Il microcontrollore Microchip pic18f2420

Il testo non sostituisce in alcun modo il manuale del microcontrollore, DS39631E.pdf (412 pagine!!) , a cui si fa riferimento.

Il pic18f2420 fa parte di una famiglia di microcontrollori le cui caratteristiche sono riassunte nella tabella seguente. Si tratta di un microcontrollore moderno, allo stato dell'arte, di fascia alta, completo, programmabile in assembly e/o in C, dotato di moltissime periferiche. L'unica periferica non presente è la periferica USB. Per trovare qualcosa di analogo in ambiente MICROCHIP (e completo di USB) bisogna considerare il PIC18F4550. Il pic18f2420 costa attualmente intorno a 2 dollari al pezzo. La assenza della periferica USB costituisce certo una limitazione rispetto al PIC18F4550, ma in ambiente didattico essa sarebbe comunque difficilmente introducibile per la sua complessità; riuscire ad esaminare tutte le periferiche contenute nel microcontrollore in esame è già un obbiettivo più che soddisfacente. 3) avendo scelto un microcontrollore di fascia alta è possibile studiare solo le periferiche o approfondire lo studio a seconda della situazione didattica. Per le periferiche "standard" vengono esaminati dei programmini di esempio. L'esame delle periferiche più complesse è lasciato al lettore. Il microcontrollore può essere programmato sia lasciandolo a bordo del target che mediante un apposiito programmatore o caricando un bootloader. Il software può essere sviluppato in diversi linguaggi fra i quali ovviamente l'assembly (l'assemblatore è fornito gratuitamente dalla microchip) che in linguaggio C. Il pic18F2420 è ottimizzato per la programmazione C, infatti si tratta di un microcontrollore RISC (a set di istruzione ridotto) per il quale la programmazione manuale in assembly risulta abbastanza difficoltosa e inefficiente. La famiglia di microcontrollori in esame è oltretutto disponibile sul simulatore interattivo PROTEUS LABCENTER UK.

CARATTERISTICHE GENERALI

MICROCHIP PIC18F2420/2520/4420/4520

28/40/44-Pin Enhanced Flash Microcontrollers with 10-Bit A/D and nanoWatt Technology

Power Management Features:

- Run: CPU on, Peripherals on
- Idle: CPU off, Peripherals on
- Sleep: CPU off, Peripherals off
- Ultra Low 50nA Input Leakage
- Run mode Currents Down to 11 µA Typical
- Idle mode Currents Down to 2.5 µA Typical
- Sleep mode Current Down to 100 nA Typical
- Timer1 Oscillator: 900 nA, 32 kHz, 2V
- Watchdog Timer: 1.4 µA, 2V Typical
- Two-Speed Oscillator Start-up

Flexible Oscillator Structure:

- Four Crystal modes, up to 40 MHz
- 4x Phase Lock Loop (PLL) – Available for Crystal and Internal Oscillators
- Two External RC modes, up to 4 MHz
- Two External Clock modes, up to 40 MHz
- Internal Oscillator Block:
  - Fast wake from Sleep and Idle, 1 µs typical
  - 8 use-selectable frequencies, from 31 kHz to 8 MHz
  - Provides a complete range of clock speeds from 31 kHz to 32 MHz when used with PLL
  - User-tunable to compensate for frequency drift
- Secondary Oscillator using Timer1 @ 32 kHz
- Fail-Safe Clock Monitor:
  - Allows for safe shutdown if peripheral clock stops

Peripheral Highlights:

- High-Current Sink/Source 25 mA/25 mA
- Three Programmable External Interrupts
- Four Input Change Interrupts
- Up to 2 Capture/Compare/PWM (CCP) modules, one with Auto-Shutdown (28-pin devices)
- Enhanced Capture/Compare/PWM (ECCP) module (40/44-pin devices only):
  - One, two or four PWM outputs
  - Selectable polarity
  - Programmable dead time
  - Auto-shutdown and auto-restart

Peripheral Highlights (Continued):

- Master Synchronous Serial Port (MSSP) module Supporting 3-Wire SPI (all 4 modes) and I^2C™ Master and Slave modes
- Enhanced Addressable USART module:
  - Supports RS-485, RS-232 and LIN/J2602
  - RS-232 operation using internal oscillator block (no external crystal required)
  - Auto-wake-up on Start bit
  - Auto-Baud Detect
- 10-Bit, up to 13-Channel Analog-to-Digital (A/D) Converter module:
  - Auto-acquisition capability
  - Conversion available during Sleep
- Dual Analog Comparators with Input Multiplexing
- Programmable 16-Level High/Low-Voltage Detection (HLVD) module:
  - Supports interrupt on High/Low-Voltage Detection

Special Microcontroller Features:

- C Compiler Optimized Architecture:
  - Optional extended instruction set designed to optimize re-entrant code
- 100,000 Erase/Write Cycle Enhanced Flash Program Memory Typical
- 1,000,000 Erase/Write Cycle Data EEPROM Memory Typical
- Flash/Data EEPROM Retention: 100 Years Typical
- Self-Programmable under Software Control
- Priority Levels for Interrupts
- 8 x 8 Single-Cycle Hardware Multiplier
- Extended Watchdog Timer (WDT):
  - Programmable period from 4 ms to 131s
- Single-Supply 5V In-Circuit Serial Programming™ (ICSP™) via Two Pins
- In-Circuit Debug (ICD) via Two Pins
- Wide Operating Voltage Range: 2.0V to 5.5V
- Programmable Brown-out Reset (BOR) with Software Enable Option

Device	Program Memory		Data Memory		I/O	10-Bit A/D (ch)	CCP/ ECCP (PWM)	MSSP		EUSART	Comp.	Timers 8/16-Bit
	Flash (bytes)	# Single-Word Instructions	SRAM (bytes)	EEPROM (bytes)				SPI	Master I^2C™			
PIC18F2420	16K	8192	768	256	25	10	2/0	Y	Y	1	2	1/3
PIC18F2520	32K	16384	1536	256	25	10	2/0	Y	Y	1	2	1/3
PIC18F4420	16K	8192	768	256	36	13	1/1	Y	Y	1	2	1/3
PIC18F4520	32K	16384	1536	256	36	13	1/1	Y	Y	1	2	1/3

pinout

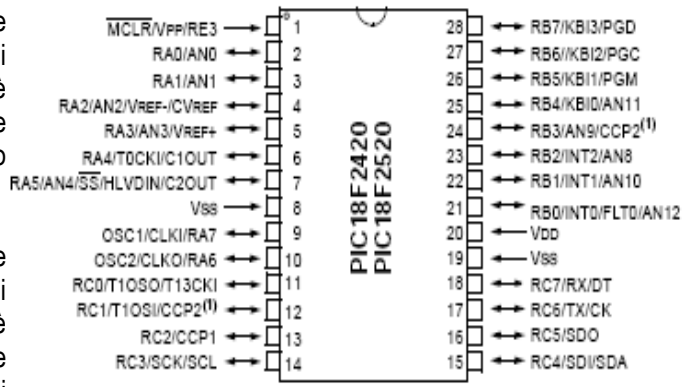
TABLE 1-1: DEVICE FEATURES

Features	PIC18F2420
Operating Frequency	DC – 40 MHz
Program Memory (Bytes)	16384
Program Memory (Instructions)	8192
Data Memory (Bytes)	768
Data EEPROM Memory (Bytes)	256
Interrupt Sources	19
I/O Ports	Ports A, B, C, (E)
Timers	4
Capture/Compare/PWM Modules	2
Enhanced Capture/Compare/PWM Modules	0
Serial Communications	MSSP, Enhanced USART
Parallel Communications (PSP)	No
10-Bit Analog-to-Digital Module	10 Input Channels
Resets (and Delays)	POR, BOR, RESET Instruction, Stack Full, Stack Underflow (PWRT, OST), MCLR (optional), WDT
Programmable High/Low-Voltage Detect	Yes
Programmable Brown-out Reset	Yes
Instruction Set	75 Instructions; 83 with Extended Instruction Set Enabled
Packages	28-Pin SPDIP 28-Pin SOIC 28-Pin QFN

Il microcontrollore dispone di diverse sorgenti di reset; la causa del reset è tracciabile e configurabile utilizzando il registro RCON.

Il microcontrollore dispone di diverse sorgenti di clock; una di essa è selezionabile e configurabile utilizzando i registri OSCCON e OSCTUNE.

28-Pin SPDIP, SOIC



4.0 RESET

The PIC18F2420/2520/4420/4520 devices differentiate between various kinds of Reset.

a) Power-on Reset (POR)
b) MCLR Reset during normal operation
c) MCLR Reset during power-managed modes
d) Watchdog Timer (WDT) Reset (during execution)
e) Programmable Brown-out Reset (BOR)
f) RESET Instruction
g) Stack Full Reset
h) Stack Underflow Reset

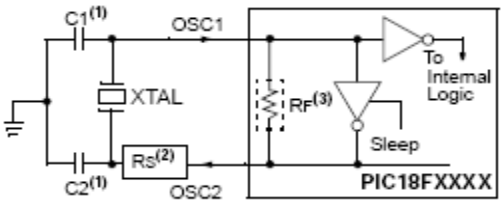
2.0 OSCILLATOR CONFIGURATIONS

2.1 Oscillator Types

PIC18F2420/2520/4420/4520 devices can be operated in ten different oscillator modes. The user can program the Configuration bits, FOSC<3:0>, in Configuration Register 1H to select one of these ten modes:

1. LP Low-Power Crystal
2. XT Crystal/Resonator
3. HS High-Speed Crystal/Resonator
4. HSPLL High-Speed Crystal/Resonator with PLL Enabled
5. RC External Resistor/Capacitor with FOSC/4 Output on RA6
6. RCIO External Resistor/Capacitor with I/O on RA6
7. INTIO1 Internal Oscillator with FOSC/4 Outpu on RA6 and I/O on RA7
8. INTIO2 Internal Oscillator with I/O on RA6 and RA7
9. EC External Clock with FOSC/4 Output
10. ECIO External Clock with I/O on RA6

FIGURE 2-1: CRYSTAL/CERAMIC RESONATOR OPERATION (XT, LP, HS OR HSPLL CONFIGURATION)



Note 1: See Table 2-1 and Table 2-2 for initial values of C1 and C2.
2: A series resistor (RS) may be required for AT strip cut crystals.
3: RF varies with the oscillator mode chosen.

TABLE 2-1: CAPACITOR SELECTION FOR CERAMIC RESONATORS

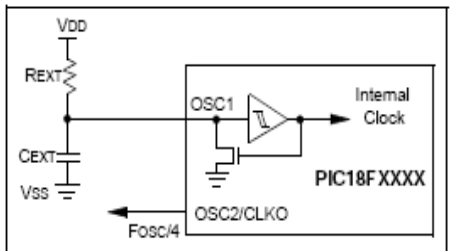
Typical Capacitor Values Used:			
Mode	**Freq**	**OSC1**	**OSC2**
XT	3.58 MHz	15 pF	15 pF
	4.19 MHz	15 pF	15 pF
	4 MHz	30 pF	30 pF
	4 MHz	50 pF	50 pF

Capacitor values are for design guidance only.

Different capacitor values may be required to produce acceptable oscillator operation. The user should test the performance of the oscillator over the expected VDD and temperature range for the application.

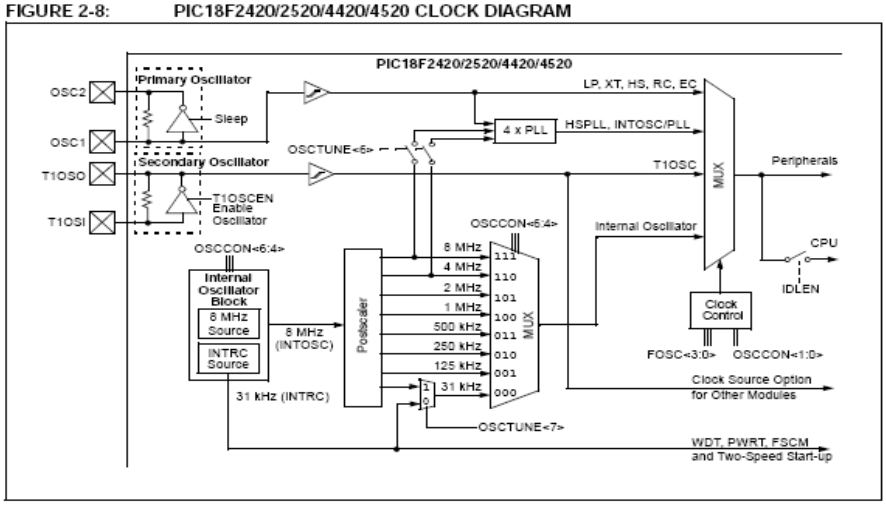
See the notes following Table 2-2 for additional information.

FIGURE 2-5: RC OSCILLATOR MODE



Recommended values: 3 kΩ ≤ REXT ≤ 100 kΩ
CEXT > 20 pF

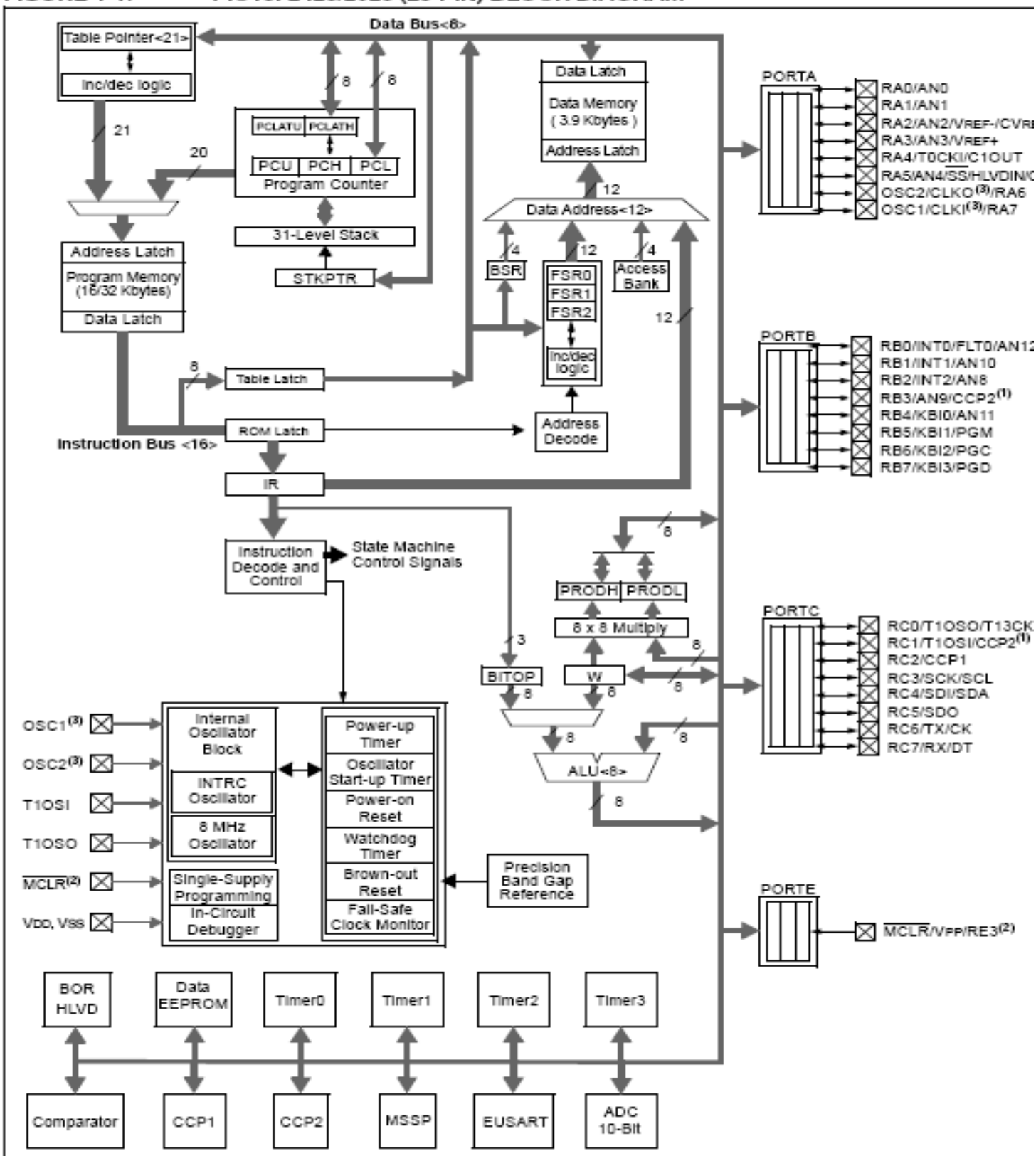
FIGURE 2-8: PIC18F2420/2520/4420/4520 CLOCK DIAGRAM



STRUTTURA INTERNA

Nella figura seguente è rappresentata la architettura della macchina con le varie periferiche. A seconda della piedinatura, quindi della versione, alcune periferiche possono essere non presenti o ridotte.

FIGURE 1-1: PIC18F2420/2520 (28-PIN) BLOCK DIAGRAM



Note 1: CCP2 is multiplexed with RC1 when Configuration bit, CCP2MX, is set, or RB3 when CCP2MX is not set.

2: RE3 is only available when MCLR functionality is disabled.

3: OSC1/CLKI and OSC2/CLKO are only available in select oscillator modes and when these pins are not being used as digital I/O. Refer to **Section 2.0 "Oscillator Configurations"** for additional information.

DESCRIZIONE DELLE FUNZIONALITA' DEI PIN

TABLE 1-2: PIC18F2420/2520 PINOUT I/O DESCRIPTIONS

Pin Name	Pin Number		Pin Type	Buffer Type	Description
	SPDIP, SOIC	QFN			
$\overline{MCLR}$/VPP/RE3	1	26			Master Clear (input) or programming voltage (input).
$\overline{MCLR}$			I	ST	Master Clear (Reset) input. This pin is an active-low Reset to the device.
VPP			P		Programming voltage input.
RE3			I	ST	Digital input.
OSC1/CLKI/RA7	9	6			Oscillator crystal or external clock input.
OSC1			I	ST	Oscillator crystal input or external clock source input. ST buffer when configured in RC mode; CMOS otherwise.
CLKI			I	CMOS	External clock source input. Always associated with pin function, OSC1. (See related OSC1/CLKI, OSC2/CLKO pins.)
RA7			I/O	TTL	General purpose I/O pin.
OSC2/CLKO/RA6	10	7			Oscillator crystal or clock output.
OSC2			O	—	Oscillator crystal output. Connects to crystal or resonator in Crystal Oscillator mode.
CLKO			O	—	In RC mode, OSC2 pin outputs CLKO which has 1/4 the frequency of OSC1 and denotes the instruction cycle rate.
RA6			I/O	TTL	General purpose I/O pin.
					PORTA is a bidirectional I/O port.
RA0/AN0	2	27			
RA0			I/O	TTL	Digital I/O.
AN0			I	Analog	Analog input 0.
RA1/AN1	3	28			
RA1			I/O	TTL	Digital I/O.
AN1			I	Analog	Analog input 1.
RA2/AN2/VREF-/CVREF	4	1			
RA2			I/O	TTL	Digital I/O.
AN2			I	Analog	Analog input 2.
VREF-			I	Analog	A/D reference voltage (low) input.
CVREF			O	Analog	Comparator reference voltage output.
RA3/AN3/VREF+	5	2			
RA3			I/O	TTL	Digital I/O.
AN3			I	Analog	Analog input 3.
VREF+			I	Analog	A/D reference voltage (high) input.
RA4/T0CKI/C1OUT	6	3			
RA4			I/O	ST	Digital I/O.
T0CKI			I	ST	Timer0 external clock input.
C1OUT			O	—	Comparator 1 output.
RA5/AN4/$\overline{SS}$/HLVDIN/C2OUT	7	4			
RA5			I/O	TTL	Digital I/O.
AN4			I	Analog	Analog input 4.
$\overline{SS}$			I	TTL	SPI slave select input.
HLVDIN			I	Analog	High/Low-Voltage Detect input.
C2OUT			O	—	Comparator 2 output.
RA6					See the OSC2/CLKO/RA6 pin.
RA7					See the OSC1/CLKI/RA7 pin.

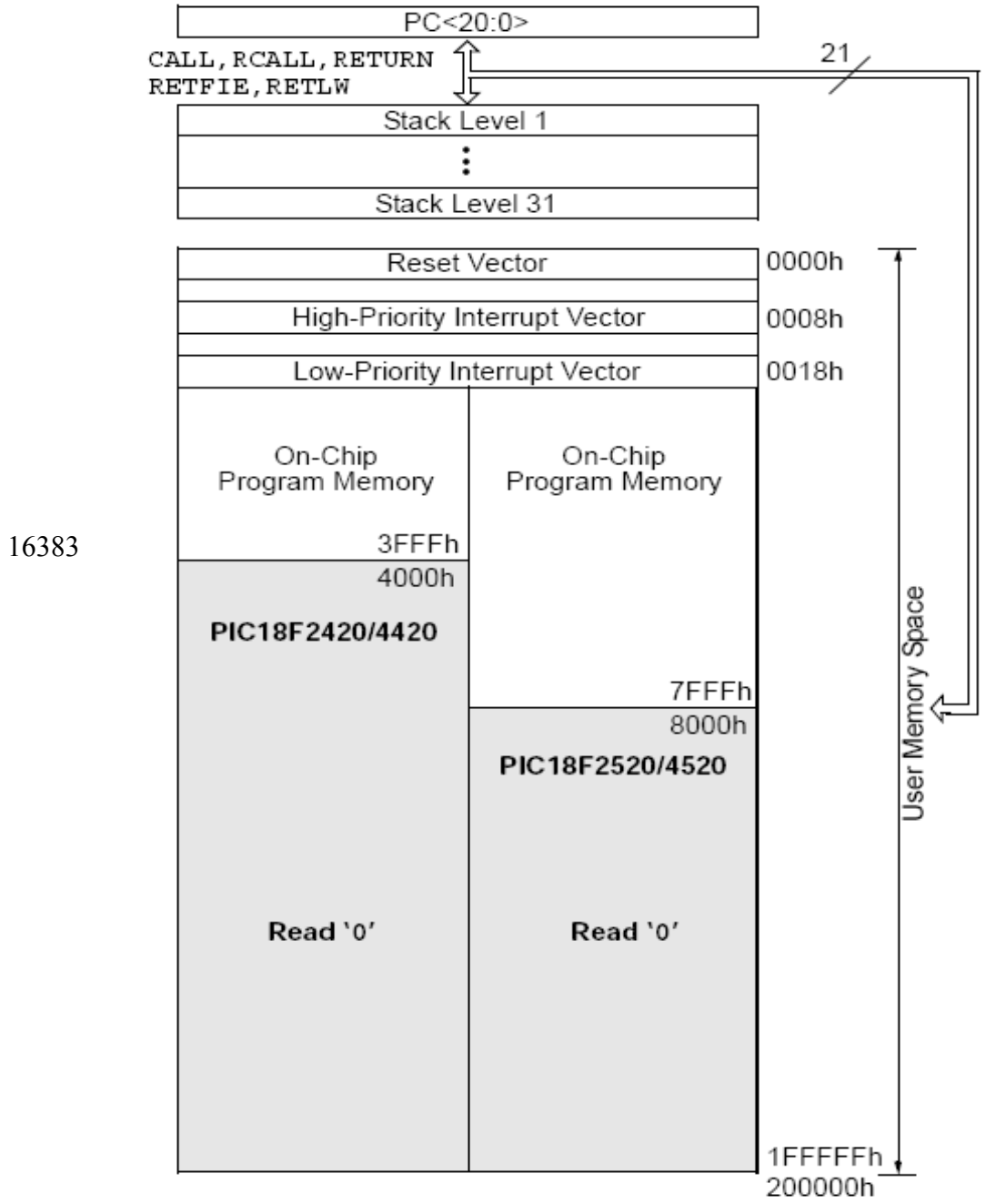
TABLE 1-2: PIC18F2420/2520 PINOUT I/O DESCRIPTIONS (CONTINUED)

Pin Name	Pin Number		Pin Type	Buffer Type	Description
	SPDIP, SOIC	QFN			
					PORTC is a bidirectional I/O port.
RC0/T1OSO/T13CKI	11	8			
RC0			I/O	ST	Digital I/O.
T1OSO			O	—	Timer1 oscillator output.
T13CKI			I	ST	Timer1/Timer3 external clock input.
RC1/T1OSI/CCP2	12	9			
RC1			I/O	ST	Digital I/O.
T1OSI			I	Analog	Timer1 oscillator input.
CCP2(2)			I/O	ST	Capture 2 input/Compare 2 output/PWM2 output.
RC2/CCP1	13	10			
RC2			I/O	ST	Digital I/O.
CCP1			I/O	ST	Capture 1 input/Compare 1 output/PWM1 output.
RC3/SCK/SCL	14	11			
RC3			I/O	ST	Digital I/O.
SCK			I/O	ST	Synchronous serial clock input/output for SPI mode.
SCL			I/O	ST	Synchronous serial clock input/output for I^2C™ mode.
RC4/SDI/SDA	15	12			
RC4			I/O	ST	Digital I/O.
SDI			I	ST	SPI data in.
SDA			I/O	ST	I^2C data I/O.
RC5/SDO	16	13			
RC5			I/O	ST	Digital I/O.
SDO			O	—	SPI data out.
RC6/TX/CK	17	14			
RC6			I/O	ST	Digital I/O.
TX			O	—	EUSART asynchronous transmit.
CK			I/O	ST	EUSART synchronous clock (see related RX/DT).
RC7/RX/DT	18	15			
RC7			I/O	ST	Digital I/O.
RX			I	ST	EUSART asynchronous receive.
DT			I/O	ST	EUSART synchronous data (see related TX/CK).
RE3	—	—	—	—	See $\overline{MCLR}$/VPP/RE3 pin.
VSS	8, 19	5, 16	P	—	Ground reference for logic and I/O pins.
VDD	20	17	P	—	Positive supply for logic and I/O pins.

Legend: TTL = TTL compatible input; CMOS = CMOS compatible input or output; ST = Schmitt Trigger input with CMOS levels; I = Input; O = Output; P = Power

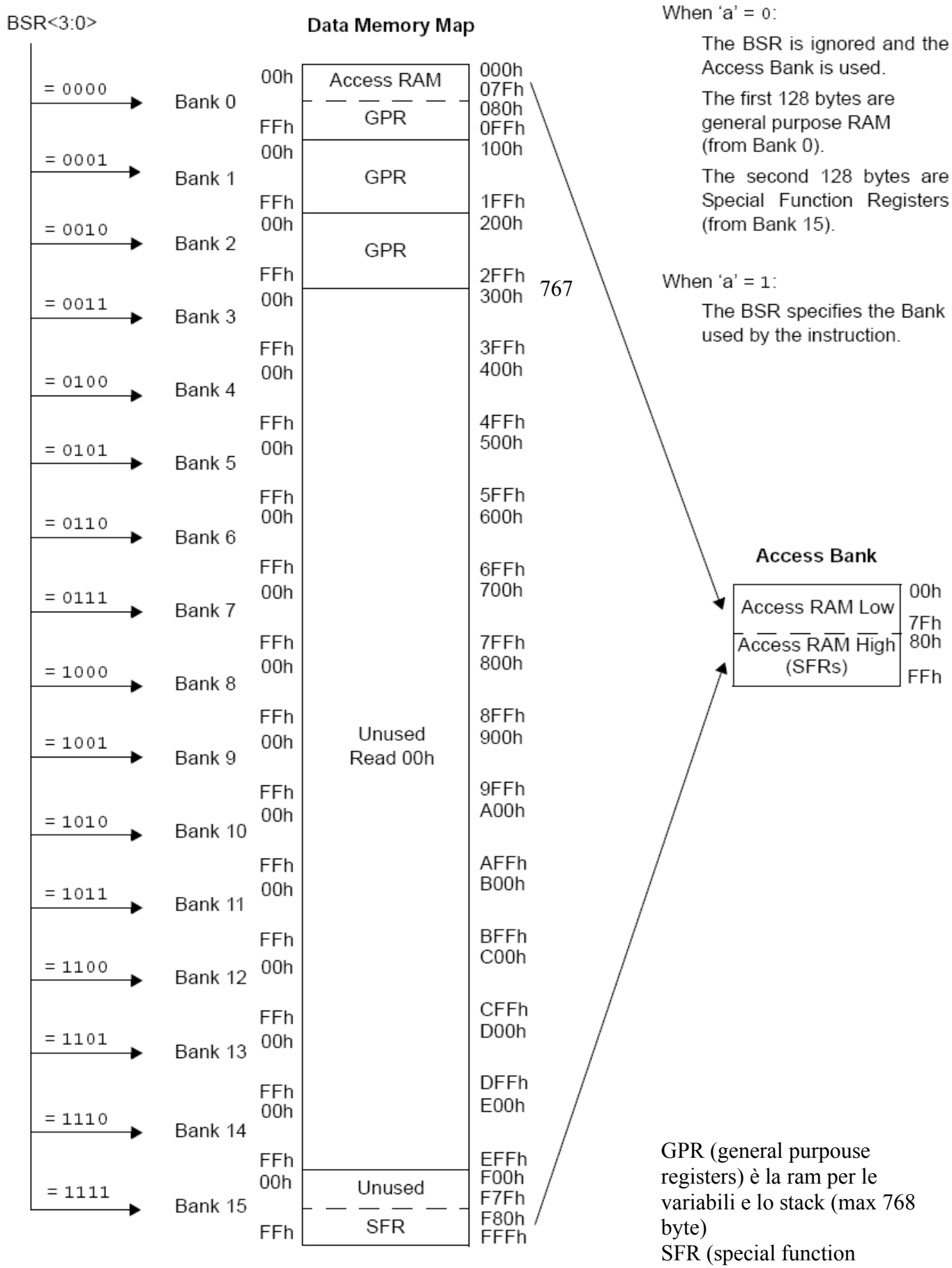
Note 1: Default assignment for CCP2 when Configuration bit, CCP2MX, is set.
2: Alternate assignment for CCP2 when Configuration bit, CCP2MX, is cleared.

PROGRAM MEMORY MAP AND STACK FOR PIC18F2420/2520/4420/4520 DEVICES



16383

FIGURE 5-5: DATA MEMORY MAP FOR PIC18F2420/4420 DEVICES



GPR (general purpouse registers) è la ram per le variabili e lo stack (max 768 byte)
SFR (special function registers) sono i registri dati, stato e controllo delle periferiche

Nelle seguenti tabelline sono riportati i valori di inizializzazione (per i vari reset) di tutti i registri delle periferiche (SFR)

TABLE 4-4: INITIALIZATION CONDITIONS FOR ALL REGISTERS

Register	Applicable Devices				Power-on Reset, Brown-out Reset	MCLR Resets, WDT Reset, RESET Instruction, Stack Resets	Wake-up via WDT or Interrupt
TOSU	2420	2520	4420	4520	---0 0000	---0 0000	---0 uuuu(3)
TOSH	2420	2520	4420	4520	0000 0000	0000 0000	uuuu uuuu(3)
TOSL	2420	2520	4420	4520	0000 0000	0000 0000	uuuu uuuu(3)
STKPTR	2420	2520	4420	4520	00-0 0000	uu-0 0000	uu-u uuuu(3)
PCLATU	2420	2520	4420	4520	---0 0000	---0 0000	---u uuuu
PCLATH	2420	2520	4420	4520	0000 0000	0000 0000	uuuu uuuu
PCL	2420	2520	4420	4520	0000 0000	0000 0000	PC + 2(2)
TBLPTRU	2420	2520	4420	4520	--00 0000	--00 0000	--uu uuuu
TBLPTRH	2420	2520	4420	4520	0000 0000	0000 0000	uuuu uuuu
TBLPTRL	2420	2520	4420	4520	0000 0000	0000 0000	uuuu uuuu
TABLAT	2420	2520	4420	4520	0000 0000	0000 0000	uuuu uuuu
PRODH	2420	2520	4420	4520	xxxx xxxx	uuuu uuuu	uuuu uuuu
PRODL	2420	2520	4420	4520	xxxx xxxx	uuuu uuuu	uuuu uuuu
INTCON	2420	2520	4420	4520	0000 000x	0000 000u	uuuu uuuu(1)
INTCON2	2420	2520	4420	4520	1111 -1-1	1111 -1-1	uuuu -u-u(1)
INTCON3	2420	2520	4420	4520	11-0 0-00	11-0 0-00	uu-u u-uu(1)
INDF0	2420	2520	4420	4520	N/A	N/A	N/A
POSTINC0	2420	2520	4420	4520	N/A	N/A	N/A
POSTDEC0	2420	2520	4420	4520	N/A	N/A	N/A
PREINC0	2420	2520	4420	4520	N/A	N/A	N/A
PLUSW0	2420	2520	4420	4520	N/A	N/A	N/A
FSR0H	2420	2520	4420	4520	---- 0000	---- 0000	---- uuuu
FSR0L	2420	2520	4420	4520	xxxx xxxx	uuuu uuuu	uuuu uuuu
WREG	2420	2520	4420	4520	xxxx xxxx	uuuu uuuu	uuuu uuuu
INDF1	2420	2520	4420	4520	N/A	N/A	N/A
POSTINC1	2420	2520	4420	4520	N/A	N/A	N/A
POSTDEC1	2420	2520	4420	4520	N/A	N/A	N/A
PREINC1	2420	2520	4420	4520	N/A	N/A	N/A
PLUSW1	2420	2520	4420	4520	N/A	N/A	N/A

Legend: u = unchanged, x = unknown, - = unimplemented bit, read as '0', q = value depends on condition.
Shaded cells indicate conditions do not apply for the designated device.

Note 1: One or more bits in the INTCONx or PIRx registers will be affected (to cause wake-up).

2: When the wake-up is due to an interrupt and the GIEL or GIEH bit is set, the PC is loaded with the interrupt vector (0008h or 0018h).

3: When the wake-up is due to an interrupt and the GIEL or GIEH bit is set, the TOSU, TOSH and TOSL are updated with the current value of the PC. The STKPTR is modified to point to the next location in the hardware stack.

4: See Table 4-3 for Reset value for specific condition.

5: Bits 6 and 7 of PORTA, LATA and TRISA are enabled, depending on the oscillator mode selected. When not enabled as PORTA pins, they are disabled and read '0'.

TABLE 4-4: INITIALIZATION CONDITIONS FOR ALL REGISTERS (CONTINUED)

Register	Applicable Devices				Power-on Reset, Brown-out Reset	$\overline{\text{MCLR}}$ Resets, WDT Reset, RESET Instruction, Stack Resets	Wake-up via WDT or Interrupt
FSR1H	2420	2520	4420	4520	---- 0000	---- 0000	---- uuuu
FSR1L	2420	2520	4420	4520	xxxx xxxx	uuuu uuuu	uuuu uuuu
BSR	2420	2520	4420	4520	---- 0000	---- 0000	---- uuuu
INDF2	2420	2520	4420	4520	N/A	N/A	N/A
POSTINC2	2420	2520	4420	4520	N/A	N/A	N/A
POSTDEC2	2420	2520	4420	4520	N/A	N/A	N/A
PREINC2	2420	2520	4420	4520	N/A	N/A	N/A
PLUSW2	2420	2520	4420	4520	N/A	N/A	N/A
FSR2H	2420	2520	4420	4520	---- 0000	---- 0000	---- uuuu
FSR2L	2420	2520	4420	4520	xxxx xxxx	uuuu uuuu	uuuu uuuu
STATUS	2420	2520	4420	4520	---x xxxx	---u uuuu	---u uuuu
TMR0H	2420	2520	4420	4520	0000 0000	0000 0000	uuuu uuuu
TMR0L	2420	2520	4420	4520	xxxx xxxx	uuuu uuuu	uuuu uuuu
T0CON	2420	2520	4420	4520	1111 1111	1111 1111	uuuu uuuu
OSCCON	2420	2520	4420	4520	0100 q000	0100 q000	uuuu uuqu
HLVDCON	2420	2520	4420	4520	0-00 0101	0-00 0101	u-uu uuuu
WDTCON	2420	2520	4420	4520	---- ---0	---- ---0	---- ---u
RCON(4)	2420	2520	4420	4520	0q-1 11q0	0q-q qquu	uq-u qquu
TMR1H	2420	2520	4420	4520	xxxx xxxx	uuuu uuuu	uuuu uuuu
TMR1L	2420	2520	4420	4520	xxxx xxxx	uuuu uuuu	uuuu uuuu
T1CON	2420	2520	4420	4520	0000 0000	u0uu uuuu	uuuu uuuu
TMR2	2420	2520	4420	4520	0000 0000	0000 0000	uuuu uuuu
PR2	2420	2520	4420	4520	1111 1111	1111 1111	1111 1111
T2CON	2420	2520	4420	4520	-000 0000	-000 0000	-uuu uuuu
SSPBUF	2420	2520	4420	4520	xxxx xxxx	uuuu uuuu	uuuu uuuu
SSPADD	2420	2520	4420	4520	0000 0000	0000 0000	uuuu uuuu
SSPSTAT	2420	2520	4420	4520	0000 0000	0000 0000	uuuu uuuu
SSPCON1	2420	2520	4420	4520	0000 0000	0000 0000	uuuu uuuu
SSPCON2	2420	2520	4420	4520	0000 0000	0000 0000	uuuu uuuu

TABLE 4-4: INITIALIZATION CONDITIONS FOR ALL REGISTERS (CONTINUED)

Register	Applicable Devices				Power-on Reset, Brown-out Reset	$\overline{\text{MCLR}}$ Resets, WDT Reset, RESET Instruction, Stack Resets	Wake-up via WDT or Interrupt
ADRESH	2420	2520	4420	4520	xxxx xxxx	uuuu uuuu	uuuu uuuu
ADRESL	2420	2520	4420	4520	xxxx xxxx	uuuu uuuu	uuuu uuuu
ADCON0	2420	2520	4420	4520	--00 0000	--00 0000	--uu uuuu
ADCON1	2420	2520	4420	4520	--00 0qqq	--00 0qqq	--uu uuuu
ADCON2	2420	2520	4420	4520	0-00 0000	0-00 0000	u-uu uuuu
CCPR1H	2420	2520	4420	4520	xxxx xxxx	uuuu uuuu	uuuu uuuu
CCPR1L	2420	2520	4420	4520	xxxx xxxx	uuuu uuuu	uuuu uuuu
CCP1CON	2420	2520	4420	4520	0000 0000	0000 0000	uuuu uuuu
	2420	2520	4420	4520	--00 0000	--00 0000	--uu uuuu
CCPR2H	2420	2520	4420	4520	xxxx xxxx	uuuu uuuu	uuuu uuuu
CCPR2L	2420	2520	4420	4520	xxxx xxxx	uuuu uuuu	uuuu uuuu
CCP2CON	2420	2520	4420	4520	--00 0000	--00 0000	--uu uuuu
BAUDCON	2420	2520	4420	4520	01-0 0-00	01-0 0-00	--uu uuuu
PWM1CON	2420	2520	4420	4520	0000 0000	0000 0000	uuuu uuuu
ECCP1AS	2420	2520	4420	4520	0000 0000	0000 0000	uuuu uuuu
	2420	2520	4420	4520	0000 00--	0000 00--	uuuu uu--
CVRCON	2420	2520	4420	4520	0000 0000	0000 0000	uuuu uuuu
CMCON	2420	2520	4420	4520	0000 0111	0000 0111	uuuu uuuu
TMR3H	2420	2520	4420	4520	xxxx xxxx	uuuu uuuu	uuuu uuuu
TMR3L	2420	2520	4420	4520	xxxx xxxx	uuuu uuuu	uuuu uuuu
T3CON	2420	2520	4420	4520	0000 0000	uuuu uuuu	uuuu uuuu
SPBRGH	2420	2520	4420	4520	0000 0000	0000 0000	uuuu uuuu
SPBRG	2420	2520	4420	4520	0000 0000	0000 0000	uuuu uuuu
RCREG	2420	2520	4420	4520	0000 0000	0000 0000	uuuu uuuu
TXREG	2420	2520	4420	4520	0000 0000	0000 0000	uuuu uuuu
TXSTA	2420	2520	4420	4520	0000 0010	0000 0010	uuuu uuuu
RCSTA	2420	2520	4420	4520	0000 000x	0000 000x	uuuu uuuu
EEADR	2420	2520	4420	4520	0000 0000	0000 0000	uuuu uuuu
EEDATA	2420	2520	4420	4520	0000 0000	0000 0000	uuuu uuuu
EECON2	2420	2520	4420	4520	0000 0000	0000 0000	0000 0000
EECON1	2420	2520	4420	4520	xx-0 x000	uu-0 u000	uu-0 u000

TABLE 4-4: INITIALIZATION CONDITIONS FOR ALL REGISTERS (CONTINUED)

Register	Applicable Devices				Power-on Reset, Brown-out Reset	$\overline{\text{MCLR}}$ Resets, WDT Reset, RESET Instruction, Stack Resets	Wake-up via WDT or Interrupt
IPR2	2420	2520	4420	4520	11-1 1111	11-1 1111	uu-u uuuu
PIR2	2420	2520	4420	4520	00-0 0000	00-0 0000	uu-u uuuu(1)
PIE2	2420	2520	4420	4520	00-0 0000	00-0 0000	uu-u uuuu
IPR1	2420	2520	4420	4520	1111 1111	1111 1111	uuuu uuuu
	2420	2520	4420	4520	-111 1111	-111 1111	-uuu uuuu
PIR1	2420	2520	4420	4520	0000 0000	0000 0000	uuuu uuuu(1)
	2420	2520	4420	4520	-000 0000	-000 0000	-uuu uuuu(1)
PIE1	2420	2520	4420	4520	0000 0000	0000 0000	uuuu uuuu
	2420	2520	4420	4520	-000 0000	-000 0000	-uuu uuuu
OSCTUNE	2420	2520	4420	4520	00-0 0000	00-0 0000	uu-u uuuu
TRISE	2420	2520	4420	4520	0000 -111	0000 -111	uuuu -uuu
TRISD	2420	2520	4420	4520	1111 1111	1111 1111	uuuu uuuu
TRISC	2420	2520	4420	4520	1111 1111	1111 1111	uuuu uuuu
TRISB	2420	2520	4420	4520	1111 1111	1111 1111	uuuu uuuu
TRISA(5)	2420	2520	4420	4520	1111 1111(5)	1111 1111(5)	uuuu uuuu(5)
LATE	2420	2520	4420	4520	---- -xxx	---- -uuu	---- -uuu
LATD	2420	2520	4420	4520	xxxx xxxx	uuuu uuuu	uuuu uuuu
LATC	2420	2520	4420	4520	xxxx xxxx	uuuu uuuu	uuuu uuuu
LATB	2420	2520	4420	4520	xxxx xxxx	uuuu uuuu	uuuu uuuu
LATA(5)	2420	2520	4420	4520	xxxx xxxx(5)	uuuu uuuu(5)	uuuu uuuu(5)
PORTE	2420	2520	4420	4520	---- xxxx	---- uuuu	---- uuuu
PORTD	2420	2520	4420	4520	xxxx xxxx	uuuu uuuu	uuuu uuuu
PORTC	2420	2520	4420	4520	xxxx xxxx	uuuu uuuu	uuuu uuuu
PORTB	2420	2520	4420	4520	xxxx xxxx	uuuu uuuu	uuuu uuuu
PORTA(5)	2420	2520	4420	4520	xx0x 0000(5)	uu0u 0000(5)	uuuu uuuu(5)

In questa tabella sono riportati tutti gli indirizzi dei vari registri delle periferich (SFR). Questi indirizzi sono utilizzati nel file 18Fxxx.h che viene incluso all'inizio del codice C di un programma. Esso contiene delle definizioni del tipo: #define PORTA *(unsigned char *)0x0F80. In tal modo possiamo operare con dei simboli invece che con dei numeri e scrivere PORTA = valore invece di *(unsigned char *)0x0F80 = valore;

TABLE 5-1: SPECIAL FUNCTION REGISTER MAP FOR PIC18F2420/2520/4420/4520 DEVICES

Address	Name	Address	Name	Address	Name	Address	Name
FFFh	TOSU	FDFh	INDF2(1)	FBFh	CCPR1H	F9Fh	IPR1
FFEh	TOSH	FDEh	POSTINC2(1)	FBEh	CCPR1L	F9Eh	PIR1
FFDh	TOSL	FDDh	POSTDEC2(1)	FBDh	CCP1CON	F9Dh	PIE1
FFCh	STKPTR	FDCh	PREINC2(1)	FBCh	CCPR2H	F9Ch	—(2)
FFBh	PCLATU	FDBh	PLUSW2(1)	FBBh	CCPR2L	F9Bh	OSCTUNE
FFAh	PCLATH	FDAh	FSR2H	FBAh	CCP2CON	F9Ah	—(2)
FF9h	PCL	FD9h	FSR2L	FB9h	—(2)	F99h	—(2)
FF8h	TBLPTRU	FD8h	STATUS	FB8h	BAUDCON	F98h	—(2)
FF7h	TBLPTRH	FD7h	TMR0H	FB7h	PWM1CON(3)	F97h	—(2)
FF6h	TBLPTRL	FD6h	TMR0L	FB6h	ECCP1AS(3)	F96h	TRISE(3)
FF5h	TABLAT	FD5h	T0CON	FB5h	CVRCON	F95h	TRISD(3)
FF4h	PRODH	FD4h	—(2)	FB4h	CMCON	F94h	TRISC
FF3h	PRODL	FD3h	OSCCON	FB3h	TMR3H	F93h	TRISB
FF2h	INTCON	FD2h	HLVDCON	FB2h	TMR3L	F92h	TRISA
FF1h	INTCON2	FD1h	WDTCON	FB1h	T3CON	F91h	—(2)
FF0h	INTCON3	FD0h	RCON	FB0h	SPBRGH	F90h	—(2)
FEFh	INDF0(1)	FCFh	TMR1H	FAFh	SPBRG	F8Fh	—(2)
FEEh	POSTINC0(1)	FCEh	TMR1L	FAEh	RCREG	F8Eh	—(2)
FEDh	POSTDEC0(1)	FCDh	T1CON	FADh	TXREG	F8Dh	LATE(3)
FECh	PREINC0(1)	FCCh	TMR2	FACh	TXSTA	F8Ch	LATD(3)
FEBh	PLUSW0(1)	FCBh	PR2	FABh	RCSTA	F8Bh	LATC
FEAh	FSR0H	FCAh	T2CON	FAAh	—(2)	F8Ah	LATB
FE9h	FSR0L	FC9h	SSPBUF	FA9h	EEADR	F89h	LATA
FE8h	WREG	FC8h	SSPADD	FA8h	EEDATA	F88h	—(2)
FE7h	INDF1(1)	FC7h	SSPSTAT	FA7h	EECON2(1)	F87h	—(2)
FE6h	POSTINC1(1)	FC6h	SSPCON1	FA6h	EECON1	F86h	—(2)
FE5h	POSTDEC1(1)	FC5h	SSPCON2	FA5h	—(2)	F85h	—(2)
FE4h	PREINC1(1)	FC4h	ADRESH	FA4h	—(2)	F84h	PORTE(3)
FE3h	PLUSW1(1)	FC3h	ADRESL	FA3h	—(2)	F83h	PORTD(3)
FE2h	FSR1H	FC2h	ADCON0	FA2h	IPR2	F82h	PORTC
FE1h	FSR1L	FC1h	ADCON1	FA1h	PIR2	F81h	PORTB
FE0h	BSR	FC0h	ADCON2	FA0h	PIE2	F80h	PORTA

NELLA TABELLA DI SEGUITO SONO RIPORTATI I FUSIBILI DI CONFIGURAZIONE.

Il loro valore viene impostato in sede di programmazione (quando si scarica il programma), mediante il software di comando del programmatore collegato al PC, e non a tempo di esecuzione del programma. E' anche possibile, mediante le direttive #pragma fare in modo che il compilatore stesso inserisca nel codice die comandi per il programmatore (la macchinetta) affinchè esso bruci i fusibili voluti. Una volta che questi sono modificati, per alterarli nuovamente occorre riprogrammare il microcontrollore, ripetiamo: non si può fare a run time, tranne casi particolari. Inoltre bisogna prestare molta attenzione al fatto che alcuni fusibili possono impedire ulteriori programmazioni; il loro scopo è di protezione antipiracy.

In genere, per quasi tutti i fusibili i valori di default possono andare bene per le nostre applicazioni

Oscillator Selection bits:

OSC = LP	LP oscillator
OSC = XT	XT oscillator
OSC = HS	HS oscillator
OSC = RC	External RC oscillator, CLKO function on RA6
OSC = EC	EC oscillator, CLKO function on RA6
OSC = ECIO6	EC oscillator, port function on RA6
OSC = HSPLL	HS oscillator, PLL enabled (Clock Frequency = 4 x FOSC1)
OSC = RCIO6	External RC oscillator, port function on RA6
OSC = INTIO67	Internal oscillator block, port function on RA6 and RA7
OSC = INTIO7	Internal oscillator block, CLKO function on RA6, port function on RA7

Fail-Safe Clock Monitor Enable bit:

FCMEN = OFF	Fail-Safe Clock Monitor disabled
FCMEN = ON	Fail-Safe Clock Monitor enabled

Internal/External Oscillator Switchover bit:

IESO = OFF	Oscillator Switchover mode disabled
IESO = ON	Oscillator Switchover mode enabled

Power-up Timer Enable bit:

PWRT = ON	PWRT enabled
PWRT = OFF	PWRT disabled

Brown-out Reset Enable bits:

BOREN = OFF	Brown-out Reset disabled in hardware and software
BOREN = ON	Brown-out Reset enabled and controlled by software (SBOREN is enabled)
BOREN = NOSLP	Brown-out Reset enabled in hardware only and disabled in Sleep mode (SBOREN is disabled)
BOREN = SBORDIS	Brown-out Reset enabled in hardware only (SBOREN is disabled)

Brown-out Reset Voltage bits:

BORV = 0	Maximum setting
BORV = 1	
BORV = 2	
BORV = 3	Minimum setting

Watchdog Timer Enable bit:

WDT = OFF	WDT disabled (control is placed on the SWDTEN bit)
WDT = ON	WDT enabled

Watchdog Timer Postscale Select bits:

WDTPS = 1	1:1
WDTPS = 2	1:2
WDTPS = 4	1:4
WDTPS = 8	1:8
WDTPS = 16	1:16
WDTPS = 32	1:32
WDTPS = 64	1:64
WDTPS = 128	1:128
WDTPS = 256	1:256
WDTPS = 512	1:512
WDTPS = 1024	1:1024
WDTPS = 2048	1:2048
WDTPS = 4096	1:4096
WDTPS = 8192	1:8192
WDTPS = 16384	1:16384
WDTPS = 32768	1:32768

MCLR Pin Enable bit:

MCLRE = OFF	RE3 input pin enabled; MCLR disabled
MCLRE = ON	MCLR pin enabled; RE3 input pin disabled

Low-Power Timer1 Oscillator Enable bit:

LPT1OSC = OFF	Timer1 configured for higher power operation
LPT1OSC = ON	Timer1 configured for low-power operation

PORTB A/D Enable bit:

PBADEN = OFF	PORTB<4:0> pins are configured as digital I/O on Reset
PBADEN = ON	PORTB<4:0> pins are configured as analog input channels on Reset

CCP2 MUX bit:

CCP2MX = PORTBE	CCP2 input/output is multiplexed with RB3
CCP2MX = PORTC	CCP2 input/output is multiplexed with RC1

Stack Full/Underflow Reset Enable bit:

STVREN = OFF	Stack full/underflow will not cause Reset
STVREN = ON	Stack full/underflow will cause Reset

Single-Supply ICSP Enable bit:

LVP = OFF	Single-Supply ICSP disabled
LVP = ON	Single-Supply ICSP enabled

Extended Instruction Set Enable bit:

XINST = OFF	Instruction set extension and Indexed Addressing mode disabled (Legacy mode)
XINST = ON	Instruction set extension and Indexed Addressing mode enabled

Background Debugger Enable bit:

DEBUG = ON	Background debugger enabled, RB6 and RB7 are dedicated to In-Circuit Debug
DEBUG = OFF	Background debugger disabled, RB6 and RB7 configured as general purpose I/O pins

Code Protection bit Block 0:

CP0 = ON	Block 0 (000800-001FFFh) code-protected
CP0 = OFF	Block 0 (000800-001FFFh) not code-protected

Code Protection bit Block 1:

CP1 = ON	Block 1 (002000-003FFFh) code-protected
CP1 = OFF	Block 1 (002000-003FFFh) not code-protected

Boot Block Code Protection bit:

CPB = ON	Boot block (000000-0007FFh) code-protected
CPB = OFF	Boot block (000000-0007FFh) not code-protected

Data EEPROM Code Protection bit:

CPD = ON	Data EEPROM code-protected
CPD = OFF	Data EEPROM not code-protected

Write Protection bit Block 0:

WRT0 = ON	Block 0 (000800-001FFFh) write-protected
WRT0 = OFF	Block 0 (000800-001FFFh) not write-protected

Write Protection bit Block 1:

WRT1 = ON	Block 1 (002000-003FFFh) write-protected
WRT1 = OFF	Block 1 (002000-003FFFh) not write-protected

Boot Block Write Protection bit:

WRTB = ON	Boot block (000000-0007FFh) write-protected
WRTB = OFF	Boot block (000000-0007FFh) not write-protected

Configuration Register Write Protection bit:

WRTC = ON	Configuration registers (300000-3000FFh) write-protected
WRTC = OFF	Configuration registers (300000-3000FFh) not write-protected

Data EEPROM Write Protection bit:

WRTD = ON	Data EEPROM write-protected
WRTD = OFF	Data EEPROM not write-protected

Table Read Protection bit Block 0:

EBTR0 = ON	Block 0 (000800-001FFFh) protected from table reads executed in other blocks
EBTR0 = OFF	Block 0 (000800-001FFFh) not protected from table reads executed in other blocks

Table Read Protection bit Block 1:

EBTR1 = ON	Block 1 (002000-003FFFh) protected from table reads executed in other blocks
EBTR1 = OFF	Block 1 (002000-003FFFh) not protected from table reads executed in other blocks

Boot Block Table Read Protection bit:

EBTRB = ON	Boot block (000000-0007FFh) protected from table reads executed in other blocks
EBTRB = OFF	Boot block (000000-0007FFh) not protected from table reads executed in other blocks

Le porte di I/O

Ogni porta di i/o (di nome A,B,C,D sul 18F2420) ha tre registri associati: LAT PORT TRIS; il bit y di TRISX stabilisce se il bit y della porta X deve essere di input (1) o di output (0). Nel caso il bit y della porta X sia definito come ingresso il valore dell'ingresso si troverà nel bit y di PORTX; nel caso sia definito come uscita il bit y della porta X verrà impostato scrivendo zero o uno nel bit y di PORTX. Come abbiamo visto molti pin hanno più di una funzione; è possibile stabilire la funzione con appositi registri di controllo e di stato. In realtà è anche presente il bit 3 della porta E multiplexato con MCLR ma non verrà utilizzato per i nostri scopi. Occorre prestare attenzione a abilitare o meno la seconda funzione. A differenza di quasi tutti gli altri microcontrollori

TABLE 10-1: PORTA I/O SUMMARY

Pin	Function	TRIS Setting	I/O	I/O Type	Description
RA0/AN0	RA0	0	O	DIG	LATA<0> data output; not affected by analog input.
		1	I	TTL	PORTA<0> data input; disabled when analog input enabled.
	AN0	1	I	ANA	A/D input channel 0 and comparator C1- input. Default input configuration on POR; does not affect digital output.
RA1/AN1	RA1	0	O	DIG	LATA<1> data output; not affected by analog input.
		1	I	TTL	PORTA<1> data input; disabled when analog input enabled.
	AN1	1	I	ANA	A/D input channel 1 and comparator C2- input. Default input configuration on POR; does not affect digital output.
RA2/AN2/ VREF-/CVREF	RA2	0	O	DIG	LATA<2> data output; not affected by analog input. Disabled when CVREF output enabled.
		1	I	TTL	PORTA<2> data input. Disabled when analog functions enabled; disabled when CVREF output enabled.
	AN2	1	I	ANA	A/D input channel 2 and comparator C2+ input. Default input configuration on POR; not affected by analog output.
	VREF-	1	I	ANA	A/D and comparator voltage reference low input.
	CVREF	x	O	ANA	Comparator voltage reference output. Enabling this feature disables digital I/O.
RA3/AN3/VREF+	RA3	0	O	DIG	LATA<3> data output; not affected by analog input.
		1	I	TTL	PORTA<3> data input; disabled when analog input enabled.
	AN3	1	I	ANA	A/D input channel 3 and comparator C1+ input. Default input configuration on POR.
	VREF+	1	I	ANA	A/D and comparator voltage reference high input.
RA4/T0CKI/C1OUT	RA4	0	O	DIG	LATA<4> data output.
		1	I	ST	PORTA<4> data input; default configuration on POR.
	T0CKI	1	I	ST	Timer0 clock input.
	C1OUT	0	O	DIG	Comparator 1 output; takes priority over port data.
RA5/AN4/$\overline{SS}$/ HLVDIN/C2OUT	RA5	0	O	DIG	LATA<5> data output; not affected by analog input.
		1	I	TTL	PORTA<5> data input; disabled when analog input enabled.
	AN4	1	I	ANA	A/D input channel 4. Default configuration on POR.
	$\overline{SS}$	1	I	TTL	Slave select input for MSSP module.
	HLVDIN	1	I	ANA	High/Low-Voltage Detect external trip point input.
	C2OUT	0	O	DIG	Comparator 2 output; takes priority over port data.
OSC2/CLKO/RA6	RA6	0	O	DIG	LATA<6> data output. Enabled in RCIO, INTIO2 and ECIO modes only.
		1	I	TTL	PORTA<6> data input. Enabled in RCIO, INTIO2 and ECIO modes only.
	OSC2	x	O	ANA	Main oscillator feedback output connection (XT, HS and LP modes).
	CLKO	x	O	DIG	System cycle clock output (FOSC/4) in RC, INTIO1 and EC Oscillator modes.
OSC1/CLKI/RA7	RA7	0	O	DIG	LATA<7> data output. Disabled in external oscillator modes.
		1	I	TTL	PORTA<7> data input. Disabled in external oscillator modes.
	OSC1	x	I	ANA	Main oscillator input connection.
	CLKI	x	I	ANA	Main clock input connection.

Legend: DIG = Digital level output; TTL = TTL input buffer; ST = Schmitt Trigger input buffer; ANA = Analog level input/output; x = Don't care (TRIS bit does not affect port direction or is overridden for this option).

molte seconde funzioni risultano abilitate di default.

TABLE 10-2: SUMMARY OF REGISTERS ASSOCIATED WITH PORTA

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Reset Values on page
PORTA	RA7(1)	RA6(1)	RA5	RA4	RA3	RA2	RA1	RA0	52
LATA	LATA7(1)	LATA6(1)	PORTA Data Latch Register (Read and Write to Data Latch)						52
TRISA	TRISA7(1)	TRISA6(1)	PORTA Data Direction Register						52
ADCON1	—	—	VCFG1	VCFG0	PCFG3	PCFG2	PCFG1	PCFG0	51
CMCON	C2OUT	C1OUT	C2INV	C1INV	CIS	CM2	CM1	CM0	51
CVRCON	CVREN	CVROE	CVRR	CVRSS	CVR3	CVR2	CVR1	CVR0	51

Legend: — = unimplemented, read as '0'. Shaded cells are not used by PORTA.

Note 1: RA<7:6> and their associated latch and data direction bits are enabled as I/O pins based on oscillator configuration; otherwise, they are read as '0'.

TABLE 10-3: PORTB I/O SUMMARY

Pin	Function	TRIS Setting	I/O	I/O Type	Description
RB0/INT0/FLT0/ AN12	RB0	0	O	DIG	LATB<0> data output; not affected by analog input.
		1	I	TTL	PORTB<0> data input; weak pull-up when $\overline{RBPU}$ bit is cleared. Disabled when analog input enabled.(1)
	INT0	1	I	ST	External interrupt 0 input.
	FLT0	1	I	ST	Enhanced PWM Fault input (ECCP1 module); enabled in software.
	AN12	1	I	ANA	A/D input channel 12.(1)
RB1/INT1/AN10	RB1	0	O	DIG	LATB<1> data output; not affected by analog input.
		1	I	TTL	PORTB<1> data input; weak pull-up when $\overline{RBPU}$ bit is cleared. Disabled when analog input enabled.(1)
	INT1	1	I	ST	External Interrupt 1 input.
	AN10	1	I	ANA	A/D input channel 10.(1)
RB2/INT2/AN8	RB2	0	O	DIG	LATB<2> data output; not affected by analog input.
		1	I	TTL	PORTB<2> data input; weak pull-up when $\overline{RBPU}$ bit is cleared. Disabled when analog input enabled.(1)
	INT2	1	I	ST	External interrupt 2 input.
	AN8	1	I	ANA	A/D input channel 8.(1)
RB3/AN9/CCP2	RB3	0	O	DIG	LATB<3> data output; not affected by analog input.
		1	I	TTL	PORTB<3> data input; weak pull-up when $\overline{RBPU}$ bit is cleared. Disabled when analog input enabled.(1)
	AN9	1	I	ANA	A/D input channel 9.(1)
	CCP2(2)	0	O	DIG	CCP2 compare and PWM output.
		1	I	ST	CCP2 capture input
RB4/KBI0/AN11	RB4	0	O	DIG	LATB<4> data output; not affected by analog input.
		1	I	TTL	PORTB<4> data input; weak pull-up when $\overline{RBPU}$ bit is cleared. Disabled when analog input enabled.(1)
	KBI0	1	I	TTL	Interrupt-on-pin change.
	AN11	1	I	ANA	A/D input channel 11.(1)
RB5/KBI1/PGM	RB5	0	O	DIG	LATB<5> data output.
		1	I	TTL	PORTB<5> data input; weak pull-up when $\overline{RBPU}$ bit is cleared.
	KBI1	1	I	TTL	Interrupt-on-pin change.
	PGM	x	I	ST	Single-Supply In-Circuit Serial Programming™ mode entry (ICSP™). Enabled by LVP Configuration bit; all other pin functions disabled.
RB6/KBI2/PGC	RB6	0	O	DIG	LATB<6> data output.
		1	I	TTL	PORTB<6> data input; weak pull-up when $\overline{RBPU}$ bit is cleared.
	KBI2	1	I	TTL	Interrupt-on-pin change.
	PGC	x	I	ST	Serial execution (ICSP) clock input for ICSP and ICD operation.(3)
RB7/KBI3/PGD	RB7	0	O	DIG	LATB<7> data output.
		1	I	TTL	PORTB<7> data input; weak pull-up when $\overline{RBPU}$ bit is cleared.
	KBI3	1	I	TTL	Interrupt-on-pin change.
	PGD	x	O	DIG	Serial execution data output for ICSP and ICD operation.(3)
		x	I	ST	Serial execution data input for ICSP and ICD operation.(3)

Legend: DIG = Digital level output; TTL = TTL input buffer; ST = Schmitt Trigger input buffer; ANA = Analog level input/output; x = Don't care (TRIS bit does not affect port direction or is overridden for this option).

Note 1: Configuration on POR is determined by the PBADEN Configuration bit. Pins are configured as analog inputs by default when PBADEN is set and digital inputs when PBADEN is cleared.

2: Alternate assignment for CCP2 when the CCP2MX Configuration bit is '0'. Default assignment is RC1.

3: All other pin functions are disabled when ICSP or ICD are enabled.

TABLE 10-4: SUMMARY OF REGISTERS ASSOCIATED WITH PORTB

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Reset Values on page
PORTB	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0	52
LATB	PORTB Data Latch Register (Read and Write to Data Latch)								52
TRISB	PORTB Data Direction Register								52
INTCON	GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF	49
INTCON2	$\overline{\text{RBPU}}$	INTEDG0	INTEDG1	INTEDG2	—	TMR0IP	—	RBIP	49
INTCON3	INT2IP	INT1IP	—	INT2IE	INT1IE	—	INT2IF	INT1IF	49
ADCON1	—	—	VCFG1	VCFG0	PCFG3	PCFG2	PCFG1	PCFG0	51

Legend: — = unimplemented, read as '0'. Shaded cells are not used by PORTB.

TABLE 10-5: PORTC I/O SUMMARY

Pin	Function	TRIS Setting	I/O	I/O Type	Description
RC0/T1OSO/ T13CKI	RC0	0	O	DIG	LATC<0> data output.
		1	I	ST	PORTC<0> data input.
	T1OSO	x	O	ANA	Timer1 oscillator output; enabled when Timer1 oscillator enabled. Disables digital I/O.
	T13CKI	1	I	ST	Timer1/Timer3 counter input.
RC1/T1OSI/CCP2	RC1	0	O	DIG	LATC<1> data output.
		1	I	ST	PORTC<1> data input.
	T1OSI	x	I	ANA	Timer1 oscillator input; enabled when Timer1 oscillator enabled. Disables digital I/O.
	CCP2(1)	0	O	DIG	CCP2 compare and PWM output; takes priority over port data.
		1	I	ST	CCP2 capture input.
RC2/CCP1/P1A	RC2	0	O	DIG	LATC<2> data output.
		1	I	ST	PORTC<2> data input.
	CCP1	0	O	DIG	ECCP1 compare or PWM output; takes priority over port data.
		1	I	ST	ECCP1 capture input.
	P1A(2)	0	O	DIG	ECCP1 Enhanced PWM output, channel A. May be configured for tri-state during Enhanced PWM shutdown events. Takes priority over port data.
RC3/SCK/SCL	RC3	0	O	DIG	LATC<3> data output.
		1	I	ST	PORTC<3> data input.
	SCK	0	O	DIG	SPI clock output (MSSP module); takes priority over port data.
		1	I	ST	SPI clock input (MSSP module).
	SCL	0	O	DIG	I^2C™ clock output (MSSP module); takes priority over port data.
		1	I	I^2C/SMB	I^2C clock input (MSSP module); input type depends on module setting.
RC4/SDI/SDA	RC4	0	O	DIG	LATC<4> data output.
		1	I	ST	PORTC<4> data input.
	SDI	1	I	ST	SPI data input (MSSP module).
	SDA	1	O	DIG	I^2C data output (MSSP module); takes priority over port data.
		1	I	I^2C/SMB	I^2C data input (MSSP module); input type depends on module setting.
RC5/SDO	RC5	0	O	DIG	LATC<5> data output.
		1	I	ST	PORTC<5> data input.
	SDO	0	O	DIG	SPI data output (MSSP module); takes priority over port data.
RC6/TX/CK	RC6	0	O	DIG	LATC<6> data output.
		1	I	ST	PORTC<6> data input.
	TX	1	O	DIG	Asynchronous serial transmit data output (EUSART module); takes priority over port data. User must configure as output.
	CK	1	O	DIG	Synchronous serial clock output (EUSART module); takes priority over port data.
		1	I	ST	Synchronous serial clock input (EUSART module)
RC7/RX/DT	RC7	0	O	DIG	LATC<7> data output.
		1	I	ST	PORTC<7> data input.
	RX	1	I	ST	Asynchronous serial receive data input (EUSART module).
	DT	1	O	DIG	Synchronous serial data output (EUSART module); takes priority over port data.
		1	I	ST	Synchronous serial data input (EUSART module). User must configure as an input.

Legend: DIG = Digital level output; TTL = TTL input buffer; ST = Schmitt Trigger input buffer; ANA = Analog level input/output; I^2C/SMB = I^2C/SMBus input buffer; x = Don't care (TRIS bit does not affect port direction or is overridden for this option).

Note 1: Default assignment for CCP2 when the CCP2MX Configuration bit is set. Alternate assignment is RB3.

2: Enhanced PWM output is available only on PIC18F4520 devices.

TABLE 10-6: SUMMARY OF REGISTERS ASSOCIATED WITH PORTC

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Reset Values on page
PORTC	RC7	RC6	RC5	RC4	RC3	RC2	RC1	RC0	52
LATC	PORTC Data Latch Register (Read and Write to Data Latch)								52
TRISC	PORTC Data Direction Register								52

TABLE 10-7: PORTD I/O SUMMARY

Pin	Function	TRIS Setting	I/O	I/O Type	Description
RD0/PSP0	RD0	0	O	DIG	LATD<0> data output.
		1	I	ST	PORTD<0> data input.
	PSP0	x	O	DIG	PSP read data output (LATD<0>); takes priority over port data.
		x	I	TTL	PSP write data input.
RD1/PSP1	RD1	0	O	DIG	LATD<1> data output.
		1	I	ST	PORTD<1> data input.
	PSP1	x	O	DIG	PSP read data output (LATD<1>); takes priority over port data.
		x	I	TTL	PSP write data input.
RD2/PSP2	RD2	0	O	DIG	LATD<2> data output.
		1	I	ST	PORTD<2> data input.
	PSP2	x	O	DIG	PSP read data output (LATD<2>); takes priority over port data.
		x	I	TTL	PSP write data input.
RD3/PSP3	RD3	0	O	DIG	LATD<3> data output.
		1	I	ST	PORTD<3> data input.
	PSP3	x	O	DIG	PSP read data output (LATD<3>); takes priority over port data.
		x	I	TTL	PSP write data input.
RD4/PSP4	RD4	0	O	DIG	LATD<4> data output.
		1	I	ST	PORTD<4> data input.
	PSP4	x	O	DIG	PSP read data output (LATD<4>); takes priority over port data.
		x	I	TTL	PSP write data input.
RD5/PSP5/P1B	RD5	0	O	DIG	LATD<5> data output.
		1	I	ST	PORTD<5> data input.
	PSP5	x	O	DIG	PSP read data output (LATD<5>); takes priority over port data.
		x	I	TTL	PSP write data input.
	P1B	0	O	DIG	ECCP1 Enhanced PWM output, channel B; takes priority over port and PSP data. May be configured for tri-state during Enhanced PWM shutdown events.
RD6/PSP6/P1C	RD6	0	O	DIG	LATD<6> data output.
		1	I	ST	PORTD<6> data input.
	PSP6	x	O	DIG	PSP read data output (LATD<6>); takes priority over port data.
		x	I	TTL	PSP write data input.
	P1C	0	O	DIG	ECCP1 Enhanced PWM output, channel C; takes priority over port and PSP data. May be configured for tri-state during Enhanced PWM shutdown events.
RD7/PSP7/P1D	RD7	0	O	DIG	LATD<7> data output.
		1	I	ST	PORTD<7> data input.
	PSP7	x	O	DIG	PSP read data output (LATD<7>); takes priority over port data.
		x	I	TTL	PSP write data input.
	P1D	0	O	DIG	ECCP1 Enhanced PWM output, channel D; takes priority over port and PSP data. May be configured for tri-state during Enhanced PWM shutdown events.

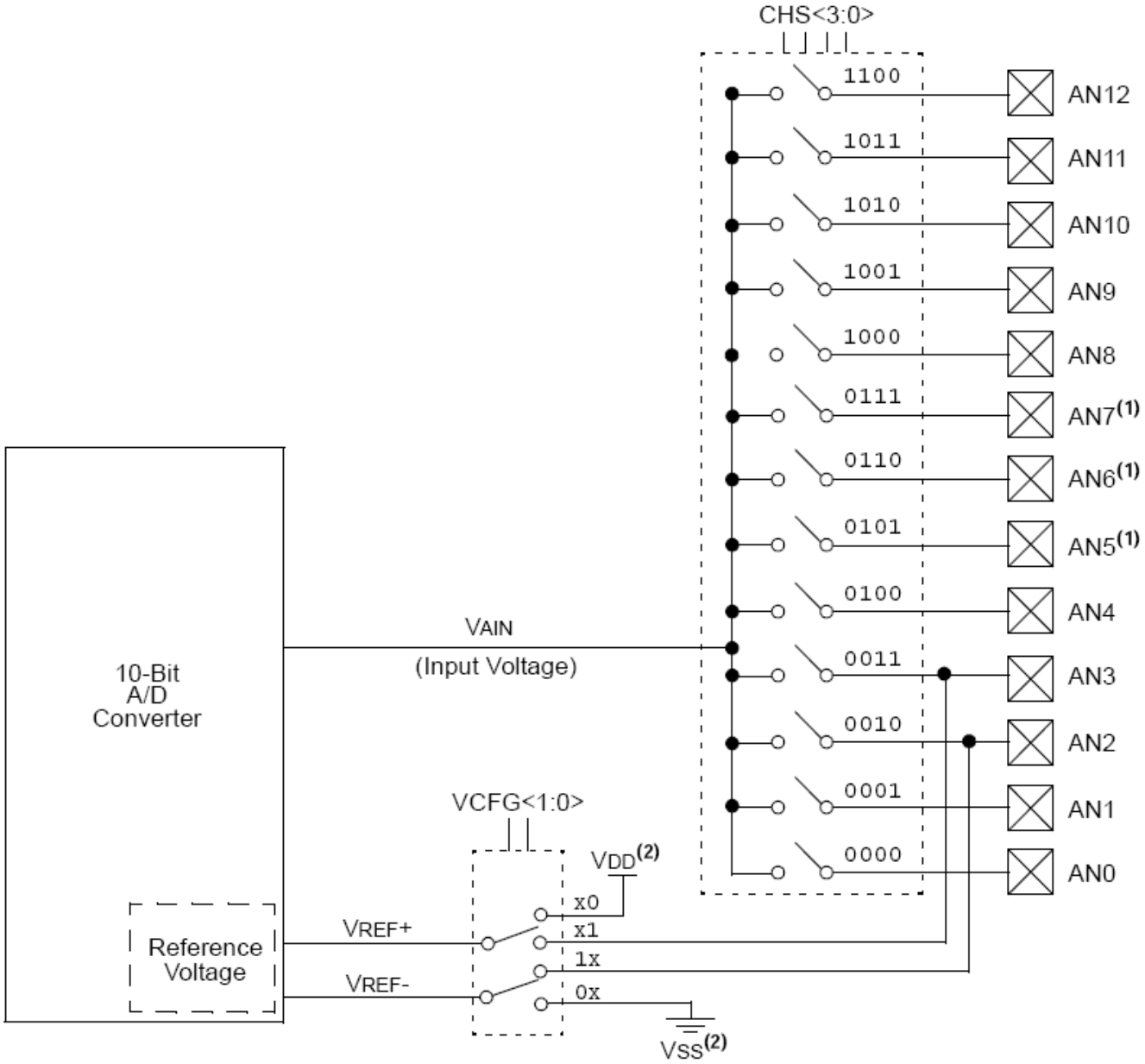
Legend: DIG = Digital level output; TTL = TTL input buffer; ST = Schmitt Trigger input buffer; x = Don't care (TRIS bit does not affect port direction or is overridden for this option).

TABLE 10-8: SUMMARY OF REGISTERS ASSOCIATED WITH PORTD

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Reset Values on page
PORTD	RD7	RD6	RD5	RD4	RD3	RD2	RD1	RD0	52
LATD	PORTD Data Latch Register (Read and Write to Data Latch)								52
TRISD	PORTD Data Direction Register								52
TRISE(1)	IBF	OBF	IBOV	PSPMODE	—	TRISE2	TRISE1	TRISE0	52
CCP1CON	P1M1(1)	P1M0(1)	DC1B1	DC1B0	CCP1M3	CCP1M2	CCP1M1	CCP1M0	51

Legend: — = unimplemented, read as '0'. Shaded cells are not used by PORTD.

Note 1: These registers and/or bits are unimplemented on 28-oin devices.



The following steps should be followed to perform an A/D conversion:

1. 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)

2. Configure A/D interrupt (if desired):

• Clear ADIF bit
• Set ADIE bit
• Set GIE bit

3. Wait the required acquisition time (if required).
4. Start conversion:

• Set GO/DONE bit (ADCON0 register)

5. 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

6. Read A/D Result registers (ADRESH:ADRESL); clear bit, ADIF, if required.
7. For next conversion, go to step 1 or step 2, as required. The A/D conversion time per bit is defined as TAD. A minimum wait of 2 TAD is required before the next acquisition starts.

E' molto complesso, qui analizzeremo le funzioni base.

Il convertitore AD, di tipo SAR permette di acquisire un canale fra fino a 13 (nei chip piu grossi) a scelta.
E' governato dai seguenti registri:
ADCON0: impostazione canale, SOC/EOC abilitazione
ADCON1: tensione di riferimento e abilitazione/disabilitazione bit porta A/B come seconda funzione.
ADCON2: impostazione intervallo fra una acquisizione e l'altra e divisore clock della macchina.
Il risultato della conversione è immagazzinato nei due registri a 8 bit ADRESH e ADRESL; ADRESL contiene in realtà solo 2 bit il bit 7 e 6, quindi tutto va traslato e riunito opportunamente per ottenere il risultato: (H << 8 + L) >> 6.
Il convertitore può generare inoltre una interruzione a fine conversione. Questo verrà analizzato in seguito. Pin associati al modulo: RA0,RA1,RA2, RA3, RA5, RB0,RB1,RB2,RB3,RB4, in modo disordinato!!!

REGISTER 19-1: ADCON0: A/D CONTROL REGISTER 0

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	CHS3	CHS2	CHS1	CHS0	GO/$\overline{\text{DONE}}$	ADON
bit 7							bit 0

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

bit 7-6 **Unimplemented:** Read as '0'

bit 5-2 **CHS<3:0>:** Analog Channel Select bits

0000 = Channel 0 (AN0)
0001 = Channel 1 (AN1)
0010 = Channel 2 (AN2)
0011 = Channel 3 (AN3)
0100 = Channel 4 (AN4)
0101 = Channel 5 (AN5)(1,2)
0110 = Channel 6 (AN6)(1,2)
0111 = Channel 7 (AN7)(1,2)
1000 = Channel 8 (AN8)
1001 = Channel 9 (AN9)
1010 = Channel 10 (AN10)
1011 = Channel 11 (AN11)
1100 = Channel 12 (AN12)
1101 = Unimplemented)(2)
1110 = Unimplemented)(2)
1111 = Unimplemented)(2)

bit 1 **GO/$\overline{\text{DONE}}$:** A/D Conversion Status bit

When ADON = 1:
1 = A/D conversion in progress
0 = A/D Idle

bit 0 **ADON:** A/D On bit

1 = A/D Converter module is enabled
0 = A/D Converter module is disabled

Note 1: These channels are not implemented on 28-pin devices.
2: Performing a conversion on unimplemented channels will return a floating input measurement.

REGISTER 19-2: ADCON1: A/D CONTROL REGISTER 1

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-q(1)	R/W-q(1)	R/W-q
—	—	VCFG1	VCFG0	PCFG3	PCFG2	PCFG1	PCFG
bit 7							

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

bit 7-6 **Unimplemented:** Read as '0'

bit 5 **VCFG1:** Voltage Reference Configuration bit (V_{REF-} source)
1 = V_{REF-} (AN2)
0 = V_{SS}

bit 4 **VCFG0:** Voltage Reference Configuration bit (V_{REF+} source)
1 = V_{REF+} (AN3)
0 = V_{DD}

bit 3-0 **PCFG<3:0>:** A/D Port Configuration Control bits:

PCFG3: PCFG0	AN12	AN11	AN10	AN9	AN8	AN7(2)	AN6(2)	AN5(2)	AN4	AN3	AN2	AN1	AN0
0000(1)	A	A	A	A	A	A	A	A	A	A	A	A	A
0001	A	A	A	A	A	A	A	A	A	A	A	A	A
0010	A	A	A	A	A	A	A	A	A	A	A	A	A
0011	D	A	A	A	A	A	A	A	A	A	A	A	A
0100	D	D	A	A	A	A	A	A	A	A	A	A	A
0101	D	D	D	A	A	A	A	A	A	A	A	A	A
0110	D	D	D	D	A	A	A	A	A	A	A	A	A
0111(1)	D	D	D	D	D	A	A	A	A	A	A	A	A
1000	D	D	D	D	D	D	A	A	A	A	A	A	A
1001	D	D	D	D	D	D	D	A	A	A	A	A	A
1010	D	D	D	D	D	D	D	D	A	A	A	A	A
1011	D	D	D	D	D	D	D	D	D	A	A	A	A
1100	D	D	D	D	D	D	D	D	D	D	A	A	A
1101	D	D	D	D	D	D	D	D	D	D	D	A	A
1110	D	D	D	D	D	D	D	D	D	D	D	D	A
1111	D	D	D	D	D	D	D	D	D	D	D	D	D

A = Analog input D = Digital I/O

Note 1: The POR value of the PCFG bits depends on the value of the PBADEN Configuration bit. When PBADEN = 1, PCFG<2:0> = 000; when PBADEN = 0, PCFG<2:0> = 111.

2: AN5 through AN7 are available only on 40/44-pin devices.

Nel seguito TAD indica il tempo necessario per converire un bit. La conversione completa di un campione analogico richiede 11 TAD. TAD dipende dal clock della macchina e dal divisore di frequenza. Fra una conversione e l'altra può intercorrere un tempo ACQT come da tabella seguente

TABLE 19-1: TAD vs. DEVICE OPERATING FREQUENCIES

AD Clock Source (TAD)		Maximum Device Frequency	
Operation	ADCS<2:0>	PIC18F2X20/4X20	PIC18LF2X2X/4X20(4)
2 TOSC	000	2.86 MHz	1.43 kHz
4 TOSC	100	5.71 MHz	2.86 MHz
8 TOSC	001	11.43 MHz	5.72 MHz
16 TOSC	101	22.86 MHz	11.43 MHz
32 TOSC	010	40.0 MHz	22.86 MHz
64 TOSC	110	40.0 MHz	22.86 MHz
RC(3)	x11	1.00 MHz(1)	1.00 MHz(2)

Note 1: The RC source has a typical TAD time of 1.2 µs.
2: The RC source has a typical TAD time of 2.5 µs.
3: For device frequencies above 1 MHz, the device must be in Sleep for the entire conversion or the A/D accuracy may be out of specification.
4: Low-power (PIC18LFXXXX) devices only.

REGISTER 19-3: 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	—	ACQT2	ACQT1	ACQT0	ADCS2	ADCS1	ADCS0
bit 7							bit 0

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

bit 7 **ADFM:** A/D Result Format Select bit
1 = Right justified
0 = Left justified

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

bit 5-3 **ACQT<2:0>:** A/D Acquisition Time Select bits
111 = 20 TAD
110 = 16 TAD
101 = 12 TAD
100 = 8 TAD
011 = 6 TAD
010 = 4 TAD
001 = 2 TAD
000 = 0 TAD(1)

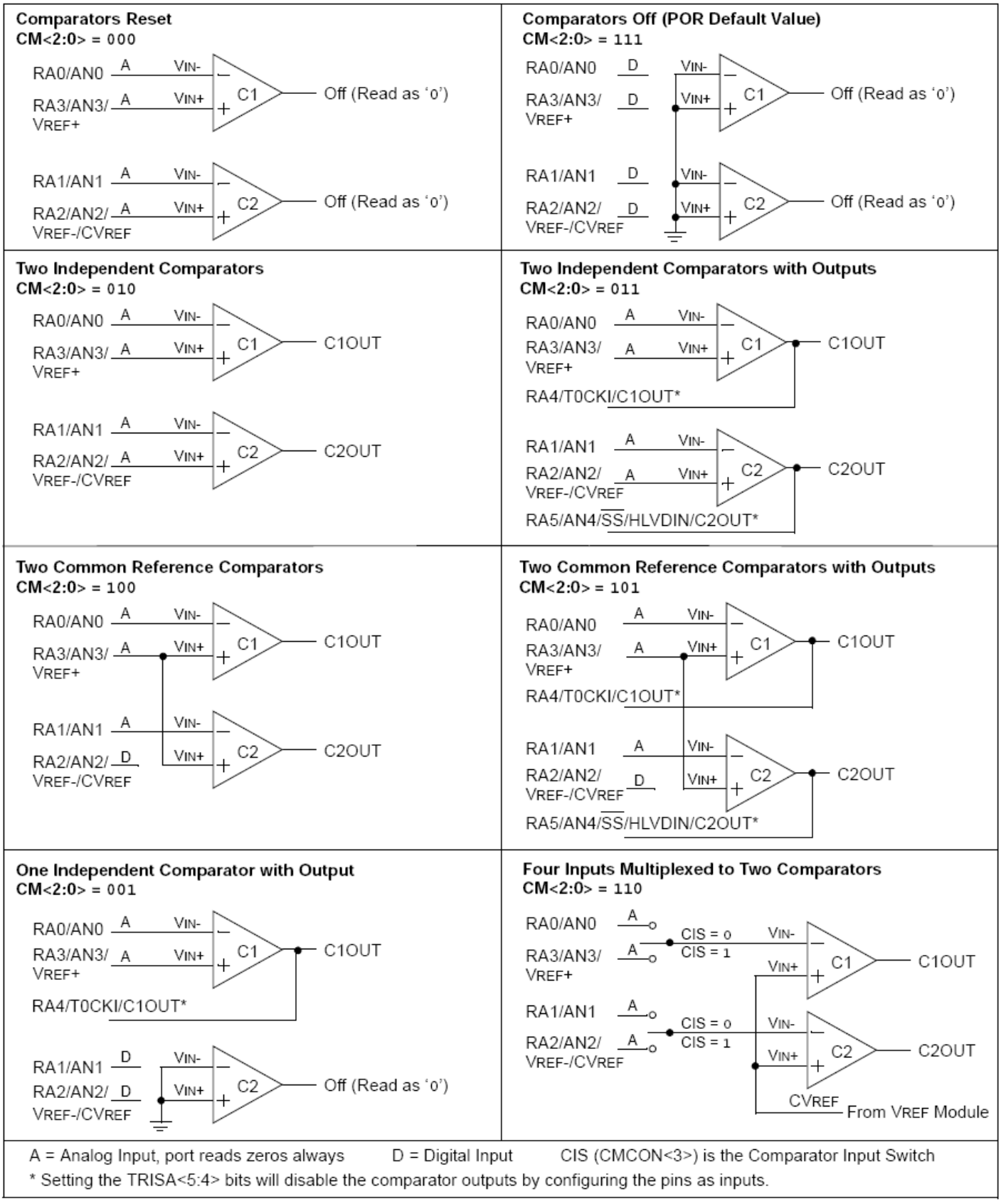
bit 2-0 **ADCS<2:0>:** A/D Conversion Clock Select bits
111 = FRC (clock derived from A/D RC oscillator)(1)
110 = FOSC/64
101 = FOSC/16
100 = FOSC/4
011 = FRC (clock derived from A/D RC oscillator)(1)
010 = FOSC/32
001 = FOSC/8
000 = FOSC/2

Note 1: If the A/D FRC clock source is selected, a delay of one TCY (instruction cycle) is added before the A/D clock starts. This allows the SLEEP instruction to be executed before starting a conversion.

I comparatori:

Il pic18F2420 possiede al suo interno due comparatori, con gli ingressi facenti capo ad alcuni bit della porta A e che possono essere commutati e configurati in vario modo utilizzando i registri e le modalità descritte dalle figure e tabelle seguenti. Il registro fondamentale per il loro utilizzo base è CMCON. I due comparatori possono inoltre essere definiti come sorgenti di interruzione, come verrà approfondito in seguito. I comparatori possiedono inoltre un generatore programmabile di tensione di riferimento e possono essere impostati per generare una interruzione non appena uno dei due ingressi si discosta da un valore di riferimento programmabile. Queste funzionalità complesse esulano dai nostri scopi.Pin associati al modulo: RA0, RA1, RA2, RA3, RA4,RA5.

FIGURE 20-1: COMPARATOR I/O OPERATING MODES



A = Analog Input, port reads zeros always D = Digital Input CIS (CMCON<3>) is the Comparator Input Switch

* Setting the TRISA<5:4> bits will disable the comparator outputs by configuring the pins as inputs.

REGISTER 20-1: CMCON: COMPARATOR CONTROL REGISTER

R-0	R-0	R/W-0	R/W-0	R/W-0	R/W-1	R/W-1	R/W-1
C2OUT	C1OUT	C2INV	C1INV	CIS	CM2	CM1	CM0
bit 7							bit 0

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

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> = 110:
1 = C1 VIN- connects to RA3/AN3/VREF+
C2 VIN- connects to RA2/AN2/VREF-/CVREF
0 = C1 VIN- connects to RA0/AN0
C2 VIN- connects to RA1/AN1

bit 2-0 **CM<2:0>**: Comparator Mode bits
Figure 20-1 shows the Comparator modes and the CM<2:0> bit settings.

LE INTERRUZIONI

Praticamente ogni dispositivo a bordo può generare una o più interruzioni. A differenza della quasi totalità dei microcontrollori in commercio le interruzioni non sono vettorizzate, ovvero non esiste un vettore di interruzione che può essere fatto puntare ad una routine di servizio per ogni singola periferica e per ogni sottosistema della periferica. Piuttosto sono divise in due gruppi, a bassa e ad alta priorità. Ciascuna periferica può essere assegnata al gruppo ad alta o bassa priorità. Esistono due soli vettori di interruzione agli indirizzi 08h per quelle ad alta priorità e 018h per quelle a bassa priorità. E' lasciato al programmatore scrivere le due corrispondenti routine che, attraverso la lettura dei registri di richiesta di interruzione, analizzino quali e quante periferiche hanno chiesto una interruzione e stabiliscano l'ordine in cui servire richieste contemporanee da parte di più periferiche. In ogni caso una richiesta ad alta priorità interrompe una a bassa priorità (ma non viceversa). Le richieste ad alta e a bassa priorità possono essere abilitate o disabilitate indipendentemente. Infine il programmatore, anche qui a differenza della quasi totalità dei microcontrollori in commercio, deve resettare esplicitamente i flag deile richieste di interruzione pendenti. Esiste anche la possibilità di disabilitare il meccanismo di priorità (abilitato di default al reset); in tal caso le interruzioni sono tutte ad alta priorità e gestite da una unica routine con interrupt vector a 08h. Tutte le sorgenti sono di default assegnate al gruppo ad alta priorità ad eccezione del TIMER0 che è sempre e solo a bassa priorità.

LOGICA DEL SISTEMA DI ABILITAZIONE E PRIORITA'

I segnali che terminano con F (flag) sono le richieste e si trovano nei registri PIR; i segnali che terminano con E sono le abilitazioni e si trovano nei registri PIE; i segnali che terminano con P sono le priorità e si trovano nei registri IPR. Per gestire una periferica ad interruzione bisogna: (1) abilitare o meno il sitema delle priorità (IPEN in RCON); (2) stabilire se è ad alta o bassa priorità impostando o resettando il bit ad essa relativo nel registro IPR opportuno, (3) abilitare la periferica ad interrompere impostando ad uno il bit relativo nel registro PIE opportuno; (4) abilitare le interruzioni a priorità alta o bassa a seconda del gruppo in cui si trova, nel registro INTCON. (5) La routine di gestione dovrà, alla fine del compito ad essa assegnato, resettare il flag relativo alla periferica nel registro PIR opportuno. Si noti che anche se le interruzioni sono disabilitate i flag nei registri PIR vengono comunque impostati al verificarsi di un evento atteso su una periferica. La cpu può pertanto controllare in modo programmato se l'evento è avvenuto controllando il bit F relativo ad una periferica nel registro PIR. Tale modalità è detta **POLLED INTERRUPT**.

FIGURE 9-1: PIC18 INTERRUPT LOGIC

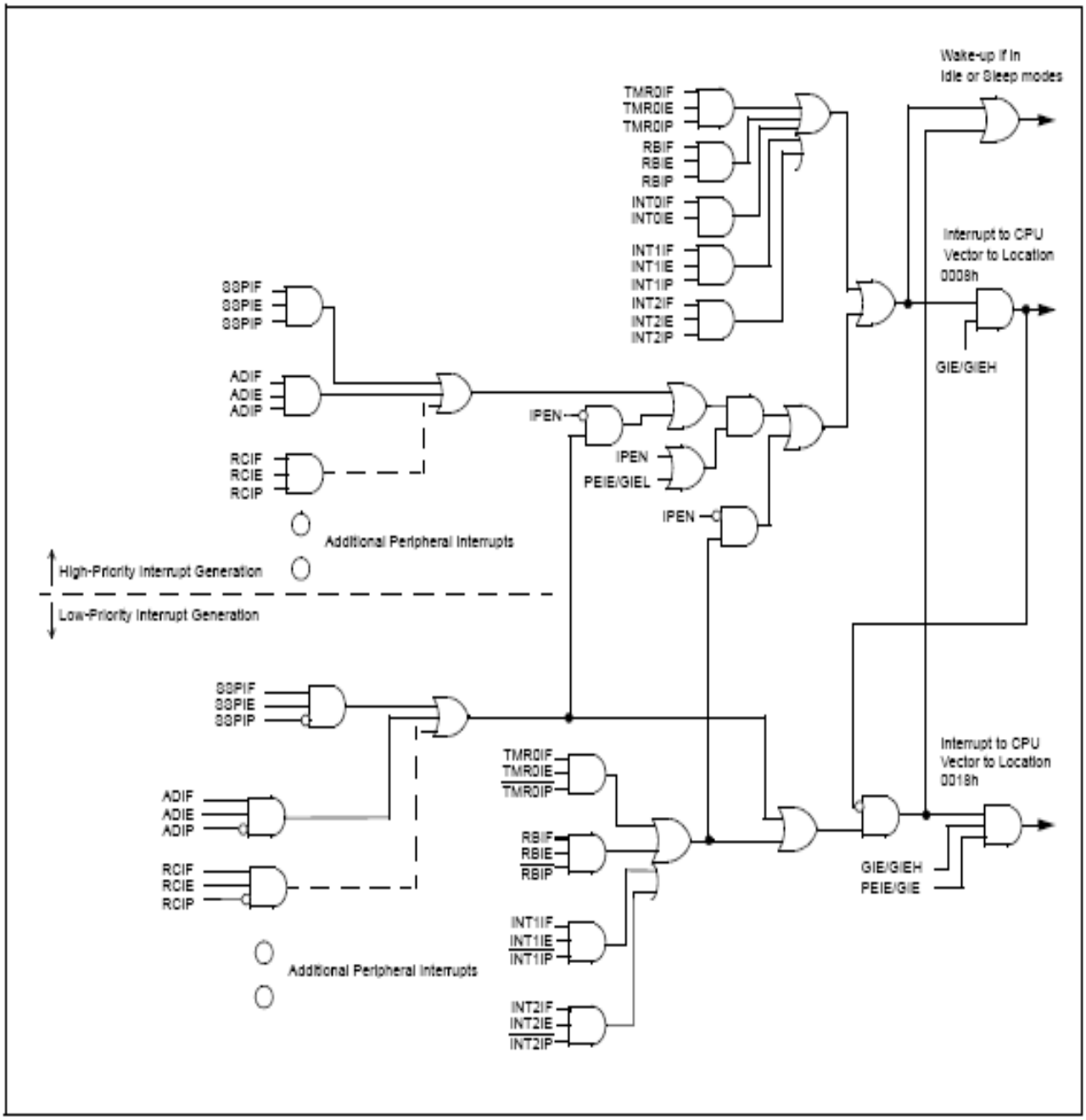
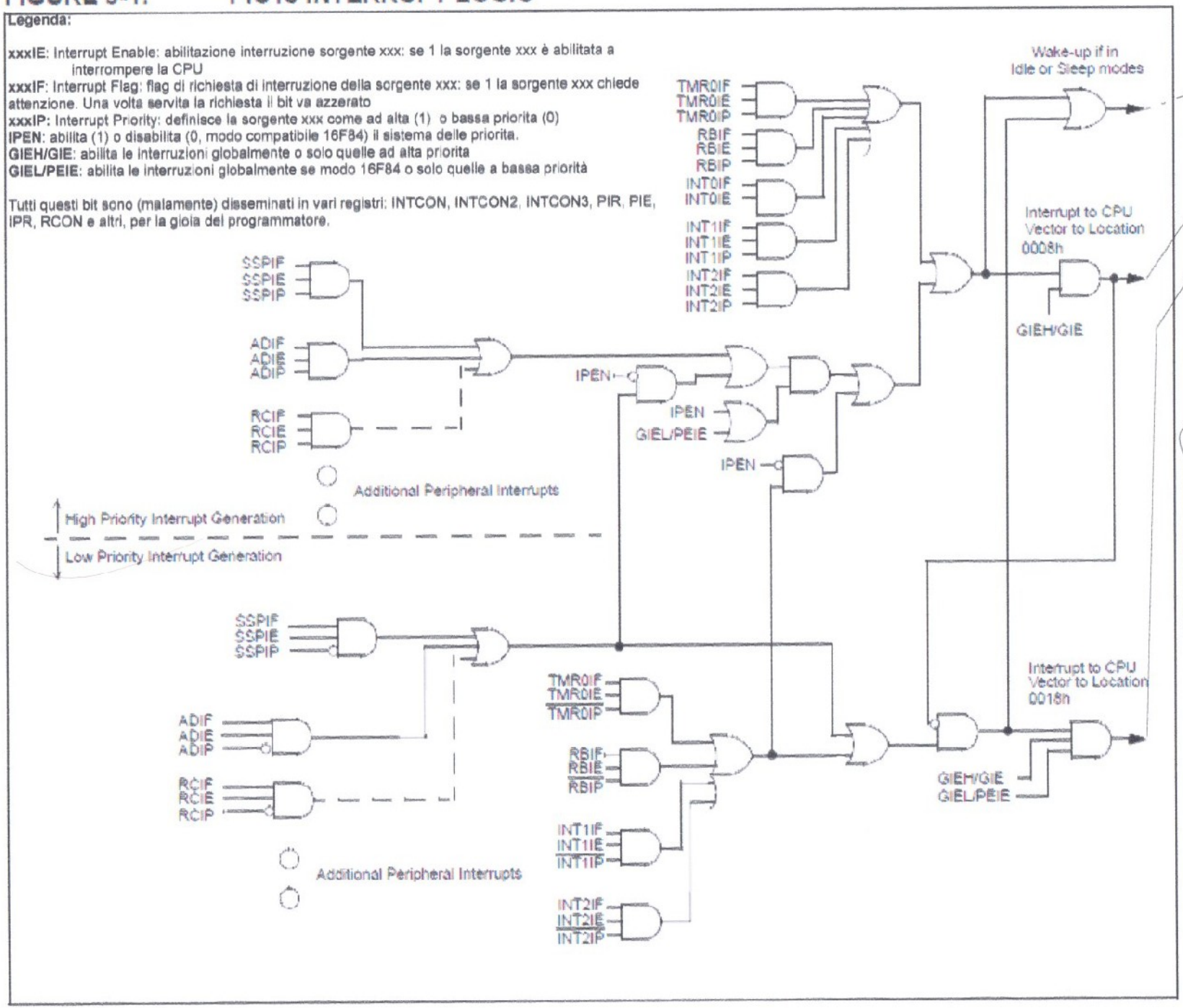


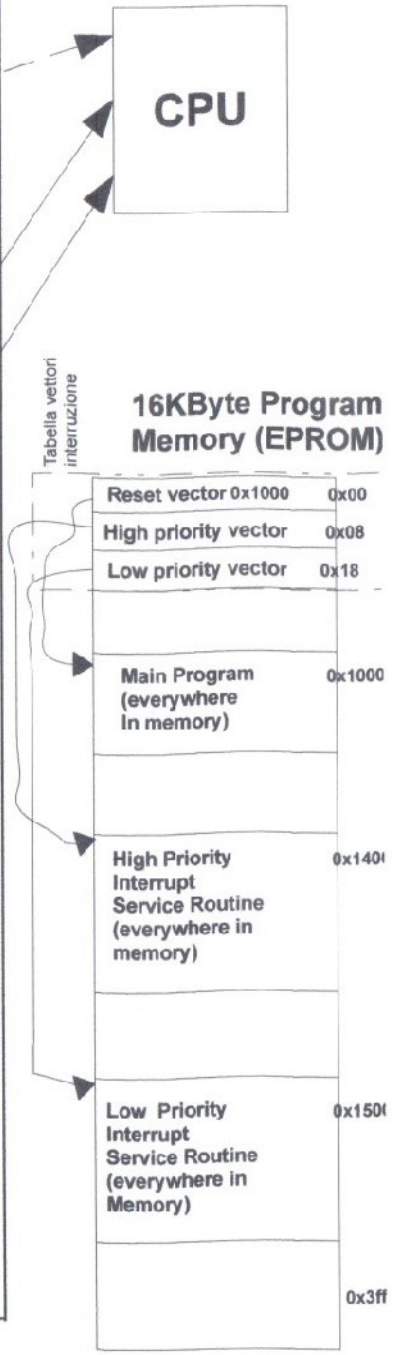
FIGURE 9-1: PIC18 INTERRUPT LOGIC

Legenda:

xxxIE: Interrupt Enable: abilitazione interruzione sorgente xxx: se 1 la sorgente xxx è abilitata a interrompere la CPU
xxxIF: Interrupt Flag: flag di richiesta di interruzione della sorgente xxx: se 1 la sorgente xxx chiede attenzione. Una volta servita la richiesta il bit va azzerato
xxxIP: Interrupt Priority: definisce la sorgente xxx come ad alta (1) o bassa priorita (0)
IPEN: abilita (1) o disabilita (0, modo compatibile 16F84) il sistema delle priorita.
GIEH/GIE: abilita le interruzioni globalmente o solo quelle ad alta priorita
GIEL/PEIE: abilita le interruzioni globalmente se modo 16F84 o solo quelle a bassa priorità

Tutti questi bit sono (malamente) disseminati in vari registri: INTCON, INTCON2, INTCON3, PIR, PIE, IPR, RCON e altri, per la gioia del programmatore.





REGISTER 9-1: INTCON: INTERRUPT CONTROL REGISTER

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-x
GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF(1)
bit 7							bit 0

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

bit 7 **GIE/GIEH:** Global Interrupt Enable bit
When IPEN = 0:
1 = Enables all unmasked interrupts
0 = Disables all interrupts
When IPEN = 1:
1 = Enables all high-priority interrupts
0 = Disables all interrupts

bit 6 **PEIE/GIEL:** Peripheral Interrupt Enable bit
When IPEN = 0:
1 = Enables all unmasked peripheral interrupts
0 = Disables all peripheral interrupts
When IPEN = 1:
1 = Enables all low-priority peripheral interrupts
0 = Disables all low-priority peripheral interrupts

bit 5 **TMR0IE:** TMR0 Overflow Interrupt Enable bit
1 = Enables the TMR0 overflow interrupt
0 = Disables the TMR0 overflow interrupt

bit 4 **INT0IE:** INT0 External Interrupt Enable bit
1 = Enables the INT0 external interrupt
0 = Disables the INT0 external interrupt

bit 3 **RBIE:** RB Port Change Interrupt Enable bit
1 = Enables the RB port change interrupt
0 = Disables the RB port change interrupt

bit 2 **TMR0IF:** TMR0 Overflow Interrupt Flag bit
1 = TMR0 register has overflowed (must be cleared in software)
0 = TMR0 register did not overflow

bit 1 **INT0IF:** INT0 External Interrupt Flag bit
1 = The INT0 external interrupt occurred (must be cleared in software)
0 = The INT0 external interrupt did not occur

bit 0 **RBIF:** RB Port Change Interrupt Flag bit(1)
1 = At least one of the RB<7:4> pins changed state (must be cleared in software)
0 = None of the RB<7:4> pins have changed state

REGISTER 9-2: INTCON2: INTERRUPT CONTROL REGISTER 2

R/W-1	R/W-1	R/W-1	R/W-1	U-0	R/W-1	U-0	R/W-1
$\overline{\text{RBPU}}$	INTEDG0	INTEDG1	INTEDG2	—	TMR0IP	—	RBIP
bit 7							bit 0

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

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

bit 6 **INTEDG0:** External Interrupt 0 Edge Select bit
1 = Interrupt on rising edge
0 = Interrupt on falling edge

bit 5 **INTEDG1:** External Interrupt 1 Edge Select bit
1 = Interrupt on rising edge
0 = Interrupt on falling edge

bit 4 **INTEDG2:** External Interrupt 2 Edge Select bit
1 = Interrupt on rising edge
0 = Interrupt on falling edge

bit 3 **Unimplemented:** Read as '0'

bit 2 **TMR0IP:** TMR0 Overflow Interrupt Priority bit
1 = High priority
0 = Low priority

bit 1 **Unimplemented:** Read as '0'

bit 0 **RBIP:** RB Port Change Interrupt Priority bit
1 = High priority
0 = Low priority

REGISTER 9-3: INTCON3: INTERRUPT CONTROL REGISTER 3

R/W-1	R/W-1	U-0	R/W-0	R/W-0	U-0	R/W-0	R/W-0
INT2IP	INT1IP	—	INT2IE	INT1IE	—	INT2IF	INT1IF
bit 7							bit 0

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

bit 7 **INT2IP:** INT2 External Interrupt Priority bit
1 = High priority
0 = Low priority

bit 6 **INT1IP:** INT1 External Interrupt Priority bit
1 = High priority
0 = Low priority

bit 5 **Unimplemented:** Read as '0'

bit 4 **INT2IE:** INT2 External Interrupt Enable bit
1 = Enables the INT2 external interrupt
0 = Disables the INT2 external interrupt

bit 3 **INT1IE:** INT1 External Interrupt Enable bit
1 = Enables the INT1 external interrupt
0 = Disables the INT1 external interrupt

bit 2 **Unimplemented:** Read as '0'

bit 1 **INT2IF:** INT2 External Interrupt Flag bit
1 = The INT2 external interrupt occurred (must be cleared in software)
0 = The INT2 external interrupt did not occur

bit 0 **INT1IF:** INT1 External Interrupt Flag bit
1 = The INT1 external interrupt occurred (must be cleared in software)
0 = The INT1 external interrupt did not occur

Note: Interrupt flag bits are set when an interrupt condition occurs, regardless of the state of its corresponding enable bit or the global enable bit. User software should ensure the appropriate interrupt flag bits are clear prior to enabling an interrupt. This feature allows for software polling.

REGISTER 9-4: PIR1: PERIPHERAL INTERRUPT REQUEST (FLAG) REGISTER 1

R/W-0	R/W-0	R-0	R-0	R/W-0	R/W-0	R/W-0	R/W-0
PSPIF(1)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF
bit 7							bit 0

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

bit 7 **PSPIF:** Parallel Slave Port Read/Write Interrupt Flag bit(1)
1 = A read or a write operation has taken place (must be cleared in software)
0 = No read or write has occurred

bit 6 **ADIF:** A/D Converter Interrupt Flag bit
1 = An A/D conversion completed (must be cleared in software)
0 = The A/D conversion is not complete

bit 5 **RCIF:** EUSART Receive Interrupt Flag bit
1 = The EUSART receive buffer, RCREG, is full (cleared when RCREG is read)
0 = The EUSART receive buffer is empty

bit 4 **TXIF:** EUSART Transmit Interrupt Flag bit
1 = The EUSART transmit buffer, TXREG, is empty (cleared when TXREG is written)
0 = The EUSART transmit buffer is full

bit 3 **SSPIF:** Master Synchronous Serial Port Interrupt Flag bit
1 = The transmission/reception is complete (must be cleared in software)
0 = Waiting to transmit/receive

bit 2 **CCP1IF:** CCP1 Interrupt Flag bit
Capture mode:
1 = A TMR1 register capture occurred (must be cleared in software)
0 = No TMR1 register capture occurred
Compare mode:
1 = A TMR1 register compare match occurred (must be cleared in software)
0 = No TMR1 register compare match occurred
PWM mode:
Unused in this mode.

bit 1 **TMR2IF:** TMR2 to PR2 Match Interrupt Flag bit
1 = TMR2 to PR2 match occurred (must be cleared in software)
0 = No TMR2 to PR2 match occurred

bit 0 **TMR1IF:** TMR1 Overflow Interrupt Flag bit
1 = TMR1 register overflowed (must be cleared in software)
0 = TMR1 register did not overflow

Note 1: This bit is unimplemented on 28-pin devices and will read as '0'.

REGISTER 9-5: PIR2: PERIPHERAL INTERRUPT REQUEST (FLAG) REGISTER 2

R/W-0	R/W-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
OSCFIF	CMIF	—	EEIF	BCLIF	HLVDIF	TMR3IF	CCP2IF
bit 7							bit 0

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

bit 7 **OSCFIF:** Oscillator Fail Interrupt Flag bit
1 = Device oscillator failed, clock input has changed to INTOSC (must be cleared in software)
0 = Device clock operating

bit 6 **CMIF:** Comparator Interrupt Flag bit
1 = Comparator input has changed (must be cleared in software)
0 = Comparator input has not changed

bit 5 **Unimplemented:** Read as '0'

bit 4 **EEIF:** Data EEPROM/Flash Write Operation Interrupt Flag bit
1 = The write operation is complete (must be cleared in software)
0 = The write operation is not complete or has not been started

bit 3 **BCLIF:** Bus Collision Interrupt Flag bit
1 = A bus collision occurred (must be cleared in software)
0 = No bus collision occurred

bit 2 **HLVDIF:** High/Low-Voltage Detect Interrupt Flag bit
1 = A high/low-voltage condition occurred (direction determined by VDIRMAG bit, HLVDCON<7>)
0 = A high/low-voltage condition has not occurred

bit 1 **TMR3IF:** TMR3 Overflow Interrupt Flag bit
1 = TMR3 register overflowed (must be cleared in software)
0 = TMR3 register did not overflow

bit 0 **CCP2IF:** CCP2 Interrupt Flag bit
Capture mode:
1 = A TMR1 register capture occurred (must be cleared in software)
0 = No TMR1 register capture occurred
Compare mode:
1 = A TMR1 register compare match occurred (must be cleared in software)
0 = No TMR1 register compare match occurred
PWM mode:
Unused in this mode.

REGISTER 9-6: PIE1: PERIPHERAL INTERRUPT ENABLE REGISTER 1

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
PSPIE(1)	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE
bit 7							bit 0

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

bit 7 **PSPIE:** Parallel Slave Port Read/Write Interrupt Enable bit(1)
1 = Enables the PSP read/write interrupt
0 = Disables the PSP read/write interrupt

bit 6 **ADIE:** A/D Converter Interrupt Enable bit
1 = Enables the A/D interrupt
0 = Disables the A/D interrupt

bit 5 **RCIE:** EUSART Receive Interrupt Enable bit
1 = Enables the EUSART receive interrupt
0 = Disables the EUSART receive interrupt

bit 4 **TXIE:** EUSART Transmit Interrupt Enable bit
1 = Enables the EUSART transmit interrupt
0 = Disables the EUSART transmit interrupt

bit 3 **SSPIE:** Master Synchronous Serial Port Interrupt Enable bit
1 = Enables the MSSP interrupt
0 = Disables the MSSP interrupt

bit 2 **CCP1IE:** CCP1 Interrupt Enable bit
1 = Enables the CCP1 interrupt
0 = Disables the CCP1 interrupt

bit 1 **TMR2IE:** TMR2 to PR2 Match Interrupt Enable bit
1 = Enables the TMR2 to PR2 match interrupt
0 = Disables the TMR2 to PR2 match interrupt

bit 0 **TMR1IE:** TMR1 Overflow Interrupt Enable bit
1 = Enables the TMR1 overflow interrupt
0 = Disables the TMR1 overflow interrupt

Note 1: This bit is unimplemented on 28-pin devices and will read as '0'.

REGISTER 9-7: PIE2: PERIPHERAL INTERRUPT ENABLE REGISTER 2

R/W-0	R/W-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
OSCFIE	CMIE	—	EEIE	BCLIE	HLVDIE	TMR3IE	CCP2IE
bit 7							bit 0

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

bit 7 **OSCFIE:** Oscillator Fail Interrupt Enable bit
1 = Enabled
0 = Disabled

bit 6 **CMIE:** Comparator Interrupt Enable bit
1 = Enabled
0 = Disabled

bit 5 **Unimplemented:** Read as '0'

bit 4 **EEIE:** Data EEPROM/Flash Write Operation Interrupt Enable bit
1 = Enabled
0 = Disabled

bit 3 **BCLIE:** Bus Collision Interrupt Enable bit
1 = Enabled
0 = Disabled

bit 2 **HLVDIE:** High/Low-Voltage Detect Interrupt Enable bit
1 = Enabled
0 = Disabled

bit 1 **TMR3IE:** TMR3 Overflow Interrupt Enable bit
1 = Enabled
0 = Disabled

bit 0 **CCP2IE:** CCP2 Interrupt Enable bit
1 = Enabled
0 = Disabled

REGISTER 9-8: IPR1: PERIPHERAL INTERRUPT PRIORITY REGISTER 1

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
PSPIP(1)	ADIP	RCIP	TXIP	SSPIP	CCP1IP	TMR2IP	TMR1IP
bit 7							bit 0

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

bit 7 **PSPIP:** Parallel Slave Port Read/Write Interrupt Priority bit(1)
1 = High priority
0 = Low priority

bit 6 **ADIP:** A/D Converter Interrupt Priority bit
1 = High priority
0 = Low priority

bit 5 **RCIP:** EUSART Receive Interrupt Priority bit
1 = High priority
0 = Low priority

bit 4 **TXIP:** EUSART Transmit Interrupt Priority bit
1 = High priority
0 = Low priority

bit 3 **SSPIP:** Master Synchronous Serial Port Interrupt Priority bit
1 = High priority
0 = Low priority

bit 2 **CCP1IP:** CCP1 Interrupt Priority bit
1 = High priority
0 = Low priority

bit 1 **TMR2IP:** TMR2 to PR2 Match Interrupt Priority bit
1 = High priority
0 = Low priority

bit 0 **TMR1IP:** TMR1 Overflow Interrupt Priority bit
1 = High priority
0 = Low priority

Note 1: This bit is unimplemented on 28-pin devices and will read as '0'.

REGISTER 9-9: IPR2: PERIPHERAL INTERRUPT PRIORITY REGISTER 2

R/W-1	R/W-1	U-0	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
OSCFIP	CMIP	—	EEIP	BCLIP	HLVDIP	TMR3IP	CCP2IP
bit 7							bit 0

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

bit 7 **OSCFIP:** Oscillator Fail Interrupt Priority bit
1 = High priority
0 = Low priority

bit 6 **CMIP:** Comparator Interrupt Priority bit
1 = High priority
0 = Low priority

bit 5 **Unimplemented:** Read as '0'

bit 4 **EEIP:** Data EEPROM/Flash Write Operation Interrupt Priority bit
1 = High priority
0 = Low priority

bit 3 **BCLIP:** Bus Collision Interrupt Priority bit
1 = High priority
0 = Low priority

bit 2 **HLVDIP:** High/Low-Voltage Detect Interrupt Priority bit
1 = High priority
0 = Low priority

bit 1 **TMR3IP:** TMR3 Overflow Interrupt Priority bit
1 = High priority
0 = Low priority

bit 0 **CCP2IP:** CCP2 Interrupt Priority bit
1 = High priority
0 = Low priority

REGISTER 9-10: RCON: RESET CONTROL REGISTER

R/W-0	R/W-1(1)	U-0	R/W-1	R-1	R-1	R/W-0(1)	R/W-0
IPEN	SBOREN	—	$\overline{RI}$	$\overline{TO}$	$\overline{PD}$	$\overline{POR}$	$\overline{BOR}$
bit 7							bit 0

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

bit 7 **IPEN:** Interrupt Priority Enable bit
1 = Enable priority levels on interrupts
0 = Disable priority levels on interrupts (PIC16CXXX Compatibility mode)

bit 6 **SBOREN:** Software BOR Enable bit(1)
For details of bit operation, see Register 4-1.

bit 5 **Unimplemented:** Read as '0'

bit 4 $\overline{RI}$: RESET Instruction Flag bit
For details of bit operation, see Register 4-1.

bit 3 $\overline{TO}$: Watchdog Timer Time-out Flag bit
For details of bit operation, see Register 4-1.

bit 2 $\overline{PD}$: Power-Down Detection Flag bit
For details of bit operation, see Register 4-1.

bit 1 $\overline{POR}$: Power-on Reset Status bit(1)
For details of bit operation, see Register 4-1.

bit 0 $\overline{BOR}$: Brown-out Reset Status bit
For details of bit operation, see Register 4-1.

Note 1: Actual Reset values are determined by device configuration and the nature of the device Reset. See Register 4-1 for additional information.

LA UART

La uart è un dispositivo abbastanza complesso e ricco di funzionalità inserito in un altro dispositivo (USART) ancora più complesso dotato della possibilità di effettuare anche trasmissioni sincrone e multidrop con indirizzamento hardware del nodo destinatario. Qui ci limiteremo a descrivere le parti coinvolte nel funzionamento come seriale asincrona half o full-duplex. Anche questo dispositivo può essere usato a polling o ad interrupt previo impostazione degli adeguati bit nei registri opportuni così come descritto al capitolo sulle interruzioni. Come vedremo la UART è prevista come periferica di uscita standard dalla maggioranza dei compilatori C che forniscono delle librerie ad alto livello per potersi interfacciare subito e con poca fatica con il mondo esterno e cominciare a fare i primi esperimenti. Tali librerie ci fanno vedere le funzioni standard di io del C che conosciamo bene: printf, getch, putch, scanf. In realtà a basso livello queste funzioni di libreria programmano e controllano i registri indicati di seguito.

E' comunque possibile costruire le nostre funzioni di gestione della UART. Pin associati: RC7, RC6 (Rx, Tx)

REGISTER 18-1: TXSTA: TRANSMIT STATUS AND CONTROL REGISTER

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-1	R/W-0
CSRC	TX9	TXEN(1)	SYNC	SENDB	BRGH	TRMT	TX9D
bit 7							bit 0

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

bit 7 **CSRC:** Clock Source Select bit
Asynchronous mode:
Don't care.
Synchronous mode:
1 = Master mode (clock generated internally from BRG)
0 = Slave mode (clock from external source)

bit 6 **TX9:** 9-Bit Transmit Enable bit
1 = Selects 9-bit transmission
0 = Selects 8-bit transmission

bit 5 **TXEN:** Transmit Enable bit(1)
1 = Transmit enabled
0 = Transmit disabled

bit 4 **SYNC:** EUSART Mode Select bit
1 = Synchronous mode
0 = Asynchronous mode

bit 3 **SENDB:** Send Break Character bit
Asynchronous mode:
1 = Send Sync Break on next transmission (cleared by hardware upon completion)
0 = Sync Break transmission completed
Synchronous mode:
Don't care.

bit 2 **BRGH:** High Baud Rate Select bit
Asynchronous mode:
1 = High speed
0 = Low speed
Synchronous mode:
Unused in this mode.

bit 1 **TRMT:** Transmit Shift Register Status bit
1 = TSR empty
0 = TSR full

bit 0 **TX9D:** 9th Bit of Transmit Data
Can be address/data bit or a parity bit.

Note 1: SREN/CREN overrides TXEN in Sync mode.

REGISTER 18-2: RCSTA: RECEIVE STATUS AND CONTROL REGISTER

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-0	R-0	R-x
SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D
bit 7							bit 0

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

bit 7 **SPEN:** Serial Port Enable bit
1 = Serial port enabled (configures RX/DT and TX/CK pins as serial port pins)
0 = Serial port disabled (held in Reset)

bit 6 **RX9:** 9-Bit Receive Enable bit
1 = Selects 9-bit reception
0 = Selects 8-bit reception

bit 5 **SREN:** Single Receive Enable bit
Asynchronous mode:
Don't care.
Synchronous mode – Master:
1 = Enables single receive
0 = Disables single receive
This bit is cleared after reception is complete.
Synchronous mode – Slave:
Don't care.

bit 4 **CREN:** Continuous Receive Enable bit
Asynchronous mode:
1 = Enables receiver
0 = Disables receiver
Synchronous mode:
1 = Enables continuous receive until enable bit, CREN, is cleared (CREN overrides SREN)
0 = Disables continuous receive

bit 3 **ADDEN:** Address Detect Enable bit
Asynchronous mode 9-Bit (RX9 = 1):
1 = Enables address detection, enables interrupt and loads the receive buffer when RSR<8> is set
0 = Disables address detection, all bytes are received and ninth bit can be used as parity bit
Asynchronous mode 9-Bit (RX9 = 0):
Don't care.

bit 2 **FERR:** Framing Error bit
1 = Framing error (can be cleared by reading RCREG register and receiving next valid byte)
0 = No framing error

bit 1 **OERR:** Overrun Error bit
1 = Overrun error (can be cleared by clearing bit, CREN)
0 = No overrun error

bit 0 **RX9D:** 9th Bit of Received Data
This can be address/data bit or a parity bit and must be calculated by user firmware.

REGISTER 18-3: BAUDCON: BAUD RATE CONTROL REGISTER

R/W-0	R-1	R/W-0	R/W-0	R/W-0	U-0	R/W-0	R/W-0
ABDOVF	RCIDL	RXDTP	TXCKP	BRG16	—	WUE	ABDEN
bit 7							bit 0

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

bit 7 **ABDOVF:** Auto-Baud Acquisition Rollover Status bit
1 = A BRG rollover has occurred during Auto-Baud Rate Detect mode (must be cleared in software)
0 = No BRG rollover has occurred

bit 6 **RCIDL:** Receive Operation Idle Status bit
1 = Receive operation is Idle
0 = Receive operation is active

bit 5 **RXDTP:** Data/Receive Polarity Select bit
Asynchronous mode:
1 = Receive data (RX) is inverted (active-low)
0 = Receive data (RX) is not inverted (active-high)
Synchronous mode:
1 = Data (DT) is inverted (active-low)
0 = Data (DT) is not inverted (active-high)

bit 4 **TXCKP:** Clock and Data Polarity Select bit
Asynchronous mode:
1 = Idle state for transmit (TX) is a low level
0 = Idle state for transmit (TX) is a high level
Synchronous mode:
1 = Idle state for clock (CK) is a high level
0 = Idle state for clock (CK) is a low level

bit 3 **BRG16:** 16-Bit Baud Rate Register Enable bit
1 = 16-bit Baud Rate Generator – SPBRGH and SPBRG
0 = 8-bit Baud Rate Generator – SPBRG only (Compatible mode), SPBRGH value ignored

bit 2 **Unimplemented:** Read as '0'

bit 1 **WUE:** Wake-up Enable bit
Asynchronous mode:
1 = EUSART will continue to sample the RX pin – interrupt generated on falling edge; bit cleared in hardware on following rising edge
0 = RX pin not monitored or rising edge detected
Synchronous mode:
Unused in this mode.

bit 0 **ABDEN:** Auto-Baud Detect Enable bit
Asynchronous mode:
1 = Enable baud rate measurement on the next character. Requires reception of a Sync field (55h); cleared in hardware upon completion.
0 = Baud rate measurement disabled or completed
Synchronous mode:
Unused in this mode.

FORMULA PER IL BAUD RATE:

TABLE 18-1: BAUD RATE FORMULAS

Configuration Bits			BRG/EUSART Mode	Baud Rate Formula
SYNC	BRG16	BRGH		
0	0	0	8-Bit/Asynchronous	FOSC/[64 (n + 1)]
0	0	1	8-Bit/Asynchronous	FOSC/[16 (n + 1)]
0	1	0	16-Bit/Asynchronous	
0	1	1	16-Bit/Asynchronous	FOSC/[4 (n + 1)]
1	0	x	8-Bit/Synchronous	
1	1	x	16-Bit/Synchronous	

Legend: x = Don't care, n = value of SPBRGH:SPBRG register pair

IMPOSTAZIONI PER ALCUNI VALORI DI BAUD RATE COMUNI:

TABLE 18-3: BAUD RATES FOR ASYNCHRONOUS MODES

BAUD RATE (K)	SYNC = 0, BRGH = 0, BRG16 = 0											
	FOSC = 40.000 MHz			FOSC = 20.000 MHz			FOSC = 10.000 MHz			FOSC = 8.000 MHz		
	Actual Rate (K)	% Error	SPBRG Value (decimal)	Actual Rate (K)	% Error	SPBRG Value (decimal)	Actual Rate (K)	% Error	SPBRG Value (decimal)	Actual Rate (K)	% Error	SPBRG Value (decimal)
0.3	—	—	—	—	—	—	—	—	—	—	—	—
1.2	—	—	—	1.221	1.73	255	1.202	0.16	129	1.201	-0.16	103
2.4	2.441	1.73	255	2.404	0.16	129	2.404	0.16	64	2.403	-0.16	51
9.6	9.615	0.16	64	9.766	1.73	31	9.766	1.73	15	9.615	-0.16	12
19.2	19.531	1.73	31	19.531	1.73	15	19.531	1.73	7	—	—	—
57.6	56.818	-1.36	10	62.500	8.51	4	52.083	-9.58	2	—	—	—
115.2	125.000	8.51	4	104.167	-9.58	2	78.125	-32.18	1	—	—	—

TABLE 18-2: REGISTERS ASSOCIATED WITH BAUD RATE GENERATOR

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Reset Values on page
TXSTA	CSRC	TX9	TXEN	SYNC	SENDB	BRGH	TRMT	TX9D	51
RCSTA	SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D	51
BAUDCON	ABDOVF	RCIDL	—	SCKP	BRG16	—	WUE	ABDEN	51
SPBRGH	EUSART Baud Rate Generator Register, High Byte								51
SPBRG	EUSART Baud Rate Generator Register, Low Byte								51

Legend: — = unimplemented, read as '0'. Shaded cells are not used by the BRG.

I TIMER

Il pic18f2420 contiene al suo interno 4 timer Timer0,1,2,3. Un timer è un dispositivo che può essere utilizzato come contatore (UP) di eventi esterni (ad esempio le transizioni di un bit di una porta) o di eventi interni (ad esempio i fronti del clock della macchina o lo scatto di un comparatore). I timer sono governati nel loro funzionamento da un registro di configurazione, col quale è possibile abilitare il conteggio, impostare un prescaler (ovvero un divisore della frequenza del segnale di clock), e da un registro dati nel quale è contenuto il valore istantaneo del conteggio. Il registro è anche scrivibile in modo da far partire il conteggio da un valore prefissato. I timer possono generare interruzioni a fine conteggio. A tal fine nei registri PIR, PIE, e INTCON sono presenti appositi bit di abiltazione, richiesta e priorità. I timer sono inoltre utilizzabili in strettissima congiunzione con le funzionalità capture, compare e pwm.

TIMER0: 8 o 16 bit

Pin associato: RA4 = clock esterno

REGISTER 11-1: T0CON: TIMER0 CONTROL REGISTER

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
TMR0ON	T08BIT	T0CS	T0SE	PSA	T0PS2	T0PS1	T0PS0
bit 7							bit 0

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

bit 7 **TMR0ON:** Timer0 On/Off Control bit
1 = Enables Timer0
0 = Stops Timer0

bit 6 **T08BIT**: Timer0 8-Bit/16-Bit Control bit
1 = Timer0 is configured as an 8-bit timer/counter
0 = Timer0 is configured as a 16-bit timer/counter

bit 5 **T0CS**: Timer0 Clock Source Select bit
1 = Transition on T0CKI pin
0 = Internal instruction cycle clock (CLKO)

bit 4 **T0SE**: Timer0 Source Edge Select bit
1 = Increment on high-to-low transition on T0CKI pin
0 = Increment on low-to-high transition on T0CKI pin

bit 3 **PSA**: Timer0 Prescaler Assignment bit
1 = TImer0 prescaler is not assigned. Timer0 clock input bypasses prescaler.
0 = Timer0 prescaler is assigned. Timer0 clock input comes from prescaler output.

bit 2-0 **T0PS<2:0>**: Timer0 Prescaler Select bits
111 = 1:256 Prescale value
110 = 1:128 Prescale value
101 = 1:64 Prescale value
100 = 1:32 Prescale value
011 = 1:16 Prescale value
010 = 1:8 Prescale value

TABLE 11-1: REGISTERS ASSOCIATED WITH TIMER0

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Reset Values on page
TMR0L	Timer0 Register Low Byte								50
TMR0H	Timer0 Register High Byte								50
INTCON	GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF	49
T0CON	TMR0ON	T08BIT	T0CS	T0SE	PSA	T0PS2	T0PS1	T0PS0	50
TRISA	RA7(1)	RA6(1)	RA5	RA4	RA3	RA2	RA1	RA0	52

Legend: Shaded cells are not used by Timer0.

Note 1: PORTA<7:6> and their direction bits are individually configured as port pins based on various primary oscillator modes. When disabled, these bits read as '0'.

TIMER1: 16 bit

Pin associato: RC0 = clock esterno

REGISTER 12-1: T1CON: TIMER1 CONTROL REGISTER

R/W-0	R-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
RD16	T1RUN	T1CKPS1	T1CKPS0	T1OSCEN	$\overline{\text{T1SYNC}}$	TMR1CS	TMR1ON
bit 7							bit 0

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

bit 7 **RD16:** 16-Bit Read/Write Mode Enable bit
1 = Enables register read/write of TImer1 in one 16-bit operation
0 = Enables register read/write of Timer1 in two 8-bit operations

bit 6 **T1RUN:** Timer1 System Clock Status bit
1 = Device clock is derived from Timer1 oscillator
0 = Device clock is derived from another source

bit 5-4 **T1CKPS<1:0>:** Timer1 Input Clock Prescale Select bits
11 = 1:8 Prescale value
10 = 1:4 Prescale value
01 = 1:2 Prescale value
00 = 1:1 Prescale value

bit 3 **T1OSCEN:** Timer1 Oscillator Enable bit
1 = Timer1 oscillator is enabled
0 = Timer1 oscillator is shut off
The oscillator inverter and feedback resistor are turned off to eliminate power drain.

bit 2 **$\overline{\text{T1SYNC}}$:** Timer1 External Clock Input Synchronization Select bit
When TMR1CS = 1:
1 = Do not synchronize external clock input
0 = Synchronize external clock input
When TMR1CS = 0:
This bit is ignored. Timer1 uses the internal clock when TMR1CS = 0.

bit 1 **TMR1CS:** Timer1 Clock Source Select bit
1 = External clock from pin RC0/T1OSO/T13CKI (on the rising edge)
0 = Internal clock (FOSC/4)

bit 0 **TMR1ON:** Timer1 On bit
1 = Enables Timer1
0 = Stops Timer1

TABLE 12-2: REGISTERS ASSOCIATED WITH TIMER1 AS A TIMER/COUNTER

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Reset Values on page
INTCON	GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF	49
PIR1	PSPIF(1)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	52
PIE1	PSPIE(1)	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	52
IPR1	PSPIP(1)	ADIP	RCIP	TXIP	SSPIP	CCP1IP	TMR2IP	TMR1IP	52
TMR1L	Timer1 Register Low Byte								50
TMR1H	Timer1 Register High Byte								50
T1CON	RD16	T1RUN	T1CKPS1	T1CKPS0	T1OSCEN	$\overline{\text{T1SYNC}}$	TMR1CS	TMR1ON	50

Legend: Shaded cells are not used by the Timer1 module.
Note 1: These bits are unimplemented on 28-pin devices; always maintain these bits clear.

TIMER2: 8 bit con prescaler e postscaler

REGISTER 13-1: T2CON: TIMER2 CONTROL REGISTER

U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	T2OUTPS3	T2OUTPS2	T2OUTPS1	T2OUTPS0	TMR2ON	T2CKPS1	T2CKPS0
bit 7							bit 0

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

bit 7 **Unimplemented:** Read as '0'

bit 6-3 **T2OUTPS<3:0>:** Timer2 Output Postscale Select bits
0000 = 1:1 Postscale
0001 = 1:2 Postscale
•
•
•
1111 = 1:16 Postscale

bit 2 **TMR2ON:** Timer2 On bit
1 = Timer2 is on
0 = Timer2 is off

bit 1-0 **T2CKPS<1:0>:** Timer2 Clock Prescale Select bits
00 = Prescaler is 1
01 = Prescaler is 4
1x = Prescaler is 16

FIGURE 13-1: TIMER2 BLOCK DIAGRAM

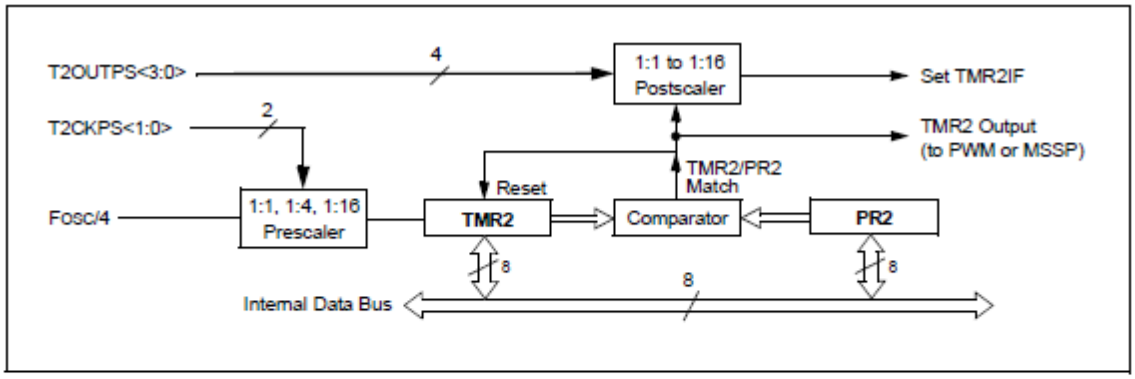


TABLE 13-1: REGISTERS ASSOCIATED WITH TIMER2 AS A TIMER/COUNTER

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Reset Values on page
INTCON	GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF	49
PIR1	PSPIF(1)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	52
PIE1	PSPIE(1)	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	52
IPR1	PSPIP(1)	ADIP	RCIP	TXIP	SSPIP	CCP1IP	TMR2IP	TMR1IP	52
TMR2	Timer2 Register								50
T2CON	—	T2OUTPS3	T2OUTPS2	T2OUTPS1	T2OUTPS0	TMR2ON	T2CKPS1	T2CKPS0	50
PR2	Timer2 Period Register								50

Legend: — = unimplemented, read as '0'. Shaded cells are not used by the Timer2 module.
Note 1: These bits are unimplemented on 28-pin devices; always maintain these bits clear.

TIMER3: 16 bit

Pin associato: RC0 = clock esterno

REGISTER 14-1: T3CON: TIMER3 CONTROL REGISTER

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
RD16	T3CCP2	T3CKPS1	T3CKPS0	T3CCP1	$\overline{\text{T3SYNC}}$	TMR3CS	TMR3ON
bit 7							bit 0

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

bit 7 **RD16:** 16-Bit Read/Write Mode Enable bit
1 = Enables register read/write of Timer3 in one 16-bit operation
0 = Enables register read/write of Timer3 in two 8-bit operations

bit 6,3 **T3CCP<2:1>:** Timer3 and Timer1 to CCPx Enable bits
1x = Timer3 is the capture/compare clock source for the CCP modules
01 = Timer3 is the capture/compare clock source for CCP2;
Timer1 is the capture/compare clock source for CCP1
00 = Timer1 is the capture/compare clock source for the CCP modules

bit 5-4 **T3CKPS<1:0>**: Timer3 Input Clock Prescale Select bits
11 = 1:8 Prescale value
10 = 1:4 Prescale value
01 = 1:2 Prescale value
00 = 1:1 Prescale value

bit 2 **T3SYNC:** Timer3 External Clock Input Synchronization Control bit
(Not usable if the device clock comes from Timer1/Timer3.)
When TMR3CS = 1:
1 = Do not synchronize external clock input
0 = Synchronize external clock input
When TMR3CS = 0:
This bit is ignored. Timer3 uses the internal clock when TMR3CS = 0.

bit 1 **TMR3CS:** Timer3 Clock Source Select bit
1 = External clock input from Timer1 oscillator or T13CKI (on the rising edge after the first falling edge)
0 = Internal clock (FOSC/4)

bit 0 **TMR3ON:** Timer3 On bit
1 = Enables Timer3
0 = Stops Timer3

TABLE 14-1: REGISTERS ASSOCIATED WITH TIMER3 AS A TIMER/COUNTER

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Reset Values on page
INTCON	GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF	49
PIR2	OSCFIF	CMIF	—	EEIF	BCLIF	HLVDIF	TMR3IF	CCP2IF	52
PIE2	OSCFIE	CMIE	—	EEIE	BCLIE	HLVDIE	TMR3IE	CCP2IE	52
IPR2	OSCFIP	CMIP	—	EEIP	BCLIP	HLVDIP	TMR3IP	CCP2IP	52
TMR3L	Timer3 Register Low Byte								51
TMR3H	Timer3 Register High Byte								51
T1CON	RD16	T1RUN	T1CKPS1	T1CKPS0	T1OSCEN	$\overline{\text{T1SYNC}}$	TMR1CS	TMR1ON	50
T3CON	RD16	T3CCP2	T3CKPS1	T3CKPS0	T3CCP1	$\overline{\text{T3SYNC}}$	TMR3CS	TMR3ON	51

Legend: — = unimplemented, read as '0'. Shaded cells are not used by the Timer3 module.

IL MODULO CCP (capture-compare-pwm)

Input Capture: il valore del contatore del timer viene congelato in un registro, detto di cattura, all'accadere di un evento particolare come ad esempio una transizione su un pin dedicato di una porta. Questa modalità può servire per la misurazione di intervalli di tempo.

Output Compare: il valore del contatore del timer viene continuamente confrontato con il valore contenuto in un registro detto di confronto; quando il contatore diventa uguale a tale registro viene intrapesa una azione, ad esempio la commutazione di un pin esterno. Questa modalità può servire per la generazione di forme d'onda.

PWM: il contenuto del contatore di un timer viene confrontato con due registri: uno detto del duty cycle e l'altro del periodo: quando il contatore diventa uguale a quello del duty cycle un pin predefinito esterno viene impostato; quando il contatore diventa uguale al registro del periodo il contatore viene resettato e il pin esterno viene ricommutato. Questa modalità può servire al controllo di potenza o alla conversione DA (con un filtro passa basso) o alla comunicazione dati (modulazione PWM).

Nel 18f2420 vi sono due moduli CCP eguali CCP1 e CCP2. I moduli funzionano in associazione con i timer come descritto dalla seguente tabella:

I pin associati a questo modulo sono RC1, RC2 e RB3.

TABLE 15-1: CCP MODE – TIMER RESOURCE

CCP/ECCP Mode	Timer Resource
Capture	Timer1 or Timer3
Compare	Timer1 or Timer3
PWM	Timer2

REGISTER 15-1: CCPxCON: CCPx CONTROL REGISTER (28-PIN DEVICES)

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	DCxB1	DCxB0	CCPxM3	CCPxM2	CCPxM1	CCPxM0
bit 7							bit 0

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

bit 7-6 **Unimplemented:** Read as '0'

bit 5-4 **DCxB<1:0>:** PWM Duty Cycle bit 1 and bit 0 for CCPx Module

Capture mode:
Unused.

Compare mode:
Unused.

PWM mode:
These bits are the two LSbs (bit 1 and bit 0) of the 10-bit PWM duty cycle. The eight MSbs (DCxB<9:2>) of the duty cycle are found in CCPRxL.

bit 3-0 **CCPxM<3:0>:** CCPx Module Mode Select bits

0000 = Capture/Compare/PWM disabled (resets CCPx module)
0001 = Reserved
0010 = Compare mode, toggle output on match (CCPxIF bit is set)
0011 = Reserved
0100 = Capture mode, every falling edge
0101 = Capture mode, every rising edge
0110 = Capture mode, every 4th rising edge
0111 = Capture mode, every 16th rising edge
1000 = Compare mode, initialize CCPx pin low; on compare match, force CCPx pin high (CCPxIF bit is set)
1001 = Compare mode, initialize CCPx pin high; on compare match, force CCPx pin low (CCPxIF bit is set)
1010 = Compare mode, generate software interrupt on compare match (CCPxIF bit is set, CCPx pin reflects I/O state)
1011 = Compare mode, trigger special event; reset timer; CCP2 match starts A/D conversion (CCPxIF bit is set)
11xx = PWM mode

TABLE 15-3: REGISTERS ASSOCIATED WITH CAPTURE, COMPARE, TIMER1 AND TIMER3

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Reset Values on page
INTCON	GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF	49
RCON	IPEN	SBOREN	—	$\overline{RI}$	$\overline{TO}$	$\overline{PD}$	$\overline{POR}$	$\overline{BOR}$	48
PIR1	PSPIF(1)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	52
PIE1	PSPIE(1)	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	52
IPR1	PSPIP(1)	ADIP	RCIP	TXIP	SSPIP	CCP1IP	TMR2IP	TMR1IP	52
PIR2	OSCFIF	CMIF	—	EEIF	BCLIF	HLVDIF	TMR3IF	CCP2IF	52
PIE2	OSCFIE	CMIE	—	EEIE	BCLIE	HLVDIE	TMR3IE	CCP2IE	52
IPR2	OSCFIP	CMIP	—	EEIP	BCLIP	HLVDIP	TMR3IP	CCP2IP	52
TRISB	PORTB Data Direction Register								52
TRISC	PORTC Data Direction Register								52
TMR1L	Timer1 Register Low Byte								50
TMR1H	Timer1 Register High Byte								50
T1CON	RD16	T1RUN	T1CKPS1	T1CKPS0	T1OSCEN	$\overline{T1SYNC}$	TMR1CS	TMR1ON	50
TMR3H	Timer3 Register High Byte								51
TMR3L	Timer3 Register Low Byte								51
T3CON	RD16	T3CCP2	T3CKPS1	T3CKPS0	T3CCP1	$\overline{T3SYNC}$	TMR3CS	TMR3ON	51
CCPR1L	Capture/Compare/PWM Register 1 Low Byte								51
CCPR1H	Capture/Compare/PWM Register 1 High Byte								51
CCP1CON	P1M1(1)	P1M0(1)	DC1B1	DC1B0	CCP1M3	CCP1M2	CCP1M1	CCP1M0	51
CCPR2L	Capture/Compare/PWM Register 2 Low Byte								51
CCPR2H	Capture/Compare/PWM Register 2 High Byte								51
CCP2CON	—	—	DC2B1	DC2B0	CCP2M3	CCP2M2	CCP2M1	CCP2M0	51

Legend: — = unimplemented, read as '0'. Shaded cells are not used by Capture/Compare, Timer1 or Timer3.
Note 1: These bits are unimplemented on 28-pin devices; always maintain these bits clear.

TABLE 15-5: REGISTERS ASSOCIATED WITH PWM AND TIMER2

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Reset Values on page
INTCON	GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF	49
RCON	IPEN	SBOREN	—	$\overline{RI}$	$\overline{TO}$	$\overline{PD}$	$\overline{POR}$	$\overline{BOR}$	48
PIR1	PSPIF(1)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	52
PIE1	PSPIE(1)	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	52
IPR1	PSPIP(1)	ADIP	RCIP	TXIP	SSPIP	CCP1IP	TMR2IP	TMR1IP	52
TRISB	PORTB Data Direction Register								52
TRISC	PORTC Data Direction Register								52
TMR2	Timer2 Register								50
PR2	Timer2 Period Register								50
T2CON	—	T2OUTPS3	T2OUTPS2	T2OUTPS1	T2OUTPS0	TMR2ON	T2CKPS1	T2CKPS0	50
CCPR1L	Capture/Compare/PWM Register 1 Low Byte								51
CCPR1H	Capture/Compare/PWM Register 1 High Byte								51
CCP1CON	P1M1(1)	P1M0(1)	DC1B1	DC1B0	CCP1M3	CCP1M2	CCP1M1	CCP1M0	51
CCPR2L	Capture/Compare/PWM Register 2 Low Byte								51
CCPR2H	Capture/Compare/PWM Register 2 High Byte								51
CCP2CON	—	—	DC2B1	DC2B0	CCP2M3	CCP2M2	CCP2M1	CCP2M0	51
ECCP1AS	ECCPASE	ECCPAS2	ECCPAS1	ECCPAS0	PSSAC1	PSSAC0	PSSBD1(1)	PSSBD0(1)	51
PWM1CON	PRSEN	PDC6(1)	PDC5(1)	PDC4(1)	PDC3(1)	PDC2(1)	PDC1(1)	PDC0(1)	51

Legend: — = unimplemented, read as '0'. Shaded cells are not used by PWM or Timer2.
Note 1: These bits are unimplemented on 28-pin devices; always maintain these bits clear.

Le interruzioni da transizione su pin esterno: INT0, INT1, INT2

Il micro dispone di 3 pin (RB0, RB1 e RB2) che possono essere programmati per generare una interruzione in risposta ad una transizione ad essi applicata. Questa funzionalità può essere utilizzata per far si che il microcontrollore risponda istantaneamente (a parte la latenza del sistema di interruzione) ad uno stimolo esterno avviando una routine di gestione apposita (ad esempio salvataggio in memoria non volatile se un pin segnala, andando basso, che sta andando via la alimentazione). E' possibile selezionare se la transizione che scatena l'interruzione è un fronte in salita (default) o in discesa mediante i bit INTEGDx nel registro INTCON2 (x è 0,1 o 2). La funzionalità va abilitata impostando a 1 il bit INTxIE nel registro PIINTCON3, la priorità nel registro INTCON3 con i bit INTxIP. La routine di interruzione testerà e poi azzererà il bit INTxIF nel registro INTCON3. Si noti che INT0 è sempre e solo ad alta priorità. Le 3 interruzioni sono poi soggette al mascheramento complessivo con GIEH e GIEL di INTCON1.

LA EEPROM

E' possibile memorizzare dati in modo permanente nella EEPROM (memoria non volatile) a bordo del micro. Questi dati permangono e possono essere richiamati anche dopo un reset o dopo aver tolto l'alimentazione al micro. La EEPROM, nel suo funzionamento base, è controllabile con: un registro indirizzo (EEADDR) che punta alla cella (sono 256 nel 18f2420) in cui si vuol scrivere; un registro dati che contiene il dato da leggere o scrivere (EEDATA); un registro stato/comando/controllo (EECON1(R)-EECON2(W)) con il quale è possibile avviare le operazioni di lettura o scrittura e controllare lo stato della loro esecuzione. Si noti che la scrittura necessita dell'invio di una sequenza particolare per evitare che essa avvenga per errore (ad esempio a causa di un programma "impazzito", o di un malfunzionamento del software).

REGISTER 7-1: EECON1: EEPROM CONTROL REGISTER 1

R/W-x	R/W-x	U-0	R/W-0	R/W-x	R/W-0	R/S-0	R/S-0
EEPGD	CFGS	—	FREE	WRERR(1)	WREN	WR	RD
bit 7							bit 0

Legend:	S = Settable bit (cannot be cleared in software)		
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

bit 7 **EEPGD:** Flash Program or Data EEPROM Memory Select bit
1 = Access Flash program memory
0 = Access data EEPROM memory

bit 6 **CFGS:** Flash Program/Data EEPROM or Configuration Select bit
1 = Access Configuration registers
0 = Access Flash program or data EEPROM memory

bit 5 **Unimplemented:** Read as '0'

bit 4 **FREE:** Flash Row Erase Enable bit
1 = Erase the program memory row addressed by TBLPTR on the next WR command (cleared by completion of erase operation)
0 = Perform write only

bit 3 **WRERR:** Flash Program/Data EEPROM Error Flag bit(1)
1 = A write operation is prematurely terminated (any Reset during self-timed programming in normal operation, or an improper write attempt)
0 = The write operation completed

bit 2 **WREN:** Flash Program/Data EEPROM Write Enable bit
1 = Allows write cycles to Flash program/data EEPROM
0 = Inhibits write cycles to Flash program/data EEPROM

bit 1 **WR:** Write Control bit
1 = Initiates a data EEPROM erase/write cycle or a program memory erase cycle or write cycle (The operation is self-timed and the bit is cleared by hardware once write is complete. The WR bit can only be set (not cleared) in software.)
0 = Write cycle to the EEPROM is complete

bit 0 **RD:** Read Control bit
1 = Initiates an EEPROM read (Read takes one cycle. RD is cleared in hardware. The RD bit can only be set (not cleared) in software. RD bit cannot be set when EEPGD = 1 or CFGS = 1.)
0 = Does not initiate an EEPROM read

Note 1: When a WRERR occurs, the EEPGD and CFGS bits are not cleared. This allows tracing of the error condition.

SEQUENZE (assembly) per la lettura e la scrittura:

EXAMPLE 7-1: DATA EEPROM READ

```
MOVLW   DATA_EE_ADDR   ;
MOVWF   EEADR          ; Data Memory Address to read
BCF     EECON1, EEPGD  ; Point to DATA memory
BCF     EECON1, CFGS   ; Access EEPROM
BSF     EECON1, RD     ; EEPROM Read
MOVF    EEDATA, W      ; W = EEDATA
```

EXAMPLE 7-2: DATA EEPROM WRITE

```
                MOVLW   DATA_EE_ADDR    ;
                MOVWF   EEADR           ; Data Memory Address to write
                MOVLW   DATA_EE_DATA    ;
                MOVWF   EEDATA          ; Data Memory Value to write
                BCF     EECON1, EEPGD   ; Point to DATA memory
                BCF     EECON1, CFGS    ; Access EEPROM
                BSF     EECON1, WREN    ; Enable writes

                BCF     INTCON, GIE     ; Disable Interrupts
                MOVLW   55h             ;
Required        MOVWF   EECON2          ; Write 55h
Sequence        MOVLW   0AAh            ;
                MOVWF   EECON2          ; Write 0AAh
                BSF     EECON1, WR      ; Set WR bit to begin write
                BSF     INTCON, GIE     ; Enable Interrupts

                                        ; User code execution
                BCF     EECON1, WREN    ; Disable writes on write complete (EEIF set)
```

COME SI PROGRAMMA

Il microcontrollore può essere programmato in qualsiasi linguaggio: basic, C, pascal, assembly....Tuttavia la sua architettura è ottimizzata per essere programmato in C. Volendolo programmare in assembly abbiamo il vantaggio (l'unico) della gratuità dell'assemblatore fornito in freeware dalla microchip. Il compilatore C della microchip è freeware ma con delle limitazioni sulla dimensione del codice generabile, sulle librerie e sui modelli di processore. Tuttavia per applicazioni semplici di tipo didattico questo non è un problema. Un ottimo compilatore C, ma caro, è quello della IAR. La microchip fornisce il compilatore MCC18, perfettamente integrato con l'ambiente di sviluppo MPLAB IDE 8.04 ma abbastanza pieno di bug. Uno clamoroso che trae in grossi equivoci è il malfunzionamento delle variabili statiche locali, che, dichiarate dentro una funzione, viene reinizializzata tutte le volte invece che solo la prima!!! Tuttavia è possibile costruire del codice funzionante. Esistono anche compilatori OpenSource basati sul compilatore gcc di Unix.

Si noti che i compilatori mettono a disposizione un file .h per ogni modello di microcontrollore gestito dal compilatore contenente le definizioni in termini di indirizzo di memoria di tutti i registri del microcontrollore. Ad esempio TRISA equivale a qualcosa del tipo: #define *(unsigned char *)0x30.

Questo ci consente di scrivere comodamente TRISA = 0x015 e non più *(unsigned char *)0x30 = 0x15.

Inoltre nel file .h (ad esempio pic18f2420.h) sono dichiarati degli UNION e degli STRUCT che rappresentano con i loro campi i singoli bit che costituiscono i vari registri. Questi bit ci consentono di utilizzare notazioni del tipo INTCONbits.GIEH = 1 per impostare ad 1 il bit 7 di INTCON invece di dover ricorrere ad AND, OR, e mascherature varie. Dovrei altrimenti scrivere ad esempio INTCON |= (1 << GIEH) per impostarlo a 1 e INTCON &= ~(1 << GIEH) per resettarlo.

In ogni caso, una volta scelto il linguaggio e quindi il compilatore o l'assemblatore, si arriva alla generazione del file eseguibile più o meno con lo stesso processo seguito per la generazione di un file Visual C: creazione del progetto, scelta del microprocessore, popolazione con i file sorgenti, compilazione e linking. Il file generato ha estensione .HEX (non exe) in quanto è scritto in un formato particolare che contiene solo caratteri esadecimali da 0 a F e il : come delimitatore. Questo file deve essere “scaricato” nella EPROM del microcontrollore. A tal fine possiamo:

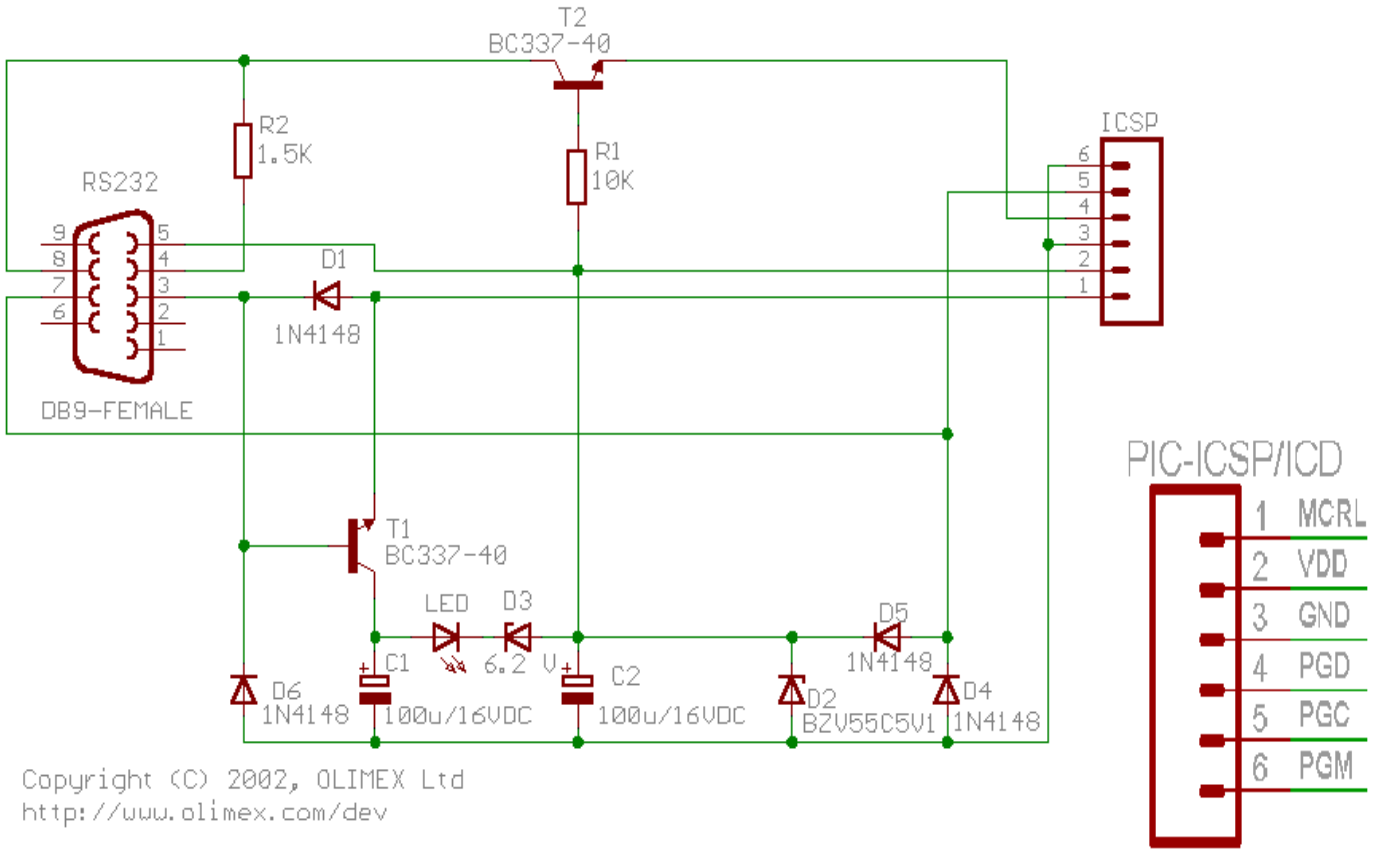
1) scaricarlo in modo parallelo utilizzando un apposito programmatore collegato al PC. Il microcontrollore va inserito in un apposito zoccolo presente sul programmatore. Questo metodo può funzionare solo per il package DIP, a, meno di acquistare costosi adattatori. Inoltre il mocro va ogni volta tolto dal target (la scheda applicativa), inserito nel programmatore, cancellato, riprogrammato e reinserito nel target. Pensiamo che nello sviluppo di un programma è possibile sbagliare migliaia di volte e capiamo che questo metodo è da disperati.
2) Scaricarlo in modo seriale usando la interfaccia seriale sincrona appositamente prevista per la programmazione e il debug (PIN PGM, PGC, PGD, MCLR). In questo modo non occorre rimuovere il micro dal target; siparla di programmazione in circuit (ICP o ISP). Occorre però procurarsi o costruirsi un piccolo adattatore (si trovano centinaia di schemi su internet) da collegare alla seriale asincrona del PC o all'USB.
3) Utilizzare un bootloader ovvero un piccolo programmino che viene memorizzato un volta per tutte in una porzione della EPROM e non viene più cancellato. Tale programmino, che viene scaricato con uno dei due metodi precedenti (quindi comunque ci risiamo, abbiamo bisogno, almeno una volta, un programmatore), ma una tantum, può ricevere i byte del nuovo programma dalla seriale asincrona. Occorre però notare che la seriale asincrona esce dal pc a livelli elettrici RS232 e non TTL e quindi occorre sempre e comunque costruire un adattatore di livello. In più occorre anche il bootloader. Questo metodo conviene quindi se: 1) possiamo sacrificare un po' di EPROM per il bootloader; 2) abbiamo già necessità di un collegamento seriale asincrono e quindi ci serve comunque l'adattatore. Anche in questo caso comunque non è necessario rimuovere il micro dal target; 3) abbiamo un pc senza seriale asincrona, ormai i notebook sono tutti provvisti solo di porte USB. Tuttavia esistono degli adattatori USB seriale asincrona che funzionano benissimo. Esistono infine dei piccoli programmatori per microcontrollori che si connettono all'usb dal costo di una ventina di euro.

Un programmatore a basso costo.

Se rinunciamo alle possibilità di debug in tempo reale offerte dai dispositivi relativamente costosi della microchip e ci accontentiamo di programmare alla cieca con un processo di “trial and error” possiamo utilizzare un dispositivo che ci consente la sola programmazione, a basso costo (non il debug).

Circuito elettrico di un programmatore da connettere alla porta seriale asincrona RS232 di un PC per programmare il PIC18F2420 (e non solo). Il connettore ICSP (in circuit serial programming) reca le connessioni per i pin di programmazione del micro.

Il programmatore va utilizzato in congiunzione ad un adeguato programma per il download dell'eseguibile che sia compatibile con il pic in oggetto e con tale programmatore. Ad esempio il WinPicPgm. Occorre installarlo, configurarlo, selezionare il micro voluto (ma dovrebbe essere in grado di rilevare il tutto da solo) caricare il file hex da programmare e avviare il download.



Programmi di Esempio

I programmi che seguono sono stati sviluppati con il compilatore C MCC18 della Microchip e provati con il simulatore interattivo PROTEUS ISIS della LABCENTER UK.

ELENCO ESEMPI:

1) pic18f2420 blink1 : lampeggio led con ritardo sw
2) pic18f2420 bal1 : eco tasti su led
3) pic18f2420 blink2 : lampeggio led con timer hw e interrupt
4) pic18f2420 AD1 : acquisizione e trasmissione su seriale asy
5) pic18f2420 expramp ad1 : conv a rampa exp con comparatore e seriale
6) pic18f2420 echo1 : ricezione e trasmissione a polling libreria
7) pic18f2420 UART1 : ricezione e trasmissione a polling
8) pic18f2420 UART2 : ricezione e trasmissione a interrupt
9) pic18f2420 extint1 : conteggio impulsi esterni ad interrupt
10) pic18f2420 ic1 : input capture (misura intervallo tempo)
11) pic18f2420 oc1 : esempio output compare
12) pic18f2420 pwm1 : esempio pwm con due tasti incr-decr
13) pic18f2420 triangle : generazione onda triangolare
14) pic18f2420 sine : generazione onda sinusoidale
15) pic18f2420 ee1 : ricezione stringa scrittura in ee ritrasmissione
16) pic18f2420 bal2bal : due sistemi bal che si scambiano lo stato di led e tasti attraverso la linea seriale
17) pic18f2420 bal2bal3 : due stistemi gemelli che si scambiano lo stato di canali analogici e led attraverso una linea seriale
18) pic18f2420 clock : orologio hh:mm:ss con display multiplexato
19) polling versus interrupt comparison
20) touch dimmer light control (phase control)
21) touch dimmer light control (burst control)

LISTATI A SEGUIRE

STRUTTURA TIPICA DI UN PROGRAMMA

// NOME PROGRAMMA, DATA, REVISIONE
// SCOPO, DESCRIZIONE
// STRUTTURA DEL SOFTWARE, MODULI, FILE E FUNZIONI CHE LO COMPONGONO, LIBRERIE NECESSARIE

#include <p18cxxx.h> // sempre per la famiglia 18F; contiene le macro per l'accesso ai registri usando simboli come PORTA al posto di indirizzi hex

#include <altri include file>

#define <definizioni costanti>

typedef <definizioni di tipi>

<dichiarazioni variabili globali (visibili in tutti i file dove sono esternalizzate)>

<dichiarazioni variabili globali statiche (visibili solo nel file che le dichiara)>

<dichiarazioni variabili esterne>

<PROTOTIPI DELLE FUNZIONI ACCESSORIE (zero o più)>

<FUNZIONE main (1 e una sola in un solo file)>

void main() { <inizializzazione periferiche necessarie e processore> <ciclo while(1) { < gestione del sistema> } > } // non può finire perchè non c'è un S.O.

<zero o più funzioni accessorie (corpi)> // è buona norma inizializzare le periferiche e/o il processore con funzioni a parte, es init_mcu, init_uart etc.

// SE SI USANO INTERRUZIONI la struttura sarà: (1) main e funzioni_normali (2) funzioni_interruzione (3) impostazione_vettori; occorre quindi aggiungere, dopo (1)
(2)
#pragma interrupt lista_nomi_routine_interruzione_separati_da_spazi (routine1 routine2 ….) // segnalo che queste routine salvano anche la PSW

<definizione (=corpo) della routine di interrupt per le sorgenti ad alta priorità>

<definizione (=corpo) della routine di interrupt per le sorgenti a bassa priorità>

(3)
// IMPOSTAZIONE VETTORI DI INTERRUZIONE CON SALTI ALLE VERE ROUTINE

#pragma code SaltoGestioneInterruptAltaPriorita = 0x08

void SaltoGestioneInterruptAltaPriorita() { GestioneInterruzioniAltaPriorita(); }

#pragma code SaltoGestioneInterruptBassPriorita = 0x18

void SaltoGestioneInterruptBassaPriorita() { GestioneInterruzioniBassaPriorita(); }

esempio gestione interruzioni: (DOPO IL MAIN e le funzioni accessorie)
(2)
#pragma interrupt GestioneInterruzioniAltaPriorita GestioneInterruzioniBassaPriorita // segnalo che queste routine salvano anche la PSW nello stack

```
void GestioneInterruzioniAltaPriorita()
{
        <esame registro di stato delle interruzioni per capire quale periferica interrompe, mediante una serie di if sui bit dei vari registri >

        // per ogni periferica gestita a interruzione e per ogni sottosistema abilitato:

        if(bit_registro_stato_interruzioni_periferica_X è alto)
        {
                <esegui gestione>

                <azzera_bit_registro_di_stato_in_questione>  // IMPORTANTE, altrimenti la ri-serve in continuazione....
        }
}

void GestioneInterruzioniBassaPriorita()  { /* analogamente a sopra ma per le periferiche definite a bassa priorità */ }
```

// IMPOSTAZIONE VETTORI DI INTERRUZIONE CON SALTI ALLE VERE ROUTINE (IN FONDO AL FILE)
(3)
#pragma code SaltoGestioneInterruptAltaPriorita = 0x08 // riempo la parola a 0x08 con un salto a GestioneInterruzioniAltaPriorita

void SaltoGestioneInterruptAltaPriorita() { GestioneInterruzioniAltaPriorita(); }

#pragma code SaltoGestioneInterruptBassPriorita = 0x18 // riempo la parola a 0x08 con un salto a GestioneInterruzioniBassaPriorita

void SaltoGestioneInterruptBassaPriorita() { GestioneInterruzioniBassaPriorita(); }

```
1.  // BLINK1.c
2.  #include <p18cxxx.h>   /* for TRISA and PORTA declarations */
3.
4.  /* Set configuration bits for use with ICD2 / PICDEM2 PLUS Demo Board:
5.   *  - set INTIO2 oscillator
6.   *  - disable watchdog timer
7.   *  - disable low voltage programming
8.   */
9.  #pragma config OSC = INTIO67
10. #pragma config WDT = OFF
11. #pragma config LVP = OFF
12.
13. void main()
14. {
15.  int i, j;
16.
17.  //ADCON1 = 7; // no AD sulla porta
18.  CMCON  = 7; // no comparatore sulla porta
19.
20.  TRISA = 0x00; // porta A in uscita
21.
22.  while(1)
23.  {
24.              for(i = 0; i < 10; i++)
25.               //  for(j = 0; j < 1000; j++)
26.              ;
27.              PORTA = 0x00;
28.
29.              for(i = 0; i < 10; i++)
30.              //   for(j = 0; j < 1000; j++)
31.              ;
32.              PORTA = 0xFF;
33.  }
34. }
```

35. // ECHO1.c comunicazione seriale asincrona 9600 N 8 1

```
#include <p18cxxx.h>   /* for TRISA and PORTA declarations */
#include <stdio.h>
#include <usart.h>

/* Set configuration bits for use with ICD2 / PICDEM2 PLUS Demo Board:
 *  - set HS oscillator
 *  - disable watchdog timer
 *  - disable low voltage programming
 */
#pragma config OSC = HS
#pragma config WDT = OFF
#pragma config LVP = OFF

char getch(void);  // funzione che riceve un carattere da RxD
void putch(char ch); // funzione che invia un carattere su TxD

void main()
{  // ATTENZIONE: INIZIALIZZAZIONE UART:
 OpenUSART( USART_TX_INT_OFF  &  /* no TX interrupt                    */
            USART_RX_INT_OFF  &  /* no RX interrupt                    */
            USART_ASYNCH_MODE &  /* UART mode                          */
            USART_EIGHT_BIT   &  /* 8 bit No parity 1 stop             */
            USART_CONT_RX     &  /* non disabilita dopo 1 RX           */
            USART_BRGH_HIGH,     /* baud rate alto                     */
            51 );              /* baud rate = (Fclock / 1024) - 1    : 9600   */

 putch('H'); putch('e'); putch('y'); putch('!'); putch('!'); putch(0x0a);putch(0x0d);

 puts("Ciao! Questo è il programma ECHO1...premi un tasto qualsiasi sulla tastiera...");

 while(1) putch(getch());
}

char getch(){ while(!DataRdyUSART()); return(getcUSART()); } // se il registro RX è pieno lo legge sennò aspetta

void putch(char ch){ while(BusyUSART()); putc(ch, stdout); } // se il registro TX è vuoto spedisce, sennò aspetta
```

72. // BAL1 (buttons and lights.c

```
73.  #include <p18cxxx.h>   /* for TRISA and PORTA declarations */
74.
75.  /* Set configuration bits for use with ICD2 / PICDEM2 PLUS Demo Board:
76.   *  - set INTIO2 oscillator
77.   *  - disable watchdog timer
78.   *  - disable low voltage programming
79.   */
80.  #pragma config OSC = INTIO67
81.  #pragma config WDT = OFF
82.  #pragma config LVP = OFF
83.
84.  void main()
85.  {
86.  // ADCON1 = 7; // no AD sulla porta
87.   CMCON  = 7; // no comparatore sulla porta
88.   TRISA = 0x00; // porta A in uscita
89.   TRISB = 0xff; // porta B in ingresso
90.
91.   while(1) PORTA = PORTB; // ciclo senza fine In cui leggo la porta B e la riecheggio sulla A
92.  }
93.
```

94. // AD1.c

```
95.  // ACQUISIZIONE AD DA DUE CANALI E TRASMISSIONE SU SERIALE
96.  #include <p18cxxx.h>   /* for TRISA and PORTA declarations */
97.  #include <stdio.h>
98.  #include <usart.h>
99.
100. #pragma config OSC = HS
101. #pragma config WDT = OFF
102. #pragma config LVP = OFF
103.
104. void main()
105. {
106.  unsigned int val0, val1;
107.  unsigned int hb, lb;
108.
109.  OpenUSART( USART_TX_INT_OFF  &  /* no TX interrupt                  */  // ATTENZIONE: INIZIALIZZAZIONE UART:
110.            USART_RX_INT_OFF  &  /* no RX interrupt                  */
111.            USART_ASYNCH_MODE &  /* UART mode                        */
112.            USART_EIGHT_BIT   &  /* 8 bit No parity 1 stop           */
113.            USART_CONT_RX     &  /* non disabilita dopo 1 RX         */
114.            USART_BRGH_HIGH,     /* baud rate alto                   */
115.            51 );                /* baud rate = (Fclock / 1024) - 1  */
116. puts("Ciao! Questo è il programma AD1:");
117.  ADCON1 = 0x0e; // + e - vrif da alimentazione, ch 0 porta A.0 ingresso analogico
118.
119.  while(1)
120.  {
121.   ADCON0 = 0x03;  // SOC:   0x02 è la maschera per il bit GO in ADCON0...seleziono il canale 0, abilito l'AD
122.
123.   while(ADCON0 & 0x02); // attesa EOC
124.   // leggo il risultato che è giustificato a sinistra
125.   lb = ADRESL; // leggo byte basso
126.   hb = ADRESH; // leggo byte alto (solo i due bit alti sono signif.)
127.
128.   val0 = ((hb << 8) | lb) >> 6; // assemblo in un intero a 10 bit giustif. a destra
129.
130.   ADCON0 = 0x07;  // SOC:   0x02 è la maschera per il bit GO in ADCON0...seleziono il canale 1, abilito l'AD
131.
132.   while(ADCON0 & 0x02); // attesa EOC
133.
134.   lb = ADRESL;       hb = ADRESH;
135.
136.   val1 = ((hb << 8) | lb) >> 6;
137.
138.   printf("ch0: %u ch1: %u \x0a\x0d", val0, val1);
139.  }
140. }
```

```
// BLINK2.c Lampeggio con timer ad Interruzione
//
// Blink2 fa lampeggiare 10 led (8 sulla porta A e 2 sulla B) usando il timer0
//        e le interruzioni (no priorità)
//
//-------------------------------------------------------------------------
#pragma config OSC = INTIO67 // oscillatore interno
#pragma config WDT = OFF
#pragma config LVP = OFF

#include <p18cxxx.h>

#define PH 0xfc // valori per contare 1000 impulsi: il contatore è up modulo
#define PL 0x18 // 65536: ci metto 64536 (0xfc18) così gli rimangono 1000 impulsi per l'overflow

volatile unsigned int timer = 0; // variabile globale per contare gli overflow

//-------------------------------------------------------------------------

void main ()
{
  // ADCON1 = 7; // no AD sulla porte
  // CMCON  = 7; // no comparatore sulla porta

  INTCON = 0x00;              // disabilito tutti le interruzioni e quelle del TMR0
                              // interruzioni in modo compatibile 16f84 senza priorità

  T0CON = 0x82;               //  set up timer0 - prescaler 1:8 --> 1MHz --> 8us x count
  TMR0H = PH;                 // setup period high byte
  TMR0L = PL;                 // setup period low byte 1 TOF = 8 ms

  TRISB = 0;                  // porta A e B in uscita...
  PORTB = 0x00;
  TRISA = 0;
  PORTA = 0x00;

  INTCON = 0xa0;              // abilito le interruzioni gloali e da TMR0   ALTERNATIVAMENTE INTCONbits.TMR0IE = 1;  e INTCONbits.GIE = 1;

  while (1)
   {
     timer = 20; // per avere 1 s ci dovrei mettere 125, ma la simulazione è più lenta...
     while(timer);  // aspetto che la routine di interrupt mi abbia azzerato timer...

     LATB ^= 0xff; // commuto le porte...
     LATA ^= 0xff;
   }
}

// Impostazione interrupt system    DEVO RISPETTARE QUEST'ORDINE delle PRAGMA e mettere questa roba esattamente qui dopo il main senno non funziona....

#pragma interrupt  GestioneInterruzione   // questa riga segnala che GestioneInterruzione è una routine di interruzione e come tale salva nello stack indirizzo di ritorno e PSW

void GestioneInterruzione ()              // questa è la routine di interrupt vera e propria
{
  if (INTCON & 0x04)            // dovrei controllare se l'interruzione ALTERNATIVAMENTE:  if(INTCONbits.TMR0IF == 1)
  {                             // viene veramente da TMR0.... posso non farlo tanto è una sola!
    if(timer) timer--;          // se si faccio il lavoro...(decremento timer, se non è 0)

    INTCON &= 0xfb;                 // devo segnale che la interruzione è stata servita   ALTERNATIVAMENTE:  INTCONbits.TMR0IF = 0;

    TMR0H = PH; TMR0L = PL;         // ricarico il timer con il valore per 1000 impulsi
  }
}

#pragma code GestioneInterruzioneAP = 0x08   // questa riga serve per caricare il vettore di interruzione a 0x08 con l'indirizzo di partenza di GestioneInterruzione

void GestioneInterruzioneAP() { GestioneInterruzione(); }   // nel vettore metto un salto alla routine perchè da 0x08 a 0x018 potrebbe non entrare, la metto insieme al programma
```

```
1.  // LAMPEGGIO LED A 1Hz CON TIMER0 su PIC (18F2420 e simili)
2.  //
3.  // Blink3 fa lampeggiare 10 led (8 sulla porta A e 2 sulla B) usando il timer0
4.  //
5.  // (non uso le interruzioni ma una tecnica di gestione detta POLLED INTERRUPT)
6.  //-------------------------------------------------------------------------
7.  #pragma config OSC = INTIO67 // oscillatore interno pin 6 e 7 utilizzabili per I/O digitali
8.
9.  //#pragma config WDT = OFF     // in realtà nel simulatore non sono implementati...
10. //#pragma config LVP = OFF     //
11.
12. #include <p18cxxx.h>
13.
14. #define PH 0xfc // valori per contare 1000 impulsi: il contatore è up modulo
15. #define PL 0x18 // 65536. ci metto 64536 (0xfc18) così gli rimangono 1000 impulsi per l'overflow
16.
17. void main ()
18. {
19.   unsigned int cnt_overflow;  // variabile per contare gli overflow ogni 8 ms (125 * 8 ms = 1s)
20.
21.   ADCON1 = 15; // no AD sulla porte (tytti i pin di portA e portB I/O digitali)
22.   CMCON  = 7; // no comparatore sulla porta
23.
24.   T0CON = 0x82;               // set up timer0 - prescaler 1:8 --> 1MHz --> 8us x count
25.                               // 1000 impulsi = 8 ms per overflow.
26.
27.   TMR0H = PH;                 // setup period high byte
28.   TMR0L = PL;                 // setup period low byte 1 TOF = 8 ms
29.
30.   TRISB = 0;                  // porta A e B in uscita...
31.   PORTB = 0x00;
32.   TRISA = 0;
33.   PORTA = 0x00;
34.
35.   while (1)
36.     {
37.       while(!INTCONbits.TMR0IF);          // aspetto l'overflow del timer (supero 65535)
38.
39.       INTCONbits.TMR0IF = 0;              // devo azzerare il flag del timer0...
40.       TMR0H = PH; TMR0L = PL;             // ricarico il timer con il valore per 1000 impulsi
41.
42.       if(cnt_overflow) cnt_overflow--;  //  decremento cnt_overflow se non è zero...
43.           else                          // quando diventa 0 commuto le porte e lo reinizializzo a 20...
44.           {
45.             LATB ^= 0xff; // commuto le porte...
46.             LATA ^= 0xff;
47.             cnt_overflow = 20; // aspetto 20 overflow del timer (supero di 65535)
48.                                // con 8 us dovrei aspettare 125 overflow ma la simulazione è lenta...
49.           }
50.     }
51. }
```

208.// EXTINT1.c

```
// ATTENZIONE: la simulazione è fatta con il 2410: con il 2420 la printf faceva casino...!!!
// EXTINT1: conteggia e invia il numero degli impulsi da INT0 rilevati ad interruzione...

#include <p18cxxx.h>   /* for TRISA and PORTA declarations */
#include <stdio.h>
#include <usart.h>

/* Set configuration bits for use with ICD2 / PICDEM2 PLUS Demo Board:
 *  - set HS oscillator
 *  - disable watchdog timer
 *  - disable low voltage programming
 */
#pragma config OSC = HS
#pragma config WDT = OFF
#pragma config LVP = OFF

volatile unsigned int EXTINTcounter = 0;
volatile unsigned char flag = 0;

void main()
{
 INTCON = 0x00;
 ADCON1 = 15; // no AD sulle porte
 CMCON  = 7; // no comparatore sulla porta
 TRISB = 0xff; // porta B in ingresso
 TRISA = 0x00;
 RCONbits.IPEN = 1;        /* enable interrupt priority levels */
 INTCONbits.GIEH = 1;      /* enable all high priority interrupts */
 INTCONbits.INT0IE = 1;    /* enable INT0 interrupt */

  // ATTENZIONE: INIZIALIZZAZIONE UART:
 OpenUSART( USART_TX_INT_OFF  &  /* no TX interrupt                    */
            USART_RX_INT_OFF  &  /* no RX interrupt                    */
            USART_ASYNCH_MODE &  /* UART mode                          */
            USART_EIGHT_BIT   &  /* 8 bit No parity 1 stop             */
            USART_CONT_RX     &  /* non disabilita dopo 1 RX           */
            USART_BRGH_HIGH,     /* baud rate alto                     */
            51 );               /* baud rate = (Fclock / 1024) - 1    */

 puts("Ciao! Questo è il programma EXTINT1:\x0d\x0a");

// while(1) printf("EXTINTcounter %u \x0d\x0a", EXTINTcounter++);
  // LA PRINTF FA CASINO, COME AL SOLITO ???

 while(1) if(flag == 1) { printf("EXTINTcounter %06u \x0d\x0a", EXTINTcounter); flag = 0; }
}

// Impostazione interrupt system    DEVO RISPETTARE QUEST'ORDINE delle PRAGMA e mettere questa roba esattamente qui dopo il main senno non funziona....

#pragma interrupt  GestioneInterruzione   // questa riga segnala che GestioneInterruzione è una routine di interruzione e come tale salva nello stack indirizzo di ritorno e PSW

void GestioneInterruzione ()               // questa è la routine di interrupt vera e propria
{
   if ((INTCON & 0xfd) && (!flag))     // dovrei controllare se l'interruzione
   {                                   // viene veramente da INT0?
     INTCONbits.INT0F = 0; // devo segnalare che la interruzione è stata servita
     EXTINTcounter++;
     LATA = EXTINTcounter; // eco su led connessi alla porta A
     flag = 1;
   }
}

#pragma code GestioneInterruzioneAP = 0x08   // questa riga serve per caricare il vettore di interruzione a 0x08 con l'indirizzo di partenza di GestioneInterruzione

void GestioneInterruzioneAP() { GestioneInterruzione();  }   // nel vettore metto un salto alla routine perchè da 0x08 a 0x018 potrebbe non entrare, la metto insieme al programma
```

274.// IC1.c ESEMPIO INPUT CAPTURE

```
// ATTENZIONE: la simulazione è fatta con il 2410: con il 2420 la printf faceva casino...!!!
// IC1: Quando viene effettuata una transizione sul pin 2 di porta C invia il numero degli impulsi al PC

#include <p18cxxx.h>   /* for TRISA and PORTA declarations */
#include <stdio.h>
#include <usart.h>

/* Set configuration bits for use with ICD2 / PICDEM2 PLUS Demo Board:
 *  - set HS oscillator
 *  - disable watchdog timer
 *  - disable low voltage programming
 */
#pragma config OSC = HS
#pragma config WDT = OFF
#pragma config LVP = OFF

volatile unsigned char flag = 0;
volatile unsigned int timer1;

void main()
{
 TRISA = 0xfe; // bit 0 out
 INTCON = 0x00;  // disabilita le interruzioni
 CCP1CON = 0x04; // CCP1 in capture mode falling edge, TIMER1
 RCONbits.IPEN = 1;         /* enable interrupt priority levels */
 INTCONbits.GIEH = 1;       /* enable all high priority interrupts */
 PIE1bits.CCP1IE = 1;       /* enable CaptureComparePWM module 1 interrupts */
 TMR1H = TMR1L = 0x00;      // azzera il timer1
 T1CON = 0x31;    // timer1 prescaler = 8, abilitato
 INTCON = 0x80;   // abilita le interruzioni ad alta priorità

  // ATTENZIONE: INIZIALIZZAZIONE UART:
 OpenUSART( USART_TX_INT_OFF  &  /* no TX interrupt                    */
            USART_RX_INT_OFF  &  /* no RX interrupt                    */
            USART_ASYNCH_MODE &  /* UART mode                          */
            USART_EIGHT_BIT   &  /* 8 bit No parity 1 stop             */
            USART_CONT_RX     &  /* non disabilita dopo 1 RX           */
            USART_BRGH_HIGH,     /* baud rate alto                     */
            51 );               /* baud rate = (Fclock / 1024) - 1    */

 puts(" Ciao! Questo è il programma IC1:\x0d\x0a");

 while(1) if(flag == 1) { printf("timer1 %06u \x0d\x0a", timer1); flag = 0; }
}

// Impostazione interrupt system    DEVO RISPETTARE QUEST'ORDINE delle PRAGMA e mettere questa roba esattamente qui dopo il main senno non funziona....

#pragma interrupt  GestioneInterruzione   // questa riga segnala che GestioneInterruzione è una routine di interruzione e come tale salva nello stack indirizzo di ritorno e PSW

void GestioneInterruzione ()               // questa è la routine di interrupt vera e propria
{
   if(PIR1bits.CCP1IF == 1)     // dovrei controllare se l'interruzione
   {                            // viene veramente da INT0?
     PIR1bits.CCP1IF = 0; // devo segnalare che la interruzione è stata servita
     timer1 = CCPR1H;     // leggo i registri di cattura, li assemblo in timer1 e segnalo...
     timer1 <<= 8;
     timer1 |= CCPR1L;
     TMR1H = TMR1L = 0x00; // riazzero il timer1
     flag = 1;
     LATA ^= 0x01;
   }
}

 #pragma code GestioneInterruzioneAP = 0x08   // questa riga serve per caricare il vettore di interruzione a 0x08 con l'indirizzo di partenza di GestioneInterruzione

void GestioneInterruzioneAP() { GestioneInterruzione();  }   // nel vettore metto un salto alla routine perchè da 0x08 a 0x018 potrebbe non entrare, la metto insieme al programma
```

341.//OC1.c ESEMPIO OUTPUT COMPARE

```
// OC1: Con i due tasti su RB0 RB1 posso regolare il duty cycle del segnale in uscita a CCP1 (RC2)

#include <p18cxxx.h>   /* for TRISA and PORTA declarations */

#pragma config OSC = HS
#pragma config WDT = OFF
#pragma config LVP = OFF

void main()
{
 unsigned int d = 32767;

 ADCON1 = 0x0f;
 CMCON = 0x07;
 TRISB = 0xff; // portB tutti input
 INTCON = 0x00;  // disabilita le interruzioni
 PIE1bits.TMR1IE = 1; // abilita interruzioni timer overflow Timer1
 CCP1CON = 0x08; // CCP1 in compare mode rising edge su RC2, TIMER1
 RCONbits.IPEN = 1;        /* enable interrupt priority levels */
 TMR1H = 0x00; TMR1L = 0x00;      // azzero il timer
 T1CON = 0x31;    // timer1 prescaler = 8, abilitato
 INTCONbits.GIEH = 1;       /* enable all high priority interrupts */

 TRISC = 0xfb;

 CCPR1L  = d & 0x00ff; CCPR1H  = d >> 8;

 while(1)
 {
                switch(PORTB & 0x03)
                {
                                case 0x02: if(d<65535) d++; CCPR1L  = d & 0x00ff; CCPR1H  = d >> 8;  break;
                                case 0x01: if(d>0) d--; CCPR1L  = d & 0x00ff; CCPR1H  = d >> 8;  break;
                }
 }
}

// Impostazione interrupt system    DEVO RISPETTARE QUEST'ORDINE delle PRAGMA e mettere questa roba esattamente qui dopo il main senno non funziona....

#pragma interrupt  GestioneInterruzione   // questa riga segnala che GestioneInterruzione è una routine di interruzione e come tale salva nello stack indirizzo di ritorno e PSW

void GestioneInterruzione ()              // questa è la routine di interrupt vera e propria
{
   if(PIR1bits.TMR1IF == 1)     // dovrei controllare se l'interruzione
   {                        // viene veramente da TIMER1?
     PIR1bits.TMR1IF = 0;   // devo segnalare che la interruzione è stata servita
     TMR1H = TMR1L = 0x00;  // riazzero il timer1
     CCP1CON = 0x08;        // resetto CCP1CON...
   }
}

#pragma code GestioneInterruzioneAP = 0x08   // questa riga serve per caricare il vettore di interruzione a 0x08 con l'indirizzo di partenza di GestioneInterruzione

void GestioneInterruzioneAP() { GestioneInterruzione();  }   // nel vettore metto un salto alla routine perchè da 0x08 a 0x018 potrebbe non entrare, la metto insieme al programma
```

396. // PWM1: Con i due tasti su RB0 RB1 regolo il duty cycle del segnale in uscita a CCP1 (RC2)

```
397.
398. #include <p18cxxx.h>   /* for TRISA and PORTA declarations */
399.
400. /*
401. The following steps should be taken when configuring
402. the CCP module for PWM operation:
403. 1. Set the PWM period by writing to the PR2
404. register.
405. 2. Set the PWM duty cycle by writing to the
406. CCPRxL register and CCPxCON<5:4> bits.
407. 3. Make the CCPx pin an output by clearing the
408. appropriate TRIS bit.
409. 4. Set the TMR2 prescale value, then enable
410. Timer2 by writing to T2CON.
411. 5. Configure the CCPx module for PWM operation.
412.
413.  */
414. #pragma config OSC = HS
415. #pragma config WDT = OFF
416. #pragma config LVP = OFF
417.
418. void main()
419. {
420.  unsigned int i;
421.
422.  ADCON1 = 0x0f;
423.  CMCON = 0x07;
424.  TRISB = 0xff; // portB tutti input
425.  INTCON = 0x00;  // disabilita le interruzioni
426.  TRISC = 0xfb;
427.
428.  PR2 = 0x80;      // conta da 0 a 128 (periodo)
429.  CCPR1L  = 0x40;  // duty a 8 bit (256 livelli) (iniziale 50%)
430.  T2CON = 0x07;    // timer2 on, prescaler = 16, abilitato
431.  CCP1CON = 0x0c;  // CCP1 in pwm mode su RC2, TIMER2 bit 5 e 6 a 0 (i = due bit bassi dei 10 del duty a 0)
432.
433.  while(1)
434.   switch(PORTB & 0x03)
435.   {
436.                case 0x02: if(CCPR1L < 255) CCPR1L++;  for(i= 0; i < 10000; i++);  break;
437.                case 0x01: if(CCPR1L > 0)   CCPR1L--;  for(i= 0; i < 10000; i++);  break;
438.   }
439. }
440.
```

441.// TRIANGLE.c

```
442. // TRIANGLE: genera un onda modulata in durata da un ond triangolare in uscita a CCP1 (RC2)
443.
444.  #include <p18cxxx.h>   /* for TRISA and PORTA declarations */
445.
446.   */
447.  #pragma config OSC = HS
448.  #pragma config WDT = OFF
449.  #pragma config LVP = OFF
450.
451.  void main()
452.  {
453.   INTCON = 0x00;  // disabilita le interruzioni
454.   TRISC = 0xfb;   // uscita su RC2
455.   RCONbits.IPEN = 1;        /* enable interrupt priority levels */
456.   PR2 = 0xff;     // periodo
457.   CCPR1L  = 0x0;  // duty a 8 bit (256 livelli)
458.   T2CON = 0x04;    // timer2 prescaler = 1, abilitato
459.   CCP1CON = 0x0c;  // CCP1 in pwm mode su RC2, TIMER2 bit 5 e 6 a 0 (i = due bit bassi dei 10 del duty a 0)
460.   PIE1bits.TMR2IE = 1;  // abilito le interruzioni di timer 2
461.   INTCONbits.GIEH = 1;      /* enable all high priority interrupts */
462.
463.   while(1);
464.  }
465.
466.  // Impostazione interrupt system    DEVO RISPETTARE QUEST'ORDINE delle PRAGMA e mettere questa roba esattamente qui dopo il main senno non funziona....
467.
468.  #pragma interrupt  GestioneInterruzione   // questa riga segnala che GestioneInterruzione è una routine di interruzione e come tale salva nello stack indirizzo di ritorno e PSW
469.
470.  void GestioneInterruzione ()              // questa è la routine di interrupt vera e propria
471.  {
472.     if(PIR1bits.TMR2IF == 1)     // dovrei controllare se l'interruzione
473.     {                        // viene veramente da TIMER2?
474.        PIR1bits.TMR2IF = 0;   // devo segnalare che la interruzione è stata servita
475.        CCPR1L++;             // ad ogni TOF incremento il duty
476.     }
477.  }
478.
479.  #pragma code GestioneInterruzioneAP = 0x08   // questa riga serve per caricare il vettore di interruzione a 0x08 con l'indirizzo di partenza di GestioneInterruzione
480.
481.  void GestioneInterruzioneAP() { GestioneInterruzione();  }   // nel vettore metto un salto alla routine perchè da 0x08 a 0x018 potrebbe non entrare, la metto insieme al programma
```

482. // SINE.c

483. //tabellina in EPROM con i valori approssimati dei campioni di 1/4 di periodo di sinusoide

```
484. const rom unsigned char samples[64] = {128, 131, 134, 137, 140, 143, 146, 149, 153, 156, 159, 162, 165, 168, 171, 174, 177, 180, 182, 185, 188, 191, 194, 196, 199, 201, 204, 207,
     209, 211, 214, 216, 218, 220, 223, 225, 227, 229, 231, 232, 234, 236, 238, 239, 241, 242,  243, 245, 246, 247, 248, 249, 250, 251, 252, 253, 253, 254, 254, 255, 255, 255, 255,
     255 };
485.
486. #pragma config OSC = HS
487. #pragma config WDT = OFF
488. #pragma config LVP = OFF
489.
490. // SINE: genera un onda modulata in durata da un onda sinusoidale in uscita a CCP1 (RC2)
491.
492. #include <p18cxxx.h>   /* for TRISA and PORTA declarations */
493.
494. void main()
495. {
496.  INTCON = 0x00;  // disabilita le interruzioni
497.  RCONbits.IPEN = 1;        /* enable interrupt priority levels */
498.  TRISC = 0xfb;   // uscita su RC2
499.  PR2 = 0xff;     // periodo
500.  CCPR1L  = 0x00;  // duty a 8 bit (256 livelli)
501.  T2CON = 0x04;    // timer2 prescaler = 1, abilitato
502.  CCP1CON = 0x0c;  // CCP1 in pwm mode su RC2, TIMER2 bit 5 e 6 a 0 (i = due bit bassi dei 10 del duty a 0)
503.  PIE1bits.TMR2IE = 1;  // abilito le interruzioni di timer 2
504.  INTCONbits.GIEH = 1;       /* enable all high priority interrupts */
505.
506.  while(1);
507. }
508.
509. // Impostazione interrupt system    DEVO RISPETTARE QUEST'ORDINE delle PRAGMA e mettere questa roba esattamente qui dopo il main senno non funziona....
510.
511. #pragma interrupt  GestioneInterruzione   // questa riga segnala che GestioneInterruzione è una routine di interruzione e come tale salva nello stack indirizzo di ritorno e PSW
512.
513. void GestioneInterruzione ()               // questa è la routine di interrupt vera e propria
514. {
515.  static unsigned char i,j; // i va da 0 a 255, j va da 0 a 63
516.
517.    if(PIR1bits.TMR2IF == 1)     // L'interruzione viene veramente da TIMER2?
518.    {
519.       j = i % 64; // j lavora su un quarto d'onda...
520.       if((i >=  0)  && (i <   64)) CCPR1L = samples[j];                 // primo quarto
521.       else if((i >= 64)  && (i <  128)) CCPR1L = samples[63 - j];        // secondo quarto
522.       else if((i >= 128) && (i <  192)) CCPR1L = 255 - samples[j];       // terzo quarto
523.       else if((i >= 192) && (i <= 255)) CCPR1L = 255 - samples[63 - j];  // quarto quarto
524.
525.       i++; // incremento l'indice del campione...
526.
527.       PIR1bits.TMR2IF = 0;   // devo segnalare che la interruzione è stata servita
528.    }
529. }
530.
531. #pragma code GestioneInterruzioneAP = 0x08   // questa riga serve per caricare il vettore di interruzione a 0x08 con l'indirizzo di partenza di GestioneInterruzione
532.
533. void GestioneInterruzioneAP() { GestioneInterruzione();  }   // nel vettore metto un salto alla routine perchè da 0x08 a 0x018 potrebbe non entrare, la metto insieme al programma
```

534.// UART1.c

535.

```
536. //  Questo programma di esempio usa la uart on chip per realizzare una trasmissione ricezione di caratteri a polling.
537.
538. #include <p18cxxx.h>   /* for TRISA and PORTA declarations */
539.
540. #define RX_VOID  0
541. #define RX_READY 1
542. #define RX_ERROR 2
543.
544.
545. void init_uart(void); // inizializza la uart
546. void TransmitByte( unsigned char data );  // trasmette un carattere a polling
547. unsigned char ReceiveByte(unsigned char *ch); // riceve un carattere e ritorna lo stato della uart
548.
549. void main()
550. {
551.  unsigned char ch;
552.
553.  init_uart(); // 12:9600 baud at 8 MHz
554.
555.  TransmitByte('U'); TransmitByte('A'); TransmitByte('R'); TransmitByte('T'); TransmitByte('1'); TransmitByte(':'); TransmitByte(0x0d); TransmitByte(0x0a);
556.
557.  while(1) if(ReceiveByte(&ch) == RX_READY) TransmitByte(ch); // forever: echoes back received chars
558. }
559. /*****************************************************************************/
560. /* initialize UART */
561. /*****************************************************************************/
562. void init_uart()
563. {
564.  SPBRGH = 0x00; SPBRG = 0x0c; /* sets the baud rate */
565.
566.  // SYNC = 0, BRGH = 0, BRG16 = 0 default values
567.
568.  RCSTAbits.SPEN = 1; // abilita la porta seriale
569.  RCSTAbits.CREN = 1; // abilita il ricevitore
570.  TXSTAbits.TXEN = 1; // abilita il trasmettitore
571. }
572. /*****************************************************************************/
573. void TransmitByte( unsigned char data )
574. {
575.  while (TXSTAbits.TRMT == 0); /* wait for empty transmit buffer */
576.  TXREG = data; /* start transmission */
577. }
578. /*****************************************************************************/
579. unsigned char ReceiveByte(unsigned char *ch)
580. {
581.             unsigned char status = RCSTA; // legge lo stato della seriale
582.
583.             if(PIR1bits.RCIF == 0) return(RX_VOID);  // se non c'è carattere ritorna
584.
585.             if(status & 0x06) // framing or overrun error...
586.             {
587.              RCSTAbits.CREN = 0; RCSTAbits.CREN = 1;  // azzera il flag e riabilita...
588.              return(RX_ERROR);
589.    }
590.
591.             *ch = RCREG;  // RX OK ritorna il carattere ricevuto
592.
593.             return(RX_READY);  // segnala stato OK
594. }
595.
```

596.// UART2.c

```
//  Questo esempio usa la uart on chip per realizzare una trasmissione e una ricezione di caratteri a interrupt.
// ATTENZIONE: selezionare: DEFAULT STORAGE CLASS = STATIC in BUILD OPTIONS (PROJECTS)....bug compilatore??!!!
//                          E DENTRO MCC18 MEMORY MODEL: multiple stack model (why???)
#include <p18cxxx.h>   /* for TRISA and PORTA declarations */

#define RX_VOID  0
#define RX_READY 1
#define RX_TIME_OUT             3000
#define PH 0xfc // valori per contare 1000 impulsi: il contatore è up modulo
#define PL 0x18 // 65536: ci metto 64536 (0xfc18) così gli rimangono 1000 impulsi per l'overflow
//*****************************************************************************
unsigned char rx_buff[21]; // buffer ricezione
unsigned int rx_timer = 0; // timer time out
unsigned char rx_cnt = 0;  // contatore caratteri ricevuti
unsigned char rx_done = 0; // flag stringa pronta
//*****************************************************************************
void init_mcu(void);  // inizializza il microcontrollore
void init_uart(void); // inizializza la uart
void TransmitByte( unsigned char data );  // trasmette un carattere a polling
void Send(unsigned char *s); // invia una stringa
unsigned char Receive(unsigned int *len); // legge il buffer RX e trova la lunghezza:  ritorna 1 se c'è una stringa pronta, 0 altrimenti
//*****************************************************************************
void main()
{
  unsigned int len;
  unsigned char s1[] = "UART2:\x0d\x0a";
  unsigned char s2[] = "Scrivere una stringa di max 20 caratteri entro 3 secondi e poi premere invio\x0d\x0a";
  unsigned char s3[] = "Echo: ";

  init_mcu(); // inizializza il micro...

  init_uart(); // inizializza la UART, N 8 1 9600 (baud rate divisor: 12:9600 baud at 8 MHz)

  Send(s1); Send(s2);

  while(1) // forever: echoes back received string.
  {
     Send(s3);

     while(Receive(&len) != RX_READY); // waits until string received

     Send(rx_buff);    // sends string back.

     TransmitByte(0x0d); TransmitByte(0x0a); // new line
  }
}
/* initialize UART */
/******************************************************************************/
void init_uart()
{
  SPBRGH = 0x00; SPBRG = 0x0c; /* sets the baud rate */
   // SYNC = 0, BRGH = 0, BRG16 = 0 default values
  RCSTAbits.SPEN = 1; // abilita la porta seriale
  RCSTAbits.CREN = 1; // abilita il ricevitore
  TXSTAbits.TXEN = 1; // abilita il trasmettitore
  PIE1bits.RCIE  = 1; // abilita interruzioni per ricezione di un carattere dalla UART
}
/******************************************************************************/
void TransmitByte( unsigned char data )
{
  while (TXSTAbits.TRMT == 0); /* wait for empty transmit buffer */
  TXREG = data; /* start transmission */
}
// generic send and receive routines
//*****************************************************************************
void Send(unsigned char *buff)
{
   unsigned char ch, i;
   i = 0; do TransmitByte(ch = buff[i++]); while(ch != 0);
}
```

```
667. //*****************************************************************************
668. unsigned char Receive(unsigned int *len) // ricezione con time_out.
669. {
670.  if(!rx_done) return(RX_VOID);
671.  *len = rx_cnt;
672.  rx_done = 0; rx_cnt = 0;
673.  return(RX_READY);
674. }
675. //*****************************************************************************
676. void init_mcu()
677. {
678.   INTCON = 0x00;               // disabilito tutti le interruzioni
679.   RCONbits.IPEN = 1;           /* enable interrupt priority levels */
680.   IPR1bits.TMR1IP = 0;       // imposta bassa priorità per interruzioni di timer1
681.   PIE1bits.TMR1IE = 1;       // abilita interruzioni timer overflow Timer1
682.   TMR1H = PH;                // da contare 1000 impulsi --> 1ms; setup period high byte
683.   TMR1L = PL;                // setup period low byte 1 TOF = 1 ms--> 1000 TOF = 1 s.
684.   T1CON = 0x10;  // Fosc = 8MHz-->Fclock = Fosc/4 = 2MHz / prescaler=2 = 1MHz <-> ius <--> valore del cnt = 64536,
685.   INTCONbits.GIEH = 1;       /* enables all high priority interrupts */
686.   INTCONbits.GIEL = 1;       /* enables all low priority interrupts */
687.   T1CONbits.TMR1ON = 1;
688. }
689. // Impostazione interrupt system
690. /*****************************************************************************/
691. // UART HANDLER
692. /*****************************************************************************/
693. #pragma interrupt UartReceive   // UartReceive è una routine di interrupt...(salva indirizzo di ritorno e PSW nello stack)
694.
695. void UartReceive() // interrupt handler uart rx
696. {
697.                 unsigned char ch;
698.
699.                 if(PIR1bits.RCIF == 0) return; // controllo se l'interruzione è dalla RX UART
700.
701.     PIR1bits.RCIF = 0; // segnalo interruzione servita...
702.
703.                 ch = RCREG; // leggo il registro ricezione
704.
705. // appena arriva il primo carattere fa partire il timer...i successivi devono arrivare entro RX_TIMEOUT sennò abort...
706.                 if(rx_timer == 0) { rx_timer = RX_TIME_OUT; rx_done = 0; rx_cnt = 0; }
707.
708. // riceve 20 caratteri e termina la stringa con uno 0.
709.
710.       if((rx_cnt == 20) || (ch == 0x0d)) { rx_buff[rx_cnt] = 0; rx_done = 1; } // se sono 20 o è invio metti fine stringa
711.                   else rx_buff[rx_cnt++] = ch;                                    // e segnala done, altrimenti immagazzina e incr.
712. }
713. // timer  handler
714. /*****************************************************************************/
715. #pragma interrupt Timer1Overflow    // Timer1Overflow è una routine di interrupt...(salva indirizzo di ritorno e PSW nello stack)
716.
717. void Timer1Overflow()              // questa è la routine di interrupt
718. {
719.     if(PIR1bits.TMR1IF == 1)       // dovrei controllare se l'interruzione
720.     {                              // viene veramente da TIMER1?
721.       PIR1bits.TMR1IF = 0;         // devo segnalare che la interruzione è stata servita
722.       if(rx_timer) rx_timer--;     // decremento il timer SW se non è zero...
723.       TMR1H = PH; TMR1L = PL;      // ricarico il timer1
724.     }
725. }
726.
727.  #pragma code GestioneInterruzioneAP = 0x08   // questa riga serve per caricare il vettore di interruzione a 0x08 con l'indirizzo di partenza di GestioneInterruzioneAP
728.
729.  void GestioneInterruzioneAP() { UartReceive();  }   // nel vettore metto un salto alla routine perchè da 0x08 a 0x018 potrebbe non entrare, la metto insieme al programma
730.  #pragma code GestioneInterruzioneLP = 0x18   // questa riga serve per caricare il vettore di interruzione a 0x08 con l'indirizzo di partenza di GestioneInterruzioneLP
731.
732.  void GestioneInterruzioneLP() { Timer1Overflow();  }   //
```

733.// EE1.c

```
734. #include <p18cxxx.h>   /* for TRISA and PORTA declarations */
735. #include <stdio.h>
736. #include <usart.h>
737.
738. //#pragma config OSC = HS
739. //#pragma config WDT = OFF
740. #pragma config LVP = OFF
741.
742. char getch(void);  // funzione che riceve un carattere da RxD
743. void putch(char ch); // funzione che invia un carattere su TxD
744.
745. void main()
746. {
747.  unsigned char i;
748.  unsigned char ch;
749.
750. // ATTENZIONE: INIZIALIZZAZIONE UART:
751.  OpenUSART( USART_TX_INT_OFF  &  /* no TX interrupt                    */
752.           USART_RX_INT_OFF  &  /* no RX interrupt                    */
753.           USART_ASYNCH_MODE &  /* UART mode                          */
754.           USART_EIGHT_BIT   &  /* 8 bit No parity 1 stop             */
755.           USART_CONT_RX     &  /* non disabilita dopo 1 RX           */
756.           USART_BRGH_HIGH,     /* baud rate alto                     */
757.           51 );              /* baud rate = (Fclock / 1024) - 1       */
758.
759.  putch('H'); putch('e'); putch('y'); putch('!'); putch('!'); putch(0x0a);putch(0x0d);
760.
761.  puts("Ciao! Questo è il programma EE1...\x0d\x0a");
762.
763.  while(1)
764.  {
765.   puts("\x0d\x0aI primi 10 byte della EEprom contengono:\x0d\x0a");
766.
767.   for(i = 0; i < 10; i++)
768.   {
769.                EEADR = i; // Data Memory Address to read
770.                EECON1bits.EEPGD = 0; // Point to DATA memory
771.                EECON1bits.CFGS = 0; // Access EEPROM
772.                EECON1bits.RD = 1; //  EEPROM Read
773.                putch(EEDATA);      // invia il dato alla seriale...
774.   }
775.
776.   puts("\x0d\x0ascrivi ora fino a 10 byte che verranno scritti nella EEprom:\x0d\x0a");
777.
778.   for(i = 0; i < 10; i++)
779.   {
780.                ch = getch();
781.                PIR2bits.EEIF == 0; // azzera il flag write complete
782.      EEADR = i; // Data Memory Address to write
783.      EEDATA = ch; // Data Memory Value to write
784.                EECON1bits.EEPGD = 0; //  Point to DATA memory
785.                EECON1bits.CFGS = 0;  // Access EEPROM
786.                EECON1bits.WREN = 1;  //  Enable writes
787.      INTCONbits.GIE = 0; // Disable Interrupts, if any
788.      EECON2 = 0x55;   // REQUIRED SECURING SEQUENCE
789.      EECON2 = 0xaa;
790.                EECON1bits.WR = 1; // Set WR bit to begin write
791.      INTCONbits.GIE = 1; // re-enable Interrupts, if any
792.                putch(ch);
793.                while(PIR2bits.EEIF == 0); // waits for write complete...
794.                EECON1bits.WREN = 0;  //  disable writes
795.   }
796.  }
797. }
798.
799. char getch(){ while(!DataRdyUSART()); return(getcUSART()); } // se il registro RX è pieno lo legge sennò aspetta
800.
801. void putch(char ch){ while(BusyUSART()); putc(ch, stdout); } // se il registro TX è vuoto spedisce, sennò aspetta
```

802.// EXRAMP AD1.c

```
803.
804.  // CONVERTITORE A RAMPA ESPONENZIALE CHE UTILIZZA UN COMPARATORE INTERNO AL PIC
805.  // condensatore da 470nF, resistenza da 47k, quarzo da 8MHz  --> numero max di cicli= 100000, poi overflow
806.
807.  #include <p18cxxx.h>   /* for TRISA and PORTA declarations */
808.  #include <stdio.h>
809.  #include <usart.h>
810.
811.  /* Set configuration bits for use with ICD2 / PICDEM2 PLUS Demo Board:
812.   *  - set HS oscillator
813.   *  - disable watchdog timer
814.   *  - disable low voltage programming
815.   */
816.  #pragma config OSC = HS
817.  #pragma config WDT = OFF
818.  #pragma config LVP = OFF
819.
820.  void main()
821.  {
822.   unsigned int count, i;
823.
824.   ADCON1 = 7; // no AD sulla porta
825.   CMCON  = 7; // no comparatore sulla porta
826.   TRISA = 0xfd; // porta A tutta in ingresso tranne il bit 1
827.
828.    // ATTENZIONE: INIZIALIZZAZIONE UART:
829.   OpenUSART( USART_TX_INT_OFF  &  /* no TX interrupt                    */
830.             USART_RX_INT_OFF  &  /* no RX interrupt                    */
831.             USART_ASYNCH_MODE &  /* UART mode                          */
832.             USART_EIGHT_BIT   &  /* 8 bit No parity 1 stop             */
833.             USART_CONT_RX     &  /* non disabilita dopo 1 RX           */
834.             USART_BRGH_HIGH,     /* baud rate alto                     */
835.             51 );                /* baud rate = (Fclock / 1024) - 1    */
836.
837.   puts("Ciao! Questo è il programma EXPRAMPAD1:");
838.
839.   while(1)
840.   {
841.                 CMCON = 0x00;  // resetto i comparatori
842.                 PORTA &= 0xfd; // azzero la tensione al C
843.                 for(i=0; i < 100; i++);
844.                 CMCON = 0x02;  // abilito i comparatori
845.                 count = 0;
846.                 PORTA |= 0x02; // inizio la carica del C
847.
848.                 while(CMCON & 0x40) // 0x40 è la maschera di CMCON per controllare il comparatore 1.
849.            {
850.                  count++; // finchè il comparatore non scatta incremento count
851.                  if(count == 10000) { puts("Voltage overflow!!!"); break; }
852.                 }
853.
854.                 printf("count (t*): %u \x0d\x0a", count); // qui ha gia trovato vin = vtrial, quindi trasmette count...
855.
856.                 while(count != 10000) count++; // serve per fare in modo, aspettando, che tutti i cicli durino uguale tempo.
857.   }
858.  }
859.
```

```
// THE SIMPLEST HH:MM:SS  MICROCONTROLLER DRIVEN CLOCK    pic18f2420 clock
// (6 digits 7-segment multiplexed display) D.Iracà Pisa 14/04/2011
//
//-------------------------------------------------------------------------
#pragma config OSC = HS // oscillatore al quarzo (esterno)
#pragma config WDT = OFF
#pragma config LVP = OFF

#include <p18cxxx.h>

#define PH 0xfc // valori per contare 1000 impulsi: il contatore è up modulo
#define PL 0x18 // 65536: ci metto 64536 (0xfc18) così gli rimangono 1000 impulsi per l'overflow

void main()
{
 unsigned int cnt_overflow = 0; // variabile globale per contare gli overflow
 unsigned char ore = 12, minuti = 34, secondi = 56, display_attivo = 0;
 // segmento display   g f e d c b a
 // bit porta B       6 5 4 3 2 1 0   (il bit 7 fa lampeggiare i punti)
 // quindi ad esempio per accendere 8 serve 0111 1111 = 0x7F, per accendere 0 serve 0011 1111 = 0x3f e cosi via....
 const unsigned char BCD27SEG[10] = {0x3f, 0x06, 0x5b, 0x4f, 0x66, 0x6d, 0x7d, 0x07, 0x7f, 0x6f };
                            // 0    1    2    3    4    5    6    7   8    9
 TRISB = 0x00;  TRISC = 0xc0; // porta B e C in uscita

 T0CON = 0x02; //  set up timer0 - prescaler 1:8 --> 1MHz --> 8us x count, fermo

 while (1)
 {
        TMR0H = PH; TMR0L = PL; // setup period high byte e setup period low byte 1 TOF = 8 ms
        T0CONbits.TMR0ON = 1;   // avvia TMR0

        while(!INTCONbits.TMR0IF);    // attesa time slice...aspetto che il flag di overflow di timer vada a 1...

        T0CONbits.TMR0ON = 0;   // ferma TMR0
        INTCONbits.TMR0IF = 0;  // resetto il flag di overflow di timer...

        LATC = (1  << display_attivo); // multiplexing dei display....seleziono il display da accendere inviandogli uno sull'anodo
        // questo comando manda sulla porta C di volta in volta: 00000001 00000010 00000100 00001000 00010000 00100000

        switch(display_attivo) // a seconda della cifra piloto i catodi che voglio accendere con zeri...
        {
                case 0:    LATB = ~BCD27SEG[secondi % 10]; break;  // trovo unita secondi (e li trasformo in 7 segmenti)
                case 1:    LATB = ~BCD27SEG[secondi / 10]; break;  // trovo decine secondi (e li trasformo in 7 segmenti)
                case 2:    LATB = ~BCD27SEG[minuti % 10];  break;  // lo stesso per ore e minuti (e li trasformo in 7 segmenti)
                case 3:    LATB = ~BCD27SEG[minuti / 10];  break;
                case 4:    LATB = ~BCD27SEG[ore % 10];     break;  // NOTA la tilde serve perchè devo mettere a massa il catodo
                case 5:    LATB = ~BCD27SEG[ore / 10];     break;  // per fare accendere il led, quindi inverto tutti i bit
        }

        display_attivo++; display_attivo %= 6; // cambio display modulo 6 cioè se ho finito il giro ricomincio..

        if(cnt_overflow) cnt_overflow--; // se cnt_overflow è > 0 decrementalo...
                else // se cnt_overflow da 125 è diventato 0 è passato 1 secondo
                {
                        switch(PORTC & 0xc0)  // gestione pulsanti aggiustamento minuti e ore
                        {
                                case 0x80: minuti++; break; // minuti (bit RC6) a massa
                                case 0x40: ore++;    break; // ore (bit RC7) a massa
                        }

                        cnt_overflow = 20;  // ci metto 6 e non 125 perchè la simulazione è lenta
                        LATB ^= 0x80; // commuto il bit dei led dei secondi

                        secondi++;    // aggiornamento secondi->ore->minuti
                        if(secondi == 60) { secondi = 0; minuti++; }
                        if(minuti == 60) { minuti = 0; ore++; }
                        if(ore == 24) { ore = 0; minuti = secondi = 0; }
                }
  }
}
```

COME FUNZIONA IL DISPLAY 7 SEGMENTI

Numero	Segmenti 6543210 gfedcba	Byte (binario) da inviare alla porta alla quale sono collegati i segmenti	Byte Hex
0	fedcba	0011 1111	0x3F
1	cb	0000 0110	0x06
2	gedba	0101 1011	0x5B
3	gdcba	0100 1111	0x3F
4	gfcb	0110 0110	0x66
5	gfdca	0110 1101	0x6D
6	gfedca	0111 1101	0x7D
7	cba	0000 0111	0x07
8	gfedcba	0111 1111	0x7F
9	gfdcba	0110 1111	0x6F

Come trovare le cifre da inviare al display per visualizzare un numero su piu cifre:

Se voglio far comparire ad esempio 25 minuti, sul display delle unita di minuti devo mandare 25 % 10 = 5 (il resto della divisione intera di 25 per 10) e sul display delle decine di minuti 25 / 10 = 2 (il quoziente della divisone intera di 25 per 10. Poichè però per il 2 devo accendere i segmenti gedba e per il 5 i segmenti gfdca dovro trasformare con una tabella (o un vettore oppure con uno switch) 2 in 0x5B e 5 in 0x6D. Posso quindi appunto mettere la colonna Byte hex della tabella dentro un vettore di 10 elementi numerati da 0 a 9, di nome ad esempio bcd27seg[10]; (si pronuncia bcd to 7 segments, ad indicare le sua operazione di decodifica da binario codificato in decimale a pilotaggio 7 segmenti). Quindi il microcontrollore deve sputare sulla porta dei catodi non i numeri di unita e decine ma i numeri trascodificati utilizzando il vettore: quindi non N ma bensi la transcodifica data da bcd27seg[N].

multiplexing:

Con un altra porta il microcontrollore selezionerà ciclicamente e in modo molto veloce (diciamo ogni 5 ms) un display a turno mettendo l'anodo comune a Vcc e invierà sui catodi il valore bcd27seg[N], essendo N il risultato di minuti % 10 e minuti / 10 e cosi via per le ore o i secondi.

In tal modo ottengo il vantaggio di ridurre enormemente il numero di collegamenti necessari: per 6 cifre avremmo 6 * 7 = 42 collegamenti (piu die pin dell'integrato, nb!!) invece col multiplexing ne servono 6 (per i display, catodi comuni) + 7 (per i segmenti, anodi) = 13.

931. // bal2bal.c invio stato tasti e stato led fra 2 sistemi gemelli

```
#include <p18cxxx.h>   /* for TRISA and PORTA declarations */
#include <stdio.h>
#include <usart.h>

/* Set configuration bits for use with ICD2 / PICDEM2 PLUS Demo Board:
 *  - set HS oscillator
 *  - disable watchdog timer
 *  - disable low voltage programming
 */
#pragma config OSC = INTIO67
#pragma config WDT = OFF
#pragma config LVP = OFF

char getch(void);  // funzione che riceve un carattere da RxD
void putch(char ch); // funzione che invia un carattere su TxD

void main()
{  // ATTENZIONE: INIZIALIZZAZIONE UART:
  OpenUSART( USART_TX_INT_OFF  &  /* no TX interrupt                    */
            USART_RX_INT_OFF  &  /* no RX interrupt                    */
            USART_ASYNCH_MODE &  /* UART mode                          */
            USART_EIGHT_BIT   &  /* 8 bit No parity 1 stop             */
            USART_CONT_RX     &  /* non disabilita dopo 1 RX           */
            USART_BRGH_HIGH,     /* baud rate alto                     */
            51 );              /* baud rate = (Fclock / 1024) - 1      */

  ADCON1 = 7; // no AD sulla porta
  CMCON  = 7; // no comparatore sulla porta
  TRISA = 0x00; // porta A in uscita
  TRISB = 0xff; // porta B in ingresso


  while(1)
  {
//   putch(PORTB);
                putc(PORTB, stdout);
//   PORTA = getch();
                PORTA = getcUSART();
  }
}

char getch(){ while(!DataRdyUSART()); return(getcUSART()); } // se il registro RX è pieno lo legge sennò aspetta

void putch(char ch){ while(BusyUSART()); putc(ch, stdout); } // se il registro TX è vuoto spedisce, sennò aspetta
```

977.// bal2bal3.c invio stato canali analogici e led fra due sistemi gemelli

```
978.  #include <p18cxxx.h>
979.  #include <stdio.h>
980.  #include <usart.h>
981.
982.  #pragma config OSC = INTIO67
983.  #pragma config WDT = OFF
984.  #pragma config LVP = OFF
985.
986.  void main()
987.  {
988.   unsigned char f = 0;
989.
990.   OpenUSART (USART_TX_INT_OFF & USART_RX_INT_OFF & USART_ASYNCH_MODE & USART_EIGHT_BIT & USART_CONT_RX &
      USART_BRGH_HIGH, 51);
991.
992.   TRISB = 0x00;
993.   TRISA = 0xff;
994.
995.  ADCON1 = 0x0B;
996.
997.   while(1)
998.   {
999.    switch(PORTA & 0xf0)
1000.  {
1001.                case 0x80: ADCON0 = 0x03; break;
1002.                case 0x40: ADCON0 = 0x07; break;
1003.                case 0x20: ADCON0 = 0x0B; break;
1004.                case 0x10: ADCON0 = 0x0F; break;
1005.                default  : PORTB = getcUSART();
1006.                                                    if(f) { f = 0; putc(0, stdout); }
1007.                                                    continue;
1008.  }
1009.  while(ADCON0 & 0x02);
1010.
1011.  putc(ADRESH, stdout);
1012.
1013.  f = 1;
1014. }
1015.}
1016.
```

```
1017.// STRICT POLLING  VS RELAXED POLLING VS INTERRUPT IO COMPARISON
1018.
1019.#include <p18cxxx.h>   /* for TRISA and PORTA declarations */
1020.
1021.#pragma config OSC = INTIO67
1022.
1023.int i, j;
1024.
1025.void flashLED3(int n) // fa lampeggiare D3 su RB2 n volte
1026. {    for(j = 0; j < n; j++) {  LATAbits.LATA3 = 0;  for(i = 0; i < 10000; i++);  LATAbits.LATA3 = 1;  for(i = 0; i < 5000; i++);         }  }
1027.
1028.void Attesa(void)  // aspetta un po con ciclo di ritardo dal quale si può uscire prima premendo P1 si RC0
1029.  {  for(i = 0; i < 150; i++)  for(j = 0; j < 250; j++) if(PORTBbits.RB0 == 0) return;  }
1030.
1031.void main()   // MAIN PROGRAM
1032.{
1033. ADCON1 = 15; // no AD sulla porta
1034. CMCON  = 7; // no comparatore sulla porta
1035. TRISA = 0xf1;  TRISB = 0xff; // porta A in uscita bit 1-2-3  // porta B in ingresso
1036.
1037. RCONbits.IPEN = 1;    INTCONbits.GIEH = 1;     // enable interrupt priority levels  and enable all high priority interrupts
1038.
1039. LATAbits.LATA1 = 1; LATAbits.LATA2 = 1; LATAbits.LATA3 = 1;// spengo i led
1040.
1041. while(PORTBbits.RB0 == 1); // premi per primo test...
1042.
1043. while(1)  // MAIN CONTROL LOOP
1044. {
1045.         flashLED3(1); // primo test: polling stretto: gestisco bene solo un tasto...
1046.
1047.         while(PORTBbits.RB0 == 1)
1048.         {
1049.                 while(PORTBbits.RB1 == 0) LATAbits.LATA1 = 0;  // aspetto finchè premo il tasto, col led acceso
1050.                 LATAbits.LATA1 = 1;  // quando è premuto lo spengo
1051.
1052.                 while(PORTBbits.RB2 == 0) LATAbits.LATA2 = 0;  // lo stesso per l'altro led
1053.                 LATAbits.LATA2 = 1;
1054.         }
1055.
1056.         flashLED3(2); // secondo test: polling lasco su due eventi veloci...
1057.                         // sembra funzionare, ma se gli eventi sono lenti si comporta molto male...
1058.
1059.         while(PORTBbits.RB0 == 1)
1060.         {
1061.                 if(PORTBbits.RB1 == 0) LATAbits.LATA1 = 0;
1062.                         else LATAbits.LATA1 = 1;
1063.
1064.                 if(PORTBbits.RB2 == 0) LATAbits.LATA2 = 0;
1065.                         else LATAbits.LATA2 = 1;
1066.         }
1067.
1068.         flashLED3(3); // infatti: terzo test: polling lasco su un evento lento e uno veloce
1069.
1070.         while(PORTBbits.RB0 == 1)
1071.         {
1072.                 if(PORTBbits.RB1 == 0) { LATAbits.LATA1 = 0; Attesa(); }  else { LATAbits.LATA1 = 1; Attesa(); }
1073.
1074.                 if(PORTBbits.RB2 == 0) { LATAbits.LATA2 = 0; } // la gestione di questo evento è
1075.                         else { LATAbits.LATA2 = 1; }             // rallentata da quella dell'altro, lento
1076.         }
1077.
1078.          flashLED3(4); // quarto test: gestione di un evento (lento) a polling e uno (veloce) a interrupt
1079.
1080.         INTCON3bits.INT2IE = 1;     /* enable INT2 interrupt */
1081.
1082.         while(PORTBbits.RB0 == 1)
1083.           if(PORTBbits.RB1 == 0) { LATAbits.LATA1 = 0; Attesa(); }  else { LATAbits.LATA1 = 1; Attesa(); }
1084.
1085.         flashLED3(5); // quinto test: gestione di due eventi ad interrupt
1086.
1087.         INTCON3bits.INT1IE = 1;     // enable INT1 interrupt
1088.
1089.         while(PORTBbits.RB0 == 1); // niente: tutto il lavoro lo fanno i due rami della routine di interruzione....
1090.
```

```
1091.           INTCON3bits.INT1IE = 0;   INTCON3bits.INT2IE = 0;        // disable INT1 interrupt  and disable INT2 interrupt
1092. }
1093.}
1094.
1095.unsigned char s1 = 0, s0 = 1, cnt0 = 0;
1096.
1097.#pragma interrupt GestioneInterruzione  // il vettore di interruzione a 0x08
1098.
1099.void GestioneInterruzione() // UNA SOLA ROUTINE PER TUTTE LE SORGENTI...(PIC STYLE)
1100.{
1101.    if (INTCON3bits.INT1F)     // controllo se l'interruzione
1102.    {                          // viene veramente da INT1
1103.                        LATAbits.LATA1 = s0;
1104.                        if(cnt0 == 5) { s0 = 1-s0; cnt0 = 0; } else cnt0++;
1105.           INTCON3bits.INT1F = 0;  // devo segnalare che la interruzione è stata servita
1106.    }
1107.
1108.    if (INTCON3bits.INT2F)    // controllo se l'interruzione
1109.    {                         // viene veramente da INT2
1110.                        LATAbits.LATA2 = s1;
1111.                        s1++; s1 %= 2;
1112.           INTCON3bits.INT2F = 0;  // devo segnalare che la interruzione è stata servita
1113.    }
1114.}
1115.
1116.// Routine di interruzione: Impostazione interrupt vector
1117.
1118.#pragma code GestioneInterruzioneSorgentiAP = 0x08 // queste righe servono per caricare il vettore di interruzione
1119.
1120.void  GestioneInterruzioneSorgentiAP() { GestioneInterruzione(); } // salto alla vera routine
```

1121.// SOLUZIONE COMPITO ESAME MATURITA: GESTIONE PANNELLO FOTOVOLTAICO

```
1122.#include <p18cxxx.h> // for TRISA and PORTA declarations
1123.#include <stdio.h>   // printf, puts etc
1124.#include <usart.h>   // funzioni UART
1125.#include <eep.h>     // funzioni EEPROM
1126.
1127.#pragma config OSC = HS   #pragma config WDT = OFF   #pragma config LVP = OFF
1128.
1129.unsigned long int timer; // contatore degli overflow di timer, globale, visibile al main e alla routine di interruzione
1130.
1131.#define ACK 6 // codice ASCII carattere ACK
1132.
1133.// impostazione  USART: no tx & rx interrupt, UART mode, 8N1, no disable, high BR
1134.#define PROTOCOL USART_TX_INT_OFF & USART_RX_INT_OFF & USART_ASYNCH_MODE & USART_EIGHT_BIT & USART_CONT_RX &
      USART_BRGH_HIGH
1135.
1136.#define PH 0xfc // valori per contare 1000 impulsi: il contatore è up modulo
1137.#define PL 0x18 // 65536: ci metto 64536 (0xfc18) così gli rimangono 1000 impulsi per l'overflow
1138.
1139.void main()
1140.{
1141. unsigned int i, val, cks, ch, id; // variabili necessarie
1142.
1143. // INIZIALIZZAZIONE PERIFERICHE NECESSARIE
1144. OpenUSART(PROTOCOL, 51);   // INIZIALIZZAZIONE UART: baud rate = (Fclock / 1024) - 1
1145.
1146. ADCON1 = 0x0e; // + e - vrif da alimentazione, ch 0, ch1 porta RA0 e RA1 ingressi analogici
1147. T0CON = 0x82;  // set up timer0 - prescaler 1:8 --> 1MHz --> 1us x count
1148. TMR0H = PH;    TMR0L = PL;    // setup period  high byte low byte 1 TOF = 1 ms
1149. INTCON = 0xa0; // abilito tutti le interruzioni e quelle del TMR0
1150.
1151. while(1)  // ciclo senza fine ACQUISIZIONE-SPEDIZIONE
1152. {
1153.  for(i = 0; i < 576; i++) // ACQUISIZIONE DATI: 0gni ora 12 acq, * 24 ore * 2 (V e I) = 576. Le acquisizioni sono a 8 bit.
1154.  {
1155.   timer = 300000;     while(timer);   // imposto timer a 300000 ms = 5 minuti e aspetto che timer diventi 0
1156.   if((i % 2) == 0) ADCON0 = 0x03; else ADCON0 = 0x07; // i pari: acq V da ch0,  i dispari: acq I da ch1; più GO = 1 e ADON = 1.
1157.   while(ADCON0 & 0x02);   // attesa EOC
1158.
1159.   Write_b_eep(i, ADRESH);  // scrivo gli 8 bit più signif. del risultato conversione allo  indirizzo i della EEPROM con una funzione di libreria...
1160.   Busy_eep();                        // aspetto che la scrittura sia finita...
1161.  }
1162.  do                    // SPEDIZIONE DATI COL METODO PAR  (se elimino il do while diventa trasmissione senza error recovery)
1163.  {                     // formato: S ID ch0 ch1...ch0 ch1 checksum  1+1+576+5 byte (la cks sta al più su 5 caratteri hex: 23DC2)
1164.
1165.            printf("S%01x", id);  // invio 'S' come delimitatore di pacchetto e  l'ID (id è 0 oppure 1 (un byte))
1166.
1167.            cks = 'S' + id; // inizializzo la checksum
1168.
1169.            for(i = 0; i < 576; i++) // per ogni byte acquisito:
1170.            {
1171.                      val = Read_b_eep(i); // lo rileggo dalla EEPROM
1172.                      printf("%02x", val); // lo spedisco in hex 2 byte
1173.                      cks += val;          // lo aggiungo alla checksum
1174.            }
1175.
1176.            printf("%05x", cks);      // invio la checksum
1177.
1178.            timer = 1000; // inizializzo timer a 1000 (1s)
1179.            while(timer)  if(DataRdyUSART()) ch = getcUSART(); // aspetto ACK dalla UART.per 1 s..
1180.
1181.  } while(ch != ACK); // se ACK non arriva entro il timeout rispedisco  il solito pacchetto e il solito ID
1182.  id = id + 1; id = id % 2; // se arriva ACK esco dal ciclo, incremento ID e procedo con 576 nuove acquisizioni...
1183. }
1184.}
1185.// Routine di interruzione di timer0
1186.#pragma interrupt GestioneTimer0    // GestioneTimer0 è una routine di interrupt
1187.
1188.void GestioneTimer0 ()              // questa è la routine di interrupt; va in esecuzione ogni ms
1189.{
1190.   if (INTCONbits.TMR0IF == 1)          // controllo se l'interruzione viene veramente da TMR0
1191.   {
1192.     if(timer) timer--;                 // se si faccio il lavoro...(decremento timer, se non è 0)
1193.     INTCONbits.TMR0IF = 0;             // &= 0xfb; devo segnalare che la interruzione è stata servita
1194.     TMR0H = PH; TMR0L = PL;            // ricarico il timer con il valore per 1000 impulsi
1195.   }
1196.}
1197.
1198.#pragma code  GestioneInterruzioneAP0 = 0x08 // carico il vettore di interruzione a 0x08
1199.
1200.void  GestioneInterruzioneAP0 { GestioneTimer0 (); }
```

// PROGRAMMA CONTROLLO CENTRALINA ACQUISIZIONE DATI PANNELLO FOTOVOLTAICO
// soluzione semplificata senza interruzioni (polled interrupt)

```
1201.#include <p18cxxx.h> // for TRISA and PORTA declarations
1202.#include <stdio.h>   // printf, puts etc
1203.#include <usart.h>   // funzioni UART
1204.#include <eep.h>     // funzioni EEPROM
1205.
1206.#pragma config OSC = HS   #pragma config WDT = OFF   #pragma config LVP = OFF
1207.
1208.#define ACK 6 // codice ASCII carattere ACK
1209.
1210.// impostazione  USART: no tx & rx interrupt, UART mode, 8N1, no disable, high BR
1211.#define PROTOCOL USART_TX_INT_OFF & USART_RX_INT_OFF & USART_ASYNCH_MODE & USART_EIGHT_BIT & USART_CONT_RX &
      USART_BRGH_HIGH
1212.
1213.#define PH 0xfc // valori per contare 1000 impulsi: il contatore è up modulo
1214.#define PL 0x18 // 65536: ci metto 64536 (0xfc18) così gli rimangono 1000 impulsi per l'overflow
1215.
1216. int Aspetta(int timer);  // torna 0 se timer è diventato 0, altrimenti torna 1.
1217.
1218.void main()
1219.{
1220. unsigned int i, val, cks, ch, id; // variabili necessarie
1221.
1222. // INIZIALIZZAZIONE PERIFERICHE NECESSARIE
1223.
1224. OpenUSART(PROTOCOL, 51);   // INIZIALIZZAZIONE UART: baud rate = (Fclock / 1024) - 1
1225.
1226. ADCON1 = 0x0e; // + e - vrif da alimentazione, ch 0, ch1 porta RA0 e RA1 ingressi analogici
1227. T0CON = 0x82;  // set up timer0 - prescaler 1:8 --> 1MHz --> 1us x count
1228. TMR0H = PH;    TMR0L = PL;    // setup period  high byte low byte 1 TOF = 1 ms
1229.
1230. while(1)  // ciclo senza fine ACQUISIZIONE-SPEDIZIONE
1231. {
1232.  for(i = 0; i < 576; i++) // ACQUISIZIONE DATI: 0gni ora 12 acq, * 24 ore * 2 (V e I) = 576. Le acquisizioni sono a 8 bit.
1233.  {
1234.   while(Aspetta(300000));  i// aspetto 300000 ms = 5 minuti
1235.
1236.   if((i % 2) == 0) ADCON0 = 0x03; else ADCON0 = 0x07; // i pari: acq V da ch0,  i dispari: acq I da ch1; più GO = 1 e ADON = 1.
1237.   while(ADCON0 & 0x02);   // attesa EOC
1238.
1239.   Write_b_eep(i, ADRESH);  // scrivo gli 8 bit più signif. del risultato conversione allo  indirizzo i della EEPROM con una funzione di libreria...
1240.   Busy_eep();                        // aspetto che la scrittura sia finita...
1241.  }
1242.
1243.  do                  // SPEDIZIONE DATI COL METODO PAR  (se elimino il do while diventa trasmissione senza error recovery)
1244.  {                   // formato: S ID ch0 ch1...ch0 ch1 checksum  1+1+576+5 byte (la cks sta al più su 5 caratteri hex: 23DC2)
1245.
1246.            printf("S%01x", id);  // invio 'S' come delimitatore di pacchetto e  l'ID (id è 0 oppure 1 (un byte))
1247.
1248.            cks = 'S' + id; // inizializzo la checksum
1249.
1250.            for(i = 0; i < 576; i++) // per ogni byte acquisito:
1251.            {
1252.                        val = Read_b_eep(i); // lo rileggo dalla EEPROM
1253.                        printf("%02x", val); // lo spedisco in hex 2 byte
1254.                        cks += val;          // lo aggiungo alla checksum
1255.            }
1256.
1257.            printf("%05x", cks);      // invio la checksum
1258.
1259.            while(Aspetta(1000))  if(DataRdyUSART()) ch = getcUSART(); // aspetto ACK dalla UART.per 1 s..
1260.
1261.  } while(ch != ACK); // se ACK non arriva entro il timeout rispedisco  il solito pacchetto e il solito ID
1262.
1263.  id = id + 1; id = id % 2; // se arriva ACK esco dal ciclo, incremento ID e procedo con 576 nuove acquisizioni...
1264. }
1265.}
1266.
1267. int Aspetta(int timer)  // torna 0 se timer è diventato 0, altrimenti torna 1.
1268. {
1269.    if (INTCONbits.TMR0IF == 1)          // controllo se  TMR0 ha avuto il rollover (o overflow che dir si voglia)...è passato un millisecondo
1270.    {
1271.      INTCONbits.TMR0IF = 0;             // &= 0xfb; se si azzero il flag di overflow del timer hw
1272.      if(timer) timer--;  else return(0);      // decremento timer, se non è 0, atrimenti ritorno 0 per segnalare fine dell'attesa
1273.      TMR0H = PH; TMR0L = PL;            // ricarico il timer con il valore per 1000 impulsi
1274.    }
1275.   return(1);   // segnalo che l'attesa non è finita
1276. }
```

```
// TOUCHDIMMER.c By D. Iracà, Pisa 16/11/2012 ver 1.0  PIC18F24xx
// INT0                                -> RB0 digital; input: syncro from rectfied & reduced AC mains
// TOUCH SENSOR                        -> RB1 digital; input
// OPTOTRIAC TRIGGER PULSE             -> RC2 digital; timer1 output (CCCP1)

#include <p18cxxx.h> // library headers

// pragmas
#pragma config OSC = INTIO67 // oscillatore interno
#pragma config WDT = OFF     // no watch dog
#pragma config LVP = OFF     // no low voltage programming

// definitions & macros
typedef enum {FALSE, TRUE} BOOL;
#define PH 0xfc // valori per contare 1000 impulsi: il contatore è up modulo
#define PL 0x18 // 65536: ci metto 64536 (0xfc18) così gli rimangono 1000 impulsi per l'overflow
#define    TWAIT0            5                                        // adj  (real time * 10)
#define    TWAIT1            10                                       // adj  (real time * 10)
#define  MAXSTEPS            10
#define  RAMP_TIME           30          // tempo di uno step  (real time * 10)

// globals
volatile unsigned int  timer, ramping_timer;
volatile unsigned char steps;
volatile unsigned char FiringDelay;
volatile BOOL upd, falling = FALSE;

const unsigned int FD[MAXSTEPS+1] = { 1990, 1800, 1600, 1400, 1200, 1000, 800, 600, 400, 200, 0 }; // angoli innesco (in cicli di clock)

// 1 ms delay routine
void delay(unsigned short int ritardo) { timer = ritardo; while(timer); }

void InitMcu() // mcu initialization
{
 ADCON1 = 0x0f;  CMCON  = 0x07; // no AD e no comparatore sulle porte A e B
 INTCON = 0x00;                 // disabilito tutte le interruzioni

 T0CON = 0x82;                  //  set up timer0 - prescaler 1:8 --> 1MHz --> 8us x count
 TMR0H = PH; TMR0L = PL;        // setup period high byte setup period low byte 1 TOF = 8 ms

 TRISB = 0xff;                  // porta B in ingresso..
 TRISCbits.TRISC2 = 0;          // bit RC2 in uscita

 TRISA = 0xf0;                  // bit RA0-1-2-3 in uscita (controllo)

 T1CON = 0x01;                                  // abilita il timer1 con i parametri di default

 INTCONbits.TMR0IE = 1;                 // abilito le interruzioni  da TMR0
 INTCONbits.INT0IE = 1;                 // abilito le interruzioni da INT0
 INTCONbits.GIE = 1;                    // abilito le interruzioni globali
}

// funzionamento dell'algoritmo di ramping:
// OnTime misura il tempo in ms che il TouchPad viene attivato: se OnTime è inferiore a TWAIT0 il tocco non ha effetto;
// se OnTime è > di TWAIT0 e < di TWAIT1 il tempo di innesco viene azzerato e il carico è spento;
// se OnTime è > di TWAIT1 il tempo di innesco viene aumentato a scatti con cadenza data da RAMP_TIME in ms e quando raggiunge
// il massimo viene fatta decrescere di nuovo fino a zero e così via finche il TouchPad è attivato.

void main()
{ //0
 BOOL Slope = TRUE;  unsigned int OnTime;

 timer = 0; ramping_timer = 0;  steps = 0; upd = FALSE;

 InitMcu(); // inizializzo mcu

 while(1) // forever:
 {
  if(PORTBbits.RB1 == 0) // se c'è tocco...gestione ramping...
  { //2
   OnTime = 0; while((PORTBbits.RB1 == 0) && (OnTime < TWAIT0)) { delay(1); OnTime++; }  // se c'è TOCCO... aspetto un pò...

   if(OnTime >= TWAIT0) // se c'è ancora
   { //3
    OnTime = 0; while((PORTBbits.RB1 == 0) && (OnTime < TWAIT1)) { delay(1); OnTime++; } // se c'è RF... aspetto un altro pò...

    if(OnTime >= TWAIT1) // se c'è ancora eseguo il ramping...
    { //4
       if(Slope) // ramp up...cresce
        { //5
           steps++;

           if(steps >= MAXSTEPS) { Slope = FALSE; } // end of positive ramp
```

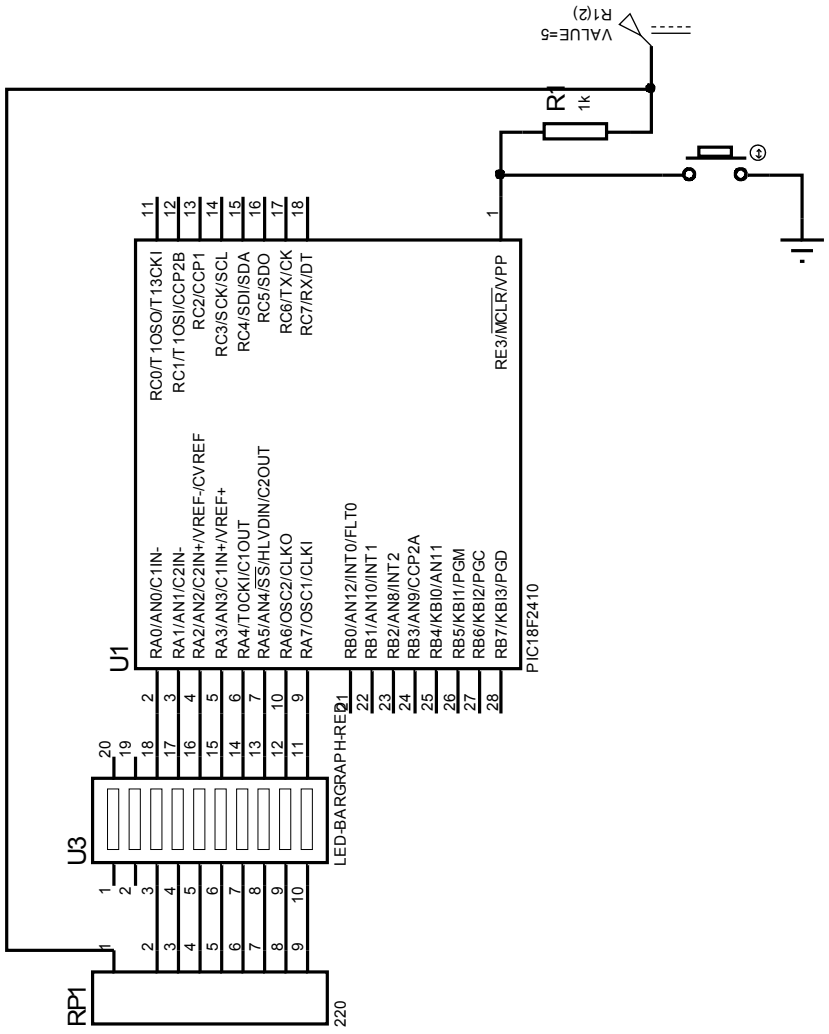
```
83            } //5
84            else // ramp down...decresce
85             { //5
86                  steps--;
87
88                  if(steps == 0) { Slope = TRUE; } // end of negative ramp
89               } //5
90
91               upd = TRUE;
92            } //4
93            else if(!ramping_timer) {/*4*/ steps = MAXSTEPS; upd = TRUE; Slope = TRUE; } //4 // altrimenti spengo.
94
95            delay(RAMP_TIME);  // dopo un pò di tempo faccio un altro ciclo do lettura touchpad...
96
97       ramping_timer = RAMP_TIME;
98      } //3
99     } //2
100  }
101 } //0
102
103 // Impostazione interrupt system  // ATTENZIONE O COSI O TUTTO IN FONDO MA DI SEGUITO!!
104
105 #pragma interrupt GestoreInterruzione     // segnala GestoreInterruzione come routine di interrupt (PSW sullo stack)
106
107 // Corpo Routine di interruzione
108
109 void GestoreInterruzione()              // questa è la routine di interrupt
110 {
111     if(INTCONbits.TMR0IF)                    // controllo se l'interruzione viene da TMR0
112     {
113               if(timer) timer--;                                        // timer generico
114
115               if(ramping_timer) ramping_timer--; // timer antirimbalzo
116
117               INTCONbits.TMR0IF = 0;           // segnalo che la interruzione è stata servita
118
119               TMR0H = PH; TMR0L = PL;          // ricarico il timer con il valore per 1000 impulsi
120     }
121
122      if(INTCONbits.INT0IF)
123       {
124               LATA ^= 0x02;          // lampeggio led controllo
125
126               CCP1CON = 0x08;        // riinizializzo il modulo compare match
127
128               if(!falling) { INTCON2bits.INTEDG0 = 0; falling = TRUE;       } else { INTCON2bits.INTEDG0 = 1; falling = FALSE; }
129
130               T1CON = 0x00; TMR1H = TMR1L = 0x00;                       // stops timer1
131
132               if(upd) { CCPR1 = FD[steps]; upd = FALSE; }     // avvio il CCP per generare il tempo di ritardo
133
134               T1CON = 0x01;                                             // starts timer1
135
136               INTCONbits.INT0IF = 0; // segnalo che la interruzione è stata servita
137        }
138 }
139
140 #pragma code GestoreInterruzioneAP = 0x08   // queste righe servono per caricare  il vettore di interruzione a 0x08
141
142 void GestoreInterruzioneAP() { GestoreInterruzione(); } // salto alla routine di interrupt
```

// TOUCHDIMMERBURST.c By D. Iracà, Pisa 16/11/2012 ver 1.0 PIC18F24xx

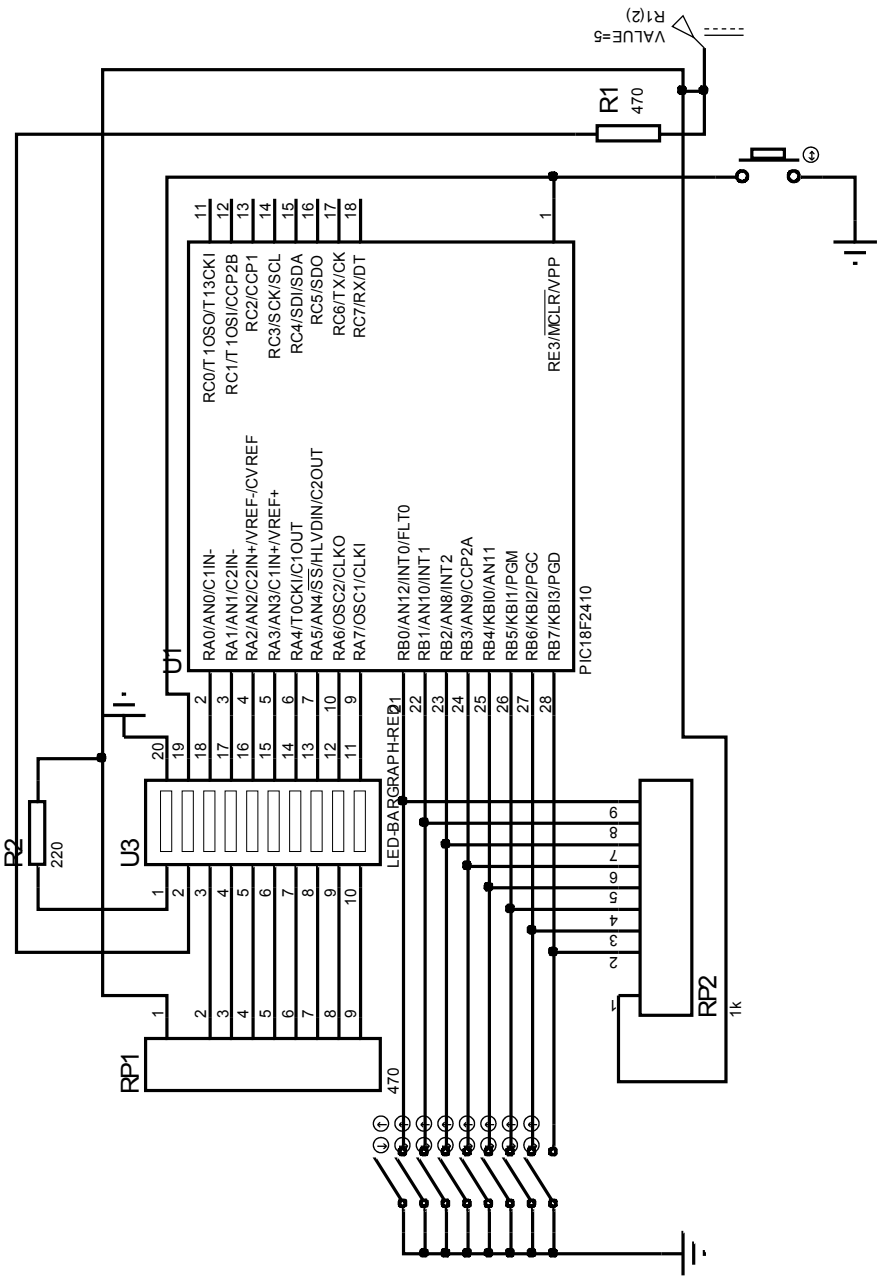
```
1   // T13CKI                                   -> RC0 digital; input: syncro from rectfied & reduced AC mains
2   // TOUCH SENSOR                             -> RB1 digital; input
3   // OPTOTRIAC TRIGGER PULSE  -> RC2 digital; timer1 output (CCCP1)
4
5   #include <p18cxxx.h> // library headers
6
7   // pragmas
8   #pragma config OSC = INTIO67 // oscillatore interno
9   #pragma config WDT = OFF     // no watch dog
10  #pragma config LVP = OFF     // no low voltage programming
11
12  // definitions & macros
13  typedef enum {FALSE, TRUE} BOOL;
14  #define PH 0xfc // valori per contare 1000 impulsi: il contatore è up modulo
15  #define PL 0x18 // 65536: ci metto 64536 (0xfc18) così gli rimangono 1000 impulsi per l'overflow
16  #define   TWAIT0                  5                                   // adj  (real time * 10)
17  #define   TWAIT1                  10                                  // adj  (real time * 10)
18  #define  MAXSTEPS      10
19  #define  RAMP_TIME     30          // tempo di uno step  (real time * 10)
20
21  // globals
22  volatile unsigned int  timer, ramping_timer;
23  volatile unsigned char steps;
24  BOOL upd;
25
26  // 1 ms delay routine
27  void delay(unsigned short int ritardo) { timer = ritardo; while(timer); }
28
29  void InitMcu() // mcu initialization
30  {
31   ADCON1 = 0x0f;  CMCON  = 0x07; // no AD e no comparatore sulle porte A e B
32   INTCON = 0x00;              // disabilito tutte le interruzioni
33
34   T0CON = 0x82;              //  set up timer0 - prescaler 1:8 --> 1MHz --> 8us x count
35   TMR0H = PH; TMR0L = PL;      // setup period high byte setup period low byte 1 TOF = 8 ms
36
37   TRISCbits.TRISC2 = 0;      // bit RC2 in uscita
38   TRISCbits.TRISC0 = 1;      // bit RC0 in ingresso
39
40   TRISA = 0xf0;             // bit RA0-1-2-3 in uscita (controllo)
41
42   TMR1H = (0xFFFF-MAXSTEPS) >> 8; TMR1L = (0xFFFF-MAXSTEPS) & 0x00FF;
43
44   CCP1CON = 0x09;            // riinizializzo il modulo compare match, RC2 high, low on compare match
45
46   PIE1bits.TMR1IE = 1;           // abilita interrupt su timer overflow da TMR1
47
48   INTCONbits.TMR0IE = 1;             // abilito le interruzioni  da TMR0
49   INTCONbits.GIE = 1;               // abilito le interruzioni globali
50
51   T1CON = 0x03;                       // abilita il timer1 con i parametri di default ma da sorgente esterna (l'AC)
52
53   CCPR1 = (0xFFFF-MAXSTEPS);   // inizializzo il compare match register
54  }
55
56  // funzionamento dell'algoritmo di ramping:
57  // OnTime misura il tempo in ms che il TouchPad viene attivato: se OnTime è inferiore a TWAIT0 il tocco non ha effetto;
58  // se OnTime è > di TWAIT0 e < di TWAIT1 il tempo di innesco viene azzerato e il carico è spento;
59  // se OnTime è > di TWAIT1 il tempo di innesco viene aumentato a scatti con cadenza data da RAMP_TIME in ms e quando
60  // raggiunge il massimo viene fatta decrescere di nuovo fino a zero e così via finche il TouchPad è attivato.
61
62  void main()
63  { //0
64   BOOL Slope = TRUE;  unsigned int OnTime;
65
66   timer = 0; ramping_timer = 0;  steps = 0; upd = FALSE;
67
68   InitMcu(); // inizializzo mcu
69
70   while(1) // forever:
71   {
```

```
72    if(PORTBbits.RB1 == 0) // se c'è tocco...gestione ramping...
73    { //2
74     OnTime = 0; while((PORTBbits.RB1 == 0) && (OnTime < TWAIT0)) { delay(1); OnTime++; }  // se c'è TOCCO... aspetto un pò...
75
76     if(OnTime >= TWAIT0) // se c'è ancora
77     { //3
78      OnTime = 0; while((PORTBbits.RB1 == 0) && (OnTime < TWAIT1)) { delay(1); OnTime++; } // se c'è RF... aspetto un altro pò...
79
80      if(OnTime >= TWAIT1) // se c'è ancora eseguo il ramping...
81              { //4
82               if(Slope) // ramp up...cresce
83               { //5
84                 steps++;
85
86               if(steps >= MAXSTEPS) { Slope = FALSE; } // end of positive ramp
87               } //5
88               else // ramp down...decresce
89               { //5
90                steps--;
91
92               if(steps == 0) { Slope = TRUE; } // end of negative ramp
93               } //5
94              upd = TRUE;
95              } //4
96              else if(!ramping_timer) {/*4*/ steps = MAXSTEPS; upd = TRUE; Slope = TRUE; } //4 // altrimenti spengo.
97
98              delay(RAMP_TIME);  // dopo un pò di tempo faccio un altro ciclo do lettura touchpad...
99
100     ramping_timer = RAMP_TIME;
101    } //3
102   } //2
103  }
104 } //0
105
106 // Impostazione interrupt system  // ATTENZIONE O COSI O TUTTO IN FONDO MA DI SEGUITO!!
107
108 #pragma interrupt GestoreInterruzione     // segnala GestoreInterruzione come routine di interrupt (PSW sullo stack)
109
110 // Corpo Routine di interruzione
111 void GestoreInterruzione()              // questa è la routine di interrupt
112 {
113             if(INTCONbits.TMR0IF)                   // controllo se l'interruzione viene da TMR0
114     {
115                         if(timer) timer--;                                      // timer generico
116
117                         if(ramping_timer) ramping_timer--; // timer antirimbalzo
118
119                INTCONbits.TMR0IF = 0;           // segnalo che la interruzione è stata servita
120
121             TMR0H = PH; TMR0L = PL;          // ricarico il timer con il valore per 1000 impulsi
122     }
123
124             if(PIR1bits.TMR1IF)                                             // controllo se l'interruzione viene da CCP1
125             {
126                         LATA ^= 0x02;          // lampeggio led controllo
127
128                         TMR1H = (0xFFFF-MAXSTEPS) >> 8; TMR1L = (0xFFFF-MAXSTEPS) & 0x00FF;
129
130                         if(upd) { CCPR1 = 0xFFFF-steps; upd = FALSE; }     // imposto il compare match register
131
132                         CCP1CON = 0x09;                         // riinizializzo il modulo compare match
133
134                         PIR1bits.TMR1IF = 0;           // segnalo che la interruzione è stata servita
135             }
136 }
137
138 #pragma code GestoreInterruzioneAP = 0x08   // queste righe servono per caricare  il vettore di interruzione a 0x08
139
140 void GestoreInterruzioneAP() { GestoreInterruzione(); } // salto alla routine di interrupt
141
```

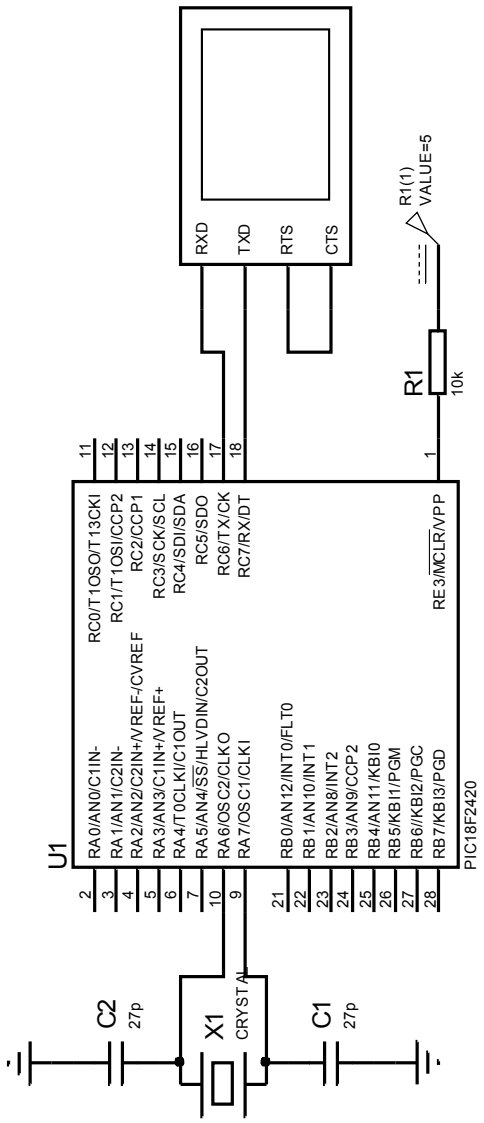
Blink1 Blink2



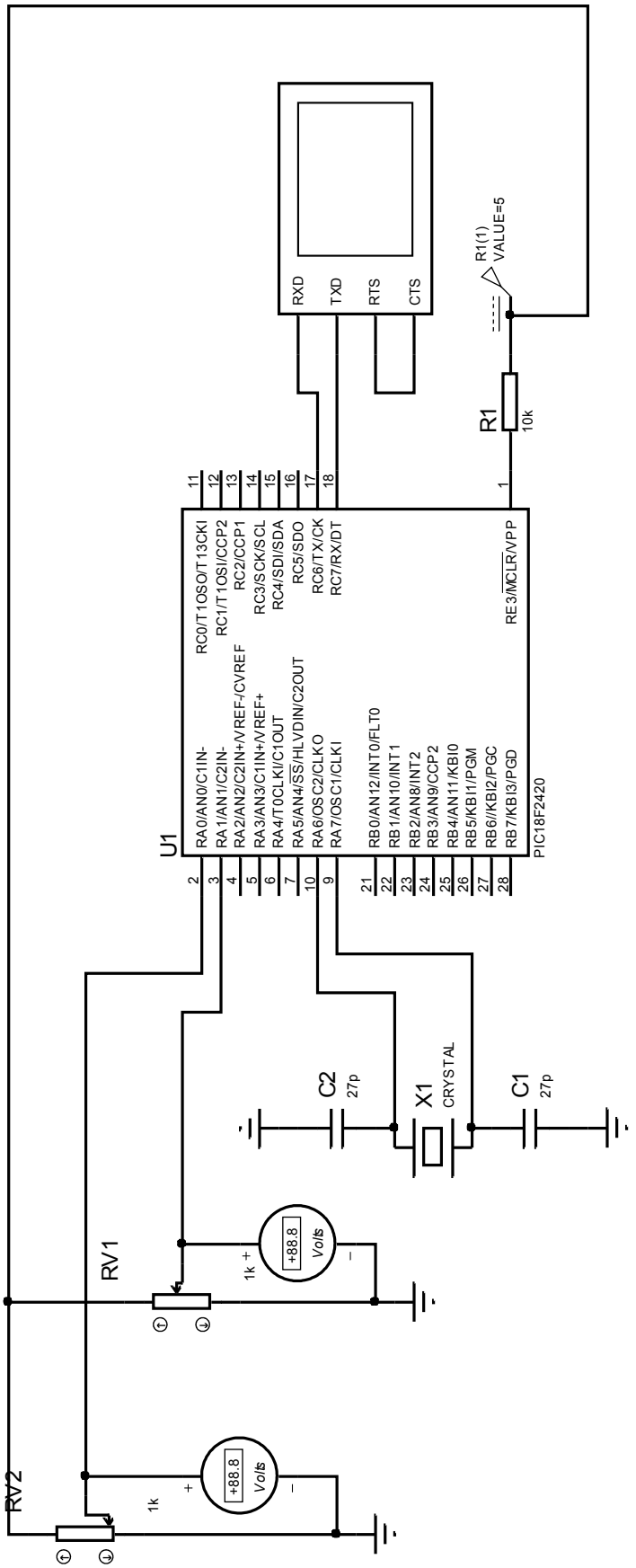
BAL1

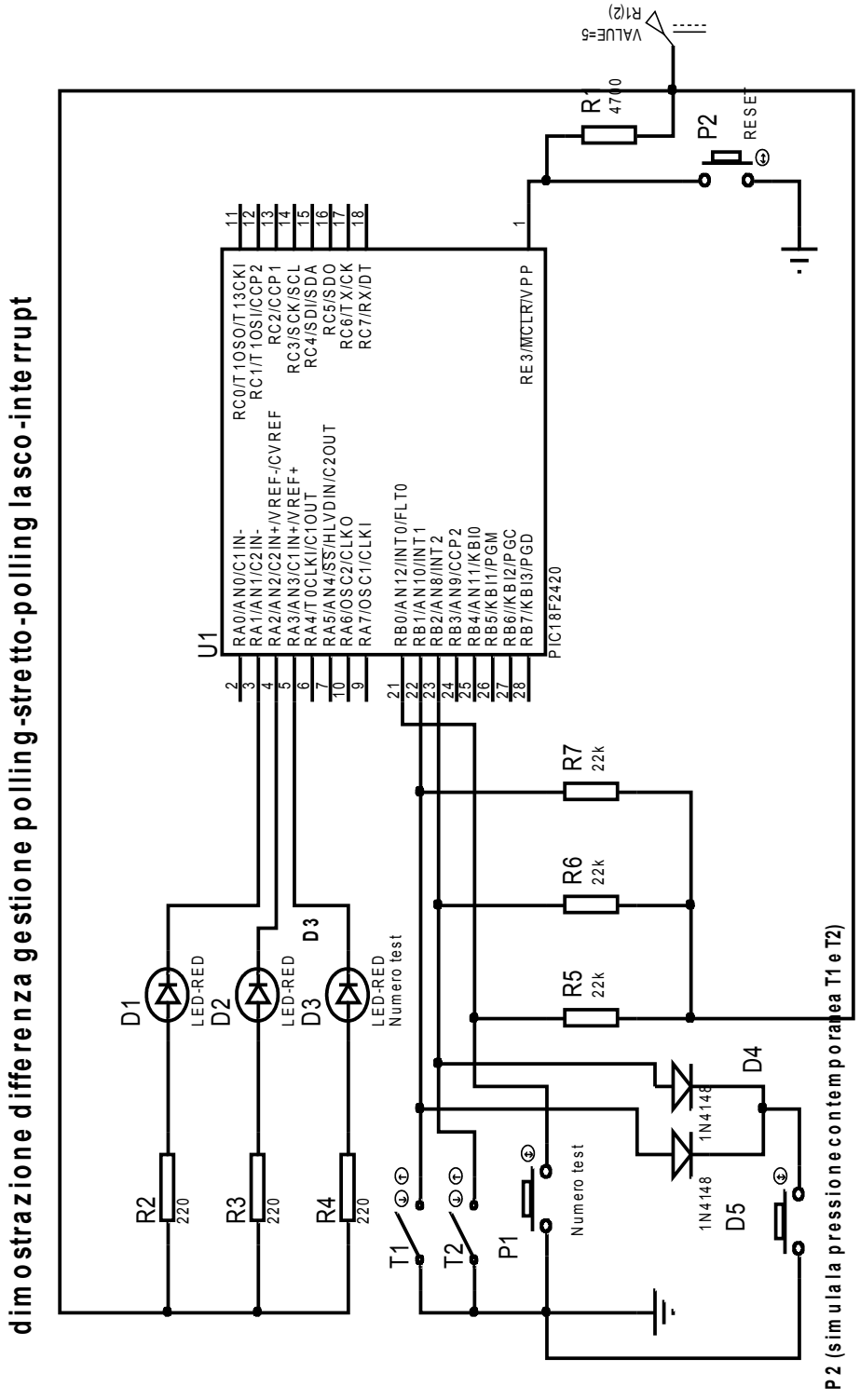

VALUE=5
R1(2)
R1
470
U1
PIC18F2410
RA0/AN0/C1IN-
RA1/AN1/C2IN-
RA2/AN2/C2IN+/VREF-/CVREF
RA3/AN3/C1IN+/VREF+
RA4/T0CKI/C1OUT
RA5/AN4/SS/HLVDIN/C2OUT
RA6/OSC2/CLKO
RA7/OSC1/CLKI
RB0/AN12/INT0/FLT0
RB1/AN10/INT1
RB2/AN8/INT2
RB3/AN9/CCP2A
RB4/KBI0/AN11
RB5/KBI1/PGM
RB6/KBI2/PGC
RB7/KBI3/PGD
RC0/T1OSO/T13CKI
RC1/T1OSI/CCP2B
RC2/CCP1
RC3/SCK/SCL
RC4/SDI/SDA
RC5/SDO
RC6/TX/CK
RC7/RX/DT
RE3/MCLR/VPP
U3
LED-BARGRAPH-RED
R2
220
RP1
470
RP2
1k

Echo1 Uart1 Uart2



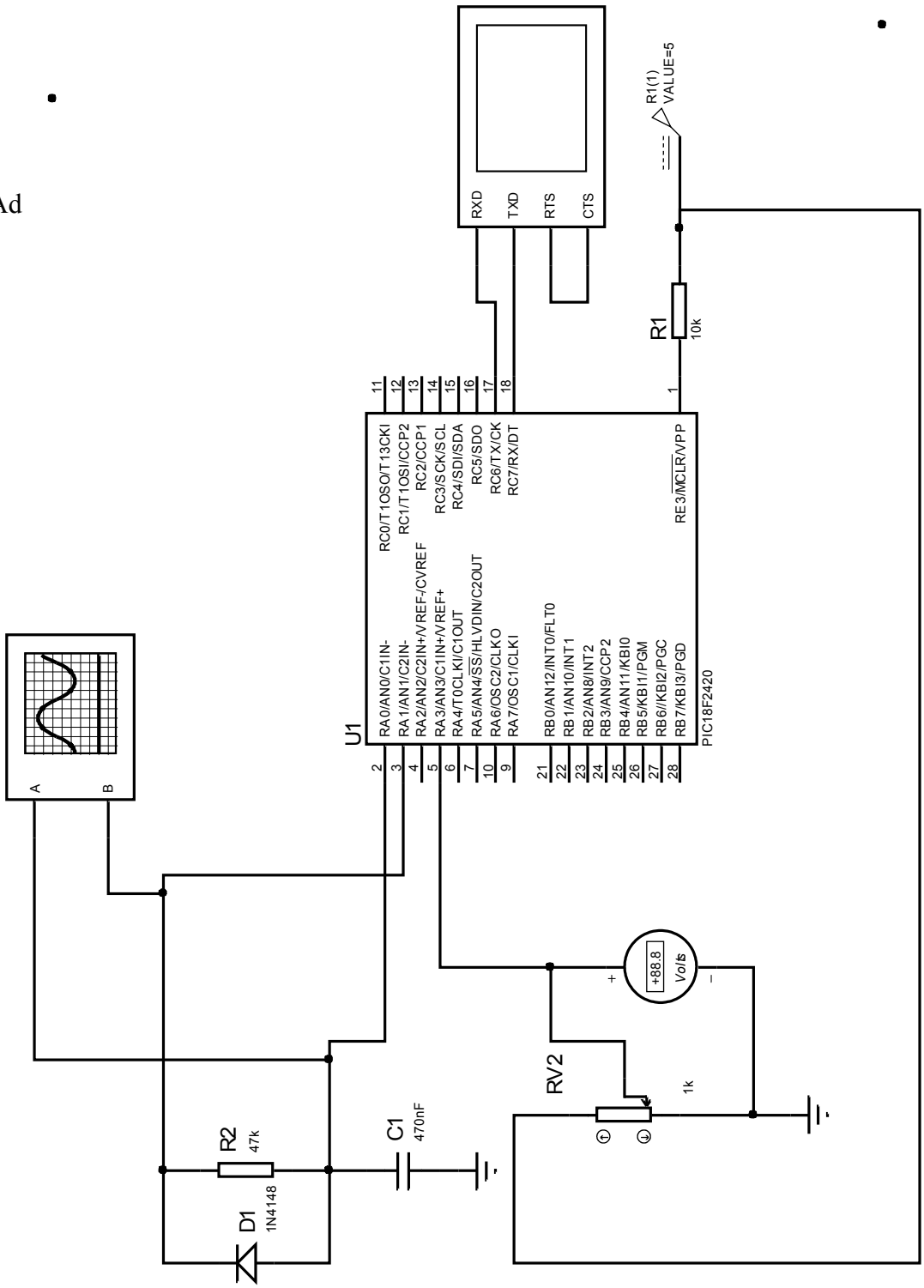
AD1 AD2


U1
PIC18F2420
RA0/AN0/C1IN-
RA1/AN1/C2IN-
RA2/AN2/C2IN+/VREF-/CVREF
RA3/AN3/C1IN+/VREF+
RA4/T0CLKI/C1OUT
RA5/AN4/SS/HLVDIN/C2OUT
RA6/OSC2/CLKO
RA7/OSC1/CLKI
RB0/AN12/INT0/FLT0
RB1/AN10/INT1
RB2/AN8/INT2
RB3/AN9/CCP2
RB4/AN11/KBI0
RB5/KBI1/PGM
RB6/KBI2/PGC
RB7/KBI3/PGD
RC0/T1OSO/T13CKI
RC1/T1OSI/CCP2
RC2/CCP1
RC3/SCK/SCL
RC4/SDI/SDA
RC5/SDO
RC6/TX/CK
RC7/RX/DT
RE3/MCLR/VPP
RXD
TXD
RTS
CTS
R1
10k
R1(1)
VALUE=5
C2
27p
X1
CRYSTAL
C1
27p
RV1
1k
RV2
1k
+88.8
Volts

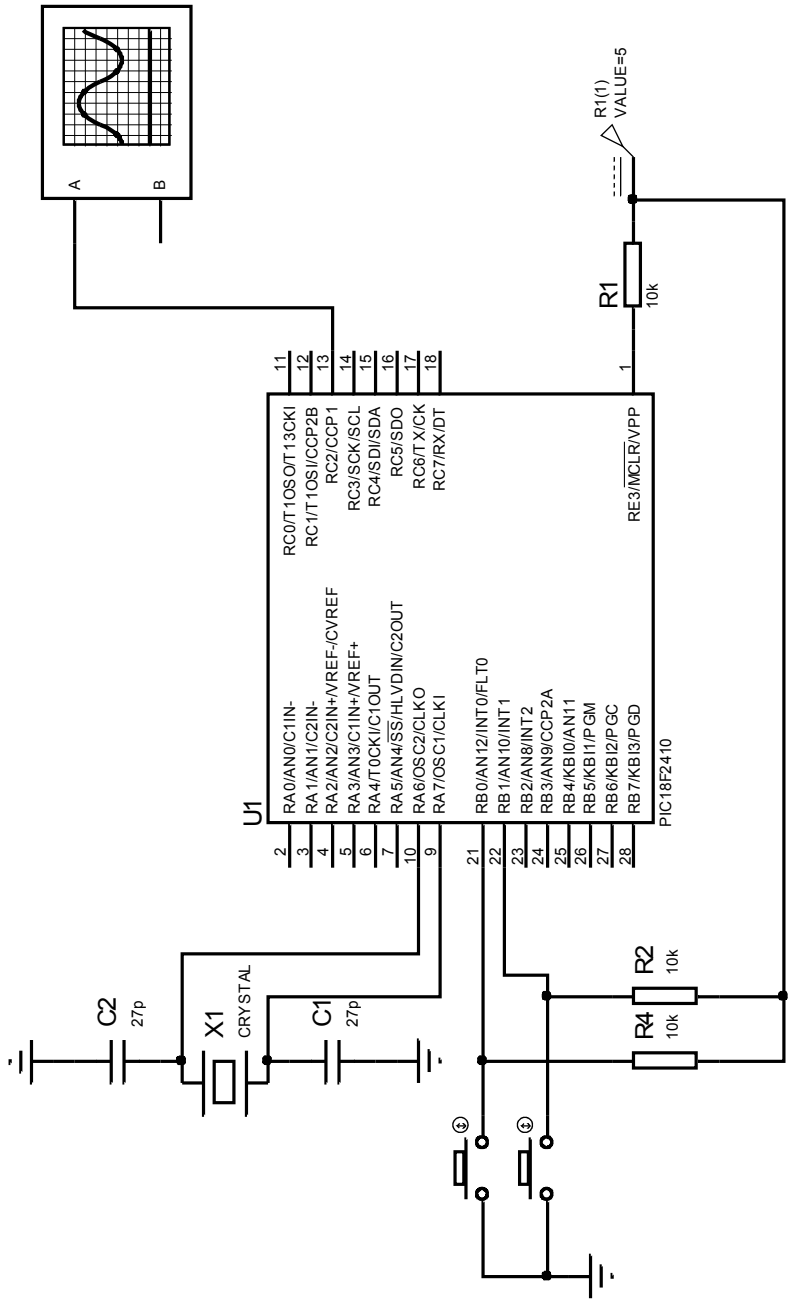

dimostrazione differenza gestione polling-stretto-polling lasco-interrupt
U1
PIC18F2420
RA0/AN0/C1IN-
RA1/AN1/C2IN-
RA2/AN2/C2IN+/VREF-/CVREF
RA3/AN3/C1IN+/VREF+
RA4/T0CLKI/C1OUT
RA5/AN4/SS/HLVDIN/C2OUT
RA6/OSC2/CLKO
RA7/OSC1/CLKI
RB0/AN12/INT0/FLT0
RB1/AN10/INT1
RB2/AN8/INT2
RB3/AN9/CCP2
RB4/AN11/KBI0
RB5/KBI1/PGM
RB6/KBI2/PGC
RB7/KBI3/PGD
RC0/T1OSO/T13CKI
RC1/T1OSI/CCP2
RC2/CCP1
RC3/SCK/SCL
RC4/SDI/SDA
RC5/SDO
RC6/TX/CK
RC7/RX/DT
RE3/MCLR/VPP
R1
4700
VALUE=5
R1(2)
P2
RESET
R2
220
R3
220
R4
220
D1
LED-RED
D2
LED-RED
D3
LED-RED
Numero test
R7
22k
R6
22k
R5
22k
D4
1N4148
1N4148
D5
T1
T2
P1
Numero test
P2 (simula la pressione contemporanea T1 e T2)

polvsint

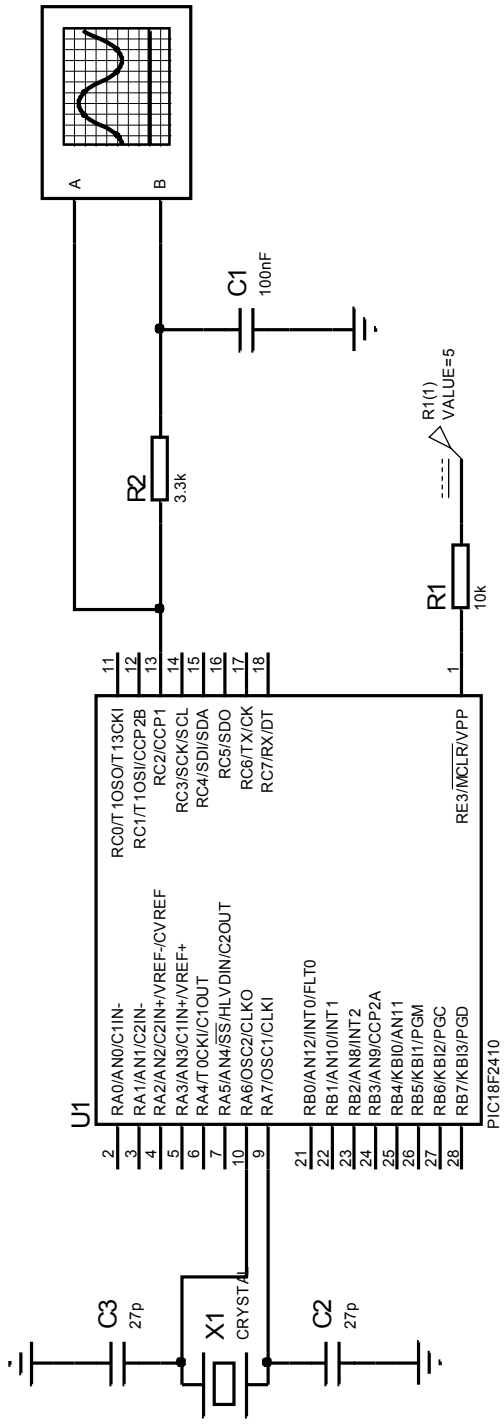
ExRampAd


RXD
TXD
RTS
CTS
R1(1)
VALUE=5
R1
10k
U1
RC0/T1OSO/T13CKI
RC1/T1OSI/CCP2
RC2/CCP1
RC3/SCK/SCL
RC4/SDI/SDA
RC5/SDO
RC6/TX/CK
RC7/RX/DT
RA0/AN0/C1IN-
RA1/AN1/C2IN-
RA2/AN2/C2IN+/VREF-/CVREF
RA3/AN3/C1IN+/VREF+
RA4/T0CLKI/C1OUT
RA5/AN4/SS/HLVDIN/C2OUT
RA6/OSC2/CLKO
RA7/OSC1/CLKI
RB0/AN12/INT0/FLT0
RB1/AN10/INT1
RB2/AN8/INT2
RB3/AN9/CCP2
RB4/AN11/KBI0
RB5/KBI1/PGM
RB6//KBI2/PGC
RB7/KBI3/PGD
RE3/MCLR/VPP
PIC18F2420
A
B
+88.8
Volts
RV2
1k
R2
47k
C1
470nF
D1
1N4148

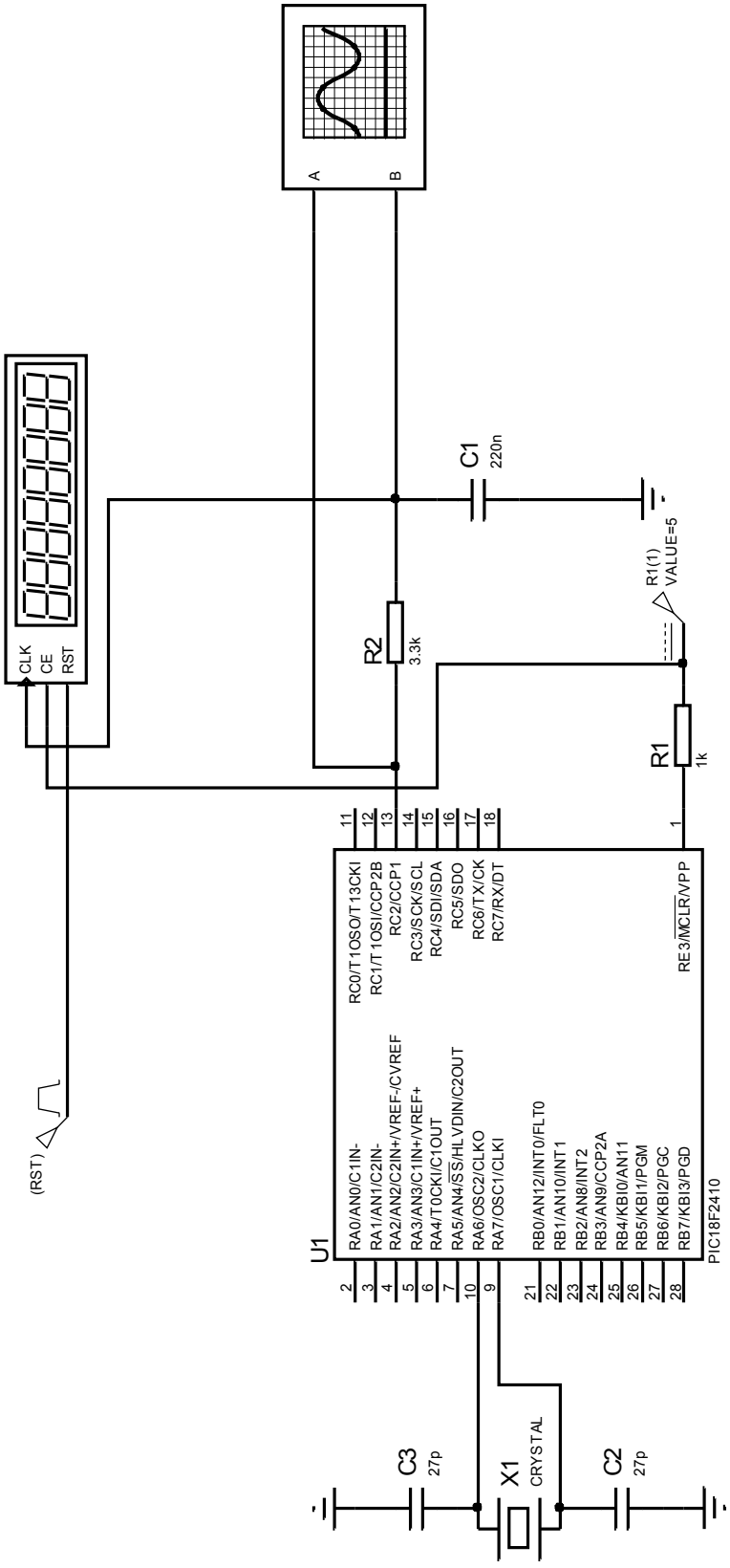
PWM1


A
B
R1(1)
VALUE=5
R1
10k
U1
RC0/T1OSO/T13CKI
RC1/T1OSI/CCP2B
RC2/CCP1
RC3/SCK/SCL
RC4/SDI/SDA
RC5/SDO
RC6/TX/CK
RC7/RX/DT
RE3/MCLR/VPP
RA0/AN0/C1IN-
RA1/AN1/C2IN-
RA2/AN2/C2IN+/VREF-/CVREF
RA3/AN3/C1IN+/VREF+
RA4/T0CKI/C1OUT
RA5/AN4/SS/HLVDIN/C2OUT
RA6/OSC2/CLKO
RA7/OSC1/CLKI
RB0/AN12/INT0/FLT0
RB1/AN10/INT1
RB2/AN8/INT2
RB3/AN9/CCP2A
RB4/KBI0/AN11
RB5/KBI1/PGM
RB6/KBI2/PGC
RB7/KBI3/PGD
PIC18F2410
C2
27p
X1
CRYSTAL
C1
27p
R2
10k
R4
10k

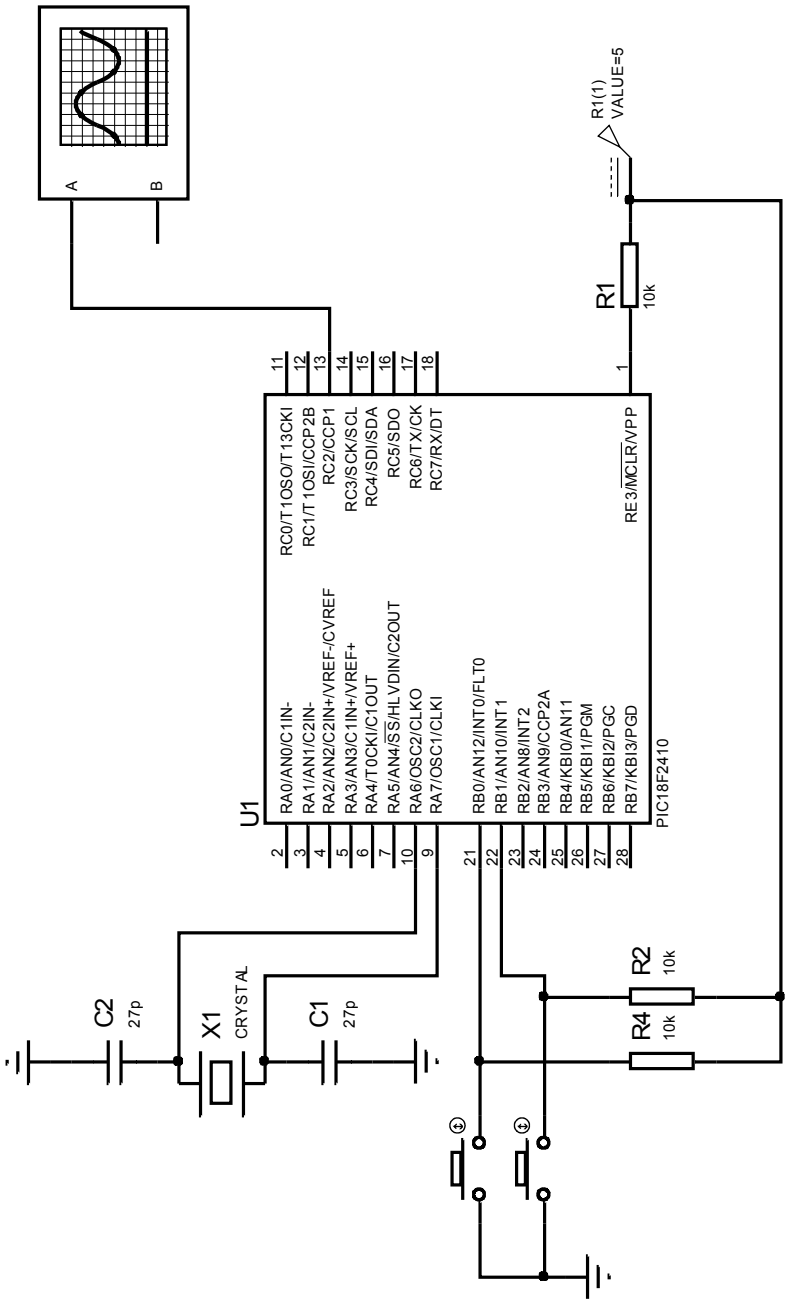
SINE


A
B
C1
100nF
R2
3.3k
R1(1)
VALUE=5
R1
10k
U1
RC0/T1OSO/T13CKI
RC1/T1OSI/CCP2B
RC2/CCP1
RC3/SCK/SCL
RC4/SDI/SDA
RC5/SDO
RC6/TX/CK
RC7/RX/DT
RA0/AN0/C1IN-
RA1/AN1/C2IN-
RA2/AN2/C2IN+/VREF-/CVREF
RA3/AN3/C1IN+/VREF+
RA4/T0CKI/C1OUT
RA5/AN4/SS/HLVDIN/C2OUT
RA6/OSC2/CLKO
RA7/OSC1/CLKI
RB0/AN12/INT0/FLT0
RB1/AN10/INT1
RB2/AN8/INT2
RB3/AN9/CCP2A
RB4/KBI0/AN11
RB5/KBI1/PGM
RB6/KBI2/PGC
RB7/KBI3/PGD
RE3/MCLR/VPP
PIC18F2410
C3
27p
X1
CRYSTAL
C2
27p

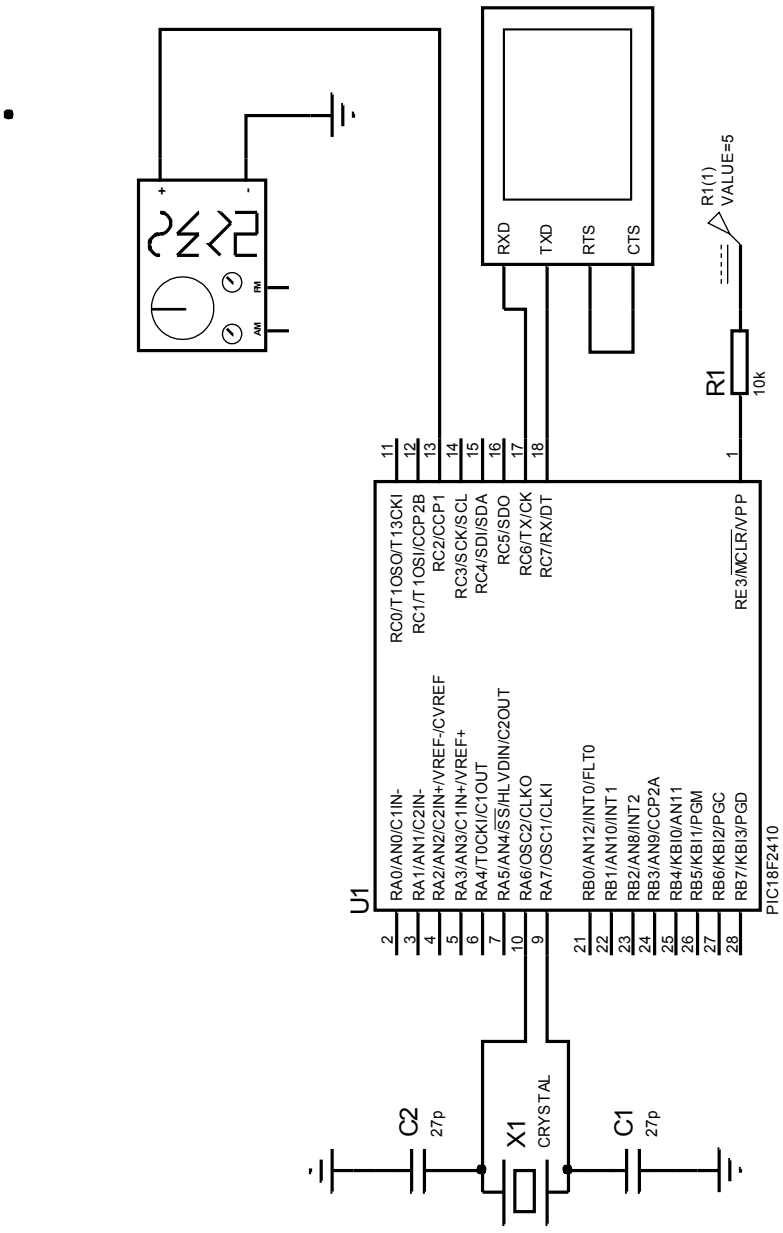
TRIANGLE


A
B
C1
220n
R2
3.3k
R1(1)
VALUE=5
R1
1k
CLK
CE
RST
(RST)
U1
PIC18F2410
RC0/T1OSO/T13CKI
RC1/T1OSI/CCP2B
RC2/CCP1
RC3/SCK/SCL
RC4/SDI/SDA
RC5/SDO
RC6/TX/CK
RC7/RX/DT
RE3/MCLR/VPP
RA0/AN0/C1IN-
RA1/AN1/C2IN-
RA2/AN2/C2IN+/VREF-/CVREF
RA3/AN3/C1IN+/VREF+
RA4/T0CKI/C1OUT
RA5/AN4/SS/HLVDIN/C2OUT
RA6/OSC2/CLKO
RA7/OSC1/CLKI
RB0/AN12/INT0/FLT0
RB1/AN10/INT1
RB2/AN8/INT2
RB3/AN9/CCP2A
RB4/KBI0/AN11
RB5/KBI1/PGM
RB6/KBI2/PGC
RB7/KBI3/PGD
C3
27p
X1
CRYSTAL
C2
27p

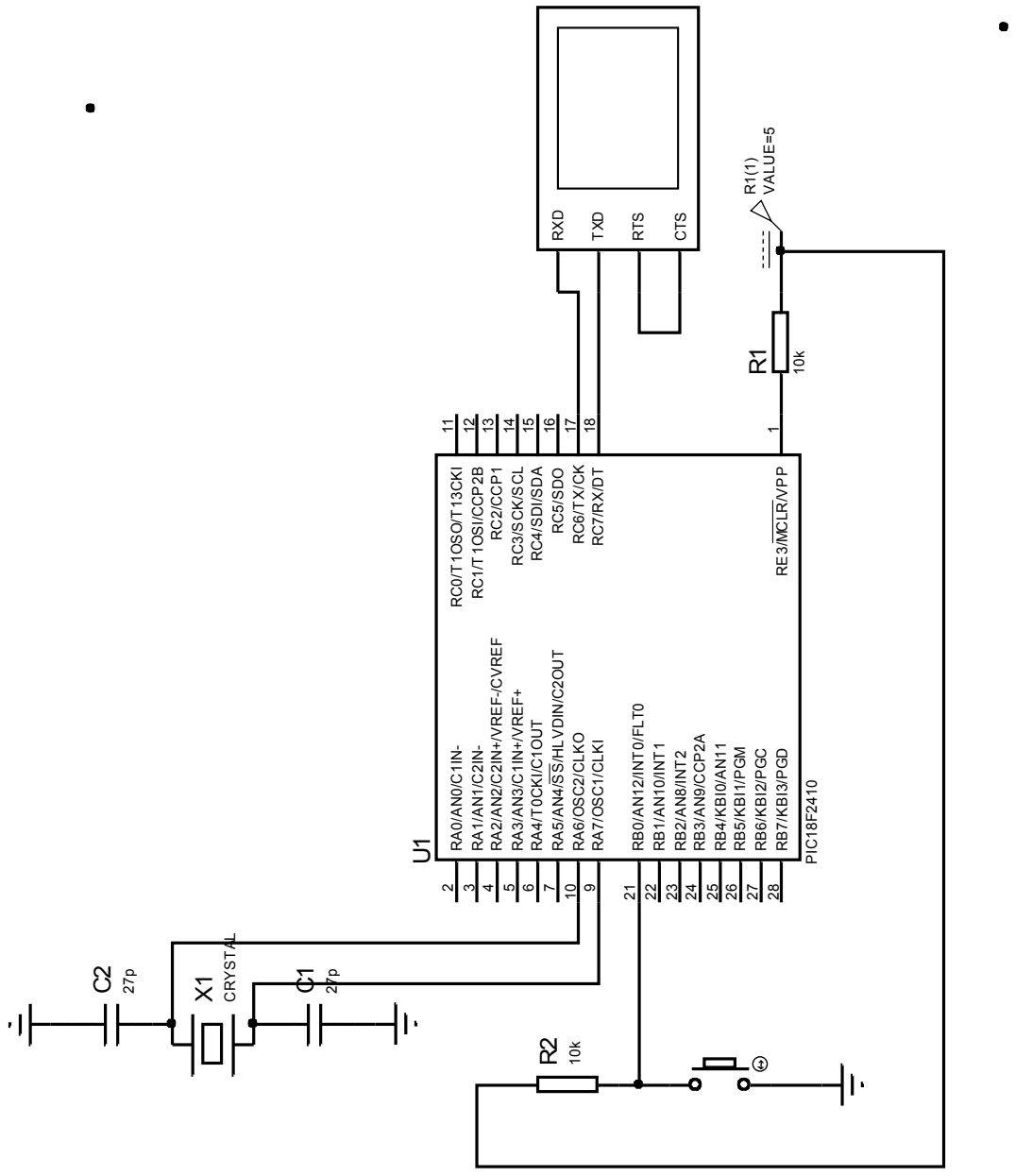
OC1

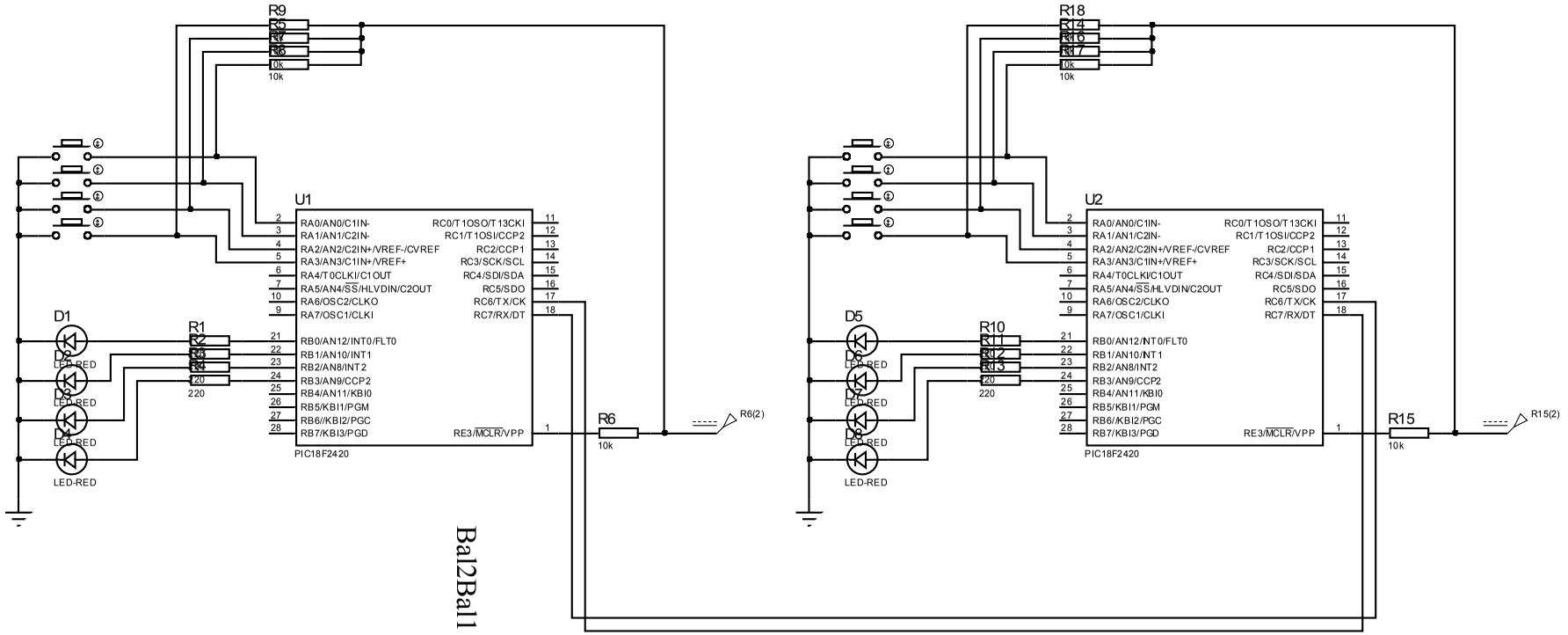

A
B
R1(1)
VALUE=5
R1
10k
U1
RC0/T1OSO/T13CKI
RC1/T1OSI/CCP2B
RC2/CCP1
RC3/SCK/SCL
RC4/SDI/SDA
RC5/SDO
RC6/TX/CK
RC7/RX/DT
RA0/AN0/C1IN-
RA1/AN1/C2IN-
RA2/AN2/C2IN+/VREF-/CVREF
RA3/AN3/C1IN+/VREF+
RA4/T0CKI/C1OUT
RA5/AN4/SS/HLVDIN/C2OUT
RA6/OSC2/CLKO
RA7/OSC1/CLKI
RB0/AN12/INT0/FLT0
RB1/AN10/INT1
RB2/AN8/INT2
RB3/AN9/CCP2A
RB4/KBI0/AN11
RB5/KBI1/PGM
RB6/KBI2/PGC
RB7/KBI3/PGD
RE3/MCLR/VPP
PIC18F2410
C2
27p
X1
CRYSTAL
C1
27p
R2
10k
R4
10k

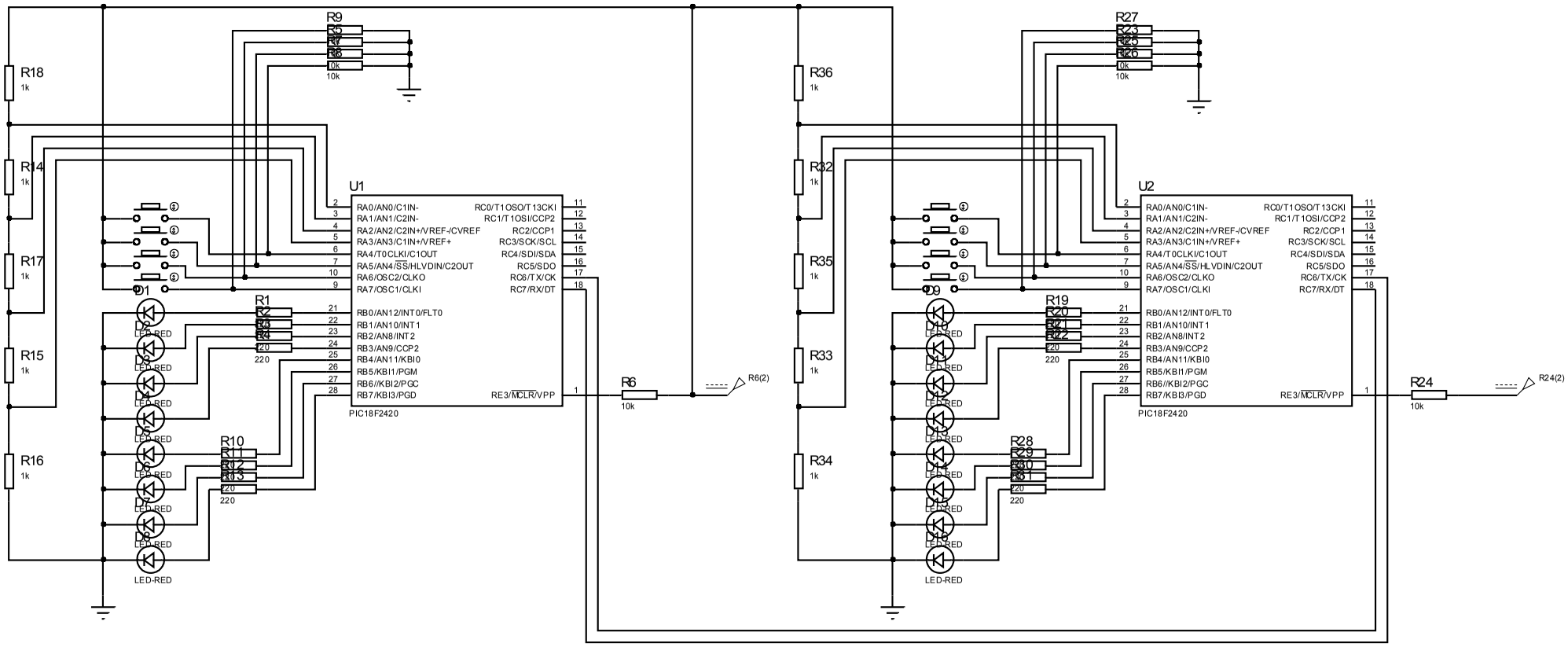
IC1


U1
PIC18F2410
RA0/AN0/C1IN-
RA1/AN1/C2IN-
RA2/AN2/C2IN+/VREF-/CVREF
RA3/AN3/C1IN+/VREF+
RA4/T0CKI/C1OUT
RA5/AN4/SS/HLVDIN/C2OUT
RA6/OSC2/CLKO
RA7/OSC1/CLKI
RB0/AN12/INT0/FLT0
RB1/AN10/INT1
RB2/AN8/INT2
RB3/AN9/CCP2A
RB4/KBI0/AN11
RB5/KBI1/PGM
RB6/KBI2/PGC
RB7/KBI3/PGD
RC0/T1OSO/T13CKI
RC1/T1OSI/CCP2B
RC2/CCP1
RC3/SCK/SCL
RC4/SDI/SDA
RC5/SDO
RC6/TX/CK
RC7/RX/DT
RE3/MCLR/VPP
RXD
TXD
RTS
CTS
R1
10k
R1(1)
VALUE=5
X1
CRYSTAL
C2
27p
C1
27p

EXTINT1

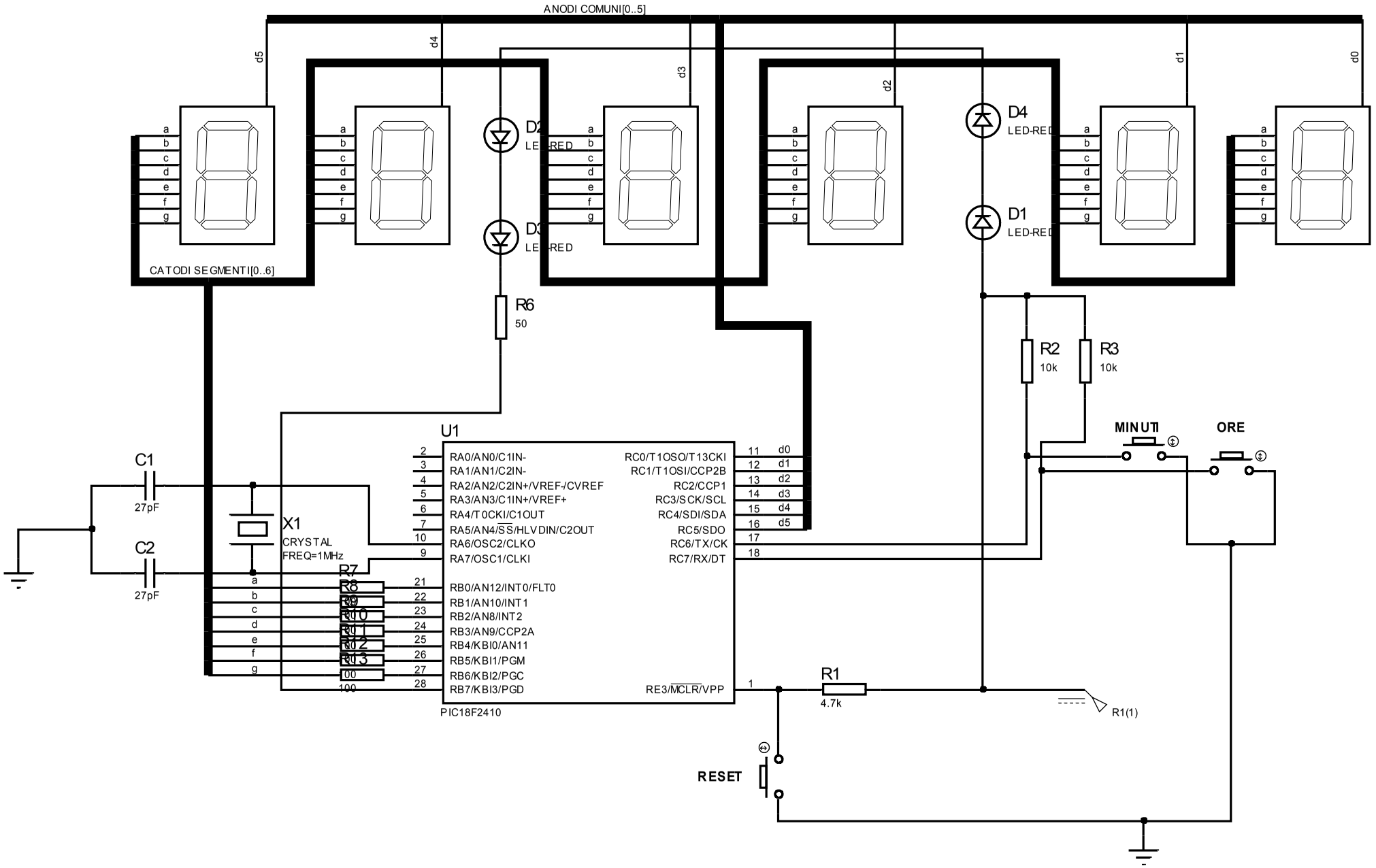

RXD
TXD
RTS
CTS
R1(1)
VALUE=5
R1
10k
U1
RC0/T1OSO/T13CKI
RC1/T1OSI/CCP2B
RC2/CCP1
RC3/SCK/SCL
RC4/SDI/SDA
RC5/SDO
RC6/TX/CK
RC7/RX/DT
RE3/MCLR/VPP
RA0/AN0/C1IN-
RA1/AN1/C2IN-
RA2/AN2/C2IN+/VREF-/CVREF
RA3/AN3/C1IN+/VREF+
RA4/T0CKI/C1OUT
RA5/AN4/SS/HLVDIN/C2OUT
RA6/OSC2/CLKO
RA7/OSC1/CLKI
RB0/AN12/INT0/FLT0
RB1/AN10/INT1
RB2/AN8/INT2
RB3/AN9/CCP2A
RB4/KBI0/AN11
RB5/KBI1/PGM
RB6/KBI2/PGC
RB7/KBI3/PGD
PIC18F2410
C2
27p
X1
CRYSTAL
C1
27p
R2
10k


R9
10k
U1
RA0/AN0/C1IN-
RA1/AN1/C2IN-
RA2/AN2/C2IN+/VREF-/CVREF
RA3/AN3/C1IN+/VREF+
RA4/T0CLKI/C1OUT
RA5/AN4/SS/HLVDIN/C2OUT
RA6/OSC2/CLKO
RA7/OSC1/CLKI
RB0/AN12/INT0/FLT0
RB1/AN10/INT1
RB2/AN8/INT2
RB3/AN9/CCP2
RB4/AN11/KBI0
RB5/KBI1/PGM
RB6/KBI2/PGC
RB7/KBI3/PGD
RC0/T1OSO/T13CKI
RC1/T1OSI/CCP2
RC2/CCP1
RC3/SCK/SCL
RC4/SDI/SDA
RC5/SDO
RC6/TX/CK
RC7/RX/DT
RE3/MCLR/VPP
PIC18F2420
D1
D2
D3
D4
LED-RED
R1
220
R6
10k
R6(2)
R18
10k
U2
PIC18F2420
D5
D6
D7
D8
LED-RED
R10
220
R15
10k
R15(2)
Bal2Ball

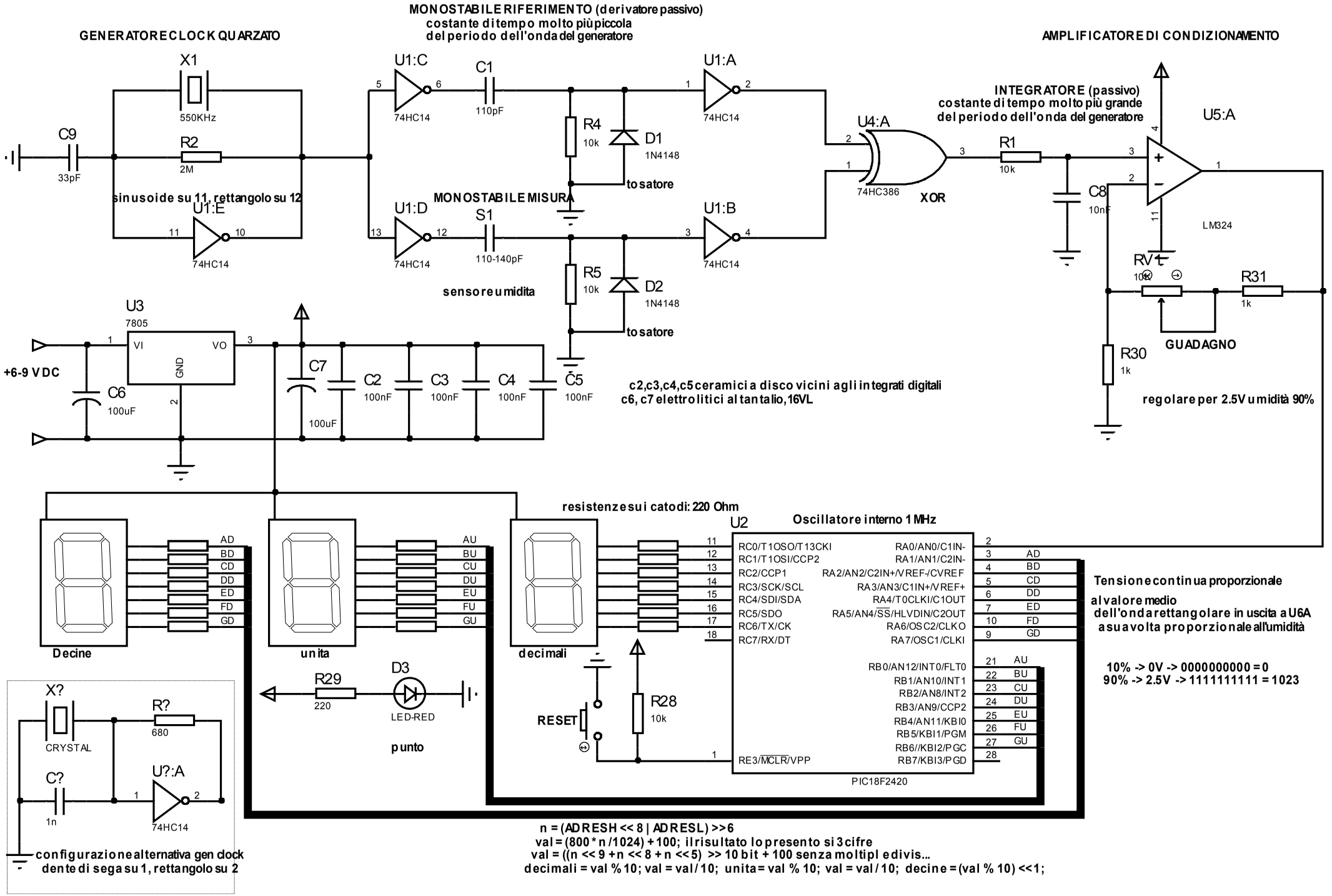

R9
R18
1k
R14
1k
R17
1k
R15
1k
R16
1k
U1
RA0/AN0/C1IN-
RA1/AN1/C2IN-
RA2/AN2/C2IN+/VREF-/CVREF
RA3/AN3/C1IN+/VREF+
RA4/T0CLKI/C1OUT
RA5/AN4/SS/HLVDIN/C2OUT
RA6/OSC2/CLKO
RA7/OSC1/CLKI
RB0/AN12/INT0/FLT0
RB1/AN10/INT1
RB2/AN8/INT2
RB3/AN9/CCP2
RB4/AN11/KBI0
RB5/KBI1/PGM
RB6//KBI2/PGC
RB7/KBI3/PGD
RC0/T1OSO/T13CKI
RC1/T1OSI/CCP2
RC2/CCP1
RC3/SCK/SCL
RC4/SDI/SDA
RC5/SDO
RC6/TX/CK
RC7/RX/DT
RE3/MCLR/VPP
PIC18F2420
D1
D2
D3
D4
D5
D6
D7
D8
LED-RED
R1
R10
220
R6
10k
R6(2)
R36
R32
R35
R33
R34
R27
R19
R28
U2
R24
R24(2)

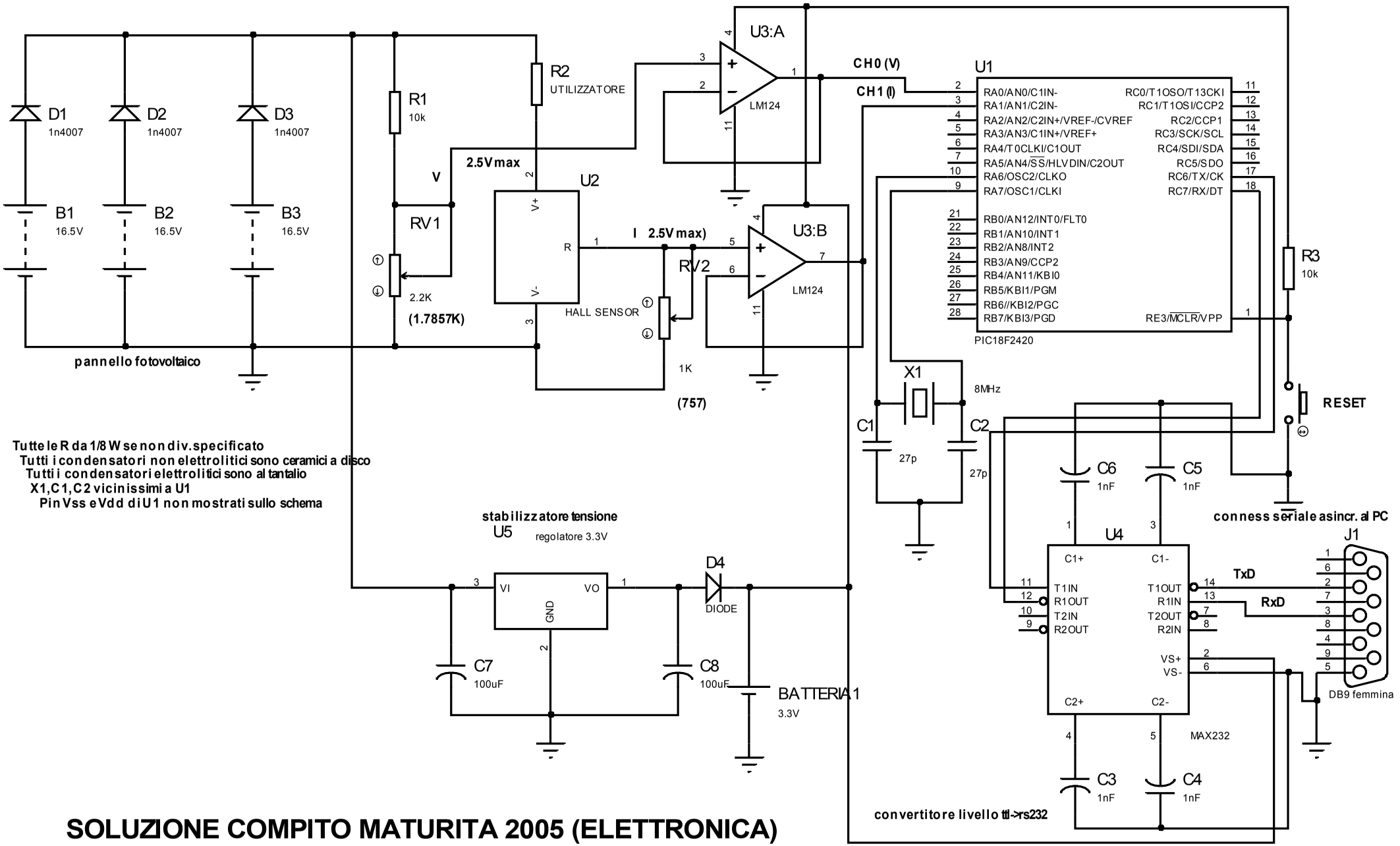
Bal2Bal3

THE SIMPLEST HH:MM:SS MICROCONTROLLER DRIVEN CLOCK
(6 digits 7-segment multiplexed display)



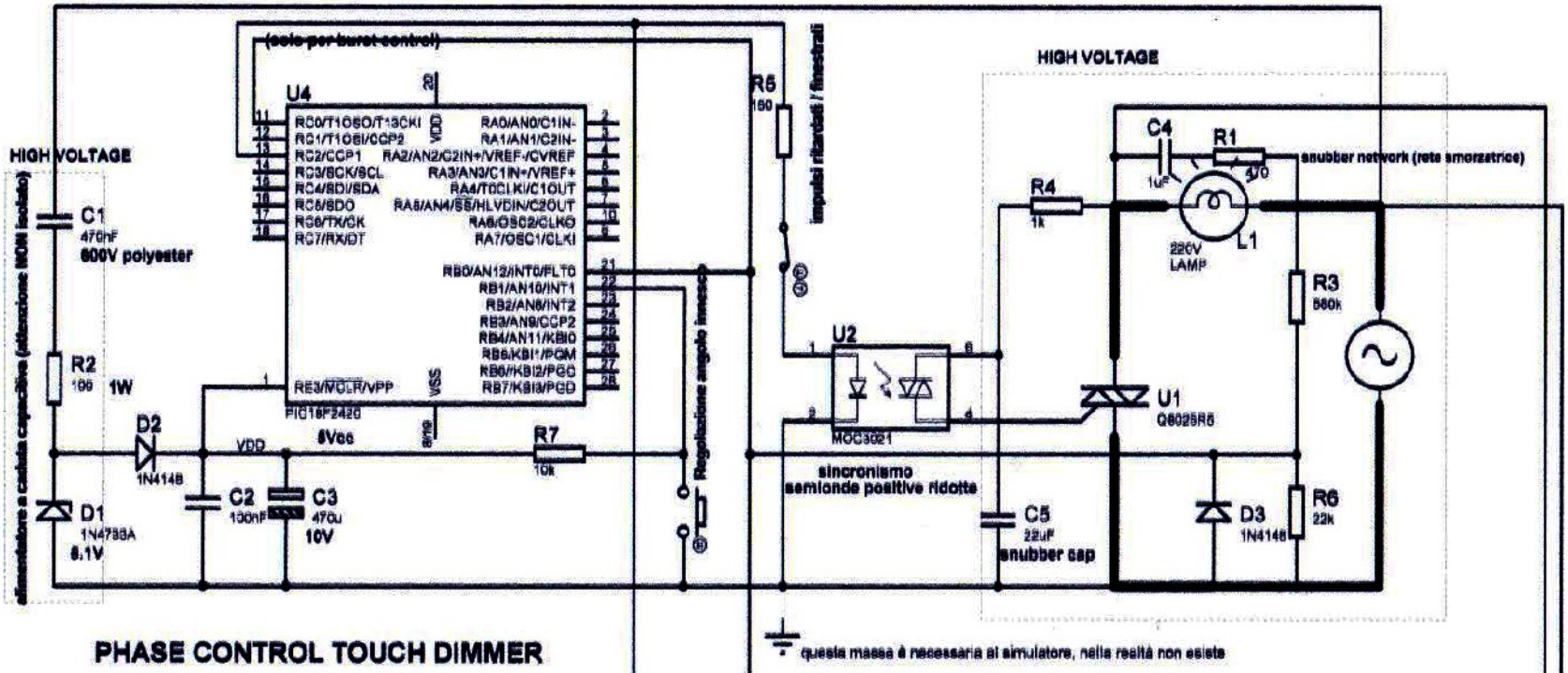
RIVELATORE UMIDITA





SOLUZIONE COMPITO MATURITA 2005 (ELETTRONICA)

MONITORAGGIO SU GIORNATA V e I PANNELLO FOTOVOLTAICO Vmax=16.5V Imax= 3.3A



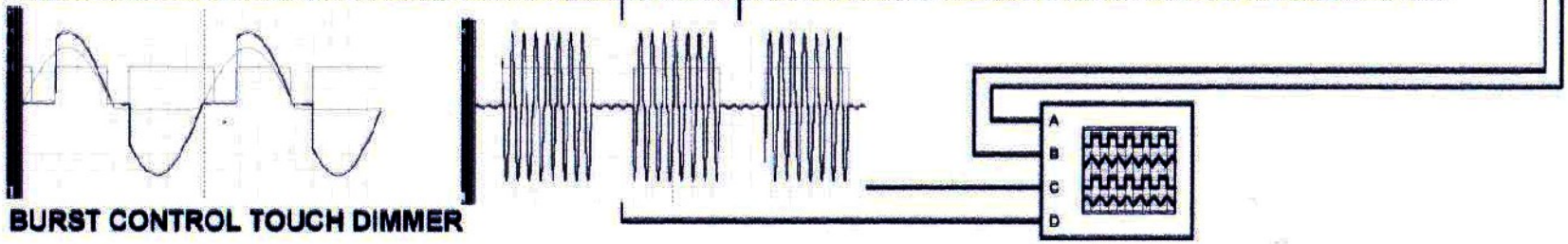
PHASE CONTROL TOUCH DIMMER

Il microcontrollore U3 fornisce (tramite il fotoaccoppiatore U2) degli impulsi di innesco al triac.

Gli impulsi sono ritardati di un angolo di innesco alpha che aumenta in 10 step di 18° premendo il pulsante (modulo 10, premendo ancora riparte da 0).

Premendo e lasciando il pulsante due volte in rapida successione alpha diventa 0 se è 180° e 180° se è zero.

Gli impulsi di innesco sono ottenuti rilevando con il comparatore interno gli attraversamenti dello zero di una versione ridotta della sinusoide .



BURST CONTROL TOUCH DIMMER

Premendo il pulsante decido in 10 steps successivi il numero di cicli interi che invio al carico mediante il triac ogni 200 ms (a 50Hz)

Va bene se l'inerzia del carico è > 200 ms; diversamente devo aumentare la frequenza (e ricalcolare C1 e R2 per fornire la giusta attenuazione)