The Analog to Digital Converter (ADC)

Corrado Santoro

ARSLAB - Autonomous and Robotic Systems Laboratory
Dipartimento di Matematica e Informatica - Università di Catania, Italy
santoro@dmi.unict.it

L.A.P. 1 Course

What is an ADC?

An ADC (Analog-to-Digital-Converter) is a circuit which gets an **analog voltage signal** and provides (to software) a **variable** proportional to the input signal.

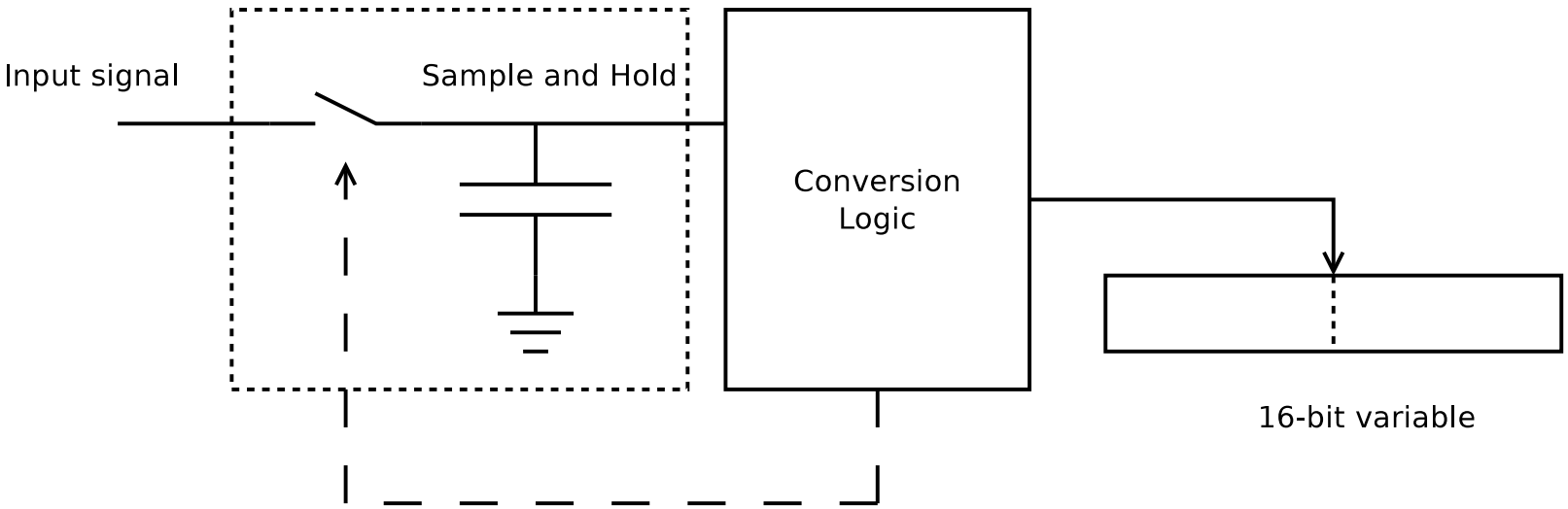
An ADC is characterised by:

- The **range** (in Volts) of the input signal (typical $[0, 5V]$ or $[0, 3.3V]$).
- The **resolution** (in **bits**) of the converter.

Example:

- **Range** = $[0, 5V]$
- **Resolution** = 12 bits
  results in range $[0, 2^{12} - 1] = [0, 4095]$
- $0V \rightarrow 0$ $2.5V \rightarrow 2048$ $5V \rightarrow 4095$

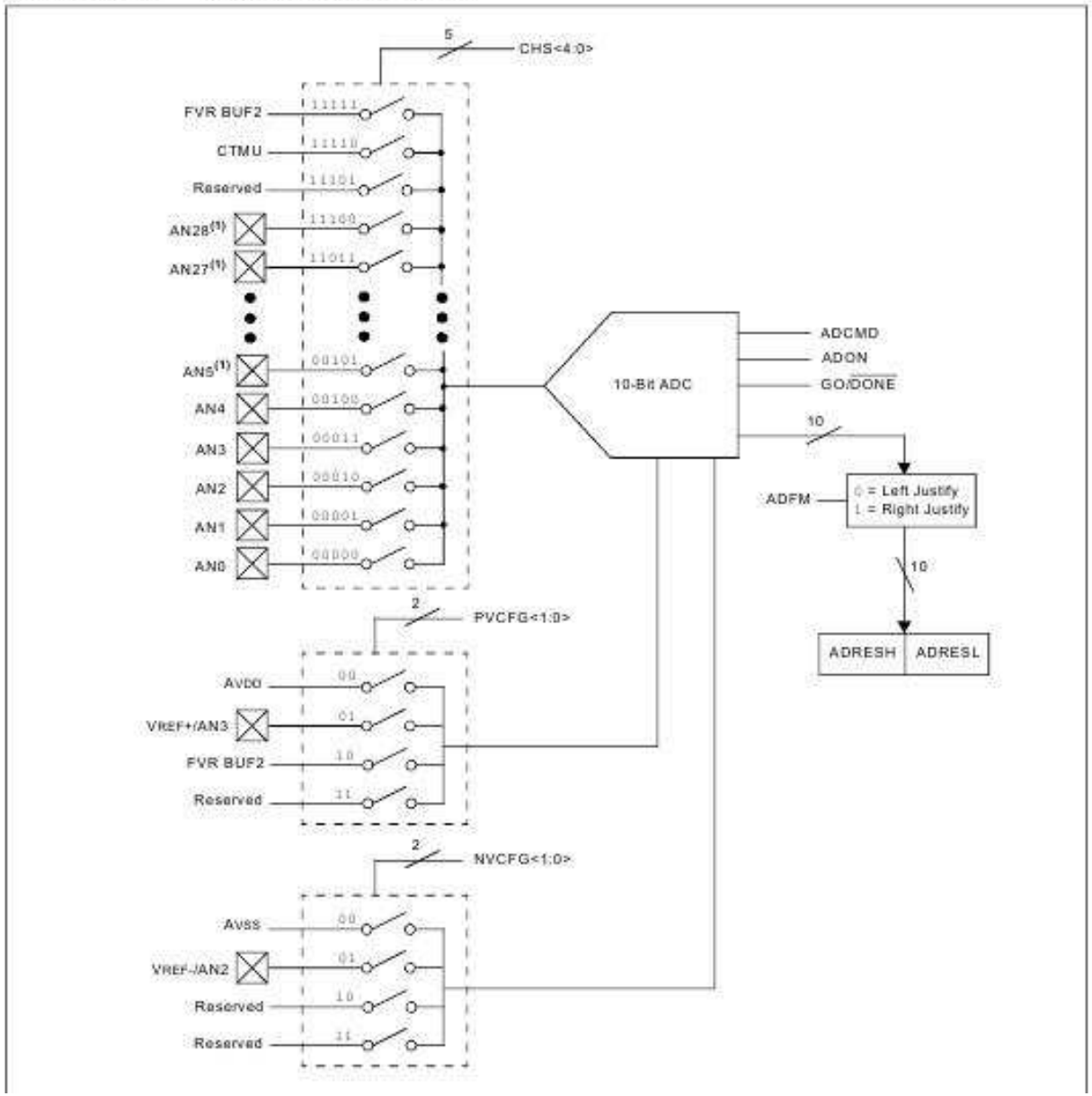
ADC: Basic working scheme



1. **Sample**: the signal is *sampled* by closing the switch and charging the capacitor.
2. **Conversion**: the switch is open and the sampled signal is *converted*; the result is stored in the 16-bit variable.
3. **End-of-conversion**: a proper bit is set to signal that the operation has been done.

The ADC Circuit of PIC18F25K22

FIGURE 17-1: ADC BLOCK DIAGRAM



ADC Registers

- Inputs are shared with pins of PORTA, PORTB, ... Registers `ANSELA`, `ANSELB`, `...` configure each **input** signal as **digital** or **analog**.
- 3 registers for configuration of the ADC circuit: `ADCON0`, `ADCON1`, `ADCON2`
- 2 registers for result: `ADRESH`, `ADRESL` (virtualized in the compiler as a unique 16-bit register named `ADRES`)
- Interrupt handling performed using bits:
  - `PIR1bits.ADIF`, end-of-conversion interrupt flag
  - `PIE1bits.ADIE`, end-of-conversion interrupt enable
  - `IPR1bits.ADIP`, end-of-conversion interrupt priority

ADC Configuration: Input Pins

1. First decide digital and analog input pins, by configuring register **ANSELx**

PIN	Digital Function	Analog Function
2	RA0	AN0
3	RA1	AN1
4	RA2	AN2
5	RA3	AN3
6	RA5	AN4
21	RB0	AN12
22	RB1	AN10
...	...	...

```
ANSELAbits.ANSA0 = 1; // RA0 as analog input --> AN0

ANSELAbits.ANSA0 = 0; // RA0 as digital input

ANSELBbits.ANSB1 = 1; // RB1 as analog input --> AN10

ANSELBbits.ANSB1 = 0; // RB1 as digital input
```

ADC Configuration: References

2. Configure the **positive** and **negative** reference for ADC inputs

REGISTER 17-2: ADCON1: A/D CONTROL REGISTER 1

R/W-0	U-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0
TRIGSEL	—	—	—	PVCFG<1:0>		NVCFG<1:0>	
bit 7							bit 0

NVCFG and **PVCFG** configure the **reference voltage range** (resp. negative and positive) for input signal.

bit 3-2 **PVCFG<1:0>:** Positive Voltage Reference Configuration bits
00 = A/D V_{REF+} connected to internal signal, AV_{DD}
01 = A/D V_{REF+} connected to external pin, V_{REF+}
10 = A/D V_{REF+} connected to internal signal, FVR BUF2
11 = Reserved (by default, A/D V_{REF+} connected to internal signal, AV_{DD})

bit 1-0 **NVCFG0<1:0>:** Negative Voltage Reference Configuration bits
00 = A/D V_{REF-} connected to internal signal, AV_{SS}
01 = A/D V_{REF-} connected to external pin, V_{REF-}
10 = Reserved (by default, A/D V_{REF+} connected to internal signal, AV_{SS})
11 = Reserved (by default, A/D V_{REF+} connected to internal signal, AV_{SS})

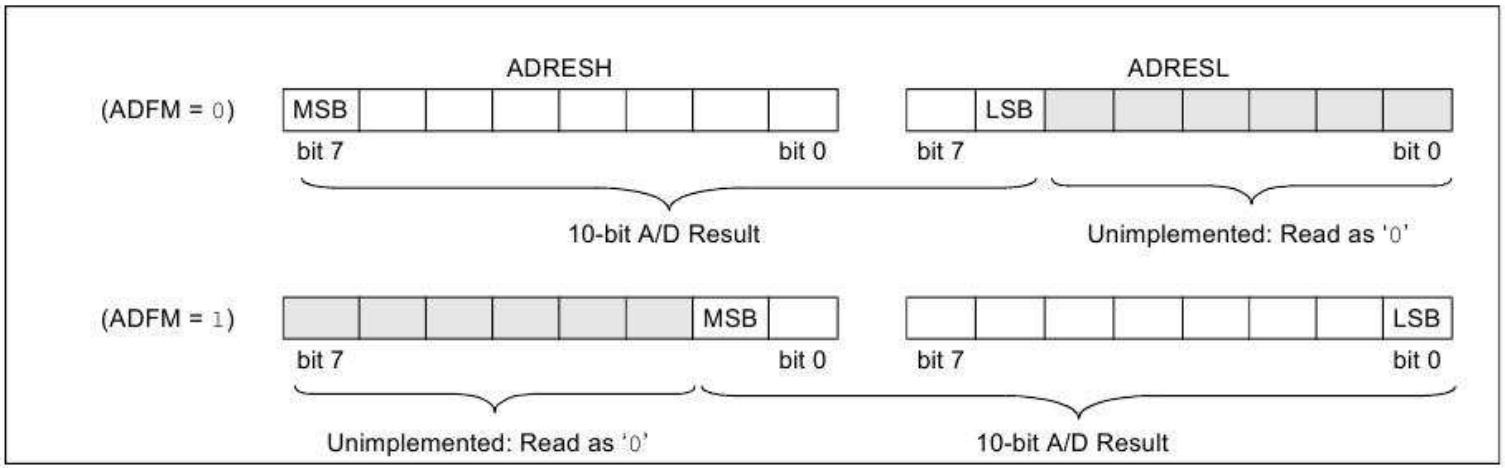
ADC Configuration: Data Format

3. Configure the format of resulting data (ADFM bit)

REGISTER 17-3: ADCON2: A/D CONTROL REGISTER 2

R/W-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
ADFM	—	ACQT<2:0>			ADCS<2:0>		
bit 7							bit 0

FIGURE 17-2: 10-BIT A/D CONVERSION RESULT FORMAT

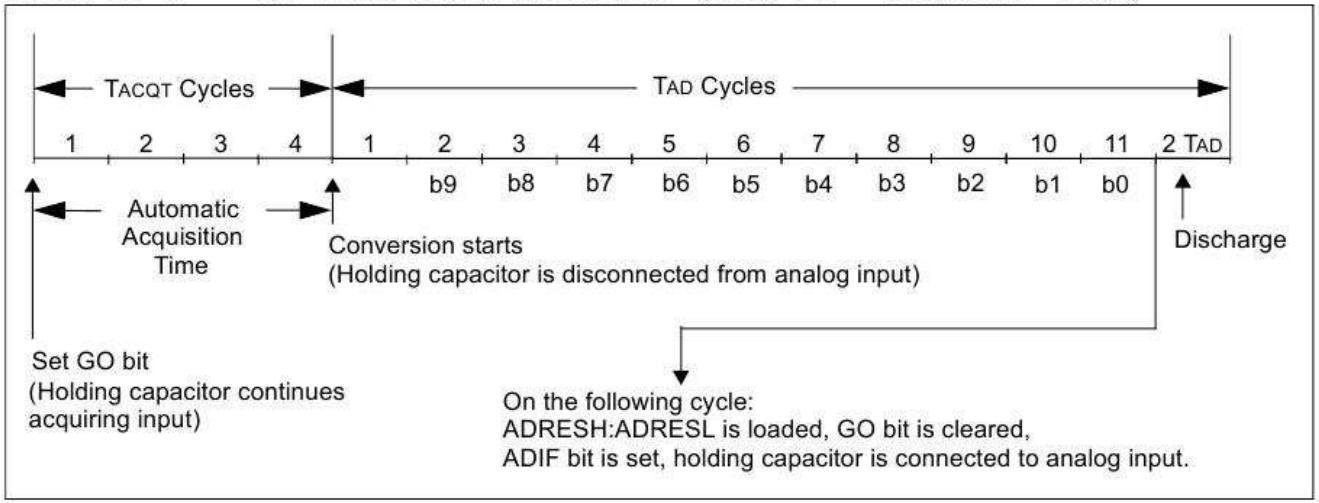


- **ADFM = 0;** is used for **fixed-point math**
- **ADFM = 1;** is used for **integer math**

ADC Configuration: Timing

The ADC is a **sequential circuit** so it has a **clock source**.

FIGURE 17-4: A/D CONVERSION TAD CYCLES (ACQT<2:0> = 010, TACQ = 4 TAD)



ADC Conversion phases

1. **Sampling phase:** T_{ACQT} ADC clock cycles (to be configured)
2. **Conversion phase:** T_{AD} ADC clock cycles (fixed to 12)

4. Configure the ADC clock source

REGISTER 17-3: ADCON2: A/D CONTROL REGISTER 2

R/W-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
ADFM	—	ACQT<2:0>			ADCS<2:0>		
bit 7							bit 0

bit 2-0 **ADCS<2:0>:** A/D Conversion Clock Select bits

000 = Fosc/2
001 = Fosc/8
010 = Fosc/32
011 = FRC(1) (clock derived from a dedicated internal oscillator = 600 kHz nominal)
100 = Fosc/4
101 = Fosc/16
110 = Fosc/64
111 = FRC(1) (clock derived from a dedicated internal oscillator = 600 kHz nominal)

ADC Configuration: Sample Timing

5. Configure the sample time

REGISTER 17-3: ADCON2: A/D CONTROL REGISTER 2

R/W-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
ADFM	—	ACQT<2:0>			ADCS<2:0>		
bit 7							bit 0

bit 5-3 **ACQT<2:0>:** A/D Acquisition time select bits. Acquisition time is the duration that the A/D charge holding capacitor remains connected to A/D channel from the instant the GO/$\overline{\text{DONE}}$ bit is set until conversions begins.

000 = 0(1)
001 = 2 TAD
010 = 4 TAD
011 = 6 TAD
100 = 8 TAD
101 = 12 TAD
110 = 16 TAD
111 = 20 TAD

ADC Configuration: Turn on ADC

6. Finally turn on the ADC

REGISTER 17-1: ADCON0: A/D CONTROL REGISTER 0

U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	CHS<4:0>					GO/$\overline{\text{DONE}}$	ADON
bit 7							bit 0

```
ANCON0bits.ADON = 1;
```

Complete ADC Configuration

```
void adc_setup(void)
{
    ANSELAbits.ANSA0 = 1; // RA0 = analog input --> AN0

    ADCON1bits.PVCFG = 0b00; // Positive reference = +VDD
    ADCON1bits.NVCFG = 0b00; // Negative reference = GND

    ADCON2bits.ADFM = 1; // format = right justified
    ADCON2bits.ACQT = 0b111; // acquisition time = 20*TAD
    ADCON2bits.ADCS = 0b110; // conversion clock = FOSC/64

    ADCON0bits.ADON = 1; // turn on ADC
}
```

Sampling a signal using the ADC

REGISTER 17-1: ADCON0: A/D CONTROL REGISTER 0

U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	CHS<4:0>					GO/$\overline{\text{DONE}}$	ADON
bit 7							bit 0

1. Select the **channel** to be sampled (**CHS**)
2. Set the **GODONE** bit to start sampling
3. When sampling is completed:
   - The **GODONE** bit goes to 0
   - An interrupt is generated (if enabled)
4. Result is stored in **ADRESH:ADRESL** (or **ADRES**)

Exercise: A LED flashing with variable frequency

We want to flash a LED using a period ranging from 5 *ms* to 500 *ms*, according to value of ADC.

Timer 0 setup:

- System clock, $FOSC = 64MHz$, therefore the basic frequency is $FOSC/4 = 16MHz$, the $P = 62.5ns$;
- Prescaler with division by 256, so the timer increments using a period $P = 62.5ns * 256 = 16\mu s$.
- 5 $ms/16\ \mu s = 312$ *counts*
- 500 $ms/16\ \mu s = 31250$ *counts*
- No interrupts

Timer Setup

```
#define PERIOD_LOW   312
#define PERIOD_HIGH  31250

void timer_setup(void)
{
  T0CONbits.TMR0ON = 0; // stop the timer
  T0CONbits.T08BIT = 0; // timer configured as 16-bit
  T0CONbits.T0CS = 0; // use FOSC
  T0CONbits.PSA = 0; // use prescaler
  T0CONbits.T0PS = 0b111; // prescaler 1:256

  INTCONbits.T0IF = 0; // reset timer interrupt flag
  INTCONbits.T0IE = 0; // no interrupts timer interrupts

  T0CONbits.TMR0ON = 1; // start the timer
}
```

Exercise: Main Program Loop

1. Start ADC conversion
2. Wait for timer overflow
3. Toggle the LED
4. **Check if the ADC conversion is completed**
5. Get the ADC result (in range $[0, 1023]$) and scale it to ($[312, 31250]$)
6. Set the new period to TMR0
7. Clear timer overflow flag
8. Restart

Exercise: Main Program

```
int main(void)
{
    int period = PERIOD_HIGH;

    TRISBbits.TRISB0 = 0; // output

    timer_setup();
    TMR0 = -period;
    adc_setup();

    ADCON0bits.CHS = 0; // select channel 0 (AN0)
    ADCON0bits.GODONE = 1;

    for (;;) {
        while (INTCONbits.TMR0IF != 1) ;

        LATBbits.LATB0 = !LATBbits.LATB0;
        INTCONbits.TMR0IF = 0;

        if (ADCON0bits.GODONE == 0) {
            // conversion completed
            period = PERIOD_LOW + ADRES * ((PERIOD_HIGH - PERIOD_LOW) / 1024);
            // retrigger conversion
            ADCON0bits.GODONE = 1;
        }

        TMR0 = -period;
    }
    return 0;
}
```

The Analog to Digital Converter (ADC)

Corrado Santoro

ARSLAB - Autonomous and Robotic Systems Laboratory
Dipartimento di Matematica e Informatica - Università di Catania, Italy
santoro@dmi.unict.it

L.A.P. 1 Course