LIÈGE université

Embedded systems

Exercise session 1

Introduction to Microcontrollers

Assembly language for Microcontrollers

In these slides

Microcontroller programming

- Data memory
- Special function registers
- Software resets
- MPASM and assembly language
- Programming a MCU in practice

Microcontroller embedded modules configuration

- I/O ports and clock source
- Timers
- Analog to digital converter
- PWM signal generator
- Capture/compare module
- UART

What you will need

These slides are designed to work with two external resources:

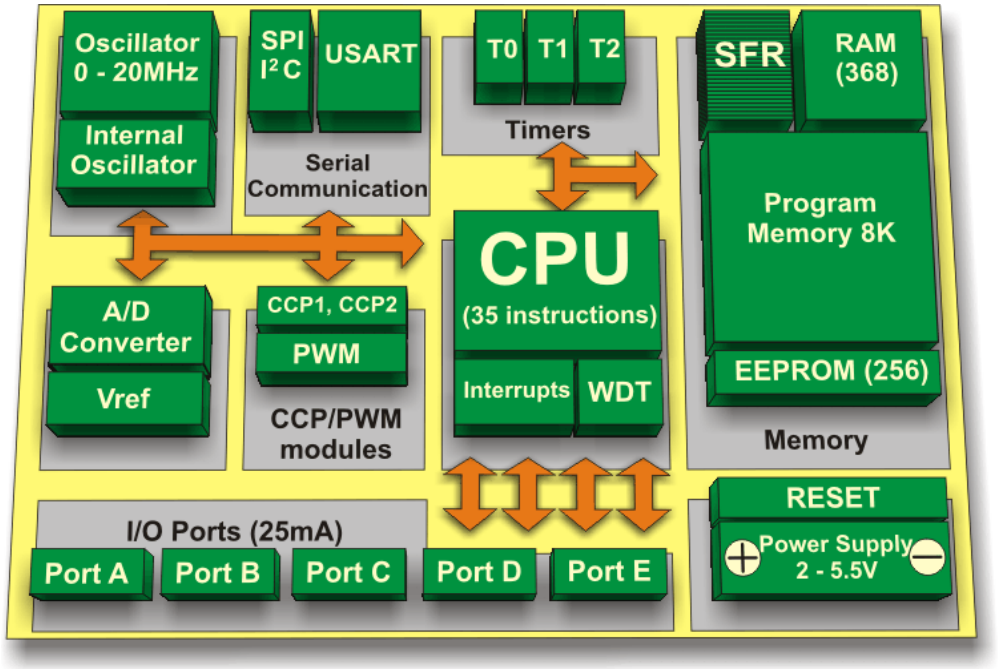
- The datasheet of the microcontroler we will use for the labs : http://ww1.microchip.com/downloads/en/DeviceDoc/40001675C.pdf
- A WooClap questionnaire : https://www.wooclap.com/MTLHOZ

Taking part to the questionnaire is not mandatory but is strongly advised. It will give you an insight on your level of understanding for the labs and the project. You will face difficulties to complete both of them if you have hard time to answer this questionnaire.

Don't hesitate to contact me by email or ask me questions during the Q&A sessions if you face some difficulties.

Microcontrollers : First insight

Microcontrollers (MCU) are small on-chip computers with a series of input and output pins. They can be programmed to perform various tasks and are generally used for applications that require real-time processing.



Microcontroller example : PIC16F1789

Microchip PIC16F1789 Core Features

- 8-bit words, 16-bit instructions
- Only 49 instructions set
- Up to 8 MIPS (Millions of Instruction Per Second)
- Free programming tool
- Program memory : 16ko
- Data EEPROM : 256 Bytes
- RAM : 2048 Bytes
- Clock frequency up to 32MHz
- 36 general purpose Input/Output pins

Note

- The program memory holds the binary data to be executed by the microcontroller.
- The EEPROM can be used to write and store data between different power cycles.
- The RAM can be used to write and store data during a single power cycle.

Microcontroller Peripheral Modules I

Additionally to the core computing unit and its digital IO pins, microcontrollers generally embed several independent peripheral modules that ease the interfacing of the MCU with the external world.

Example of Analog Peripherals

- Analog to Digital Converters (ADC)
  Used to convert an analog voltage into binary data
- Digital to Analog Converters (DAC)
  Used to generate an analog voltage from digital data
- Comparator Modules
  Used to interface analog circuits by comparing two analog voltages and providing a digital indication of their relative magnitude.
- Operational Amplifiers
  Used to process (e.g. amplify and or filter) analog signals.

Microcontroller Peripheral Modules II

Additionally to the core computing unit and its digital IO pins, microcontrollers generally embed several independent peripheral modules that ease the interfacing of the MCU with the external world.

Example of Digital Peripherals

- Timers

  Used to precisely measure elapsed time.
- Pulse Width Modulation (PWM) Signal Generators

  PWM signals are periodic binary signals which stay high during a given proportion of their period and are low everywhere else.
- Digital busses and communication lines interfaces (e.g. I2C, EUSART, ...)

  Used to transfer binary data between two digital devices according to some given protocols.

General considerations

Microcontrollers are computers. In order **to interact** with the outer world, **they need both inputs (**e.g. sensors**) and outputs (**e.g. actuators**)**.

- Inputs capture information from the world (e.g. microphones, cameras, ...)
- Outputs interact with the world (e.g. motors, pumps, LEDs, ...)

Analog vs digital

Microcontrollers are digital devices. Communication with sensors or actuators can however be analog or digital:

- Devices transmit or receivie analog information thanks to a current or voltage modulation => need to translate this information from or to binary data
- ➔ This can be done with ADCs and DACs.
- The microcontroller communicates with other digital devices via busses or communication lines (UART, SPI, I^2C,...)

How to make things work together?

In short, microcontrollers consist in a set of peripheral modules organized around a computing unit that can be programmed at will. Enabeling and configuring the modules for a specific task, as well as managing interactions between these modules and the computing unit is entirely done through software.

The MCU is configured by writing specific values in its memory.

- Some specific configuration bits are written in the flash memory with the user's software when programming the MCU.
- Other configuration bits can be set on the fly by the user's software by writing the desired configuration in a special function register.

All the information required to use and program a MCU can be found in a single document called the datasheet (datasheet for the PIC16F1789).

Data Memory

- Used for temporarily storing and keeping intermediate results and variables.
- Is reset each time the microcontroller looses power.
- Is divided is several banks.

➔ Each bank contains a set of addressable registers

Accessing a given register is performed in two instructions:

- First selecting the bank in which this slot is located,
- Then addressing this slot by its position in the bank.

Note: This system allows 8-bit devices, which can only address 256 different registers by default, to feature large data memories.

Warning: Selecting the wrong bank will lead to accessing an undesired register and lead to potential random behaviours.

Data Memory Organization

Each register has its own 12-bit address in the PIC16f1789:

- Fist five bits of the address define the Bank address
- Last seven bits select the registers in that bank.

The PIC16F1789 data memory is partitioned in 32 memory banks of 128 bytes each:

- 12 core registers
- 20 Special Function Registers (SFR)
- Up to 80 bytes of General Purpose RAM (GPR)
- 16 bytes of common RAM (each bank contains the same data for those bytes)

The active bank is selected by writing the bank number into the Bank Select Register (BSR). Unimplemented memory will read as '0'.

Note: For all information about the PIC16F1789 data memory, refer to section 3.0 of the datasheet

Core Registers

Registers essential for operating a MCU

- Duplicated in every bank
- Store information on the state of the program execution

Important core registers:

- BSR : Bank select register
- STATUS : Arithmetic status of the ALU and reset status
- WREG : Store one operand for operation requiring two operands and/or results for some operations
- PCL and PCLATH : program counter value
- INTCON : interrupt configuration

Note: Core registers are described in section 3.2.1 of the datasheet.

Special Function Registers

Specific slots of the data memory which hold the state of the device and its configuration.

- All peripherals embedded in the microcontroller are configured via the SFRs.

Note

As all other registers, the special function registers can be modified during the program execution by overwriting them.

Use

- Program execution
- IO ports configuration
- Internal peripherals configuration (interrupts, timers, A/D converters, PWM,...)
- Status flags

Watchdog Timer and Master Clear

Aim

Used to reset the microcontroller in its initial state.

Watchdog Timer

- Timer that is incremented at a given rate (based on clock transitions) and that resets the microcontroller when it reaches the overflow
- Used to cure potential dead- or livelocks during tasks execution

Master Clear

- Pin that resets the microcontroller when its input goes to 1
- Used to manually reset the microcontroller

Warning: If those functionalities are misused, they can lead to some unpredictable program behaviour due to untimely resets.

Note: Resets are described in section 5.0 of the datasheet.

PIC Programming | Instruction set

Assembler reminder

- Series of instructions (==operations) that can be directly performed by the processing unit;
- Ultra low-level programming language
- Limited set of available instructions (varies from one computing unit to another)

PIC16F1789 (see section 30 of datasheet)

- 49 instructions allowing operations
    - At bit level
    - At byte level
    - On the program counter
- Limited set ⇒ complex operations (e.g. muliplications, divisions, signed arithmetics, ...) must be implemented from these instructions

PIC Programming | Tricks for complex operations

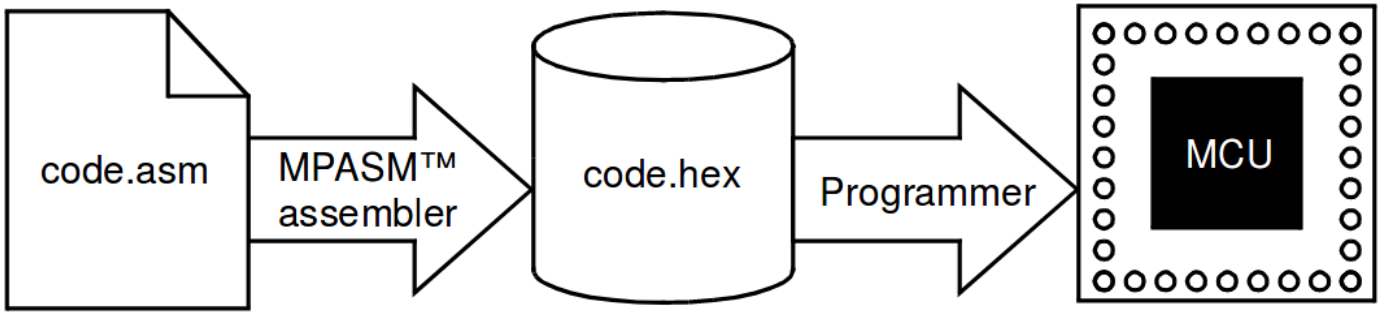
Binary shifts

- Shifting a byte to the left == multiplication by 2
- Shifting a byte to the right == division by 2
- → Multiplications (resp. divisions) can be implemented by a succession of shifts, additions (resp. subtractions), and compares

Lookup tables

- Replace operations too complex to be implemented (e.g. trigonometric functions)
- Store precomputed results of operations in memory
  - → Trade-off between computational complexity and memory use
  - → Need for a mapping between rounded function arguments and corresponding result location (memory address)

PIC Programming | Basic idea



MPASM

- Assembly language for Microchip's PIC1x 8-bit microcontrollers
- Provides several useful macros to have readable code
- Provides high-level directives to ease programming

➔ Need to be assembled before being sent to the microcontroller

Assembling assembly == replacing macros and directives accordingly, replace human-readable labels with memory addresses,...

PIC Programming | Example

Program structure

```
; header files
processor 16f1789

; configuration bits settings
…

; Set main code entry point
org     0x00
nop
goto    code_start

; declare interrupt vector
org     0x04
nop
goto    interrupt_vector
```

Loads MPASM operations and instruction set for the corresponding microcontroller

Next instruction will be written at this address of the program memory

Make program counter jump to this label (will be replaced by a static address by compiler)

PIC Programming | MPASM code example

```
processor 16f1789

Variable factor1=23
Variable factor2=6
Variable result=0
Variable counter=8

Multiplication:
    BTFSS factor2,0
            GOTO Skip
        MOVF factor1,0
        ADDW result,1

Skip:
    LSLF factor1,1
    LSRF factor2,1
    DECFZ counter,1
            GOTO Multiplication
```

8-bit multiplication

- Declare and initialize variables
- Perform binary written calculation for multiplication

Warning

- This code does not handle overflows!

Go to the WooClap and try to answer questions 1 and 2!
(precise description of each instruction given in section 30.2 of the datasheet)

PIC Programming | Example

Configuration bits (can't be modified during execution)

→For the different configuration bits which can be parametrized for each PIC16F, look at the datasheet or at the file "xc8/1.3x/docs/pic16_chipinfo.html"

→MPLABX IDE: Window >> PIC memory views >> configuration bits, set everything, then generate a config file.

Example

```
CONFIG  OSC = INTIO67          ; Internal oscillator block
CONFIG  MCLRE = ON             ; Active master clear
CONFIG  BOREN = ON            ; Enable Brown-out Reset
CONFIG  LVP = OFF              ; Low voltage icps off
CONFIG  WDT = OFF              ; Watchdog off
```

PIC Programming | IDE

Idea

The IDE is a software which helps you to write, compile and debug a program

MPLABX

- IDE for microcontrollers from Microchip.
- Available for Windows, Linux, OSx
- Supports both MPASM (assembler) and embedded C
    - For 8bits microcontrollers (PIC16Fxxxx) : XC8 compiler
    - The XC8 compiler comes with a complete documentation in its installation folder (xc8/1.3x/docs for example)

MPLABX

- Lots of documentation on the MPLAB environment available online
- Links to the IDE on the webpage of the exercise sessions

PIC Programming | Programmer

Idea

The compiled program needs to be uploaded in the microcontroller. This is done by using a programmer

Microchip programmers

Pickit 3

- "Inexpensive" (~40$)
- Low programming speed
- Available in the toolbox

Pickit 3

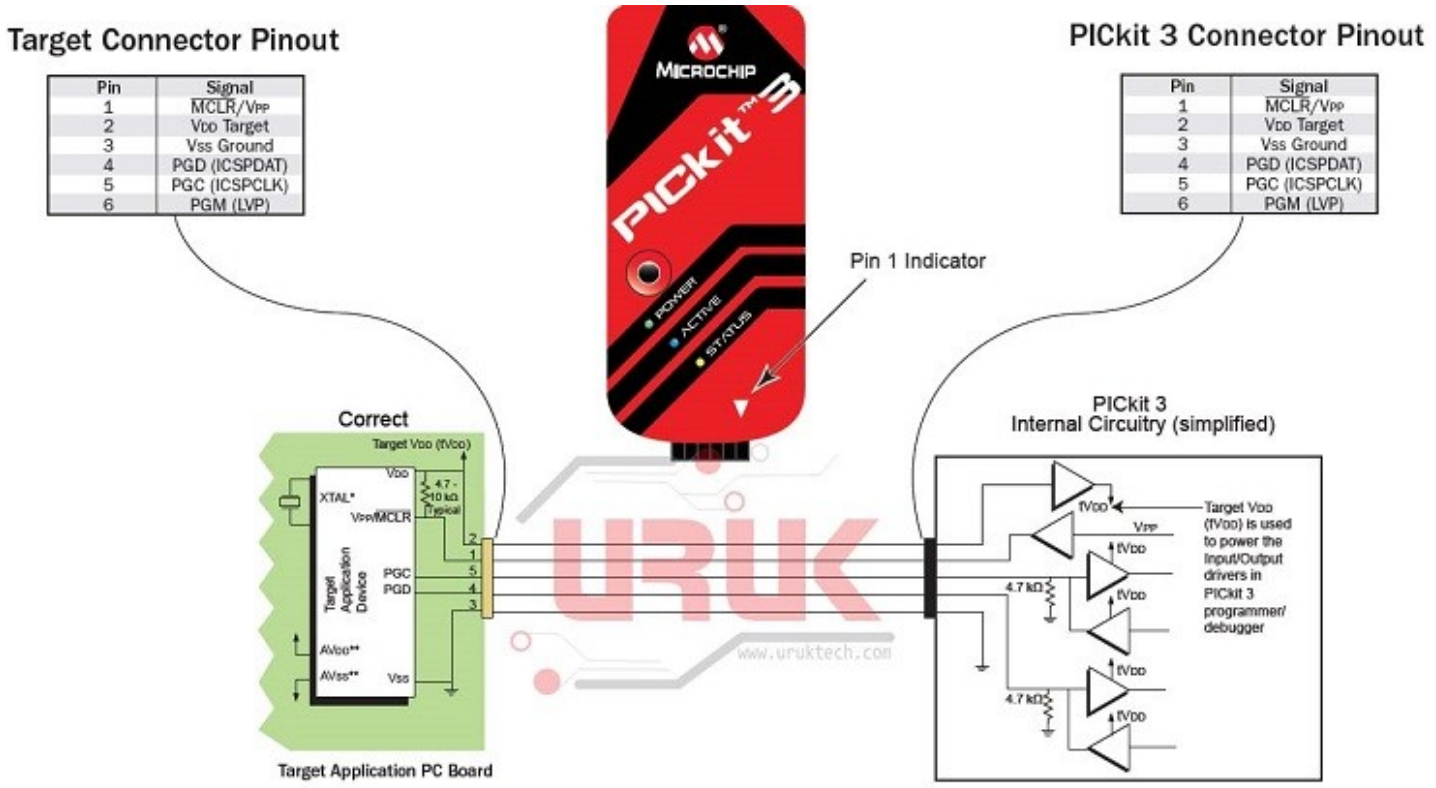
ICD3

- Expensive (~400$)
- High programming speed
- pports software breakpoints (useful for debugging)
- Available in the lab (R100)

ICD3

PIC Programming | Programmer → PIC

For the connection between the PicKit3 and the MCU, you have to look at the datasheet (of both the PicKit3 and the MCU).



Note

In order to be flashed, the microcontroller should be powered (either by an external source or by the PicKit3)

PIC Programming | Walkthrough

What to do step-by-step:

1)Download and install MPLAB X (MPASM comes with it)

2)Launch MPLAB and create a new project (specify your PIC, your compiler and your programmer)

3)Add the linker file which corresponds to your PIC (located in "MPLABX installation path"/vx.xx/mpasmx/LKR/ directory)

4)Right click on Source Files folder >> New >> .asm file

5)Write your program

6)Connect the programming tool and your PIC >> power up your PIC >> plug the programming tool to your PC (you can also use it to power up your PIC)

7)Click on "Make and program device"

I/O ports

Problem

Each microcontroller has a finite number of pins. Each of them can handle a subset of specific tasks which depend on their location.

$=>$ for a given application there is a necessity to chose a microcontroller which can handle all requirements.

Solution

Choose a microcontroller based on its specifications (those can be found in the datasheet)

Example

PIC16F1789 : pin RB2 (see Table 2 [pg 11] in datasheet)

- Digital IO
- Analogic input for channel 8 of ADC, Analogic ouput for channel 3 of DAC
- Inverting input for op-amp 2

I/O ports configuration

Idea

Select desired functionality of a pin by configuring corresponding SFR (see datasheet).

Example with the PIC16F1789

How can I set RD1 as a "1" digital output and RA0 as an binary input?

Answer this question on the wooclap!

(answer located in section 13.0 of the datasheet)

Note

By default all I/O pins of a Microchip 16Fxxxx microcontroller are set as high-impedance inputs.

Clock configuration

Idea

Select MCU clock signal source by configuring corresponding SFR (see datasheet).

Example with the PIC16F1789

How can I set the microcontroller to use its internal oscillator at 1MHz in pair with the integrated frequency multiplier (4x)?

- FOSC2:FOSC0 = 0b100 → is configured outside of the program
- OSCTUNE = 0b*0000000
- OSCCON = 0b11011*1*

Note

Clock frequency ≠ Number of instructions per second

Timer configuration

Idea

Counts clock cycles => useful to keep track of time

- Number of bits of the timer : defines the maximum timer value possible
- Prescaler value : reduces the timer incrementation speed by a given factor

Example with the PIC16F1789

How can I configure the microcontroller to count a period of 0.5s with a clock at 1MHz?

Try to answer this question by yourself before moving to the next slide.
You will find all the required information in the datasheet (sections related to timers)

Timer configuration

Idea

Counts clock cycles => useful to keep track of time

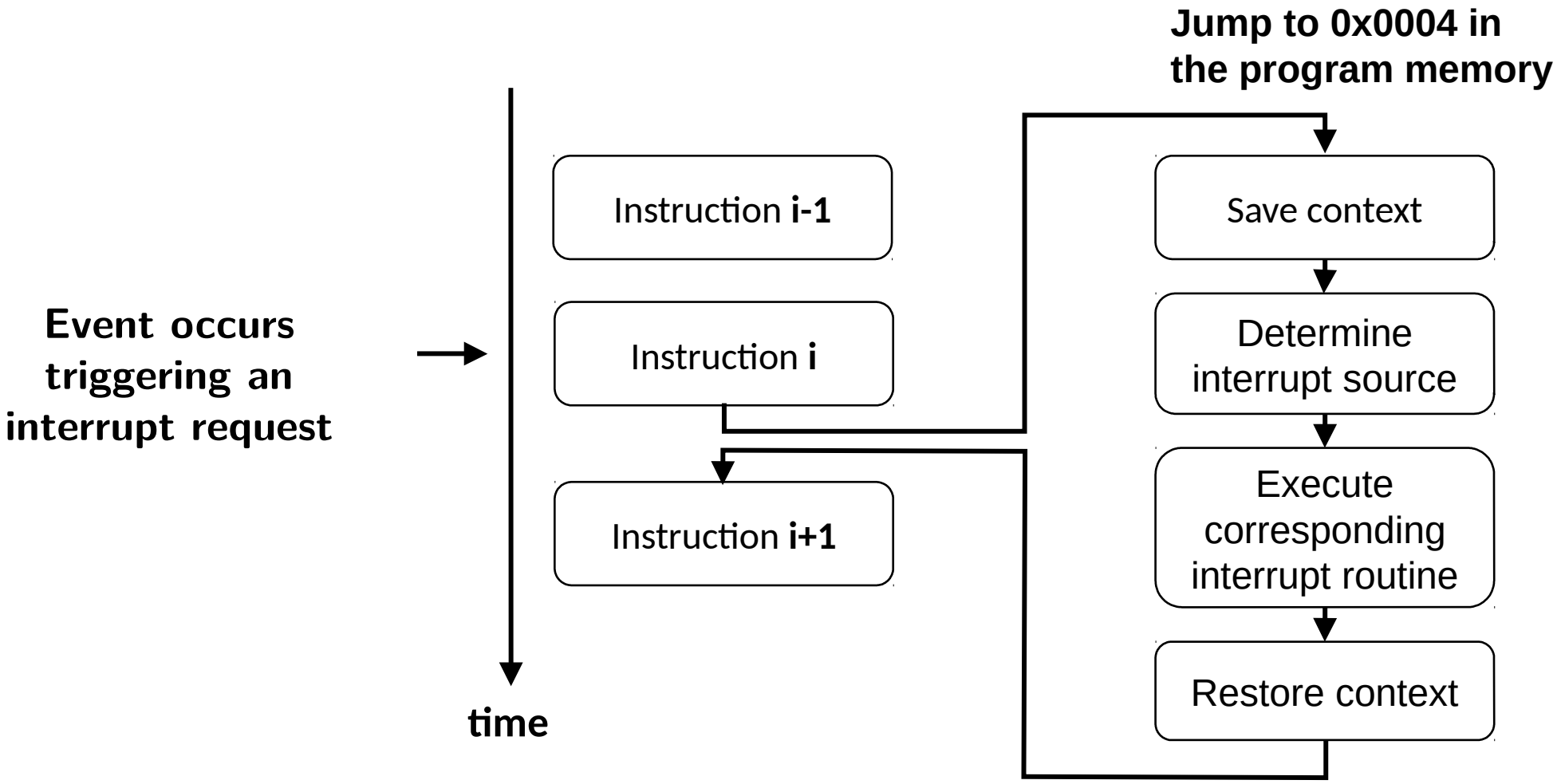
- Number of bits of the timer : defines the maximum timer value possible
- Prescaler value : reduces the timer incrementation speed by a given factor

Example with the PIC16F1789

How can I configure the microcontroller to count a period of 0.5s with a clock at 1MHz?

- 0.5s == 500,000 clock cycles or 125,000 instructions ⇒ too much for 16bits ⇒ need of a prescaler
- 125,000/65,535 = 1.9 ⇒ prescaler = 1:2
- Timer1 is the only solution for the PIC 16F1789 : T1CON = 0b000100*1
- Counter value stored in TMR0H and TMR0L
- 0.5s == 125,000/2 = 62,500 timer value

Interrupts


Jump to 0x0004 in
the program memory
Event occurs
triggering an
interrupt request
Instruction i-1
Instruction i
Instruction i+1
time
Save context
Determine
interrupt source
Execute
corresponding
interrupt routine
Restore context

Interrupts

- All interrupt sources are enabled by a certain bit in a SFR (plus the GIE bit in the INTCON register)
- Several interrupt sources: Timers, input pin, UART, ADC, etc.
- Three bits (within SFR) to control operations:
    - **Flag bit**: indicating that interrupt event occurred
    - **Enable bit**: allowing the program to jump to the interrupt vector when the flag bit is set
    - **Priority bit** : indicating if the corresponding interrupt has high or low priority
- If a flag bit is set (as well as the corresponding enable bit), the MCU jumps automatically to the interrupt vector.

Interrupts

- *When using interrupts, remember:*
  - Clear the interrupt flag before enabling the interrupt
  - Check if the interrupt is enabled before checking its interrupt flag
  - Once in the interrupt code, first thing to do: **clear the interrupt flag** (it is not done automatically!)

A/D Converter

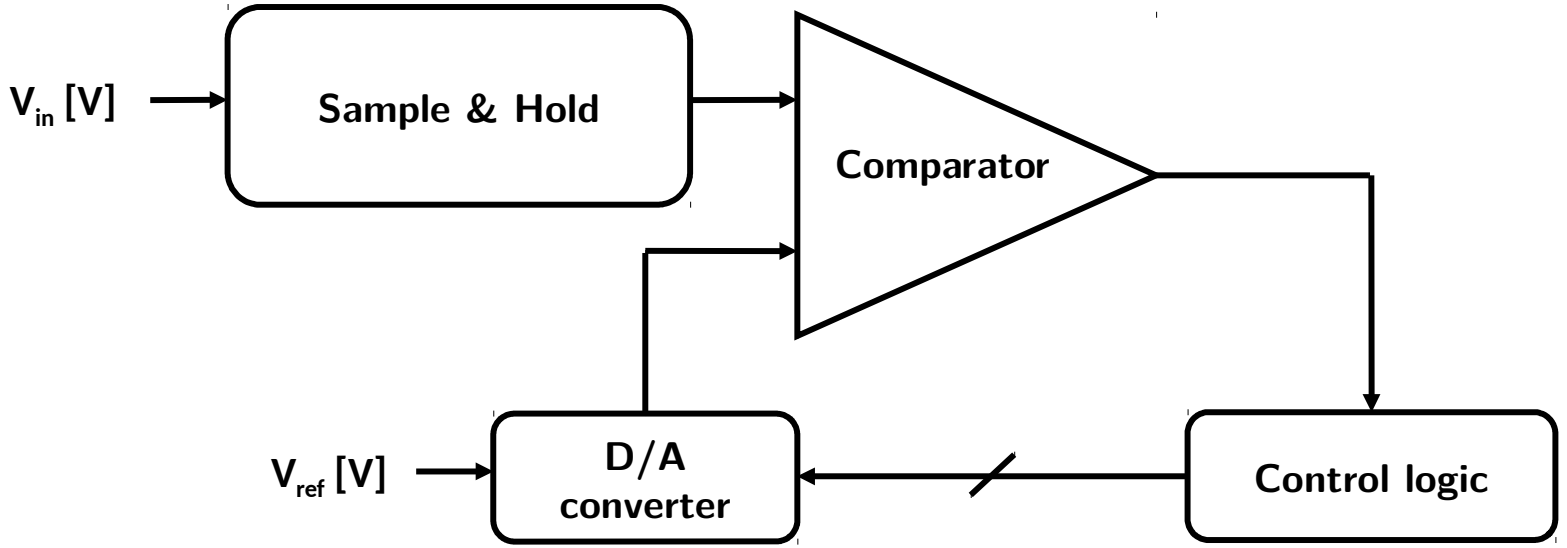
Concept

Converts an input voltage into a binary number according to a reference voltage :

$$V_{meas} = \left(\frac{N\ bit\ word}{2^N}\right) \cdot \left(V_{ref+} - V_{ref-}\right) + V_{ref-}$$

How does it work?

The conversion is made with successive comparisons with the reference voltage.



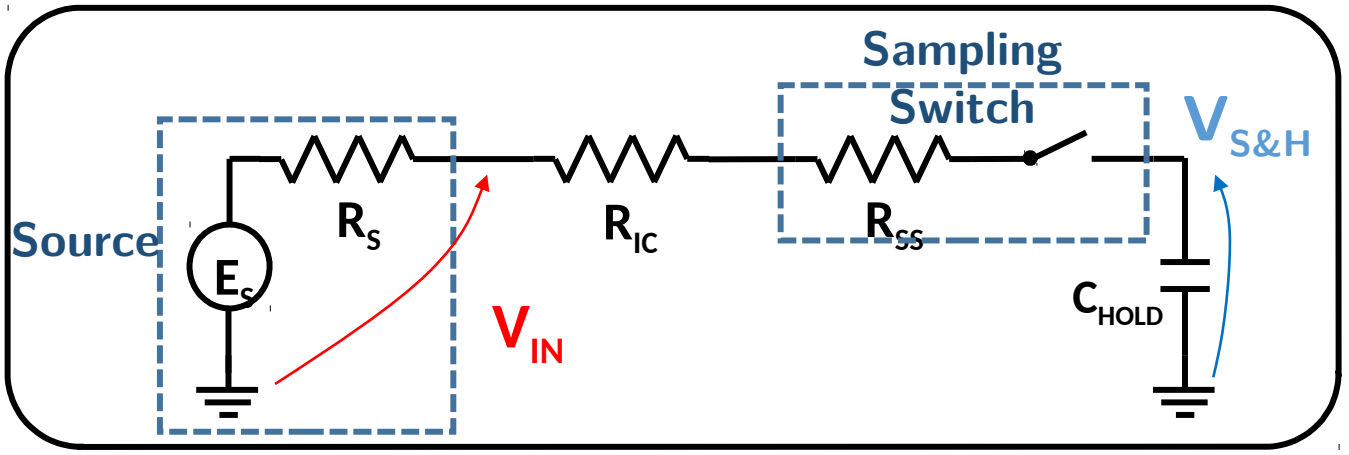
Note : V_{ref} MUST be smaller or equal to V_{dd}

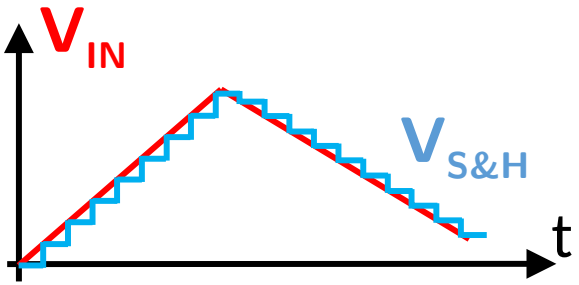
A/D Converter | Sample & Hold circuit

Concept

Samples the input signal by charging a capacitor and holds the value of the voltage constant during the time of the conversion

How does it work?





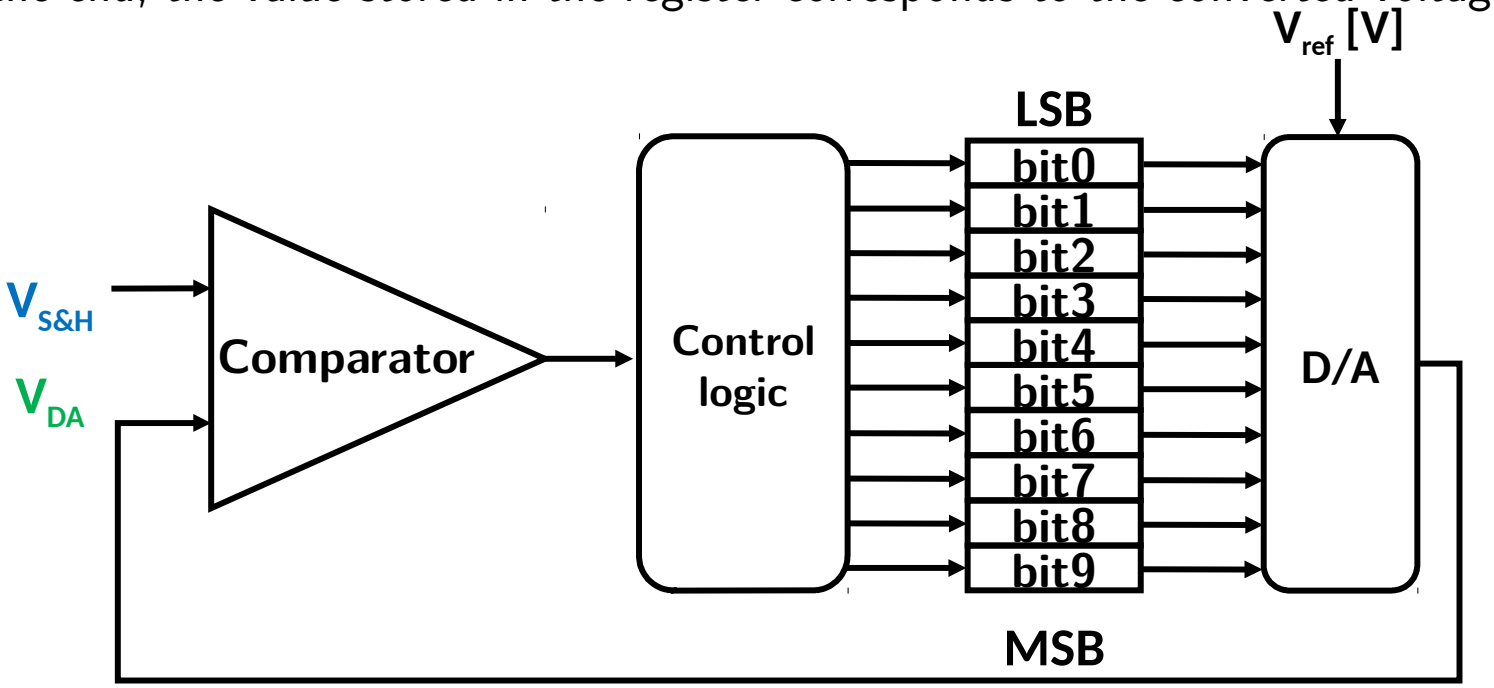
Note

The switch should remain closed long enough for the capacitor to fully charge. This charging period is called the minimum acquisition time and is proportional to $\tau = (R_S+R_{IC}+R_{SS})C_{HOLD} \Rightarrow R_S$ should be as small as possible!

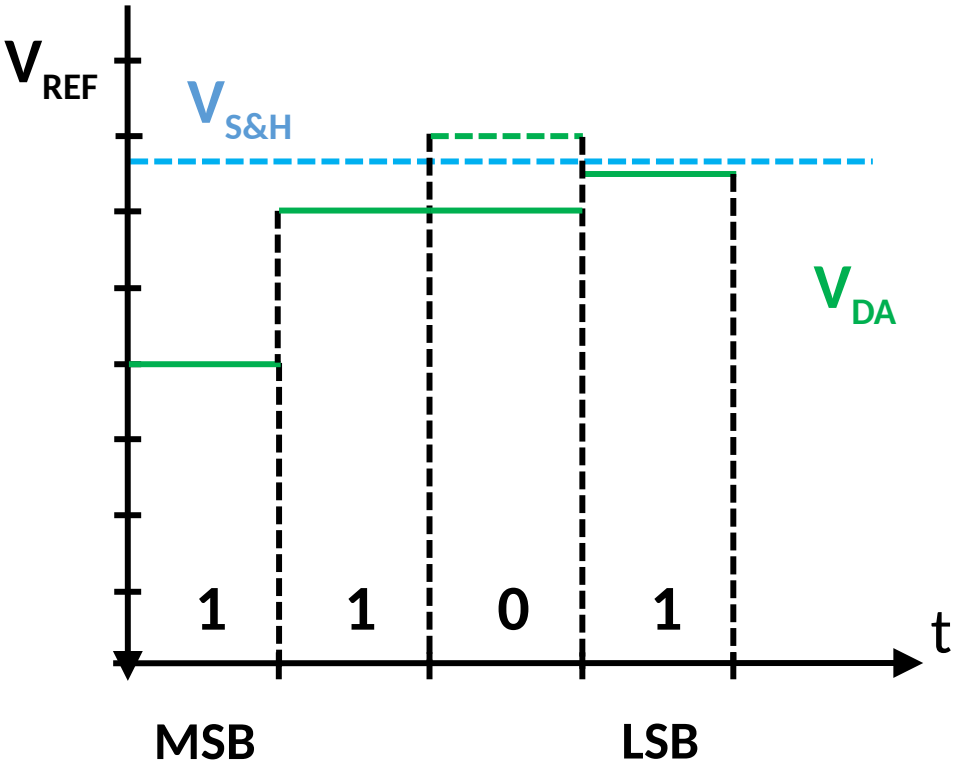
A/D Converter | Converter circuit

How does it work?

- $V_{DA} = (\text{register value}/\ 2^{10}) * V_{REF}$
- Initially, the register is cleared ($V_{DA} = 0$ V)
- Starting from the MSB to the LSB, each bit is set sequentially:
  - If $V_{DA} < V_{S\&H}$, current bit stays set
  - If $V_{DA} > V_{S\&H}$, current bit is clear
- At the end, the value stored in the register corresponds to the converted voltage



A/D Converter | 4-bit converter example



$V_{DA} = (register/2^4) * V_{ref}$

MSB LSB

1 0 0 0 ➔ $V_{DA} = V_{ref}/2$, $V_{S\&H} > V_{DA}$
» bit3 = 1

1 1 0 0 ➔ $V_{DA} = (3/4)*V_{ref}$, $V_{S\&H} > V_{DA}$
» bit2 = 1

1 1 1 0 ➔ $V_{DA} = (7/8)*V_{ref}$, $V_{S\&H} < V_{DA}$
» bit1 = 0

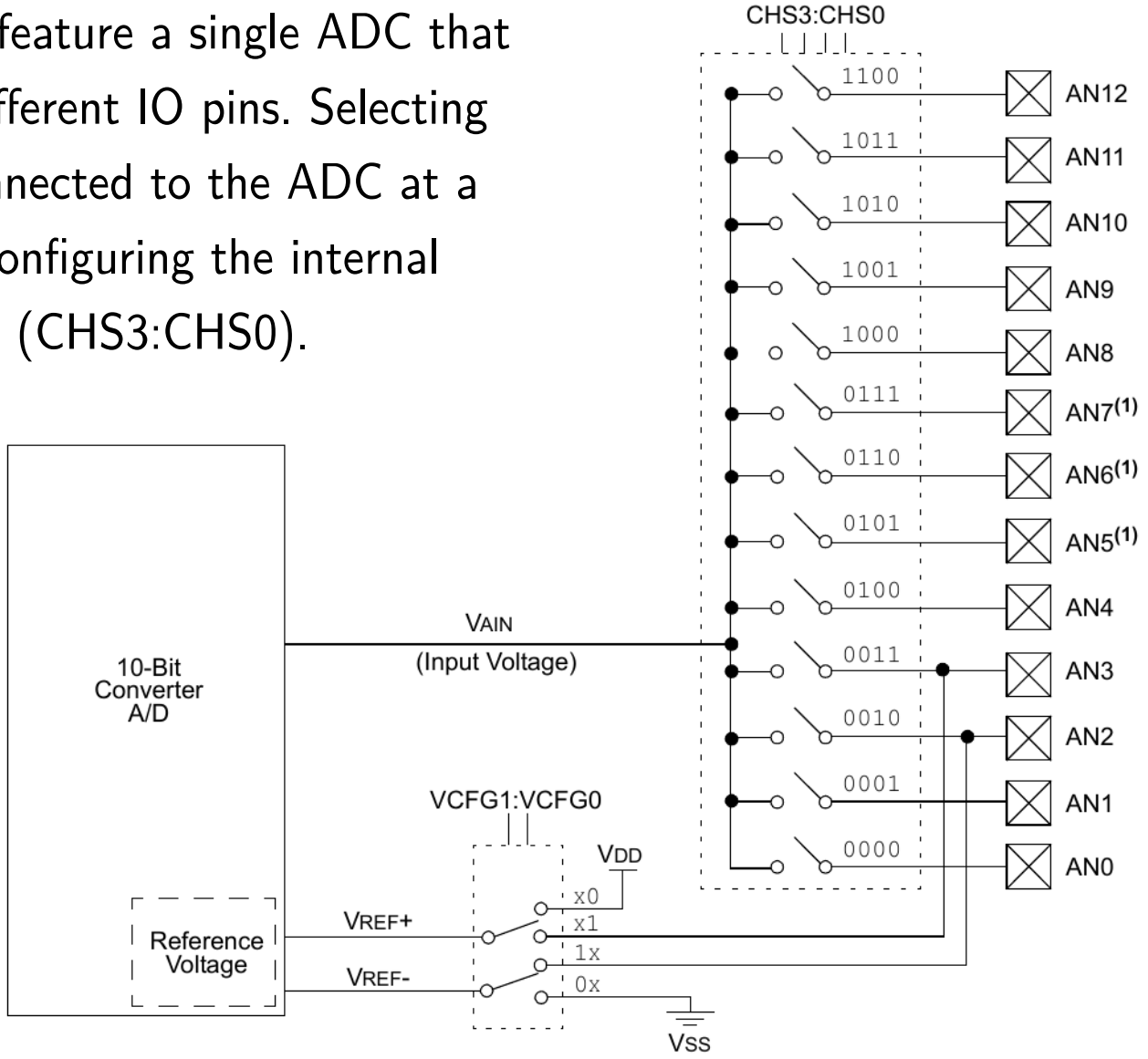
1 1 0 1 ➔ $V_{DA} = (13/16)*V_{ref}$, $V_{S\&H} > V_{DA}$
» bit0 = 1

Note

- Voltage references can be changed to adapt the resolution of the conversion.
- You **cannot** use references which are out of the Vdd-Vss range!

A/D Converter | Example : PIC18F4620

Many microcontrollers feature a single ADC that can be connected to different IO pins. Selecting the pin that will be connected to the ADC at a given time is done by configuring the internal multiplexer of the ADC (CHS3:CHS0).



A/D Converter | Configuration example : PIC16F1789

1. Configure Port:
- Disable pin output driver (Refer to the TRIS register)
- Configure pin as analog (Refer to the ANSEL register)
- Disable weak pull-ups

2.Configure the A/D module
- Configure analog pins, voltage reference and digital I/O (ADCON1)
- Select A/D input channel (ADCON0)
- Select A/D acquisition time (ADCON2)
- Select A/D conversion clock (ADCON2)
- Turn on A/D module (ADCON0)

3. Configure A/D interrupt (if desired):
- Clear A/D interrupt flag bit
- Set ADIE (interrupt enable) bit
- Set GIE (global interrupt enable) bit

4. Wait the required acquisition time (if needed).

5. Start conversion:
   Set GO/DONE bit (ADCON0 register)

6. Wait for A/D conversion to complete, by either:
- Polling for the GO/DONE bit to be cleared

OR
- Waiting for the A/D interrupt

7. Read A/D Result registers (ADRESH:ADRESL); clear bit ADIF, if required.

8. For next conversion, go to step 1 or step 2, as required. The A/D conversion time per bit is defined as T AD. A minimum wait of 2 T AD is required before the next acquisition starts.

→ All of this is given in the datasheet!

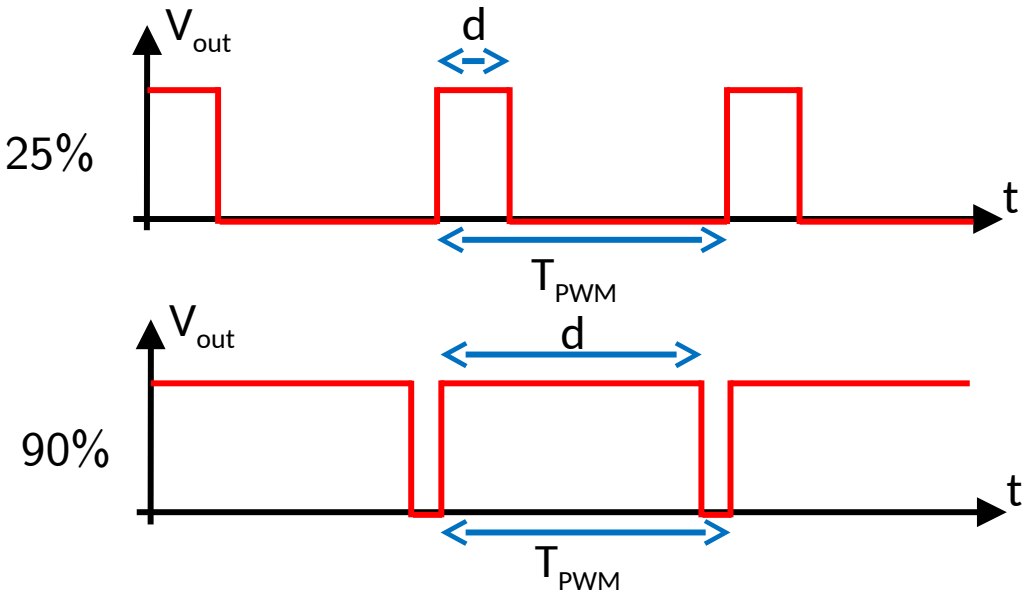
(see section 17.2.6)

PWM

The PWM (Pulse Width Modulation) is a periodic square signal that stays high for a given proportion of its period. PWM signals are widely used to drive various loads (motors, converters,...).

A PWM signal has 2 parameters:

- T_{PWM} = period of the signal
- d = duty cycle : Proportion of the period that remains high



PWM | Integrated generator

Concept

Some microcontrollers contain PWM modules which automatically generate a PWM signal on one of their output pins.

How does it work?

- The period and the duration of the duty cycle are encoded in terms of timer value.
- The value of the output pin is set depending on the timer value compared the the 2 parameters

Note

The resolution of the duty cycle is dependent of the MCU clock frequency and of the frequency of the desired PWM signal :

$$\text{PWM Resolution (max)} = \frac{\log\left(\frac{\text{FOSC}}{\text{FPWM}}\right)}{\log(2)} \text{bits}$$

PWM | Setup example for PIC18F4620

Operations

1) Set the PWM period by writing to the PR2 register.
2) Set the PWM duty cycle by writing to the CCPRxL register and CCPxCON<5:4> bits.
3) Configure the CCPx module for PWM operation.
4) Configure and start Timer2.
5) Make the CCPx pin an output by clearing the appropriate TRIS bit.

→ All of this is given in the datasheet (section 25.3.2)!

EQUATION 15-1:

$$\text{PWM Period} = [(PR2) + 1] \cdot 4 \cdot T_{OSC} \cdot (\text{TMR2 Prescale Value})$$

EQUATION 15-2:

$$\text{PWM Duty Cycle} = (CCPRxL{:}CCPxCON\langle 5{:}4\rangle) \cdot T_{OSC} \cdot (\text{TMR2 Prescale Value})$$

Capture module

Concept

Captures the value of a timer as soon as an external signal was triggered on a pin.

How does it work?

When a falling or a rising edge (one or the other, not both) is detected on the dedicated pin, the value of a given timer at this moment is copied inside of a register.

- An interrupt can be triggered once a capture occurs

Note

- Be careful, if two successive triggers of the pin are too spaced in time, several timer overflows may have been reached during this time.
- Be careful with input bounces!

Compare module

Concept

Modifies the value of an output pin when a given timer reaches a given value.

How does it work?

The user set a value in a dedicated register. This value is constantly compared with a given timer. When the timer value matches the user value, an output pin can be triggered high or low depending on the user need.

- An interrupt can be triggered when a compare event occurred

Note

- Keep in mind that the time required for a timer to reach a given value may depend of the rest of your code (timer resets for example).

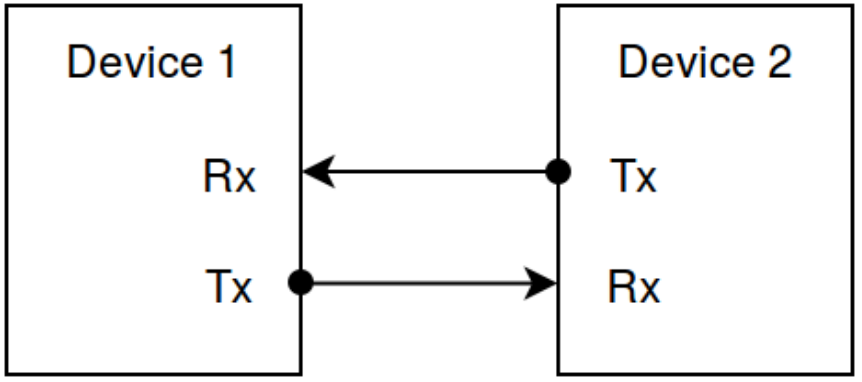
UART

Concept

The Universal Asynchronous Receiver Transmitter (UART) is used for transmitting and receiving serial data.

How does it work?

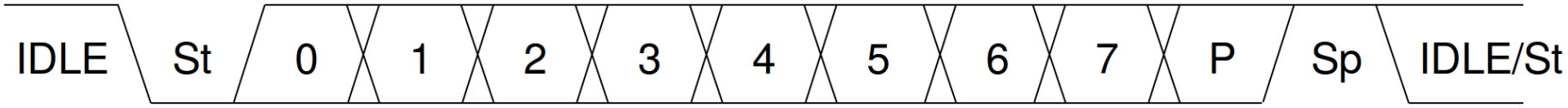
The communication is made thanks to two wires, where each wire is used to carry the communication in one direction.



Note

The transmission speed (baudrate $==$ bits/s in this case) is set by the user and should be the same for both devices.

UART | Communication protocol for a line



1	**St**	Start bit, always low
2	**(n)**	Data bits (5 to 8)
3	**P**	Parity bit (optional), can be even or odd
4	**Sp**	Stop bit, always high
5	**IDLE**	No transfer on the line, always high

UART | Inside of the microcontroller I

Configuration

- Set UART configuration registers for your need
- Set baudrate of the line
- Enable interrupts if desired

Transmission

- Load data to transmit in the dedicated register
- When the stop bit of the previous transmission is reached:
    - Transfer of the data to the transfer register (interrupt can be triggered for this event)
    - Flag is raised to tell that the transfer register is full
- Transmission begins

UART | Inside of the microcontroller II

Reception

- Flag is raised when some data reception completed (an interrupt corresponding to this flag can be triggered is desired)
- Collect the received data by reading the register which temporarily holds them
- Determine if any errors occurred during the reception and clear corresponding flags if necessary

Conclusion for this session

The datasheet is your friend!