2.20 Basic Analogue Input/Output and Synchronisation

Hardware and software aspects of input/output synchronisation

EE302 – Real time and embedded systems

1

Overview

- Aims
  - To introduce basic analogue input handling
- Learning outcomes – you should be able to…
  - Describe the purpose and operation of a comparator
  - Write code/pseudocode which acts on the current comparator state or when the state changes
  - Describe the purpose and operation of an ADC
  - Write code/pseudocode which acts on the current ADC value or some processed version of that value and prior values

2

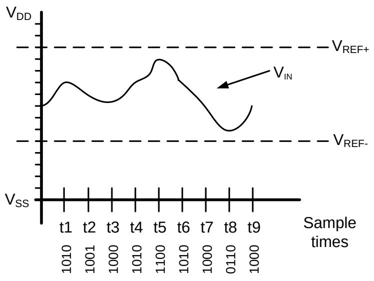
Analogue Input/Output

- Digital I/O
  - Included switches, buttons, keypads, LEDs, etc
- Analogue I/O typically applies to
  - Signals from sensors (temperature, pressure, acceleration, etc)
  - Signals to drive actuators (motors, etc.)
- For now we will focus on analogue input, specifically supported by the following peripherals 外围设备
  - Analog to Digital convertor
  - Comparator

ADC

ADC

- Analogue to Digital Convertor (ADC)
  - converts a voltage presented at the analogue input to a digital representation (a number)



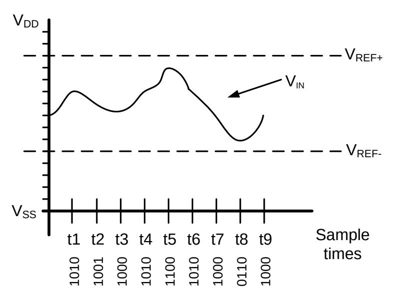
Analog to Digital Values
(Digital Inputs to CPU)

- Principle specifications
  - Analogue voltage range (min to max) that can be accepted
  - Number of bits, *b*, used for digital representation (implies 2^b voltage steps will be used to span the voltage range)
  - Acquisition time and conversion time will determine the maximum **sample rate**
  - Other parameters related to accuracy and linearity of the conversions

ADC signal conditioning for best resolution

- For maximum resolution we need the input signal to span the complete input voltage range of ADC
  - OPTION 1: Amplify and/or offset signal with external analogue h/w so that min and max approx. equal V_{SS}, V_{DD} respectively
  - OPTION 2: Provide external V_{REF-} and V_{REF+} levels which better match the input signal range (subject to device constraints)

Vref指参考电压



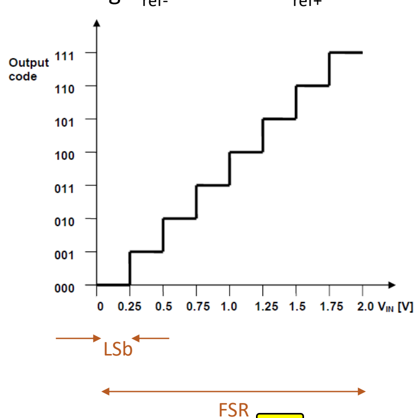
Analog to Digital Values
(Digital Inputs to CPU)

ADC concepts

- Valid voltage range and reference levels
  - ADC can represent voltages between V_{REF-} and V_{REF+} by a number. Anything outside this range is represented by the maximum or mininum valid number as appropriate.
  - If no external voltage references are supplied, V_{REF-} is usually V_{SS} and V_{REF+} is usually V_{DD}
  - The simplest (but not most robust) way to create external voltage reference levels is using a resistor divider circuit
- Resolution and step size
  - The ADC quantizes the valid voltage range into a number of equal size steps and assigns a number to each step
  - If the ADC has B bits of resolution then it will divide the voltage range in 2^B steps
  - Each step will cover $(V_{REF+} - V_{REF-}) / 2^B$ of the valid voltage range
- Relationship between digital value (number), d, and analogue voltage, V_{IN}
  - See next slide

ADC concepts (contd.)

Example 3-bit ADC transfer function assuming V_{ref-} is 0V and V_{ref+} is 2V



Full scale range (FSR)

$$FSR = V_{REF+} - V_{REF-}$$

Resolution = step size = LSb (least significant bit) size

$$LSb = FSR / 2^B$$

Digital code, d, given V_{IN}

$$d = \begin{cases} 0, & V_{IN} < V_{REF-} \\ 2^B - 1 & V_{IN} \geq V_{REF+} \\ \lfloor (V_{IN} - V_{REF-}) / LSb \rfloor & otherwise \end{cases}$$

Estimated V_{IN} given d

$$V_{IN} = (d \cdot LSb) + V_{REF-}$$

Example

9

Assume a 2-bit ADC and reference levels of 0 and 3V

Q. How many steps does the ADC resolve and what is the step size (LSb)?

Q. What number would the ADC use to represent each of the following voltages?

- *0V*
- *1.5V*
- *2.9V*

Q. Estimate the voltage represented by the ADC number 2.

03 November 2020

9

Self test questions

10

Q. If the two reference levels are 0 and 5V what number would each of the following voltages be represented by assuming a 10 bit ADC?

- *0V,*
- *5V,*
- *1.5V,*
- *-1V,*
- *6V*

03 November 2020

10

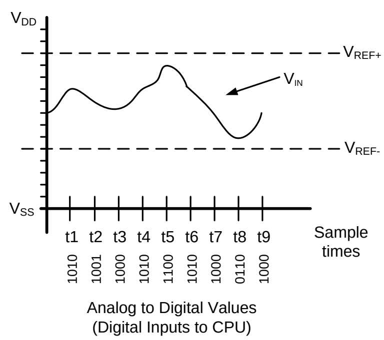
Self test questions

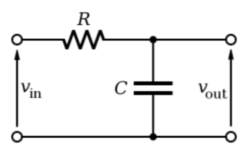
Q. If the two reference levels are 1V and 3V

- *what would be step size of an 8 bit ADC?*
- *What voltage would the number 64 represent?*

ADC signal conditioning part 2

- Sample rate (f_s or fs or Fs)
  - Number of times per second that can sample voltage and convert i number
  - Nyquist criterion: sample rate must be $2*f_{MAX}$ from signal
- To prevent aliasing
  - May need to filter signal before input to ADC to remove frequencies higher than 2 f_s
    - E.g. with a simple RC filter
- May also need to modify input impedance





$$f_c = \frac{1}{2\pi RC}$$

Self test questions

The ADC shall be used to monitor a signal whose highest frequency of interest is 200 Hz. However the signal contains significant energy at frequencies up to 10000 Hz.

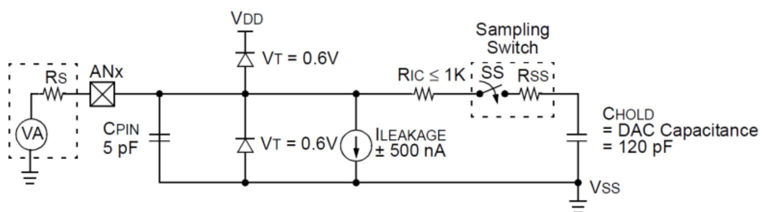
Q1. What sampling rate should we use?

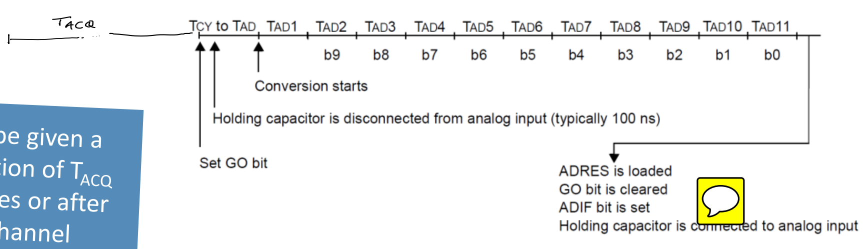
fmax is 200 Hz, so the minimum sampling frequency required is 2 * 200 Hz = 400 Hz.

Q2. Do we need an anti-aliasing filter, and if so what component values do we need?

that is less than or equal to 200 Hz to ensure it attenuates frequencies above that. A common choice is to set the cut-off frequency to around 90-95% of the Nyquist frequency

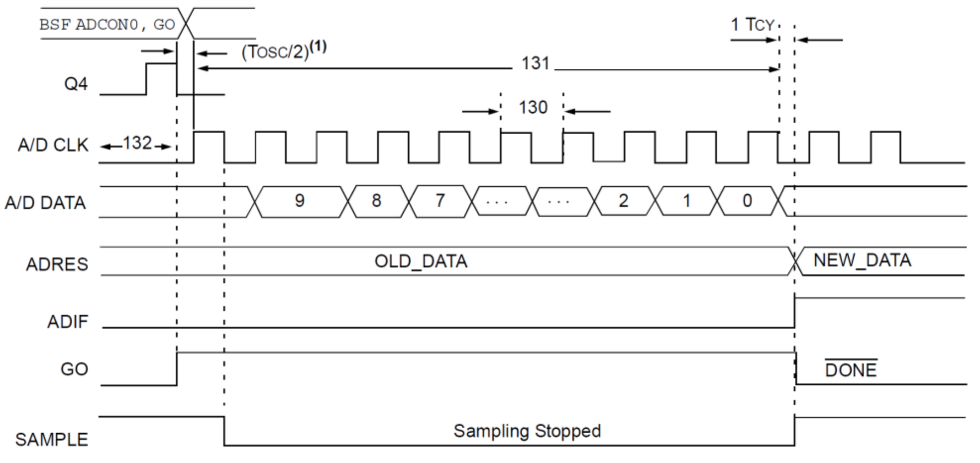
PIC ADC - sample acquisition and conversion





The ADC must be given a minimum duration of T_{ACQ} between samples or after changing ADC channel

Contd. (timing diagram)

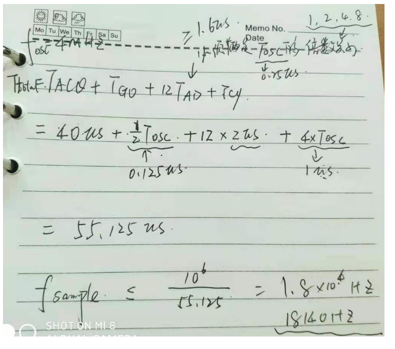


Self test questions

Assume

- T_{CY} = *instruction cycle time = 4 ** T_{OSC}
- *130 = TAD (must be >= 1.6 microsecs)*
- *131 = TCNV = 12 * TAD*
- *132 = TACQ = 10 microsecs abs min, min really about 20 microsecs. typ 40 microsecs*
- *134 = TGO: Q4 to A/D clock start (typically TOSC/2)*

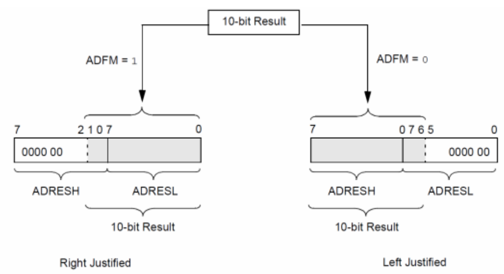
What is the maximum sample rate achievable if f_{OSC} *= 4 MHz?*



PIC ADC - 10 bit results vs. 8 or 16 bit data types

17

10 bit justification is chosen using ADFM configuration bit



Right justified best for all 10 bits
Left justified best for most significant 8 bits

```
// right justified result
int adcValue;

adcValue = (ADRESH << 8);
adcValue += ADRESL;
```

```
// left justified result (=value * 64)
int adcValue;

adcValue = (ADRESH << 8);
adcValue += ADRESL;
```

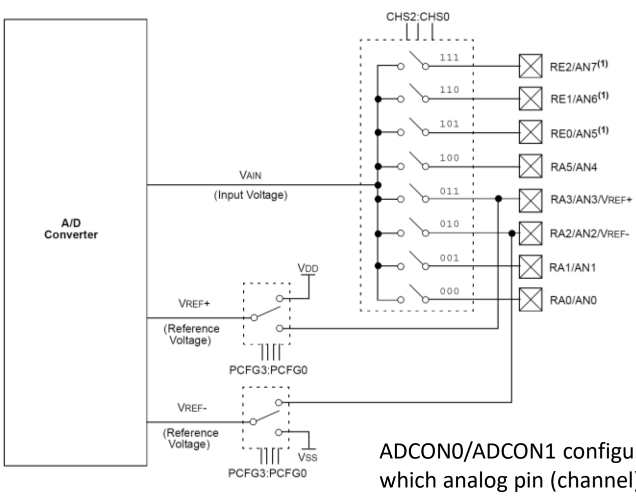
```
// left justified 8 bit result
// (ignore 2 LSb)
unsigned char adcValue;

adcValue = ADRESH;
```

03 November 2020

17

PIC ADC – multiplexing and channel selection

18



ADCON0/ADCON1 configuration bits determine which analog pin (channel) is connected to the ADC and whether internal (V_{DD}, V_{SS}) or external voltage references are used.

03 November 2020

18

ADC – general I/O pseudocode

19

```
// General app structure
loop:
  pollADC()
  …
  delay SUPERLOOP_TICK // usually equals ADC_SAMPLE_PERIOD

setup:
  configure analogue and vref inputs
  select ADC clock, ADC format
  select ADC channel and turn on ADC
  // if using multiple channels, select channel in the poll function
  // instead

// polling the ADC for a single sample
pollADC:
  select ADC channel if necessary
  delay for acquisition time if necessary
  start ADC conversion and busy-wait (poll) until done
  read ADC value (8 bit (1 register) or 10 bit (2 registers))
  …
  // process value further if desired (e.g. peak detect)
  // act on processing outcome as necessary
```

03 November 2020

19

ADC useful C code fragments

20

```
typedef unsigned int Uint16;
typedef unsigned char Uint8;
#define ADC_CHANNEL_MASK = 0b00111000

// NOTE: below are code fragments that you can use as needed -
// these are NOT all part of one function!
-----------------------------------------------------------------
// select ADC channel to be sampled next
  ADCON0 = (ADCON0 & ~ADC_CHANNEL_MASK) | (channel << 3);
-----------------------------------------------------------------
// sample the ADC and place the 10 bit result in a 16 bit variable
  Uint16 adcValue;
  ADGO = 1; // initiate I/O operation: start ADC conversion
  while (ADGO) continue; // "wait until done" polling
  adcValue = (ADRESH << 8) + ADRESL; // store the 10 bit sample
-----------------------------------------------------------------
// sample the ADC and place 8 bit result in an 8 bit variable
// NOTE: must configure ADC to use left justified results
  Uint8 adcValue;
  ADGO = 1; // initiate I/O operation: start ADC conversion
  while (ADGO) continue; // "wait until done" polling
  adcValue = ADRESH; // store the 8 bit sample
```

03 November 2020

20

[Extra] Basic signal processing – smoothing

21

```
// First order low pass filter
// (exponential averager) for smoothing

void processADC() {
  static Uint8 xn; // current adc sample
  static Uint8 yPrev = 0; // prev filter output
  Uint8 yn; // current filter output

  // sample and store current value…

  // filter/smooth it according to
  // yn = alpha*xn + (1-alpha)*y_prev
  //
  // assume we want a lot of smoothing
  // i.e. alpha < 0.5, so alpha = pow(2,-b)
  // Noting that pow(2,-b)*val == val >> b

  yn = (xn >> b) + yPrev – (yPrev >> b);

  // now use smoothed value yn as needed…
}
```

$$y_n = \alpha x_n + (1-\alpha) y_{n-1}$$

$$\tau = T_s \left(\frac{1-\alpha}{\alpha} \right)$$

$$f_c = \frac{1}{2\pi\tau}$$

$$\alpha = \frac{T_s}{T_s + \tau}$$

To use shifts and no multiplies, we constrain α as follows

$$\alpha = \begin{cases} 1-2^{-b}, & \alpha \geq 0.5 \\ 2^{-b}, & \alpha < 0.5 \end{cases}$$

03 November 2020

21

[Extra] Basic signal processing – peak detection

22

```
// Basic peak detection – looking for local minima and maxima
processSignal(x)
  static xPrev = 0 // initially no prev signal, so zero
  static wasIncreasing = TRUE // assume signal was increasing

  extremum = NONE // assume no peak unless one is found
  if wasIncreasing
    if x < xPrev // no longer increasing
      extremum = LOCAL_MAX // we've just passed a maximum
      wasIncreasing = FALSE
  else // wasIncreasing FALSE
    if x > xPrev // no longer decreasing
      extremum = LOCAL_MIN // just passed a minimum
      wasIncreasing = TRUE
  xPrev = x

  // now act on outcome, e.g. to do something when a maximum is
  // detected
  if extremum is LOCAL_MAX
    …
```

03 November 2020

22

Self test questions

Q1. Describe the purpose and operation of an ADC

Q2. How does ADC channel selection work on the PIC MCU and how does changing channel affect ADC timing?

Q3. Assume VDD=5V, VSS=0V, and the analogue input comes from a pressure sensor whose voltage ranges from 0 to 3V with a nominal "on" threshold at 0.5V

- *Show the ADC relevant connections to the PIC MCU for this signal*
- *Write the pseudocode required to light a LED for 3 superloops duration whenever the ADC output is above THRESHOLD for 5 samples in a row*

Self test questions – past exam

Consider a signal connected to the internal ADC of a PIC16 family microcontroller. The signal ranges from 0.2 to 1.6 V and the required signal resolution must be 2 mV or better.

(i) Briefly explain the relevance of the number of bits per sample and the voltage reference values for such an ADC.

(ii) Choose appropriate voltage reference levels for the input signal. Then, assuming V_{DD}=5V and V_{SS}=0V, sketch a circuit that could be used to provide these voltage reference levels specifying component values where possible.

(iii) Assuming the ADC is used in 10-bit mode and based on the reference levels chosen in (ii), specify the ADC signal resolution (i.e. the minimum voltage step size that can be resolved) and state whether this meets the signal resolution requirements given originally?

(iv) Determine the ADC value produced when the signal is 0.6 V.

(v) Determine the estimated signal voltage corresponding to an ADC value of 750.

27

03 November 2020

27

28

03 November 2020

28

Comparator

Comparator

- Comparator
  - Compares analogue input voltage (V_{IN+}) to some reference level (V_{IN-})
  - Logic 1 means V_{IN+} above V_{IN-}, 0 means below
  - What about noise and what about signal levels "near" the reference level?

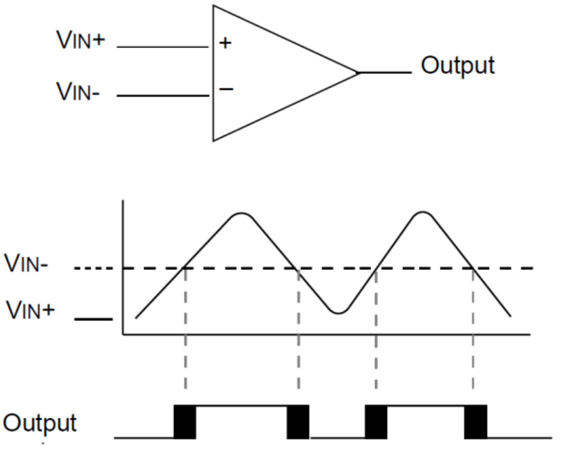
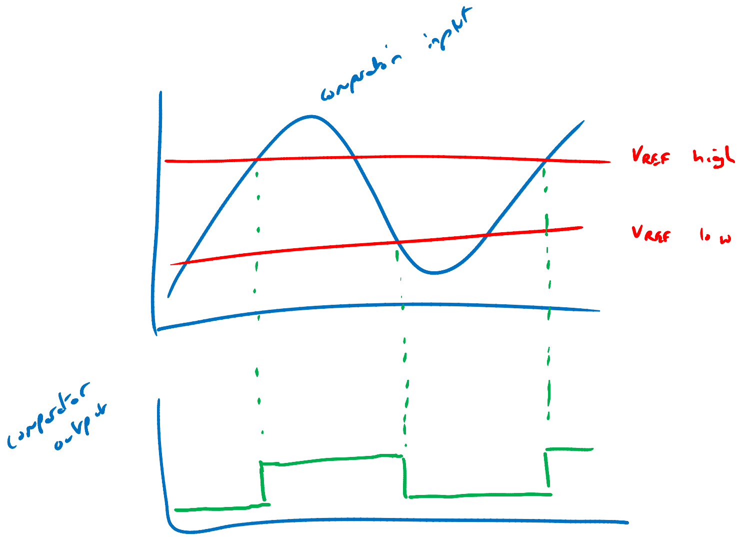


Figure reproduced from PIC16F87x data sheet

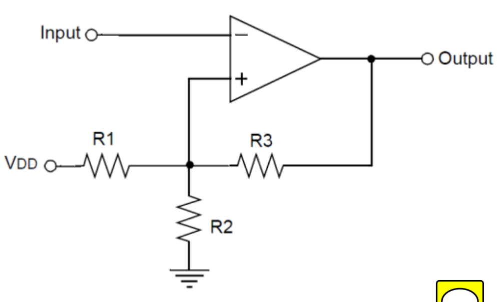
Comparator with 2 thresholds (hysteresis)

31



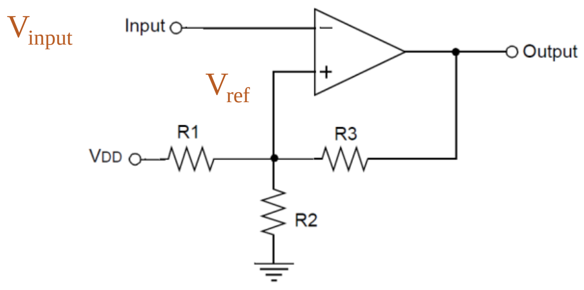
Comparator with hardware hysteresis

33



To prevent oscillation near reference level, basic solution is to implement hysteresis using 2 thresholds: low and high.

Comparator with hardware hysteresis

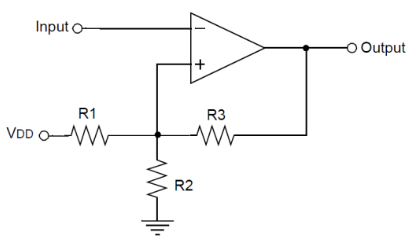


To prevent oscillation near reference level, basic solution is to implement hysteresis using 2 thresholds: low and high.

$V_{input} < V_{ref}$

$V_{input} >= V_{ref}$

Comparator hardware hysteresis contd.



Resistor values for hardware hysteresis can be chose according to the following equations

$$\frac{R3}{R1} = \frac{V_{low}}{V_{high} - V_{low}} \qquad (1)$$

$$\frac{R2}{R1} = \frac{V_{low}}{V_{DD} - V_{high}} \qquad (2)$$

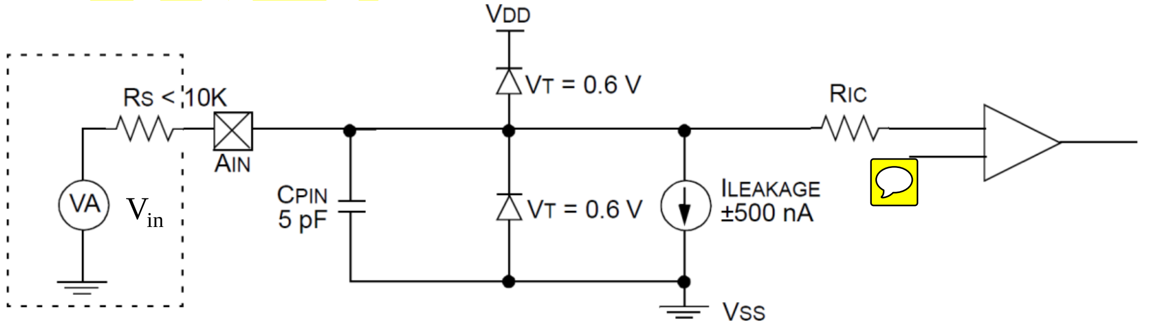
The result is that R2 and R3 can both be expressed as a multiple of R1. Therefore choose a value for R1 to minimize R1 and the other 2 can then be derived.

EXAMPLE: V_{DD}=5V, desired high threshold V_{high} = 2.7V and low threshold V_{low} = 2.3V. Sub into equation 1 and 2 yields R3 = 5.75 * R1 and R2 = 1 * R1. If we choose R1 = 100 kΩ then R2 = 100 kΩ and R3 = 575 kΩ .

PIC MCU comparator specifics

38

- Analogue interfacing issues
  - Voltage latch up (a short circuit through diodes which can destroy the hardware)
    - if Vin < V_{SS} – 0.6V or Vin > V_{DD} + 0.6V then current flows out through A_{IN}
  - Input impedance constraints



03 November 2020

38

Contd.

39

- There are 2 comparator devices integrated in the PIC16xxx
  - A control bit specifies which one to read
  - Various ways of configuring analogue inputs and the comparator reference (threshold) voltage
- Timing/synchronisation issues
  - There is a delay (around 10 microsecs) before the output is valid after switching between comparators
  - There is a delay before the reference level is stable when switching **internal** reference levels (e.g. for software based hysteresis)

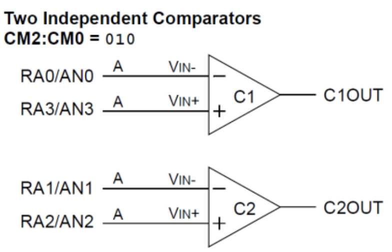
03 November 2020

39

PIC Comparator pseudocode

```
// General app structure
Main
  setup
    …

  loop forever
    pollComparator()
    …
    delay SUPERLOOP_TICK

setup()
  // details depend on use of hardware or software hysteresis

// Polling a single comparator is like polling a digital port, e.g.
pollComparator()
  // details depend on use of hardware or software hysteresis
  // and whether we need to react on change, react continuously
  // while above/below threshold etc.
  …
```


Comparator with hardware threshold (and hysteresis)



When using hardware thresholds we can choose which pin is the input and which pin is the reference (i.e. threshold). External circuitry (as shown on previous slides) would implement the hysteresis. Let's assume the same circuit as slide 6 so that the input is V_{IN-} and the reference is V_{IN+}.

C1OUT is high when $V_{IN-} < V_{IN+}$ at C1 and low otherwise.

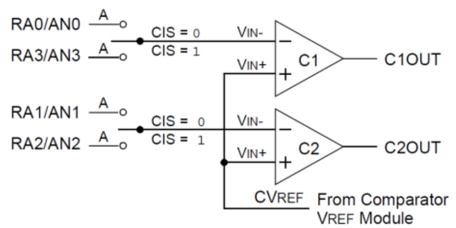
Likewise C2OUT is high when $V_{IN-} < V_{IN+}$ at C2 and low otherwise.

Pseudocode with hardware hysteresis

```
setup()
  configure port tristate
  configure comparator mode and external voltage reference
  choose initial comparator and multiplexed input (if needed)
  configure comparator output to appear on output pin (for voltage ref)

// No need to change comparator mode and reference
// And polling the comparator is like polling a digital port
pollComparator() // Version 1: act every loop when signal > threshold



// Version 2: act only when signal crosses threshold from low to high
pollComparator()
```


Comparator with software controlled threshold (and hysteresis)

Four Inputs Multiplexed to Two Comparators
CM2:CM0 = 110



When using software, we control the reference level (threshold) by setting registers which cause the voltage of CV_{REF} to change. In this case, the external input must be V_{IN-} and the reference is always V_{IN+}. To implement hysteresis software must set values which cause CV_{REF} in response to the current output (C1OUT or C2OUT).

NOTE: C1OUT is high when $V_{IN-} < CV_{REF}$ at C1 and low otherwise.

Likewise C2OUT is high when $V_{IN-} < CV_{REF}$ at C2 and low otherwise.

Pseudocode with software hysteresis

45

```
setup()
  configure port tristate
  configure comparator mode and internal voltage reference
  set initial voltage reference level (high or low as needed)
  choose initial comparator and multiplexed input (if needed)

…
```

03 November 2020

45

Pseudocode with software hysteresis

46

```
pollComparator()
  static prevComparatorOut = LOW // or HIGH depending on needs
  Boolean comparatorChanged = FALSE

  // only change voltage reference if the comparator output changes
  if comparatorOut differs from prevComparatorOut
    set comparatorChanged = TRUE
    prevComparatorOutput = comparatorOutput

    // change the threshold according to the just changed
    // comparator output high or low
    if comparatorOut is HIGH
      set voltage reference level to COMPARATOR_THRESHOLD_HIGH
    else // comparatorOut is LOW
      set voltage reference level to COMPARATOR_THRESHOLD_LOW

    // if there is no delay in the superloop, then we need to add a
    // delay here to allow comparator to stabilize after changing
    // the internal voltage reference level

  // remaining normal comparator code goes here, e.g
  if comparatorChanged          OR          if comparatorOut is HIGH
    …                                         …
```

03 November 2020

46

Self test questions

Describe the purpose and operation of a comparator

- Using hardware hysteresis
- Using "software" hysteresis

Assume VDD=5V, VSS=0V, and the analogue input comes from a pressure sensor whose voltage ranges from 0 to 3V with a nominal "on" threshold at 0.5V

- Show the connection of this signal to the PIC MCU including any external hysteresis required. Guesstimate any resistor values you can.
- Write the pseudocode required to light a LED for 3 superloops duration whenever the comparator output indicates a transition from off to on