Università degli studi di Genova

Dipartimento di Informatica, Bioingeneria, Robotica e Ingegneria dei Sistemi

Corso di Laurea Magistrale in Robotics Engineering

Embedded Systems Documentation

Authors:

Cornia Luca
Lombardi Simone
Dondero Enrico
Bono Alberto

Academic year 2023/2024

Indice

Capitolo 1

Microcontroller

The following figure shows all input/output of the microcontroller.

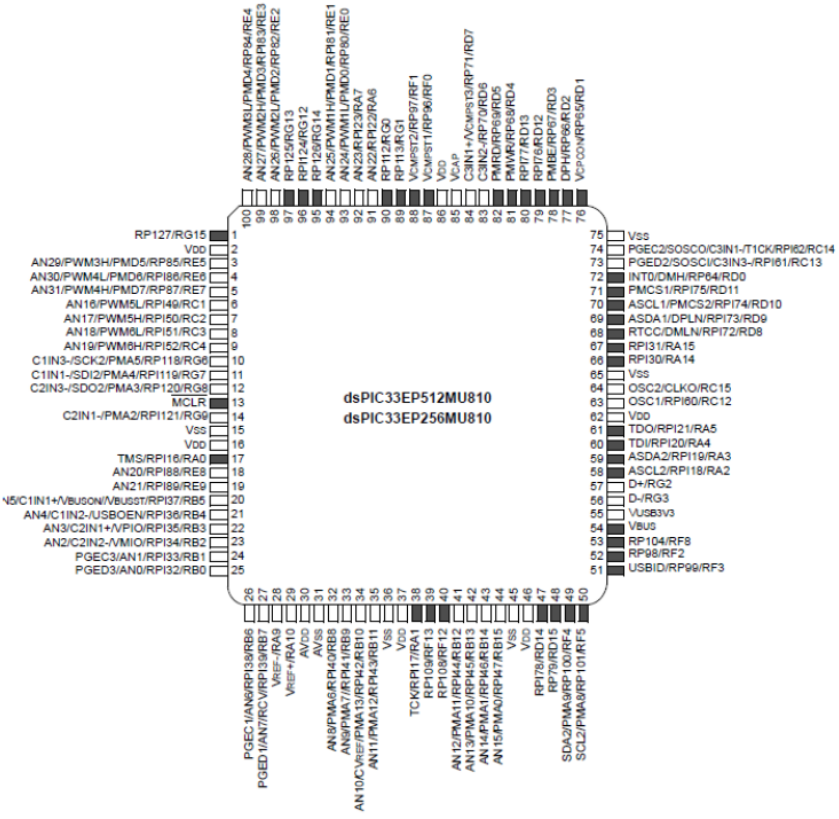

Figura 1.1: I/O Pins

General information about our microcontroller:

1. Model: dsPIC33EP512MU810
2. Architecture: 16-bit
3. External oscillator frequency: 144 MHz
4. Clock frequency: 72 MHz
5. Number of timers: 5

Capitolo 2

Digital I/O port

In this chapter we are talking about how to configure and get information to pins. To do this it's necessary the use of registers.

Registers:

- TRISx[1]: used to select pin as input or output.
- PORTx: used to **read** the voltage value of the pin.
- LATx: used to **write** a logic value as pins output.

So we have registers (e.g TRISA) and bit field data structure (e.g TRISAbits).

Important:

1. Always read inputs from PORTx and write outputs to LATx. If you need to read what you set as output: **read LATx**.

[1]x: stands for any of the available I/O registers: A, B, C, D, E, F

2. When you set pins:
   1 = input
   0 = output.

3. Deactivate analog pins (if you don't use PWM). So within the main add the following line of code:
   ANSELA=ANSELB=ANSELC=ANSELD=ANSELE=ANSELG=0x0000;

Registers of our components:

- Led LD1: RA0
- Led LD2: RG9
- Button T2: RE8
- Button T3: RE9

Capitolo 3

Oscillator and Timer

3.1 Oscillator

In our device, the configuration about oscillator is set by the bootloader and cannot be changed.
The fundamental equation to take into account is the following:

$$F_{cy}{}^{1} = \frac{F_{osc}{}^{2}}{2} \tag{3.1}$$

3.2 Timers

Here, it has been introduced the concept of the **prescaler**. This block (see 3.1) divide the frequency of clock by the integer number. This operation enables the storage of larger numbers within the registers, which cannot be initially represented due to the limited capacity of our 16-bit registers.

[1] Clock frequency: The clock frequency determines the rate at which each instruction is executed within the processor

3.2.1 Prescaler

How to choose the value of the prescaler?
11 = 1:256
10 = 1:64
01 = 1:8
00 = 1:1

N.B: There are two bits, so when you set the prescaler remember to setup both bits.

3.2.2 Registers

The timer has 2 registers:

- **TMRx**: It counts the period of the clock.
- **PRx**: It contains a number corresponding to the amount of time required to reach.

These registers have special function used for configuration and access:

- **TxCON**: Timer x Control register (e.g T1CON, T2CON)
  - TON: Starts the timer.
  - TCKPS: Set the prescaler.
  - TCS: clock source.
  - TGATE: If the gated mode should be enabled.
- **TMRx**: Represent the current value of the timer. **Set to zero** to reset the timer.

- **PRx**: Set this to the value the timer must count to.

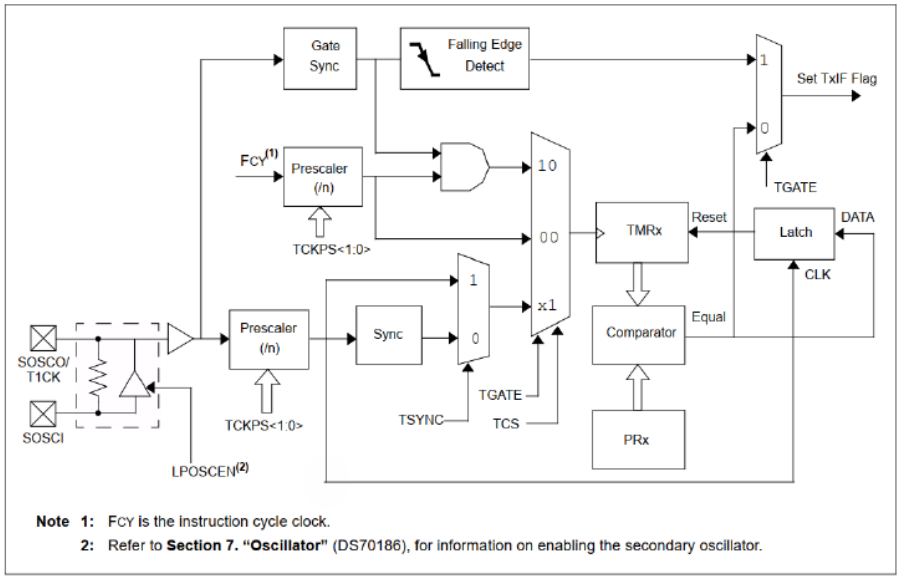

Figura 3.1: Timer A - scheme

3.2.3 Interrupt flags

Let's introduce another concept: the **interrupt flags**.
The interrupt flag signals an event related to the corresponding timer (e.g when the timer expire).
The interrupt flags[3] are the following ones:

- IFS0bits.T1IF
- IFS0bits.T2IF
- IFS0bits.T3IF
- IFS1bits.T4IF
- IFS1bits.T5IF

IMPORTANT: Must be put to zero manually to be notified of a new event!

[3]Page 58 of the datasheet shows the location of all the IFs

3.3 Setup timer

Here there are all steps for setting in a right way the timer:

1. Set **clock source** to internal (72 MHz):
   TxCONbits.TCS = 0;

2. Set the **prescaler**:
   TxCONbits.TCKPS0 = bit value;
   TxCONbits.TCKPS1 = bit value;

3. Set the value the timer must **count to**:
   PRx = value;

4. **Start** timer:
   TxCONbits.TON = 1;

5. Set **flag** to zero:
   IFSybits.TxIF = 0;

Capitolo 4

Interrupts

There are two different type of interrupt:

1. **Hardware interrupt**: an asynchronous signal from HW indicating the need for attention. It means that it does have any temporal relation to which part of the code the microcontroller is executing

2. **Software trap**: a synchronous event in SW indicating the need for a change in execution. It means that the interrupt depends on the instruction we are currently executing.

In the following picture is represented the workflow of an interrput:

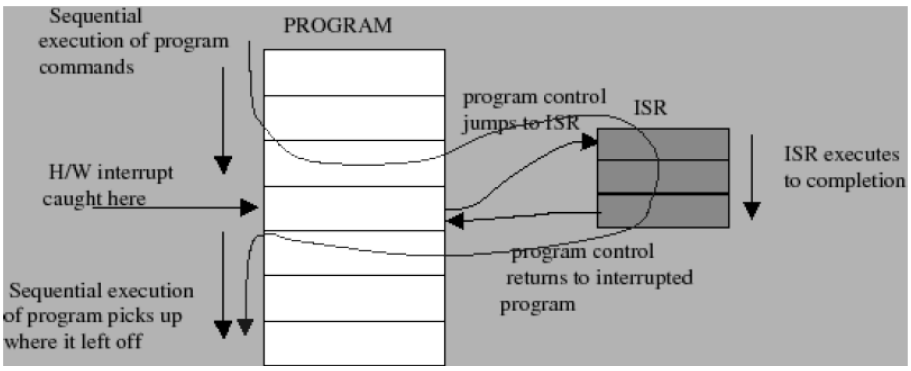

Figura 4.1: interrupt scheme

At the beginning of each instruction cycle all interrupt flags are sampled. If the interrupt flag and the enable value are high the CPU saves the information and then the **Interrupt Service Routine (ISR)** is executed. After the ISR, the previous context is recovered and the program execution continues.

Rules of ISR function:

- Each ISR has a specific name which is used for linking its code to the **Interrupt Vector Table (IVT)**[1].
- No parameters can be passed or returned to the ISR.
- The flag generating the interrupt should be generally cleared in the ISR.
- ISRs shouldn't be called by main line code.
- Care must be taken with shared data.

```
void __attribute__((__interrupt__, __auto_psv__)) _INT0Interrupt() {
  IFS0bits.INT0IF = 0; // reset interrupt flag

  [...] // do some action
}
```

Figura 4.2: ISR function

There are 2 registers:

1. IFSx[2]: Interrupt Flag Status registers, contains the flags that notify the program of certain events (e.g timer expire).
2. IECx[3]: Interrupts Enable Control registers, it is used to **enable** an interrupt.

[1]ISR names can be found in the Interrupts documentation at page 11.
[2]Page 58 of the datasheet shows the location of all the IFs
[3]Page 58 of the datasheet shows the location of all the IEs

Note: This is the main difference between the interrupt and what we do in the chapter of timers. In this case it's possible to use a timer as an interrupt by simply enabling its register.

4.1 Remappable input pin

The remappable input pin (RP) allow to define an external interrupt like a button. So, when you configure the RP pin for an input, the corresponding bit in the TRISx register must bu configured as an input (set to '1').
The **table of input mapping** is available at page 9 of the I/O documentation.

The main thing to remember is:

$$\textbf{Functionality} = \textbf{Remappable Input Pin} \tag{4.1}$$

So for example if we want to maps the U1RX to RPI20:

```
TRISAbits.TRISA4 = 1; // RPI20 is RA4
RPINR11bits.U1RXR = 20;
```

Figura 4.3: Remappable input pin

4.2 Remappable output pin

The same thing is available for remapping a pin as an output. So in this case we define a special functionality of an output pin. The **table of output mapping** is available at page 12 of the I/O documentation.

The main thing to remember is:

$$\textbf{Remappable Output Pin} = \textbf{Functionality} \tag{4.2}$$

So for example if we want to maps the U1TX to RP104:

```
RPOR11bits.RP104R = 1; // 1 = U1TX
```

Figura 4.4: Remappable output pin

Note: The value "1" is in the second column of the table at page 12.