Highlights
    - Lab #4 is due tonight (November 23rd)
    - Exam #2 next week (November 30th)
  - Project …
    - Any questions?
  - Lab #3 reports have been graded …
    - Hard copy submittals will be passed back
    - Electronic submittals have been emailed

Syllabus Review

Week	Date	Topics	Lab	Lab Report Due
1	09/07/10	Intro, Course & Lab Overview, Microcontroller Basics	1	
2	09/14/10	PIC16F684 Overview and General Input/Output	1 con't	
3	09/21/10	Switches	2	
4	09/28/10	Seven Segment LEDs	2 con't	1
5	10/05/10	**Examination 1**		
X	10/12/10	**No Class – Monday Schedule**		
6	10/19/10	Analog to Digital Conversion	3	2
7	10/26/10	Analog to Digital Conversion con't	3 con't	
8	11/02/10	Lab Work (Finish Lab #3 and start Lab #4)	3, 4	
9	11/09/10	LCD Interface and Assembly Language	4 con't	3
10	11/16/10	Comparators, Timers, Pulse Width Modulation (PWM)	5	
11	11/23/10	Mixed C & Assembly Programming/Course Project	Project	4
12	11/30/10	**Examination 2**		
13	12/07/10	Course Project	Project	5
14	12/14/10	Final Exam/Course Project Brief and Demonstration	Demo	

Microcontroller Hardware and / or Interfaces

We shall begin with last week's lecture ...

Comparators ...

PIC16F684 Hardware

- Another analog interface peripheral of the PIC16F684 …
  - Comparators

Comparator Fundamentals

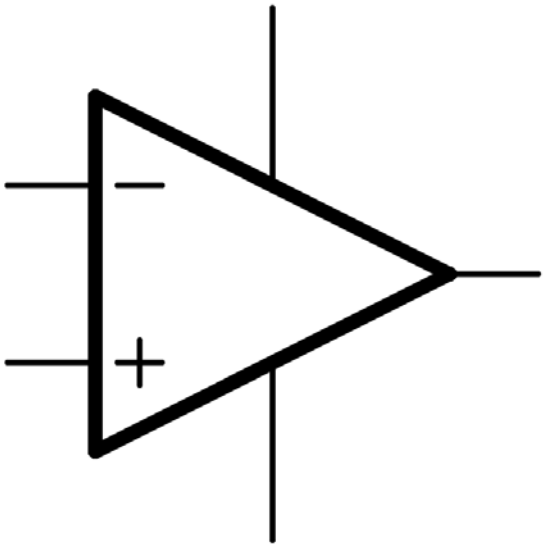
- Compares the voltage level of two analog signals … and …
  - Identifies which signal is the largest

Why would we need such a device?

- Switching on lights and heaters
- Detecting when a level in a circuit exceeds some particular threshold
- Switching power supplies
- Generating square waves from triangle waves
- And so on …

Comparator Fundamentals

- Building block of the comparator is the Operational Amplifier (Op-Amp)

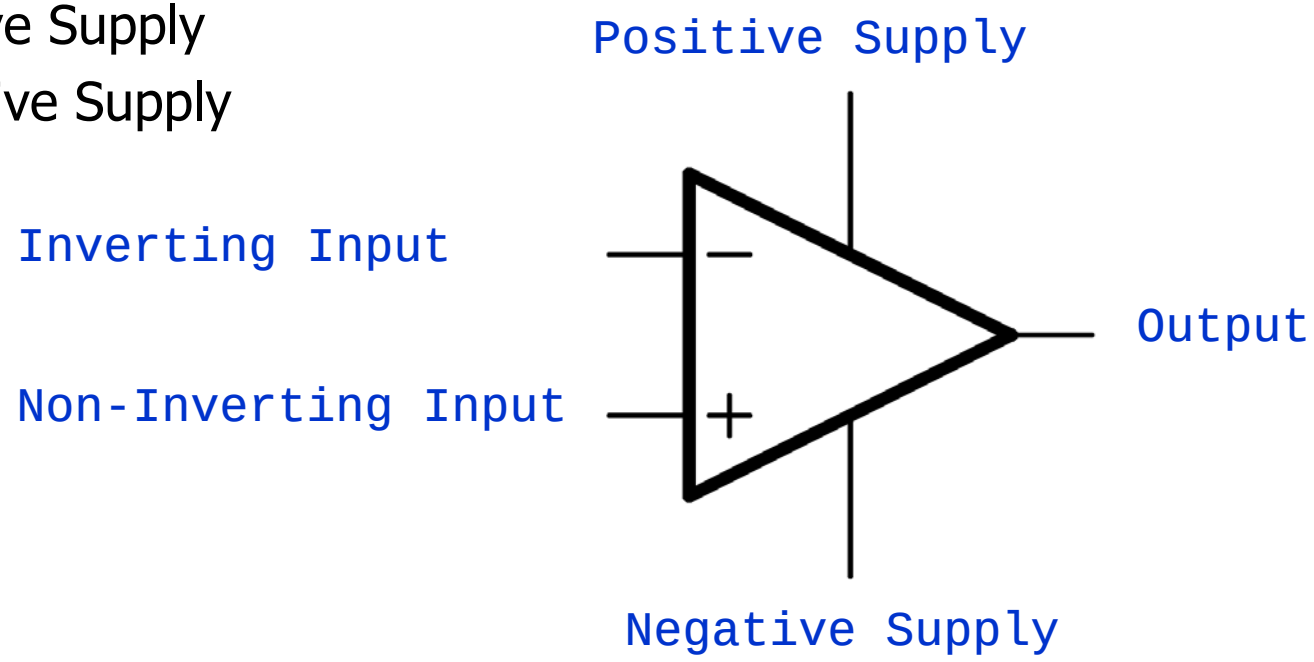


Op-Amp Fundamentals

- An op amp is a …
  - Differential input, …
  - Single-ended output …
  - Amplifier
- In other words … an Op Amp processes small input signals … developing a single-ended output

Op-Amp

- Op Amp has a minimum of 5 terminals
  - "-" Inverting input
  - "+" Non-Inverting input
  - Output
  - Positive Supply
  - Negative Supply



Comparators

- The output goes positive when the non-inverting input is more positive than the inverting input
- The output goes negative when the inverting input is more positive than the non-inverting input
- Therefore …

IF V- < V+ … output is positive

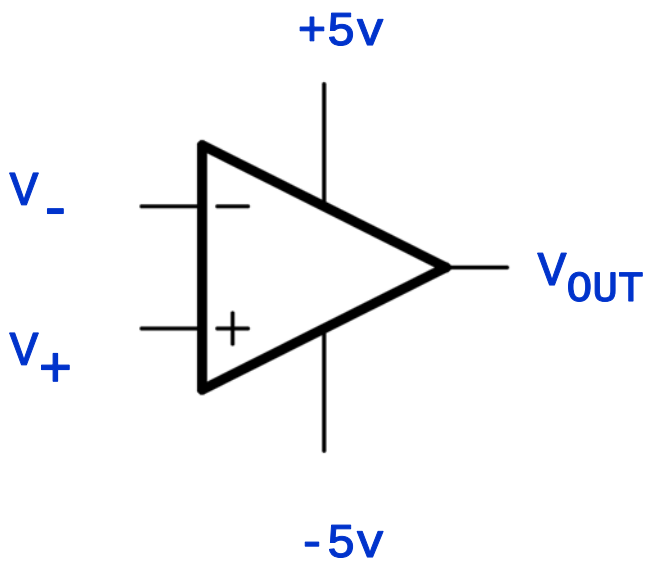
IF V+ < V- … output is negative

IF V- = V+ … output is zero

Comparator Example

Example: … we have a positive supply voltage of +5 and a negative supply voltage of -5

What is V_{OUT} for the values indicated in the table?

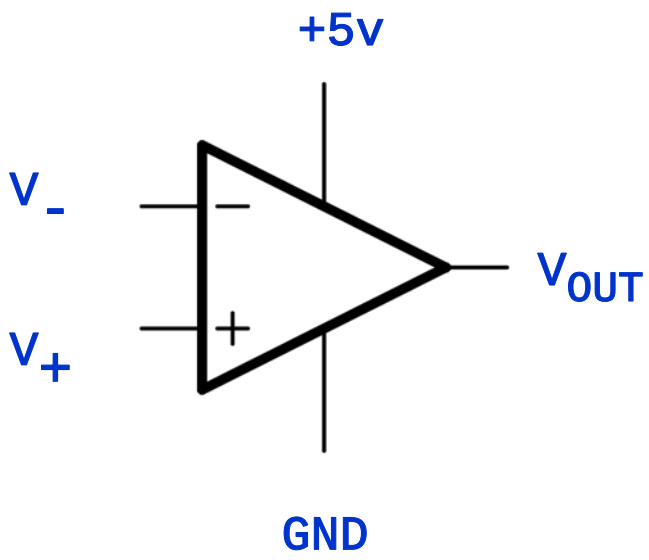


V_-	V_+	V_{OUT}
+1	-1	-5
+1	+2	+5
+2	+1	-5
0	0	0
-1	+1	+5
0	-1	-5
0	+1	+5
+3	+3	0

Comparator Example

Example 2 … we have a positive supply voltage of +5 and a negative supply voltage set to ground.

What is V_{OUT} for the values indicated in the table?

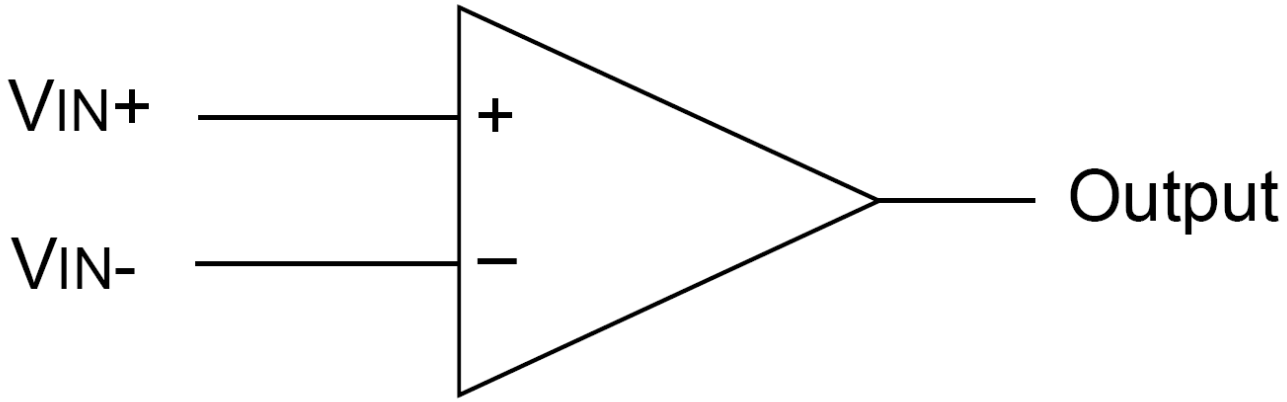


V_-	V_+	V_{OUT}
+2	-1	0
+1	+2	+5
+2	-2	0
0	-1	0
-2	-2	0
0	+1	+5

The PIC16F684 Comparator Module

- Dual comparators
- Multiple comparator configurations
- Comparator outputs are available internally/externally
- Programmable output polarity
- Interrupt-on-change
- Wake-up from Sleep
- Timer1 gate (count enable) – ONLY C2 CAN BE LINKED TO TIMER1
- Output synchronization to Timer1 clock input
- Programmable voltage reference

PIC16F684 Comparators



- When the analog voltage at VIN+ is < the analog voltage at VIN- …
  - The output of the comparator is a digital low level

- When the analog voltage at VIN+ is > the analog voltage at VIN- …
  - The output of the comparator is a digital high level

PIC16F684 Comparator Configuration

- There are eight modes of operation for the comparator
    1. Comparators Reset
    2. Three Inputs Multiplexed to Two Comparators
    3. Four Inputs Multiplexed to Two Comparators
    4. Two Common Reference Comparators
    5. Two Independent Comparators
    6. One Independent Comparator
    7. Two Common Reference Comparators with Outputs
    8. Comparators Off

- The CM<2:0> bits of the CMCON0 register are used to select these modes

- I/O lines change as a function of the mode

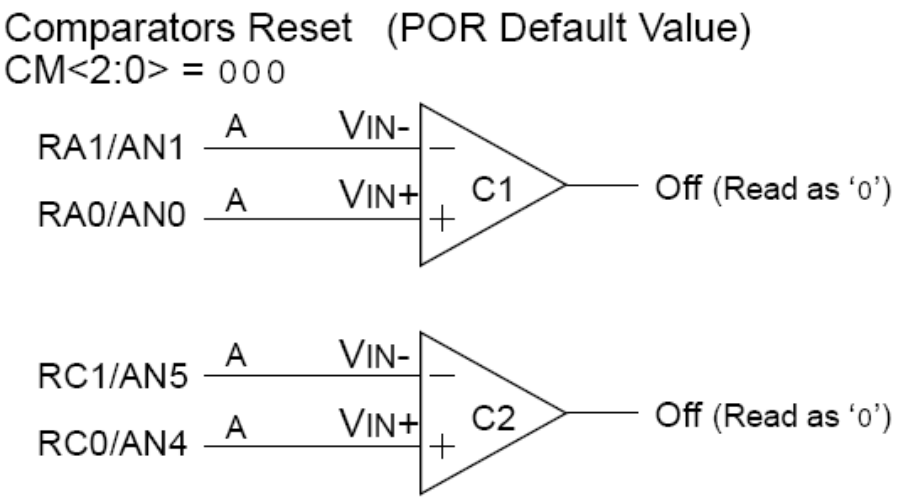
Comparator Module Control (CMCON0) Register

- The CMCON0 register (Register 8-1) provides access to the following comparator features:
    - Mode selection (Selects one of eight modes)
    - Output state
    - Output polarity
    - Input switch

PIC16F684 Comparator Modes of Operation (CMCON0) Register

1. Comparators Reset –
   `CM<2:0> = 000`
2. Three Inputs Multiplexed to Two Comparators –
   `CM<2:0> = 001`
3. Four Inputs Multiplexed to Two Comparators –
   `CM<2:0> = 010`
4. Two Common Reference Comparators –
   `CM<2:0> = 011`
5. Two Independent Comparators –
   `CM<2:0> = 100`
6. One Independent Comparator –
   `CM<2:0> = 101`
7. Two Common Reference Comparators with Outputs –
   `CM<2:0> = 110`
8. Comparators Off –
   `CM<2:0> = 111`

PIC16F684 Comparator Modes of Operation
Comparators Reset - CM<2:0> = 000

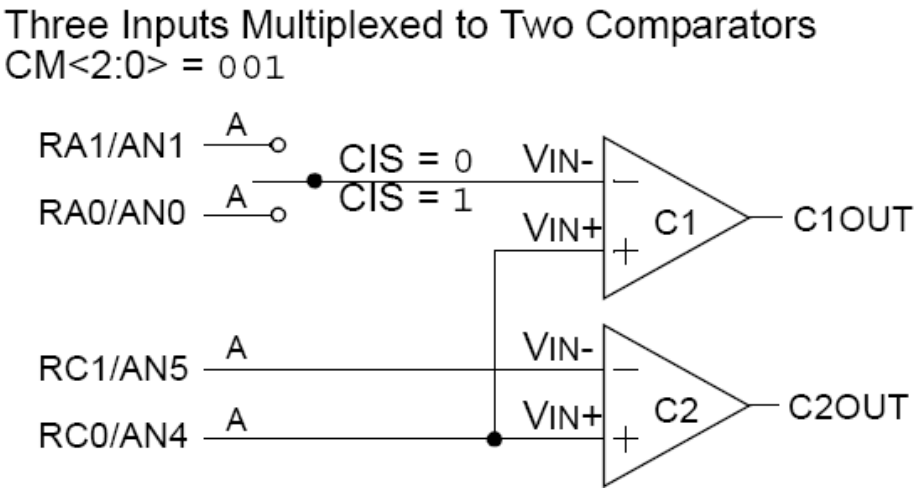


Legend: A = Analog Input, ports always read '0'
D = Digital Input

CIS (CMCON0<3>) is the Comparator Input Switch

PIC16F684 Comparator Modes of Operation

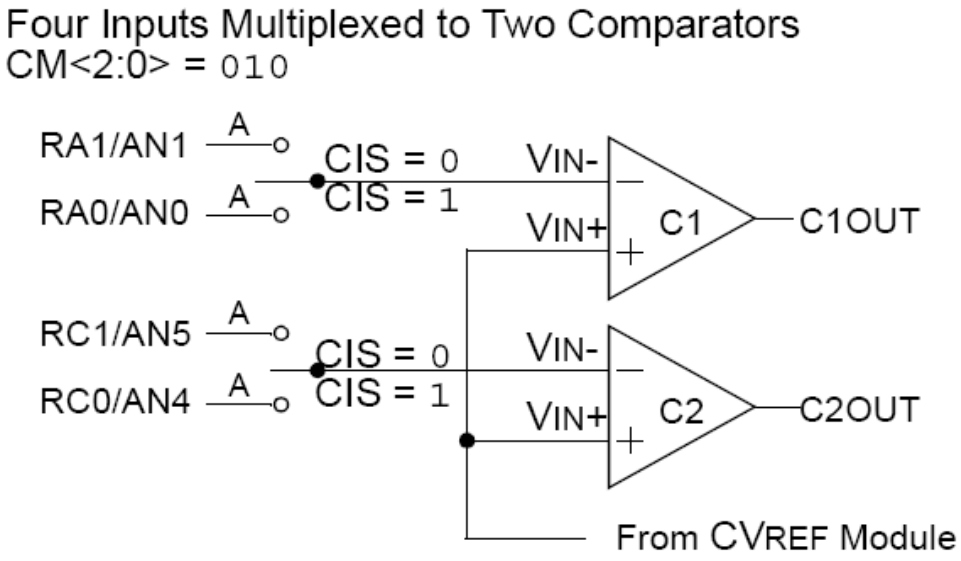
Three Inputs Multiplexed to Two Comparators – CM<2:0> = 001



Legend: A = Analog Input, ports always read '0'
D = Digital Input

CIS (CMCON0<3>) is the Comparator Input Switch

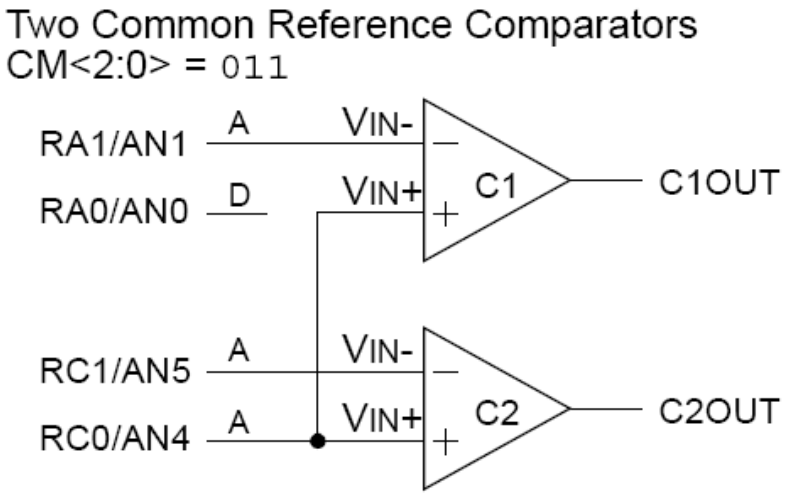
PIC16F684 Comparator Modes of Operation
Four Inputs Multiplexed to Two Comparators – CM<2:0> = 010



Legend: A = Analog Input, ports always read '0'
D = Digital Input

CIS (CMCON0<3>) is the Comparator Input Switch

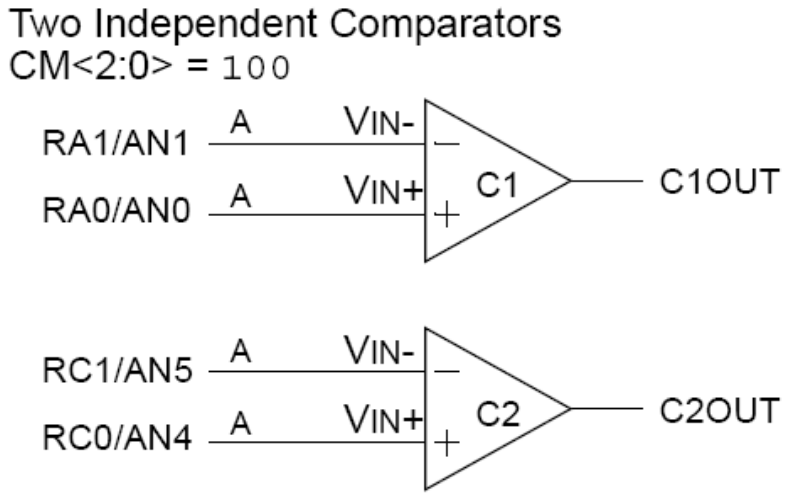
PIC16F684 Comparator Modes of Operation
Two Common Reference Comparators – CM<2:0> = 011



Legend: A = Analog Input, ports always read '0'
D = Digital Input

CIS (CMCON0<3>) is the Comparator Input Switch

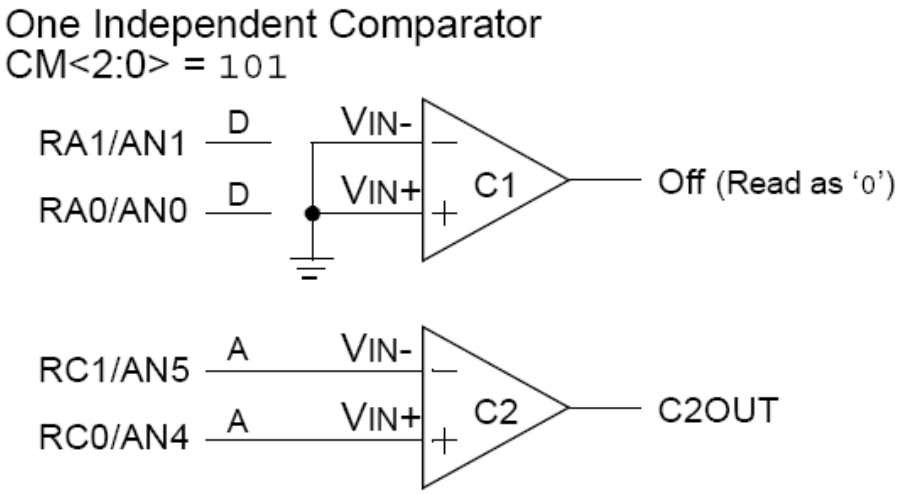
PIC16F684 Comparator Modes of Operation
Two Independent Comparators – CM<2:0> = 100



Legend: A = Analog Input, ports always read '0'
D = Digital Input

CIS (CMCON0<3>) is the Comparator Input Switch

PIC16F684 Comparator Modes of Operation
One Independent Comparator – CM<2:0> = 101

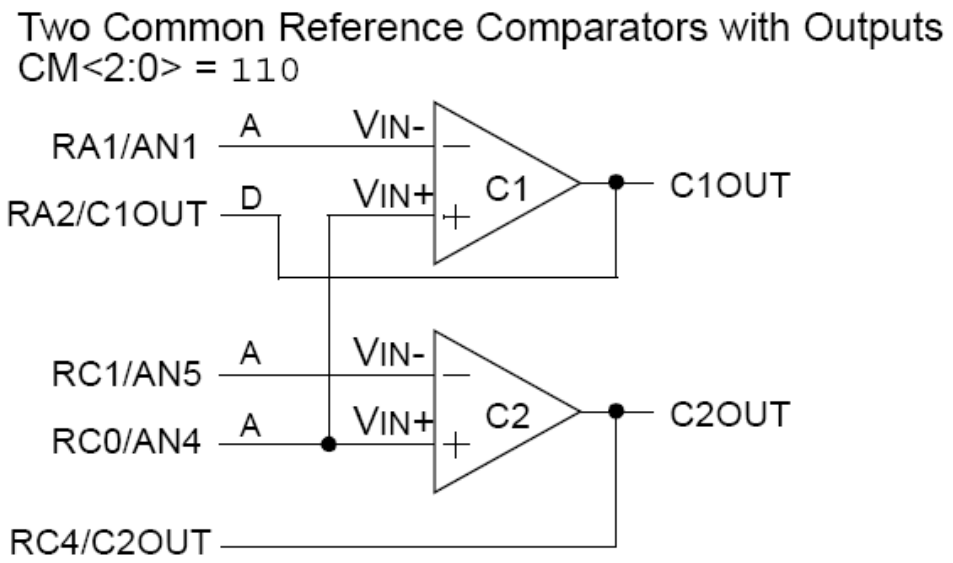


Legend: A = Analog Input, ports always read '0'
D = Digital Input

CIS (CMCON0<3>) is the Comparator Input Switch

PIC16F684 Comparator Modes of Operation

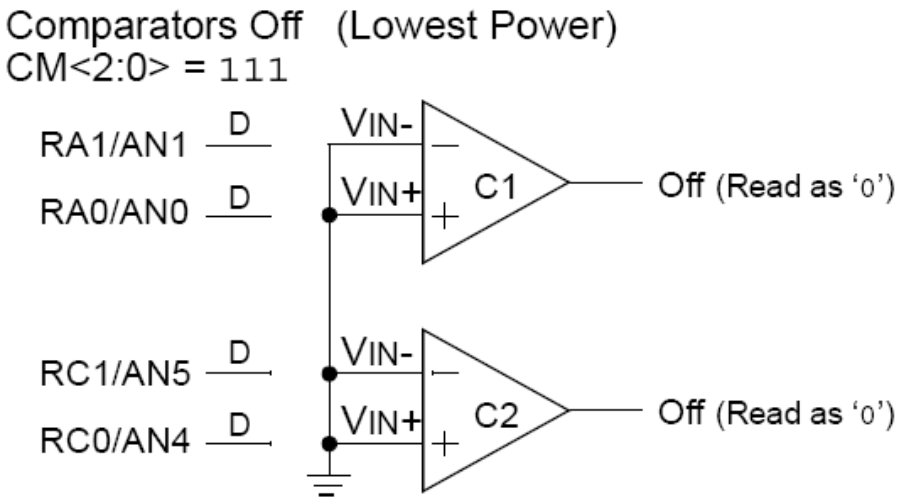
Two Common Reference Comparators with Outputs – CM<2:0> = 110



Legend: A = Analog Input, ports always read '0'
D = Digital Input

CIS (CMCON0<3>) is the Comparator Input Switch

PIC16F684 Comparator Modes of Operation
Comparators Off – CM<2:0> = 111



Legend: A = Analog Input, ports always read '0'
D = Digital Input

CIS (CMCON0<3>) is the Comparator Input Switch

PIC16F684 Comparator Output State

- Each comparator state can always be read internally via the associated CxOUT bit of the CMCON0 register
- The comparator outputs are directed to the CxOUT pins when CM<2:0> = 110 (Two Common Reference Comparators *with Outputs*)
    - C1OUT is Pin 11
    - C2OUT is Pin 6
  - When this mode is selected, the TRIS bits for the associated CxOUT pins must be cleared (0) to enable the output drivers

PIC16F684 Comparator Output Polarity

- The polarity of the comparator output can be inverted by setting the CxINV bits for the associated comparator (CMCON0<5:4>)
- Clearing CxINV results in a non-inverted output
- The following table shows the output state versus input conditions and the polarity bit

Input Conditions	CINV	CxOUT
$V_{IN-} > V_{IN+}$	0	0
$V_{IN-} < V_{IN+}$	0	1
$V_{IN-} > V_{IN+}$	1	1
$V_{IN-} < V_{IN+}$	1	0

REGISTER 8-1: CMCON0 – COMPARATOR CONFIGURATION REGISTER (ADDRESS: 19h)

R-0	R-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
C2OUT	C1OUT	C2INV	C1INV	CIS	CM2	CM1	CM0
bit 7							bit 0

bit 7 **C2OUT**: Comparator 2 Output bit
When C2INV = 0:
1 = C2 VIN+ > C2 VIN-
0 = C2 VIN+ < C2 VIN-
When C2INV = 1:
1 = C2 VIN+ < C2 VIN-
0 = C2 VIN+ > C2 VIN-

bit 6 **C1OUT**: Comparator 1 Output bit
When C1INV = 0:
1 = C1 VIN+ > C1 VIN-
0 = C1 VIN+ < C1 VIN-
When C1INV = 1:
1 = C1 VIN+ < C1 VIN-
0 = C1 VIN+ > C1 VIN-

bit 5 **C2INV**: Comparator 2 Output Inversion bit
1 = C2 output inverted
0 = C2 output not inverted

bit 4 **C1INV**: Comparator 1 Output Inversion bit
1 = C1 Output inverted
0 = C1 Output not inverted

bit 3 **CIS**: Comparator Input Switch bit
When CM<2:0> = 010:
1 = C1 VIN- connects to RA0/AN0
C2 VIN- connects to RC0/AN4
0 = C1 VIN- connects to RA1/AN1
C2 VIN- connects to RC1/AN5
When CM<2:0> = 001:
1 = C1 VIN- connects to RA0/AN0
0 = C1 VIN- connects to RA1/AN1

bit 2-0 **CM<2:0>:** Comparator Mode bits
Figure 8-3 shows the Comparator modes and CM<2:0> bit settings

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

Comparator Outputs

- The comparator outputs are read through the `CMCON0` register
- These bits are *read-only*
- The comparator outputs may also be directly output to the RA2 and RC4 I/O pins
- When enabled, multiplexers in the output path of the RA2 and RC4 pins will switch and the output of each pin will be the unsynchronized output of the comparator

Comparator Outputs

- The TRIS bits will still function as an output enable/disable for the RA2 and RC4 pins while in this mode

- The polarity of the comparator outputs can be changed using the C1INV and C2INV bits (CMCON0<5:4>)

Comparator Interrupts

- The comparator interrupt flags are set whenever there is a change in the output value of its respective comparator

- Software will need to maintain information about the status of the output bits, as read from `CMCON0<7:6>` to determine the actual change that has occurred

Comparator Reference

- The comparator module also allows the selection of an internally generated voltage reference for one of the comparator inputs

- The VRCON register (Register 8-3) controls the voltage reference module

CONFIGURING THE VOLTAGE REFERENCE

- The voltage reference can output 32 distinct voltage levels
  - 16 in a high range … and …
  - 16 in a low range
- The following equation determines the output voltages:

VRR = 1 *(low range):* $CV_{REF} = (VR3:VR0/24) \times V_{DD}$

VRR = 0 *(high range):*

$$CV_{REF} = (V_{DD}/4) + (VR3:VR0 \times V_{DD}/32)$$

Comparator Voltage References

VDD = 5 volts

VRCON, 5 = 1 (low)		VRCON, 5 = 0 (high)	
VCON 3:0	CVref	VCON 3:0	CVref
0000	0.00	0000	1.25
0001	0.21	0001	1.41
0010	0.42	0010	1.56
0011	0.63	0011	1.72
0100	0.83	0100	1.88
0101	1.04	0101	2.03
0110	1.25	0110	2.19
0111	1.46	0111	2.34
1000	1.67	1000	2.50
1001	1.88	1001	2.66
1010	2.08	1010	2.81
1011	2.29	1011	2.97
1100	2.50	1100	3.13
1101	2.71	1101	3.28
1110	2.92	1110	3.44
1111	3.13	1111	3.59

REGISTER 8-3: VRCON – VOLTAGE REFERENCE CONTROL REGISTER (ADDRESS: 99h)

R/W-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
VREN	—	VRR	—	VR3	VR2	VR1	VR0
bit 7							bit 0

bit 7 **VREN:** CVREF Enable bit
1 = CVREF circuit powered on
0 = CVREF circuit powered down, no IDD drain and CVREF = VSS

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

bit 5 **VRR:** CVREF Range Selection bit
1 = Low range
0 = High range

bit 4 **Unimplemented:** Read as '0'

bit 3-0 **VR<3:0>:** CVREF Value Selection 0 ≤ VR<3:0> ≤ 15
When VRR = 1: CVREF = (VR<3:0>/24) * VDD
When VRR = 0: CVREF = VDD/4 + (VR<3:0>/32) * VDD

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

REGISTERS ASSOCIATED WITH COMPARATOR MODULE

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR, BOD	Value on all other Resets
0Bh/8Bh	INTCON	GIE	PEIE	T0IE	INTE	RAIE	T0IF	INTF	RAIF	0000 0000	0000 0000
0Ch	PIR1	EEIF	ADIF	CCP1IF	C2IF	C1IF	OSFIF	TMR2IF	TMR1IF	0000 0000	0000 0000
19h	CMCON0	C2OUT	C1OUT	C2INV	C1INV	CIS	CM2	CM1	CM0	0000 0000	0000 0000
1Ah	CMCON1	—	—	—	—	—	—	T1GSS	C2SYNC	---- --10	---- --10
85h	TRISA	—	—	TRISA5	TRISA4	TRISA3	TRISA2	TRISA1	TRISA0	--11 1111	--11 1111
87h	TRISC	—	—	TRISC5	TRISC4	TRISC3	TRISC2	TRISC1	TRISC0	--11 1111	--11 1111
8Ch	PIE1	EEIE	ADIE	CCP1IE	C2IE	C1IE	OSFIE	TMR2IE	TMR1IE	0000 0000	0000 0000
99h	VRCON	VREN	—	VRR	—	VR3	VR2	VR1	VR0	0-0- 0000	0-0- 0000

Legend: x = unknown, u = unchanged, — = unimplemented, read as '0'. Shaded cells are not used by the Capture, Compare or Timer1 module.

Timers and Counters ...

Timers/Counters

- Why do we need timers (or counters)?
  - For accurate event timing and counting which is often needed in microcontroller applications
    - For example
      - A sensor on a motor shaft which gives one pulse per revolution of the shaft
        - » The number of pulses per second will give shaft speed

Timers/Counters

- Instead of using loops for delays … we can use hardware timers
- Advantages …
  - Processor is free to handle other tasks rather than sitting in a loop for timing and doing nothing
  - Timer is more accurate for measuring a loop than using the stopwatch function
  - You can calculate the exact time instead of using trial and error

How Timers/Counters Work

- A timer is a peripheral that measures elapsed time …
  - Typically by counting processor cycles or clocks
- A counter measures elapsed time …
  - Using external events
- A timer is setup by programming a register with a specific value
  - Some processors count up …
  - Others count down
    - An interrupt is generated when a certain point is met

How Timers/Counters Work

- Timer counts cycles from either …
  - The main clock … or …
  - An external clock fed from an external source
- Many processors include multiple internal clock that can be used to drive the timers
- Many processors have multiple timers

Timer/Counter

- Digital output waveforms are easy to generate by writing ones and zeros to a port line
- Delays between pulses create the output frequency
- Changing delays between pulses will change the output frequency
- PIC16F684 timer/counter can be used to generate repetitive waveforms pulses
    - Allows the microcontroller to perform other tasks while generating these repetitive waveforms

Timing with the Microcontroller

- All microcontrollers have timer circuits
    - Some have multiple
- The timers are …
    - Hardware binary counters
        - Allow time interval measurement … or …
        - Count
    - To be carried out separately from program execution

PIC16F684 Timers ...

Timing with the Microcontroller

- Watchdog Timer
- Timer0 …
  - 8-bit register that can count pulses up to 0xFF (255)
  - 8-bit programmable prescaler
- Timer1 ….
  - 16-bit register that can count up to 0xFFFF (65, 535)
  - Timer/counter with prescaler

Timing with the Microcontroller

- Timer 2 …
  - 8-bit register
  - Timer/counter with 8-bit period register, prescaler and postscaler
  - The values of TMR2 and PR2 are constantly compared to determine when they match.
    - TMR2 will increment from 00h until it matches the value in PR2
      - When a match occurs … two things happen …
        - » TMR2 is reset to 00h on the next increment cycle
        - » Postscaler is incremented
      - The match output of the Timer2/PR2 comparator is fed into the Timer2 postscaler
      - The output of postscaler is used to set the interrupt flag bit

Timing with the Microcontroller

- Timing is achieved by counting the clock pulses
- The `OPTION` Register allows us to slow down the these pulses (using what is called a "Prescaler") by a factor of …
  - 2, 4, 8, 16, 32, 64, 128, or 256

Timer0 ...

The Timer0 Module Register (TMR0)

- The Timer0 module is an 8-bit timer/counter with the following features:
    - 8-bit timer/counter register (TMR0)
    - 8-bit prescaler (shared with Watchdog Timer)
    - Programmable internal or external clock source
    - Programmable external clock edge selection
    - Interrupt on overflow

Timer0 Operation

- When used as a timer …
  - The Timer0 module can be used as either an …

    8-bit timer … or …

    an 8-bit counter

Timer0 --- 8-BIT TIMER MODE

- When used as a timer, the Timer0 module will …
  - Increment every instruction cycle (without prescaler)
- Timer mode is selected by clearing the T0CS bit of the OPTION register
    - Set (OPTION_REG<5>) to '0'
- When TMR0 is written, the increment is inhibited for two instruction cycles immediately following the write
    - The TMR0 register can be adjusted, in order to account for the two instruction cycle delay when TMR0 is written

Timer0 --- 8-BIT COUNTER MODE

- When used as a counter … the Timer0 module will …
  - Increment on every rising or falling edge of the T0CKI pin
    - The incrementing edge is determined by the T0SE bit of the OPTION register (OPTION_REG<4>)

      "1" = Increment on high-to-low transition on T0CKI pin

      "0" = Increment on low-to-high transition on T0CKI pin

- Counter mode is selected by setting the T0CS bit of the OPTION register to '1' (OPTION_REG<5>)

Timer 0 (TMR0) Interrupt

- A Timer0 interrupt is generated when the TMR0 register timer/counter overflows from FFh to 00h
- This overflow sets the T0IF bit (INTCON<2>)
- The interrupt can be masked by clearing the T0IE bit (INTCON<5>)
- The T0IF bit must be cleared in software by the Timer0 module Interrupt Service Routine before re-enabling this interrupt
- The Timer0 interrupt cannot wake the processor from Sleep since the timer is shut off during Sleep

Timer0 --- Prescaler

- An 8-bit counter is available as a prescaler for the Timer0 module
- The prescaler assignment is controlled in software by the control bit PSA (OPTION_REG<3>)
- Clearing the PSA bit will assign the prescaler to Timer0
- Prescale values are selectable via the PS<2:0> bits (OPTION_REG<2:0>)
- The prescaler is not readable or writable
- When assigned to the Timer0 module, all instructions writing to the TMR0 register (e.g., CLRF 1, MOVWF 1, BSF 1, x....etc.) will clear the prescaler

TMR0

- When used as a counter …
  - the register is incremented
    - each time a clock pulse is applied to pin `TOCK1`

- When used as a timer …
  - the register increments at a rate determined by
    - the system clock frequency … and …
    - a prescaler

  - Prescalers rate vary from …
    - 1:2 … to … 1:256

  - Prescalers are selected from the `OPTION_REG`

TMR0 – Timer Mode

- Microcontroller oscillator = f_{osc} = 4 MHz
- The internal oscillator frequency seen at TOCS is …

  f_{osc} divided by 4 = f_{osc} /4

- Oscillator period = T_{osc} = 1/f_{osc}= 0.25 microseconds

An Example

f_{osc} = 4 MHz

Timer speed = ¼ f_{osc} = 1 MHz

To turn an LED on for 1 sec we would need to count 1,000,000 pulses … a lot of pulses!

Prescaler can slow down the pulses … for example …

1,000,000/256 = 3906.25 or 3906 pulses

So to turn the LED on for 1 sec we need 3906 pulses, for 0.5 sec we need 1953 pulses.

Overflow

- Timer0 will generate an interrupt when the TMR0 register overflows from FFh to 00h
- The T0IF interrupt flag bit of the INTCON register is set every time the TMR0 register overflows, regardless of whether or not the Timer0 interrupt is enabled
- The T0IF bit must be cleared in software
- The Timer0 interrupt enable is the T0IE bit of the INTCON register
- *Overflow time* is the time it will take until TMR0 register overflows

Overflow Time

$$\text{Overflow time} = 4 \times T_{OSC} \times \text{Prescaler} \times (256 - \text{TMR0})$$

- Where …
  - Overflow time is in microseconds
  - 4 is as a result of fosc being divided by 4
  - T_{OSC} is the oscillator period in microseconds
  - Prescaler is the prescaler value chosen using OPTION_REG
  - TMR0 is the value loaded into TMR0 register

Example – Overflow Time

- Assume 4 MHz microcontroller oscillator
- Prescaler chosen as 1:8 (`PS2:PS0` to "`010`")
- Assume `TMR0` is decimal 100

4 MHz clock has a period T=1/f = 1/4MHz = 0.25 µsec

Using the formula …

```
Overflow time = 4 x 0.25 x 8 x (256 – 100) =
```

```
Overflow time = 1248 µsec = 1.248 msec
```

Determining TMR0 Value

- We normally need to know the value to load into `TMR0` for the required overflow time

- Modifying the prior equation … we obtain …

$$TMR0 = 256 - (Overflow\ time)/(4 \times T_{OSC} \times Prescaler)$$

- Where …
  - Overflow time is in microseconds
  - 4 is as a result of fosc being divided by 4
  - T_{OSC} is the oscillator period in microseconds
  - Prescaler is the prescaler value chosen using OPTION_REG
  - TMR0 is the value loaded into TMR0 register

Example – TMR0 Value Determination

- Assume 4 MHz microcontroller oscillator
- Prescaler chosen as 1:8 (`PS2:PS0` to "`010`")
- Interrupt to be generated after 500 µsec

4 MHz clock has a period T=1/f = 1/4MHz = 0.25 µsec

Using the formula …

```
TMR0 = 256 – 500/(4 x 0.25 x 8) = 193.5
```

The nearest number we can load into TMR0 is 193

Required TMRO Values for Different Overflow Times (4 MHz Oscillator)

Time to overflow (µs)	PRESCALER							
	2	4	8	16	32	64	128	256
100	206	231	243	250	253	254	-	-
200	156	206	231	243	250	253	254	-
300	106	181	218	237	246	251	253	255
400	56	156	206	231	243	250	253	254
500	6	131	193	224	240	248	252	254
600	-	106	181	218	237	16	251	253
700	-	81	168	212	234	245	250	253
800	-	56	156	206	231	243	250	253
1,000	-	6	131	193	225	240	248	252
5,000	-	-	-	-	100	178	77	236
10,000	-	-	-	-	-	100	178	217
20,000	-	-	-	-	-	-	100	178
30,000	-	-	-	-	-	-	-	139
40,000	-	-	-	-	-	-	-	100
50,000	-	-	-	-	-	-	-	60
60,000	-	-	-	-	-	-	-	21

TABLE 5-1: REGISTERS ASSOCIATED WITH TIMER0

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR, BOD	Value on all other Resets
01h	TMR0	Timer0 Module register								xxxx xxxx	uuuu uuuu
0Bh/8Bh	INTCON	GIE	PEIE	T0IE	INTE	RAIE	T0IF	INTF	RAIF	0000 0000	0000 0000
81h	OPTION_REG	$\overline{\text{RAPU}}$	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	1111 1111	1111 1111
85h	TRISA	—	—	TRISA5	TRISA4	TRISA3	TRISA2	TRISA1	TRISA0	--11 1111	--11 1111

Legend: — = Unimplemented locations, read as '0', u = unchanged, x = unknown. Shaded cells are not used by the Timer0 module.

REGISTER 2-3: INTCON – INTERRUPT CONTROL REGISTER (ADDRESS: 0Bh OR 8Bh)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
GIE	PEIE	T0IE	INTE	RAIE	T0IF	INTF	RAIF
bit 7							bit 0

bit 7 **GIE:** Global Interrupt Enable bit
1 = Enables all unmasked interrupts
0 = Disables all interrupts

bit 6 **PEIE:** Peripheral Interrupt Enable bit
1 = Enables all unmasked peripheral interrupts
0 = Disables all peripheral interrupts

bit 5 **T0IE:** TMR0 Overflow Interrupt Enable bit
1 = Enables the TMR0 interrupt
0 = Disables the TMR0 interrupt

bit 4 **INTE:** RA2/INT External Interrupt Enable bit
1 = Enables the RA2/INT external interrupt
0 = Disables the RA2/INT external interrupt

bit 3 **RAIE:** PORTA Change Interrupt Enable bit(1)
1 = Enables the PORTA change interrupt
0 = Disables the PORTA change interrupt

bit 2 **T0IF:** TMR0 Overflow Interrupt Flag bit(2)
1 = TMR0 register has overflowed (must be cleared in software)
0 = TMR0 register did not overflow

bit 1 **INTF:** RA2/INT External Interrupt Flag bit
1 = The RA2/INT external interrupt occurred (must be cleared in software)
0 = The RA2/INT external interrupt did not occur

bit 0 **RAIF:** PORTA Change Interrupt Flag bit
1 = When at least one of the PORTA <5:0> pins changed state (must be cleared in software)
0 = None of the PORTA <5:0> pins have changed state

Note 1: IOCA register must also be enabled.

2: T0IF bit is set when Timer0 rolls over. Timer0 is unchanged on Reset and should be initialized before clearing T0IF bit.

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

REGISTER 2-2: OPTION_REG – OPTION REGISTER (ADDRESS: 81h)

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
$\overline{\text{RAPU}}$	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
bit 7							bit 0

bit 7 **$\overline{\text{RAPU}}$:** PORTA Pull-up Enable bit
1 = PORTA pull-ups are disabled
0 = PORTA pull-ups are enabled by individual port latch values

bit 6 **INTEDG:** Interrupt Edge Select bit
1 = Interrupt on rising edge of RA2/INT pin
0 = Interrupt on falling edge of RA2/INT pin

bit 5 **T0CS:** TMR0 Clock Source Select bit
1 = Transition on RA2/T0CKI pin
0 = Internal instruction cycle clock (CLKOUT)

bit 4 **T0SE:** TMR0 Source Edge Select bit
1 = Increment on high-to-low transition on RA2/T0CKI pin
0 = Increment on low-to-high transition on RA2/T0CKI pin

bit 3 **PSA:** Prescaler Assignment bit
1 = Prescaler is assigned to the WDT
0 = Prescaler is assigned to the Timer0 module

bit 2-0 **PS<2:0>:** Prescaler Rate Select bits

BIT VALUE	TMR0 RATE	WDT RATE
000	1 : 2	1 : 1
001	1 : 4	1 : 2
010	1 : 8	1 : 4
011	1 : 16	1 : 8
100	1 : 32	1 : 16
101	1 : 64	1 : 32
110	1 : 128	1 : 64
111	1 : 256	1 : 128

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

Commands for TMR0

- **Lab # 5 uses TMR0 ...**

```
; --- New Delay Routine using TMR0 ----------------------------
Delay    CLRF     TMR0              ; Clear Timer0 register, start counting
         CLRF     INTCON            ; Disable interrupts and clear T0IF
         BSF      STATUS, RP0       ; Bank1
         MOVLW    0xC7              ; PortB pull-ups are disabled,
         MOVWF    OPTION_REG        ; Interrupt on rising edge of RB0
                                    ; Timer0 increment from internal clock
                                    ; with a prescaler of 1:126.
         BCF      STATUS, RP0       ; Bank0
         BSF      INTCON, T0IE      ; Enable TMR0 interrupt
again    btfss    INTCON,2          ; Bit 2 set?
         goto     again             ; No, bit is clear, goto again
         return
; --- End of new Delay routine ------------------------------
```

Timers1...

Timer1 Module

- Similar to TMR0
- Timer1 Features
  - 16-bit timer/counter register pair (TMR1H:TMR1L)
  - Programmable internal or external clock source
  - 3-bit prescaler
  - Optional LP oscillator
  - Synchronous or asynchronous operation
  - Timer1 gate (count enable) via comparator or T1G pin
  - Interrupt on overflow
  - Wake-up on overflow (external clock, Asynchronous mode only)
  - Time base for the Capture/Compare function
  - Special Event Trigger (with ECCP)
  - Comparator output synchronization to Timer1 clock
- See PIC16F684 Datasheet for related information

Timers2 ...

Timer2 Module

- Similar to TMR0

- Timer2 Features
  - 8-bit timer register (TMR2)
  - 8-bit period register (PR2)
  - Interrupt on TMR2 match with PR2
  - Software programmable prescaler (1:1, 1:4, 1:16)
  - Software programmable postscaler (1:1 to 1:16)

- See PIC16F684 Datasheet for related information

Watchdog Timer ...

Watchdog Timer

- A watchdog timer is a special hardware fail-safe mechanism
  - Intervenes if the software stops functioning properly
- The watchdog timer is periodically reset by software
- If the software crashes or hangs
  - The watchdog timer soon expires
    - Causing the entire system to be reset automatically

Watchdog Timer

- The inclusion and use of a watchdog timer is a common way to deal with unexpected software hangs or crashes that may occur after the system is deployed

- For example … systems deployed in space

  – If the software hangs or crashes … you cannot reset it manually

  – Instead you need to build in an automatic recovery mechanism into the system

Watchdog Timer

- Always implement the code that handles resetting the watchdog timer in the main processor loop
- Never implement the watchdog timer reset in an ISR (Interrupt Service Routine)
  - Reason …
    - The main processing loop can hang while the interrupts and ISR continue to function
      - Meaning the watchdog timer would never be able to reset the system

PIC16F684 Watchdog Timer (WDT)

- The WDT has the following features:
  - Operates from the LFINTOSC (31 kHz)
  - Contains a 16-bit prescaler
  - Shares an 8-bit prescaler with Timer0
  - Time-out period is from 1 ms to 268 seconds
  - Configuration bit and software controlled

WDT CONTROL

- The WDTE bit is located in the Configuration Word register
  - When set, the WDT runs continuously
    - When the WDTE bit in the Configuration Word register is set, the SWDTEN bit of the WDTCON register has no effect
    - If WDTE is clear, then the SWDTEN bit can be used to enable and disable the WDT
      - Setting the bit will enable it and clearing the bit will disable it

PWM
Pulse Width Modulation ...

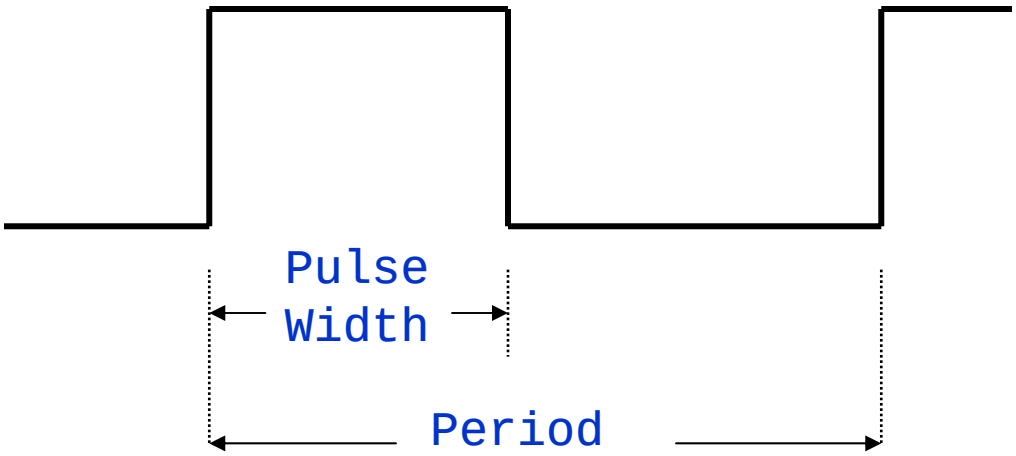
PWM – Pulse Width Modulation

- PWM Basics
  - Involves outputting a train of pulses at a fixed frequency
  - The Duty Cycle is varied to change the average output voltage

PWM

- Why use PWM?
  - PWM is a powerful technique for controlling analog circuits with a processor's digital outputs
- PWM is employed in a wide variety of applications:
  - Measurements
  - Communications
  - Power control
  - Conversions

Duty Cycle --- Logic High as a percentage of PWM period



$$\text{Duty Cycle} = \frac{\text{Pulse Width}}{\text{Period}} \times 100\%$$

Duty Cycle

- At the end of the duty cycle …
  - The output goes low
- At the end of the period …
  - The output goes high … except …
    - For special cases where the duty cycle is 100% or 0%

Analog Circuits

- An analog signal has a continuously varying value in both time and magnitude
- Digital signals take values from a finite set of predetermined possibilities
- Analog voltages and currents can be used to control things directly
    - Volume of a car radio
        - Knob is connected to a variable resistor
- Analog circuits are hard to keep tunes … the drift over time

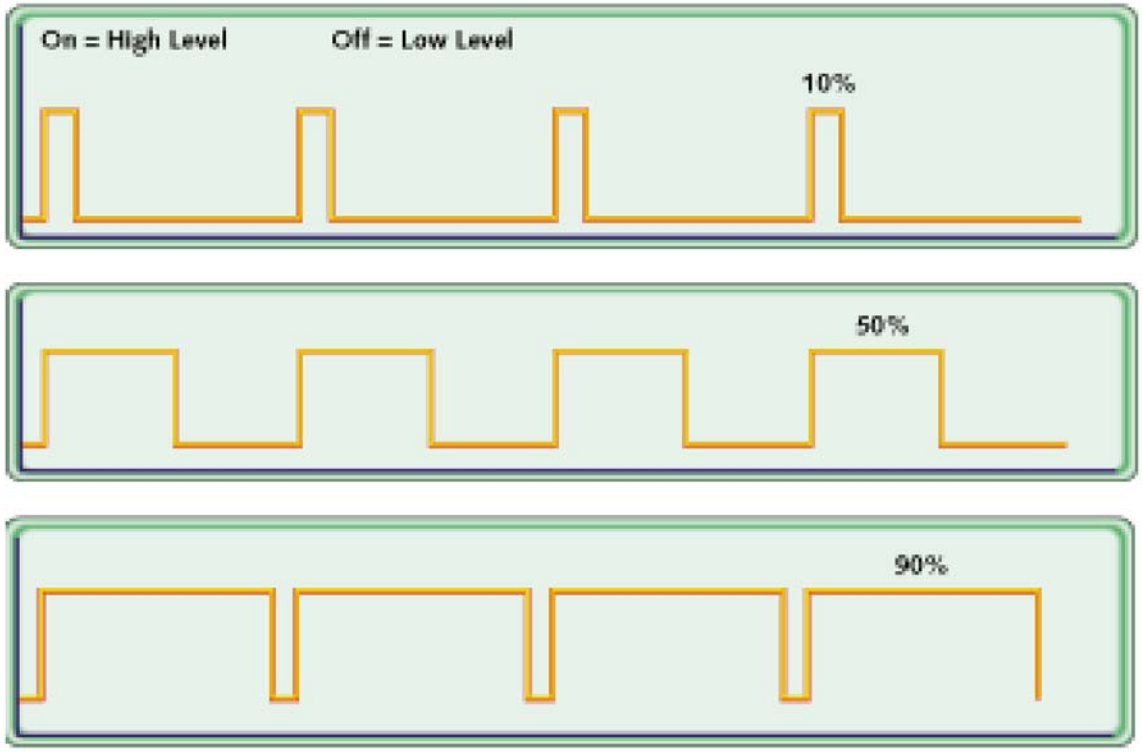
Digital Control

- Controlling analog circuits digitally can drastically reduce system costs and power consumption
- PWM is a way of digitally encoding analog signal levels
  - Use high-resolution counters
  - The duty cycle of a square wave is modulated to encode a specific analog signal level
- Many microcontrollers already contain PWM controllers

Digital Control

- PWM is still digital because at any instance in time …
  - The full DC supply is either …
    - Fully on … or …
    - Fully off
- Given a sufficiently small period of the PWM signal
  - Any analog value can be encoded with PWM

PWM Signals of Varying Duty Cycles



PWM Signals of Varying Duty Cycles

- The top signal shows a PWM output at a 10% duty cycle
  - That is, the signal is on for 10% of the period and off the other 90%
- The middle and lower signals show PWM outputs at 50% and 90% duty cycles, respectively
- These three PWM outputs encode three different analog signal values, at 10%, 50%, and 90% of the full strength
  - If, for example, the supply is 9V and the duty cycle is 10%, a 0.9V analog signal results

A Simple PWM Circuit

A Simple PWM Circuit

- The previous slide shows a simple circuit that could be driven using PWM
- A 9 V battery powers an incandescent light bulb
  - If we closed the switch connecting the battery and lamp for 50 ms, the bulb would receive 9 V during that interval
  - If we then opened the switch for the next 50 ms, the bulb would receive 0 V
  - If we repeat this cycle 10 times a second, the bulb will be lit as though it were connected to a 4.5 V battery (50% of 9 V)
- We say that the duty cycle is 50% and the modulating frequency is 10 Hz

A Simple PWM Circuit

- Most loads, inductive and capacitive alike, require a much higher modulating frequency than 10 Hz
- If the lamp was switched …
  - On for five seconds … then …
  - Off for five seconds … then …
  - On again
- The duty cycle be 50%, but the bulb would appear brightly lit for the first five seconds and off for the next
- In order for the bulb to see a voltage of 4.5 volts, the cycle period must be short relative to the load's response time to a change in the switch state

A Simple PWM Circuit

- To achieve the desired effect of a dimmer (but always lit) lamp
  - It is necessary to increase the modulating frequency
- The same is true in other applications of PWM
- **Common modulating frequencies range from 1 kHz to 200 kHz**

Advantages of PWM

- The signal remains digital all the way from the processor to the controlled system …
  - No digital-to-analog conversion is necessary
- By keeping the signal digital …
  - Noise effects are minimized
    - Noise can only affect a digital signal if it is strong enough to change a logic-1 to a logic-0, or vice versa

PWM Application - PWM-controlled brake

- To put it simply, a brake is a device that clamps down hard on something
- In many brakes, the amount of clamping pressure (or stopping power) is controlled with an analog input signal
- The more voltage or current that's applied to the brake, the more pressure the brake will exert

PWM Application - PWM-controlled brake

- The output of a PWM controller could be connected to a switch between the supply and the brake
- To produce more stopping power, the software need only increase the duty cycle of the PWM output
- If a specific amount of braking pressure is desired …
    - Measurements would need to be taken to determine the mathematical relationship between duty cycle and pressure
        - And the resulting formulae or lookup tables would be tweaked for operating temperature, surface wear, and so on

PWM Application - PWM-controlled brake

- To set the pressure on the brake to, say, 100 psi
  - The software would do a reverse lookup to determine the duty cycle that should produce that amount of force
  - It would then set the PWM duty cycle to the new value and the brake would respond accordingly
  - If a sensor is available in the system
    - The duty cycle can be tweaked, under closed – loop control, until the desired pressure is precisely achieved

PWM Controllers

- The duty cycle is the ratio of the on-time to the period
    - The modulating frequency is the inverse of the period
- To start PWM operation:
    - Set the period in the on-chip timer/counter that provides the modulating square wave
    - Set the on-time in the PWM control register
    - Set the direction of the PWM output of the general-purpose I/O pin
    - Start the timer
    - Enable the PWM controller

PIC16F684
Capture/Compare/PWM Module ...

PIC16F684

- Capture/Compare/PWM module …
  - Capture Mode …
    - Timer register starts counting when a signal is detected
    - When next signal is detected … count is saved
    - The count will correspond to the period of the input signal
  - Compare Mode …
    - A register is preloaded with a value
    - Continuously compared to Timer1 count
    - When count matches … an interrupt is generated

PIC16F684 PWM Mode

- The PWM mode generates a Pulse-Width Modulated signal on the `CCP1` pin

- The duty cycle, period and resolution are determined by the following registers:
    - `PR2`
    - `T2CON`
    - `CCPR1L`
    - `CCP1CON`

PIC16F684 PWM Mode

- In Pulse-Width Modulation (PWM) mode …
  - The CCP (Capture/Compare/PWM) module produces up to a 10-bit resolution PWM output on the CCP1 pin
    - Since the CCP1 pin is multiplexed with the PORT data latch …
      - The TRIS for that pin must be cleared to enable the CCP1 pin output driver

PIC16F684 PWM Period

- The PWM period is specified by the PR2 register of Timer2
- The PWM period can be calculated using the following Equation:

```
PWM Period =[(PR2)+ 1]•4•TOSC•(TMR2 Prescale Value)
```

PIC16F684 PWM Duty Cycle

- The PWM duty cycle is specified by writing a 10-bit value to multiple registers:
  - CCPR1L register … and …
  - CCP1<1:0> bits of the CCP1CON register
- To calculate the PWM pulse width use the following:

```
Pulse Width =(CCPR1L:CCP1CON<5:4>) •TOSC • (TMR2 Prescale Value)
```

PIC16F684 SETUP FOR PWM OPERATION

1. Disable the PWM pin (CCP1) output driver by setting the associated TRIS bit

2. Set the PWM period by loading the PR2 register

3. Configure the CCP module for the PWM mode by loading the CCP1CON register with the appropriate values

4. Set the PWM duty cycle by loading the CCPR1L register and CCP1 bits of the CCP1CON register

PIC16F684 SETUP FOR PWM OPERATION

5. Configure and start Timer2:

   - Clear the TMR2IF interrupt flag bit of the PIR1 register
   - Set the Timer2 prescale value by loading the T2CKPS bits of the T2CON register
   - Enable Timer2 by setting the TMR2ON bit of the T2CON register

6. Enable PWM output after a new PWM cycle has started:

   - Wait until Timer2 overflows (TMR2IF bit of the PIR1 register is set)
   - Enable the CCP1 pin output driver by clearing the associated TRIS bit

PIC16F684 - PWM (Enhanced Mode)

- The Enhanced PWM Mode can generate a PWM signal on up to four different output pins with up to 10-bits of resolution

- It can do this through four different PWM output modes:

    - Single PWM
    - Half-Bridge PWM
    - Full-Bridge PWM, Forward mode
    - Full-Bridge PWM, Reverse mode

- To select an Enhanced PWM mode, the `P1M` bits of the `CCP1CON` register must be set appropriately

PWM Initialization

```
        CLRF  CCP1CON    ; CCP Module is off
        CLRF  TMR2       ; Clear Timer2
        MOVLW 0x7F       ;
        MOVWF PR2        ;
        MOVLW 0x1F       ;
        MOVWF CCPR1L     ; Duty Cycle is 25% of PWM Period
        CLRF  INTCON     ; Disable interrupts and clear T0IF
        BSF   STATUS, RP0; Bank1
        BCF   TRISC, PWM1; Make pin output
        CLRF  PIE1       ; Disable peripheral interrupts
        BCF   STATUS, RP0; Bank0
        CLRF  PIR1       ; Clear peripheral interrupts Flags
        MOVLW 0x2C       ; PWM mode, 2 LSbs of Duty cycle = 10
        MOVWF CCP1CON    ;
        BSF   T2CON, TMR2ON ; Timer2 starts to increment
;
; The CCP1 interrupt is disabled,
; do polling on the TMR2 Interrupt flag bit
;
PWM_Period_Match
        BTFSS PIR1, TMR2IF
        GOTO PWM_Period_Match
;
; Update this PWM period and the following PWM Duty cycle
;
        BCF   PIR1, TMR2IF
```

Programming / Software

Programming

- Commands/instructions that we will encounter tonight
    - C commands/Assembly Language –
        - » ***Mixed C and Assembly Programming***
    - PIC16F684 control – ***None***

Mixed C and Assembly Programming ...

Mixing C and Assembler Code (Mixed Mode)

- First … Mixed Mode is …
    - Writing parts of a program in different languages
- Why would we want to write programs in Mixed Mode?
    - There are some low-level tasks that either can be better implemented in assembly, or can *only* be implemented in assembly language
    - *Hand optimize* the assembly code in ways that the compiler cannot
    - Assembly is also useful for time-critical or real-time processes …
        - The timing can be strictly controlled

Mixing C and Assembler Code

- Assembly language code can be mixed with C code using two different techniques …
  - Writing assembly code and placing it into a separate assembler module
  - OR … including it as in-line assembler in a C module

Mixing C and Assembler Code

- We will center our attention to including it as *in-line assembler in a C module*

- When including it as in-line assembler in a C module …

  – PIC instructions are directly embedded "in-line" into C code using the directives

    - #asm and #endasm

    - OR … the statement asm()

The *#asm* and *#endasm*

- The *#asm* and *#endasm* directives are
  - Used to start and end a block of assembly instructions
- The #asm and #endasm construct is not syntactically part of the C program

The *asm()* statement

- The *asm()* statement is used to embed a single assembler instruction
- This form looks and behaves like a C statement …
- However …
  - Each instruction must be encapsulated within an *asm()* statement

Mixing C and Assembler Code

- You should not use a *#asm* block within any C constructs such as …
  - if
  - while
  - do
  - etc
- In these cases, use only the asm("") form …
  - Which is a C statement and will correctly interact with all C flow-of-control structures

Local and *Global* Identifiers

- "*Global*" implies defined outside a function
- "*Local*" defined within a function
- For any non-local assembly symbol …
  - The GLOBAL directive must be used to link in with the symbol if it was defined elsewhere
- If it is a local symbol, then it may be used immediately

Header File Symbols

- If writing assembler code from within a C module …
  - SPECIAL FUNCTION REGISTERS (SFRs) …
    - Recall … SPRs are … PORTA, PORTC, TRISA, TRISC, etc
- SPRs may be accessed by referring to the symbols defined by the chip-specific C header files
  - Whenever you include <htc.h> into a C module, all the available SFRs are defined as absolute C variables
    - As the contents of this file is C code, it cannot be included into an assembler module, but assembler code can use these definitions

Header File Symbols

- To use a SFR in in-line assembler code from within the same C module that includes <htc.h> …
  - Simply use the symbol with an underscore character prepended to the name
- For example:

```
#include <htc.h>
void main(void)
{
        PORTA = 0x55;
        asm("movlw #0xAA");
        asm("movwf _PORTA");
```

Problems Encountered When Implementing ...

Problems Encountered When Implementing

- Must use the underscore character "_" in front of SPR names
- Bit names did not work … therefore …
  - Use the actual bit number vice the name … for example …
    - Instead of … asm(" BSF _PORTA, _RA4")
    - Use … asm(" BSF _PORTA, 4")
- Need a space (□) between the Register and the bit _PORTA,□4
- Place a // after the ; for example …

```
BCF     _STATUS, 5              ;  //Select Bank 0
```

Problems Encountered When Implementing

- Binary format …

  001111B vice b'001111‘

- See the require Hi-Tech number formats below …

  » These differ from the MPLAB assembler!

Table 4.3: ASPIC numbers and bases

Radix	Format
Binary	digits 0 and 1 followed by `B`
Octal	digits 0 to 7 followed by `O`, `Q`, `o` or `q`
Decimal	digits 0 to 9 followed by `D`, `d` or nothing
Hexadecimal	digits 0 to 9, A to F preceded by `0x` or followed by `H` *or* `h`

An Example ...

An Example …

- Lets look at our original C program …
  - Flashing LED D0 …
    - We wrote a delay function in C …
    - Then wrote it again when we used assembly language
- We will use the C program and "Mix" in the assembly
    - Delay code
    - Initialization routine
- Programs called Lecture_11A.c and Lecture_11B.c

An Example … (*WAS*)

```
 void PORTA_init(void)
{
                // WAS the following
                //PORTA = 0;            //  All PORTA Pins are low
                //CMCON0 = 7;           //  Turn off Comparators
                //ANSEL = 0;            //  Turn off ADC
                //TRISA = 0b001111;                   //  RA4 and 5 are outputs; RA0,1,2, and 3 are input
```

An Example … (*MIXED*)

```
 void PORTA_init(void)
{
    // New code is the following ...
   #asm
            BCF       _STATUS, 5                ;  //Select Bank 0
;
            CLRF      _PORTA                    ;  //Initialize PORTA (to all zeros)
;
                      MOVLW    7                ;  //Load w with 7
                      MOVWF    _CMCON0          ;  //Load CMCON0 with 7
                                                ;  //Turns off comparators
;
                      BSF      _STATUS, 5       ;  //Select Bank 1
;
                      CLRF     _ANSEL           ;  //Shut off ADC (digital I/O)
;
                      MOVLW    001111B          ;  //Load w – RA4 and RA5 outputs
                      MOVWF    _TRISA           ;  //copy w to TRIS PORTA
;
                      BCF      _STATUS, 5       ;  //Select Bank 0
    #endasm

            return;
}
/******** END OF PORTA_init ***************************/
```

An Example … (*Delay*)

```
 void delay_routine(void)
{
   // WAS the following
              // int i, j;

              // for (i = 0; i < 255; i++)
               //              for (j = 0; j < 255; j++);

   // New code is the following ...

              asm("                          MOVLW    255")      ; //Decimal 255
              asm("                          MOVWF    _count")   ; //Initialize counter to 10
              asm("Repeat                    DECFSZ   _count,f") ; //Decrement counter
              asm("                          GOTO     Repeat")   ; //If counter <> 0

              return;

}
/******** END OF delay_routine ************************/
```

An Example … (*Final Thoughts*)

- Need to adjust the Delay Routine to have the LED flash on and off slower …
- Two files are on the Webpage …
  - Lecture_11A.c the original Flash_Do.c file
  - Lecture_11B.c the mixed mode file

Lab

Lab

- Finish Lab # 5 … the report is optional, however, you need to run through the lab

- Work on your Course Project

Next Class …

Next Class Topics

- Exam #2 ...
  - Covers material since the last exam … however …
    - Earlier material is the building blocks for follow-on material

HomeWork

Homework

1. Perform Lab #5 (optional)

2. Read the material from Today's class which is found in the …

    - PIC16F684 Data Sheet
    - PICmicro Mid-Range MCU Family Reference Manual
    - HI-TECH C Tools for the PIC10/12/16 MCU Family – Hi-Tech Software Manual, section 3.9

3. Prepare for Exam #2, next week … November 30th

Time To ...

Start the Lab ...

References

1. PIC16F684 Data Sheet 41202F

2. PICmicro Mid-Range MCU Family Reference Manual

3. HI-TECH C Tools for the PIC10/12/16 MCU Family – Hi-Tech Software Manual, section 3.9