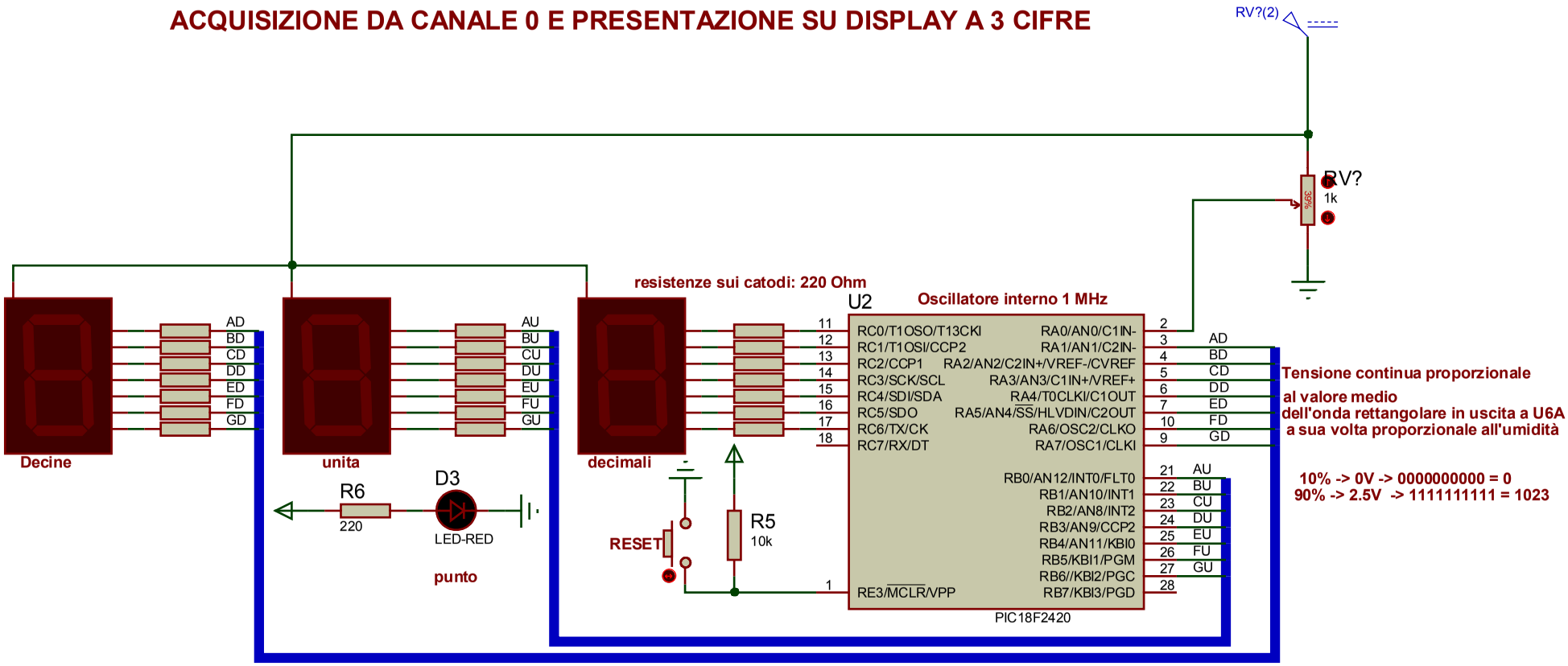
// AnalogToLedDisplay
// ACQUISIZIONE DA CANALE 0 e PRESENTAZIONE SU DISPLAY A 3 CIFRE (non multiplexato)

```
1   #pragma config OSC = INTIO67 // oscillatore al quarzo (esterno)
2   #pragma config WDT = OFF
3   #pragma config LVP = OFF
4
5   #include <p18cxxx.h>                                          disp. Segmenti          aa
6                                                                                       b       f
7   // segmento display   g f  e d c b a                                                  gg
8   // bit porta              6 5 4 3 2 1 0                                             c       e
9   // quindi ad esempio per accendere 8 serve 0111 1111 = 0x7F,                          dd
10  // per accendere 0 serve 0011 1111 = 0x3f e cosi via....   (att. In questa applicazione
11
12  // TABELLA CONVERSIONE DA Binary Coded Decimal a 7 segmenti
13  const unsigned char BCD27SEG[10] = {0x3f, 0x30, 0x6d, 0x79, 0x72, 0x5b, 0x5f, 0x31, 0x7f, 0x7b };
14                                                  // 0     1     2         3       4       5       6       7       8       9
15  void main()
16  {
17   unsigned long int val; // variabile per assemblare il valore acquisito
18   unsigned int lb, hb;     // parte bassa e alta del risultato (8 + 8 bit).
19                            // Servono 16 bit perchè devo traslare di 8 a sx.
20
21   TRISA = 0x01; TRISB = 0x00;  TRISC = 0x00; // porta A in uscita tranne RA0, porta B e C in uscita
22
23   ADCON1 = 0x0e;  // + e - vrif da alimentazione, ch0 portA bit0, ingresso analogico
24
25   while (1)
26   {
27            ADCON0 = 0x03; // SOC al ch0 (0x02 = GO, + 0x01 = abilito canale 0)
28
29           while(ADCON0 & 0x02); // attesa EOC
30
31            lb = ADRESL; hb = ADRESH; // leggo parte bassa e parte alta del risultato
32            val = ((hb << 8) | hb) >> 6; // assemblo in un intero a 10 bit giustificato a destra
33
34            // converto il campo 0<-->1023 in 100<-->900 (10.0 <--> 90.0) (val*800)/1024+100
35            // 800 = 512 + 256 + 32 ovvero 2^9 + 2^8 + 2^5; 1024 = 2^10 --> faccio tutto con << e >>  !!!
36           // però siccome 800 * 1023 NON STA in un intero a 16 bit uso un long int 32 bit.
37            // (il risultato finale sta pero in 10 bit)
38
39            val = (((val << 9) + (val << 8) + (val << 5)) >> 10) + 100;  // FAST conversion !!!
40
41            // conversione da intero a 3 cifre BCD che vengono presentate sulle 3 porte
42            LATC = ~BCD27SEG[val%10]; val = val / 10; // presenta decimali  (~ x anodo a Vcc)
43            LATB = ~BCD27SEG[val%10]; val = val / 10; // presenta unità
44            LATA = (~BCD27SEG[val%10]) << 1;            // presenta decine
45                                                        //(sui 7 bit più significativi (ecco xkè << 1 !!!))
46    }
47  }
48
```

ACQUISIZIONE DA CANALE 0 E PRESENTAZIONE SU DISPLAY A 3 CIFRE

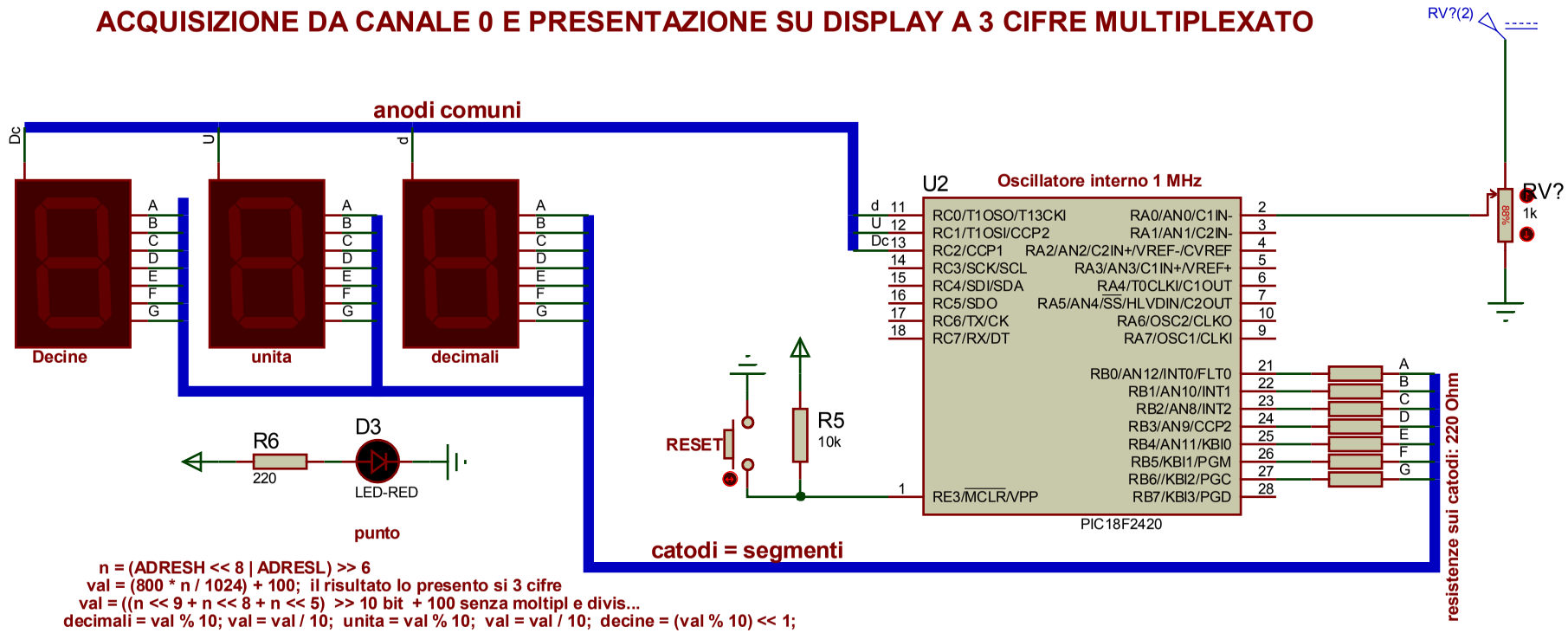



```
n = (ADRESH << 8 | ADRESL) >> 6
val = (800 * n / 1024) + 100;  il risultato lo presento si 3 cifre
val = ((n << 9 + n << 8 + n << 5)  >> 10 bit  + 100 senza moltipl e divis...
decimali = val % 10; val = val / 10;  unita = val % 10;  val = val / 10;  decine = (val % 10) << 1;
```

// ACQUISIZIONE DA CANALE 0 e PRESENTAZIONE SU DISPLAY A 3 CIFRE (multiplexato)

```
1  #pragma config OSC = INTIO67 // oscillatore al quarzo (esterno)
2  #pragma config WDT = OFF
3  #pragma config LVP = OFF
4
5  #include <p18cxxx.h>
6
7  // segmento display   g f e d c b a
8  // bit porta B            6 5 4 3 2 1 0
9  // quindi ad es. per accendere 8 serve 0111 1111 = 0x7F, per accendere 0  0011 1111 = 0x3f e cosi
   via....
10
11 // TABELLA CONVERSIONE DA Binary Coded Decimal a 7 segmenti
12 const unsigned char BCD27SEG[10] = {0x3f, 0x30, 0x6d, 0x79, 0x72, 0x5b, 0x5f, 0x31, 0x7f, 0x7b };
13                                                  // 0    1       2       3       4       5       6    7       8       9
14 void main()
15 {
16  unsigned long int val; // variabile per assemblare il valore acquisito
17  unsigned int lb, hb; // parte bassa e alta del risultato (8 + 8 bit). Servono 16 bit perchè traslo di 8 a sx.
18  int i;
19
20  TRISA = 0xff;  TRISB = 0x00; TRISC = 0xf8; // porta A ingresso , porta B uscita porta RC0-1-2 in uscita
21
22  ADCON1 = 0x0e;  // + e - vrif da alimentazione, ch0 portA bit0, ingresso analogico
23
24  LATC = 0x00;
25
26  while (1)
27  {
28          ADCON0 = 0x03; // SOC al ch0 (0x02 = GO, + 0x01 = abilito canale 0)
29          while(ADCON0 & 0x02); // attesa EOC
30
31          lb = ADRESL; hb = ADRESH; // leggo parte bassa e parte alta del risultato
32          val = ((hb << 8) | hb) >> 6; // assemblo in un intero a 10 bit giustificato a destra
33
34          // converto il campo 0<-->1023 in 100<-->900 (10.0 <--> 90.0) (val*800)/1024+100
35          // 800 = 512 + 256 + 32 ovvero 2^9 + 2^8 + 2^5; 1024 = 2^10 --> faccio tutto con << e >> |||
36          // però siccome 800 * 1023 NON STA in un intero a 16 bit uso un long int 32 bit.
37          // (il risultato finale sta pero in 10 bit)
38
39          val = (((val << 9) + (val << 8) + (val << 5)) >> 10) + 100;  // FAST conversion !!!
40
41          // conversione da intero a 3 cifre BCD che sono presentate sui 3 display a rotazione velocissima
42
43          LATC = 0x01; // attivo display decimali
44          LATB = ~BCD27SEG[val%10]; val = val / 10; // presenta decimali  (~ x anodo a Vcc)
45
46          LATC = 0x02; // attivo display unità
47          LATB = ~BCD27SEG[val%10]; val = val / 10; // presenta unità
48
49          LATC = 0x04; // attivo display decine
50          LATB = ~BCD27SEG[val%10]; // presenta decine
51   }
52 }
```

ACQUISIZIONE DA CANALE 0 E PRESENTAZIONE SU DISPLAY A 3 CIFRE MULTIPLEXATO

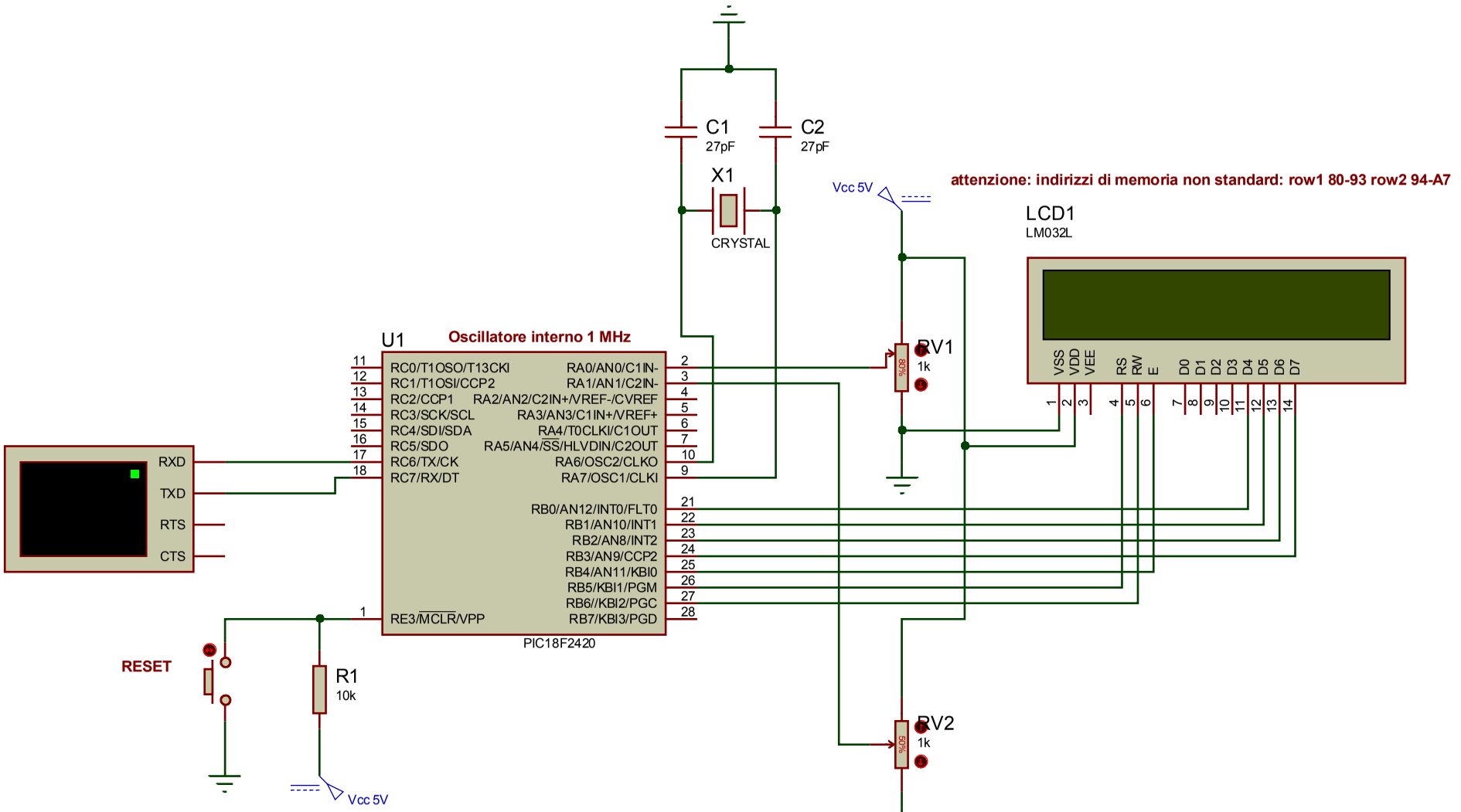



```
n = (ADRESH << 8 | ADRESL) >> 6
val = (800 * n / 1024) + 100;  il risultato lo presento si 3 cifre
val = ((n << 9 + n << 8 + n << 5)  >> 10 bit  + 100 senza moltipl e divis...
decimali = val % 10; val = val / 10;  unita = val % 10;  val = val / 10;  decine = (val % 10) << 1;
```

// AnalogToLCDDisplay.c

```
1   #include <p18Cxxx.h>
2   #include <xlcd.h>
3   #include <delays.h>
4   #include <stdio.h>
5   #include <usart.h>
6
7   #pragma config OSC = HS  // oscillatore al quarzo (esterno)
8
9   // no TX interrupt no RX interrupt UART mode 8 bit No parity 1 stop  non disabilita dopo 1 RX aud rate alto
10  #define PROTOCOL  USART_TX_INT_OFF & USART_RX_INT_OFF & USART_ASYNCH_MODE &
    USART_EIGHT_BIT & USART_CONT_RX & USART_BRGH_HIGH
11
12  void main( void )
13  {
14   char buff[41]; unsigned int val[2], ch, hb, lb;
15
16   OpenXLCD(FOUR_BIT & LINES_5X7); // configure external LCD
17
18   // configure USART ATTENZIONE: INIZIALIZZAZIONE UART:
19    OpenUSART(PROTOCOL, 100); /* baud rate = (Fclock / 1024) - 1: 100->9600 a 16MHz    */
20
21   puts("Ciao sono AnalogToLCDParallelDisplay!!\x0d\x0a");
22
23   ADCON1 = 0x0d; // + e - vrif da alimentazione, ch 0 ch1 porta A ingressi analogici
24
25   while(1)
26   {
27    for(ch = 0; ch < 2; ch++) // per ogni canale:
28    {
29     ADCON0 = 0x03 | (ch << 2);  // SOC | ADON | (ch << 2)
30
31     while(ADCON0 & 0x02); // attesa EOC
32     // leggo il risultato che è giustificato a sinistra
33     lb = ADRESL;  hb = ADRESH; // leggo byte basso e poi leggo byte alto (solo i due bit alti sono signif.)
34
35     val[ch] = ((hb << 8) | lb) >> 6; // assemblo in un intero a 10 bit giustif. a destra
36
37     ADCON0 = 0x07;  // SOC:   0x02 è la maschera per il bit GO in ADCON0...seleziono il canale 1, abilito l'AD
38    }
39    // preparo stringa in buff
40    sprintf(buff, "ch0: %04u  ch1: %04u", val[0], val[1]); //ch0: xxxx  ch1: yyyy     // 20 caratteri esatti!!
41
42    putsUSART(buff);  // stampo su seriale puts fa casino!!
43    puts("\x0d\x0a"); // stampo su LCD
44
45    putsXLCD(buff);
46
47    while(!DataRdyUSART()); ReadUSART(); // aspetto pressione carattere per nuova acquisizione
48   }
49  }
50
51
52  // CATTIVA PRATICA DI PROGRAMMAZIONE: routine necessarie contenute nel software utente invece che nella
    libreria!!
53  void DelayFor18TCY() { int i = 0; for(i = 0; i < 10; i++) Nop(); }
54  void DelayPORXLCD (){ Delay1KTCYx(60); }
55  void DelayXLCD () { Delay1KTCYx(20); }
```

ACQUISIZIONE DA CANALE 0 E CANALE 1 E PRESENTAZIONE SU LCD PARALLELO



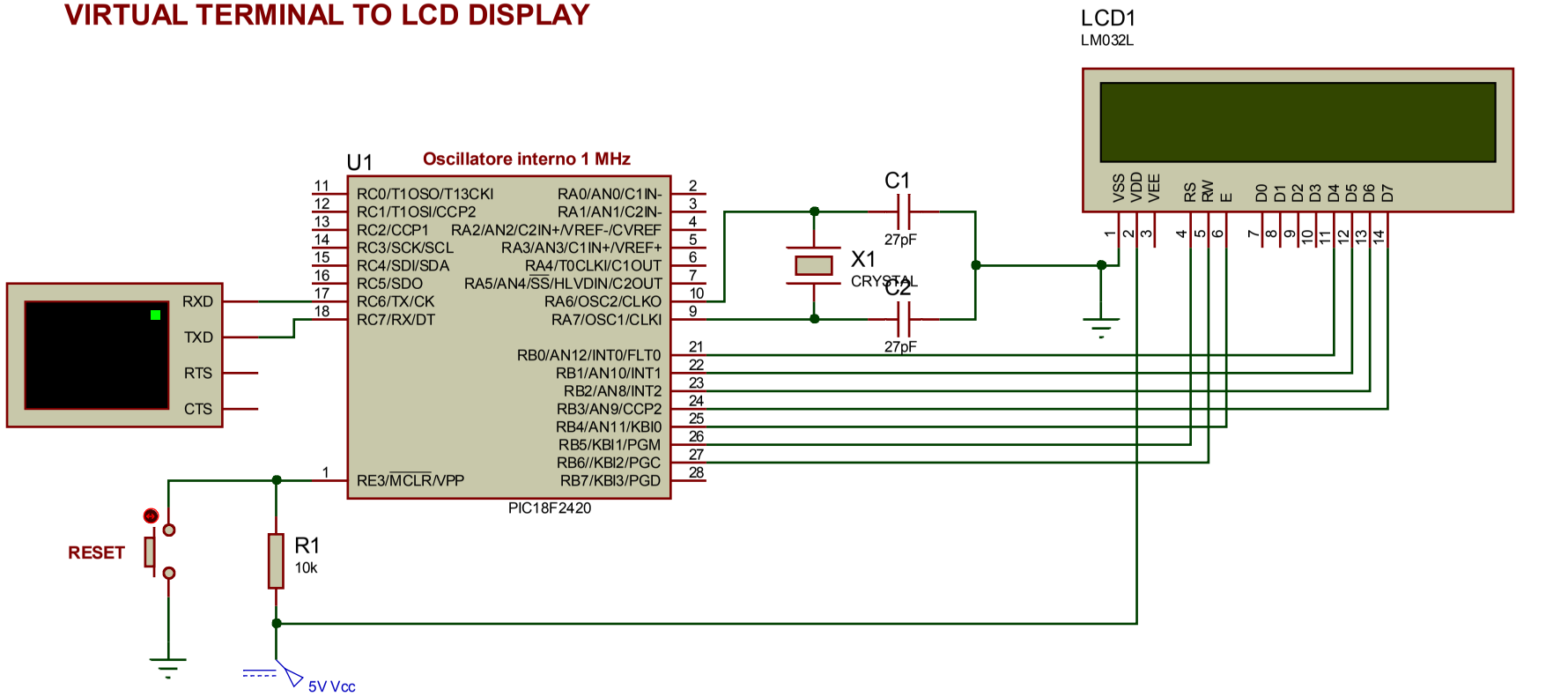

```
n = (ADRESH << 8 | ADRESL) >> 6
val = (800 * n / 1024) + 100;  il risultato lo presento si 3 cifre
val = ((n << 9 + n << 8 + n << 5)  >> 10 bit  + 100 senza moltipl e divis...
decimali = val % 10; val = val / 10;  unita = val % 10;  val = val / 10;  decine = (val % 10) << 1;
```

// VirtualTerminalToLCDParallelDisplay.c

```
1   #include <p18Cxxx.h>
2   #include <xlcd.h>
3   #include <delays.h>
4   #include <usart.h>
5
6   #pragma config OSC = HS //INTIO67 // oscillatore al quarzo (esterno)
7
8   // no TX interrupt no RX interrupt UART mode 8 bit No parity 1 stop  non disabilita dopo 1 RX aud rate alto
9   #define PROTOCOL  USART_TX_INT_OFF & USART_RX_INT_OFF & USART_ASYNCH_MODE &
    USART_EIGHT_BIT & USART_CONT_RX & USART_BRGH_HIGH
10
11  void main( void )
12  {
13   char data, s[] = " Ciao!!";
14
15   OpenXLCD(FOUR_BIT & LINES_5X7); // configure external LCD
16
17   // configure USART ATTENZIONE: INIZIALIZZAZIONE UART:
18   OpenUSART(PROTOCOL, 100);   // baud rate = (Fclock / 1024) - 1 : 100: 9600 a 16MHz
19
20   putrsXLCD("Ciao!!");  // ?? baco non scrive ?
21   putsXLCD(s); // baco si mangia il primo carattere ?
22
23   while(1)
24   {
25    while(!DataRdyUSART()); //wait for data
26    data = ReadUSART(); //read data
27    WriteUSART(data);
28    WriteDataXLCD(data); //write to LCD
29    if(data == 0x1b) break;
30   }
31
32   CloseUSART();
33  }
34
35
36
37  // CATTIVA PRATICA DI PROGRAMMAZIONE: routine necessarie ma contenute nel software utente invece che
    nella libreria!!
38  void DelayFor18TCY() { int i = 0; for(i = 0; i < 10; i++) Nop(); }
39  void DelayPORXLCD (){ Delay1KTCYx(60); }
40  void DelayXLCD () { Delay1KTCYx(20); }
```

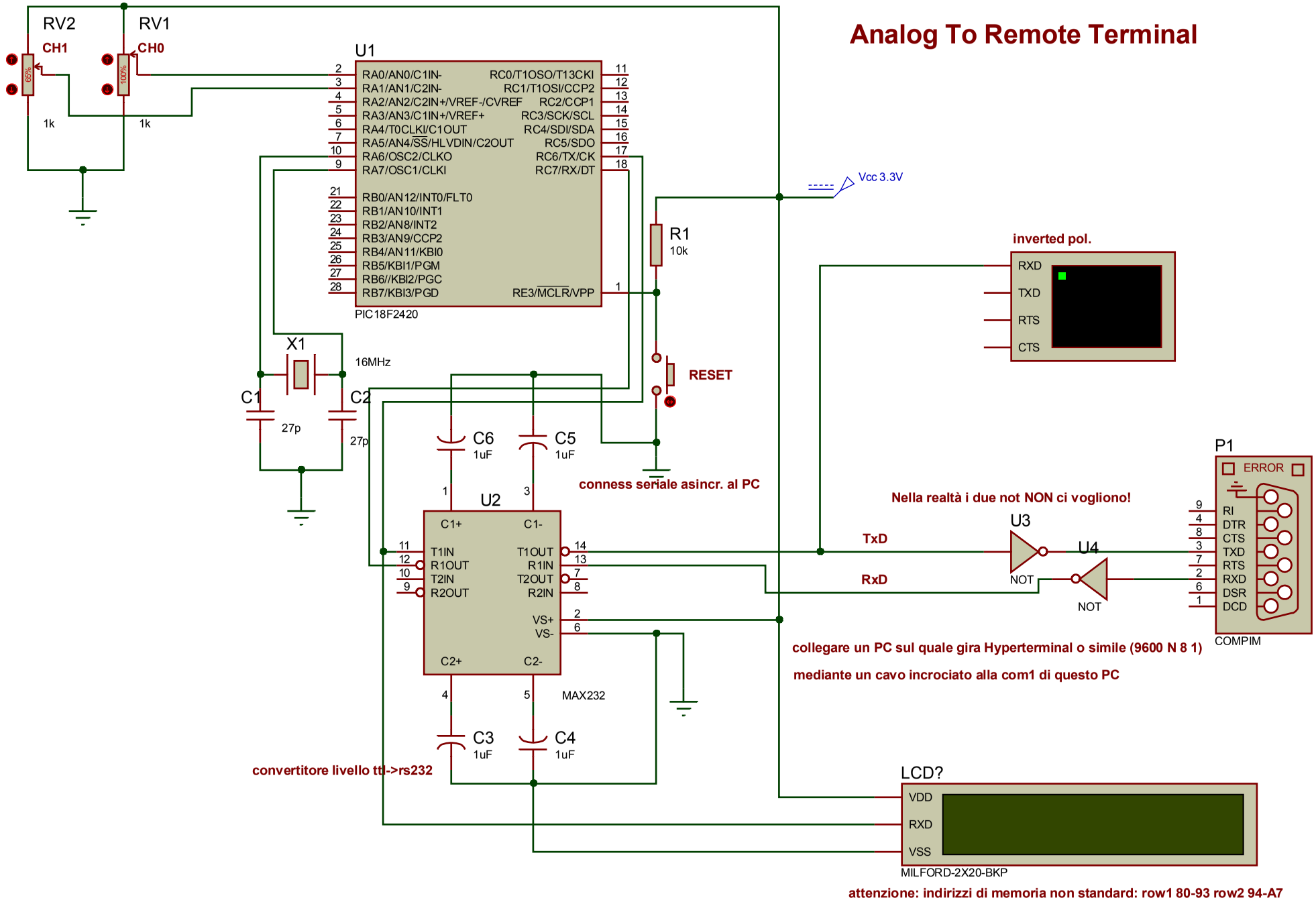
attenzione: indirizzi di memoria non standard: row1 80-93 row2 94-A7

VIRTUAL TERMINAL TO LCD DISPLAY



// AnalogToRemoteTerminal.c
// ACQUISIZIONE AD DA DUE CANALI E TRASMISSIONE SU SERIALE

```
1   #include <p18cxxx.h>   /* for TRISA and PORTA declarations */
2   #include <stdio.h>
3   #include <usart.h>
4
5   #pragma config OSC = HS
6
7   // no TX interrupt no RX interrupt UART mode 8 bit No parity 1 stop  non disabilita dopo 1 RX aud rate alto
8   #define PROTOCOL  USART_TX_INT_OFF & USART_RX_INT_OFF & USART_ASYNCH_MODE &
    USART_EIGHT_BIT & USART_CONT_RX & USART_BRGH_HIGH
9
10  void main()
11  {
12   unsigned int val[2], ch, hb, lb;
13   unsigned char buff[41];
14
15   // configure USART ATTENZIONE: INIZIALIZZAZIONE UART:
16   OpenUSART(PROTOCOL, 100); /* baud rate = (Fclock / 1024) - 1: 100->9600 a 16MHz    */
17
18   puts("Ciao sono AnalogToLCDParallelDisplay!!\x0d\x0a");
19
20   ADCON1 = 0x0e; // + e - vrif da alimentazione, ch 0 ch1 porta A. ingressi analogici
21
22   while(1)
23   {
24    for(ch = 0; ch < 2; ch++) // per ogni canale:
25    {
26     ADCON0 = 0x03 | (ch << 2);  // SOC | ADON | (ch << 2)
27
28     while(ADCON0 & 0x02); // attesa EOC
29     // leggo il risultato che è giustificato a sinistra
30     lb = ADRESL;  hb = ADRESH; // leggo byte basso e poi leggo byte alto (solo i due bit alti sono signif.)
31
32     val[ch] = ((hb << 8) | lb) >> 6; // assemblo in un intero a 10 bit giustif. a destra
33
34     ADCON0 = 0x07;  // SOC:   0x02 è la maschera per il bit GO in ADCON0...seleziono il canale 1, abilito l'AD
35    }
36    // preparo stringa in buff
37    sprintf(buff, "ch0: %04u  ch1: %04u", val[0], val[1]); //ch0: xxxx  ch1: yyyy
38                                                          // 12345678912345678912 20 caratteri esatti!!
39    putsUSART(buff);  // stampo su seriale puts fa casino!!
40    puts("\x0d\x0a"); // a capo
41
42    while(!DataRdyUSART()); ReadUSART(); // aspetto pressione carattere per nuova acquisizione
43   }
44  }
```


Analog To Remote Terminal
RV2
CH1
RV1
CH0
65%
100%
1k
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
Vcc 3.3V
R1
10k
RESET
X1
16MHz
C1
27p
C2
27p
C6
1uF
C5
1uF
conness seriale asincr. al PC
U2
C1+
C1-
T1IN
R1OUT
T2IN
R2OUT
T1OUT
R1IN
T2OUT
R2IN
VS+
VS-
C2+
C2-
MAX232
C3
1uF
C4
1uF
convertitore livello ttl->rs232
inverted pol.
RXD
TXD
RTS
CTS
Nella realtà i due not NON ci vogliono!
TxD
RxD
U3
NOT
U4
NOT
P1
ERROR
RI
DTR
CTS
TXD
RTS
RXD
DSR
DCD
COMPIM
collegare un PC sul quale gira Hyperterminal o simile (9600 N 8 1)
mediante un cavo incrociato alla com1 di questo PC
LCD?
VDD
RXD
VSS
MILFORD-2X20-BKP
attenzione: indirizzi di memoria non standard: row1 80-93 row2 94-A7

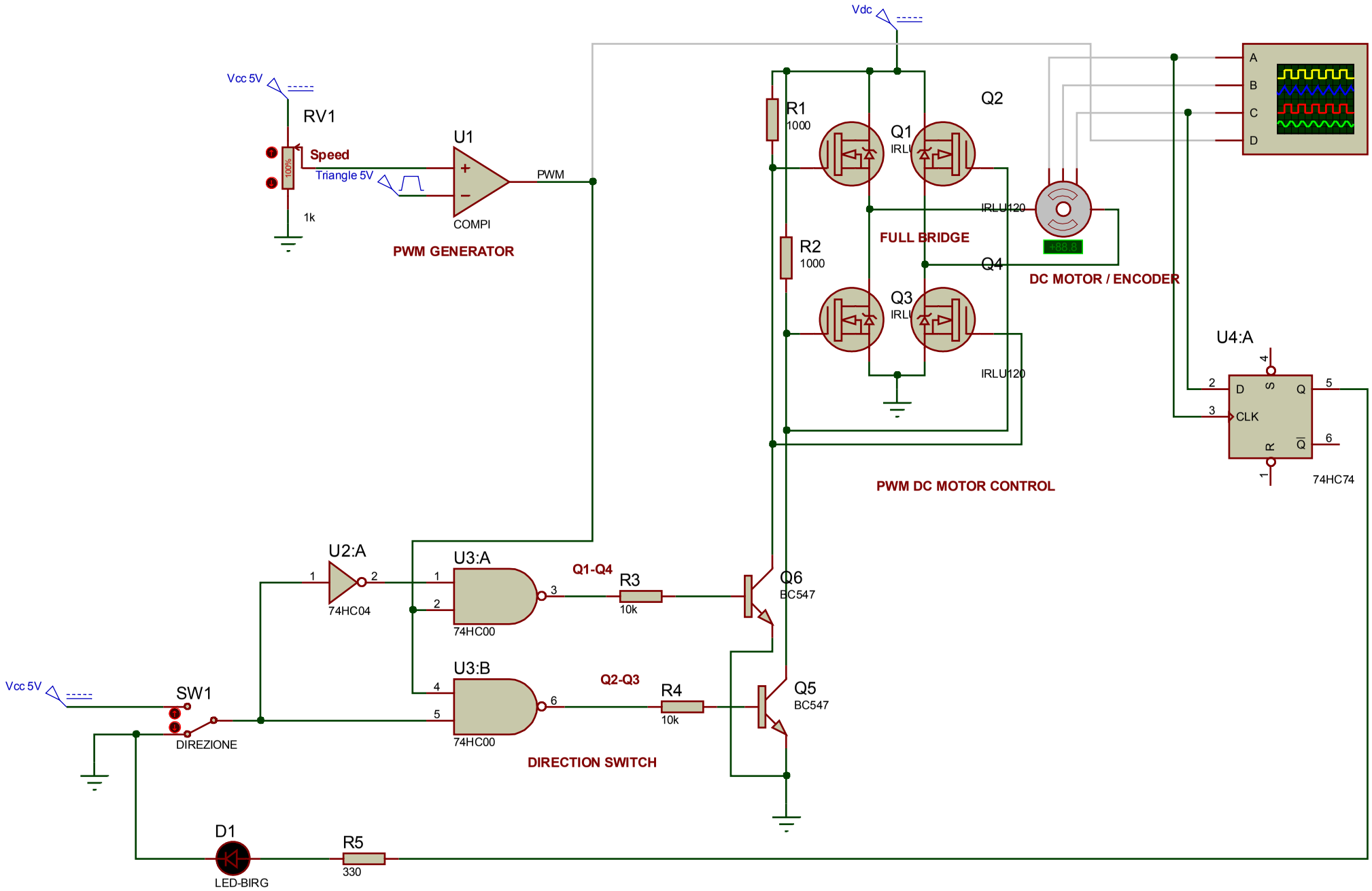
// ACQUISIZIONE AD DA DUE CANALI AD INTERRUZIONE E TRASMISSIONE SU SERIALE AD2.c

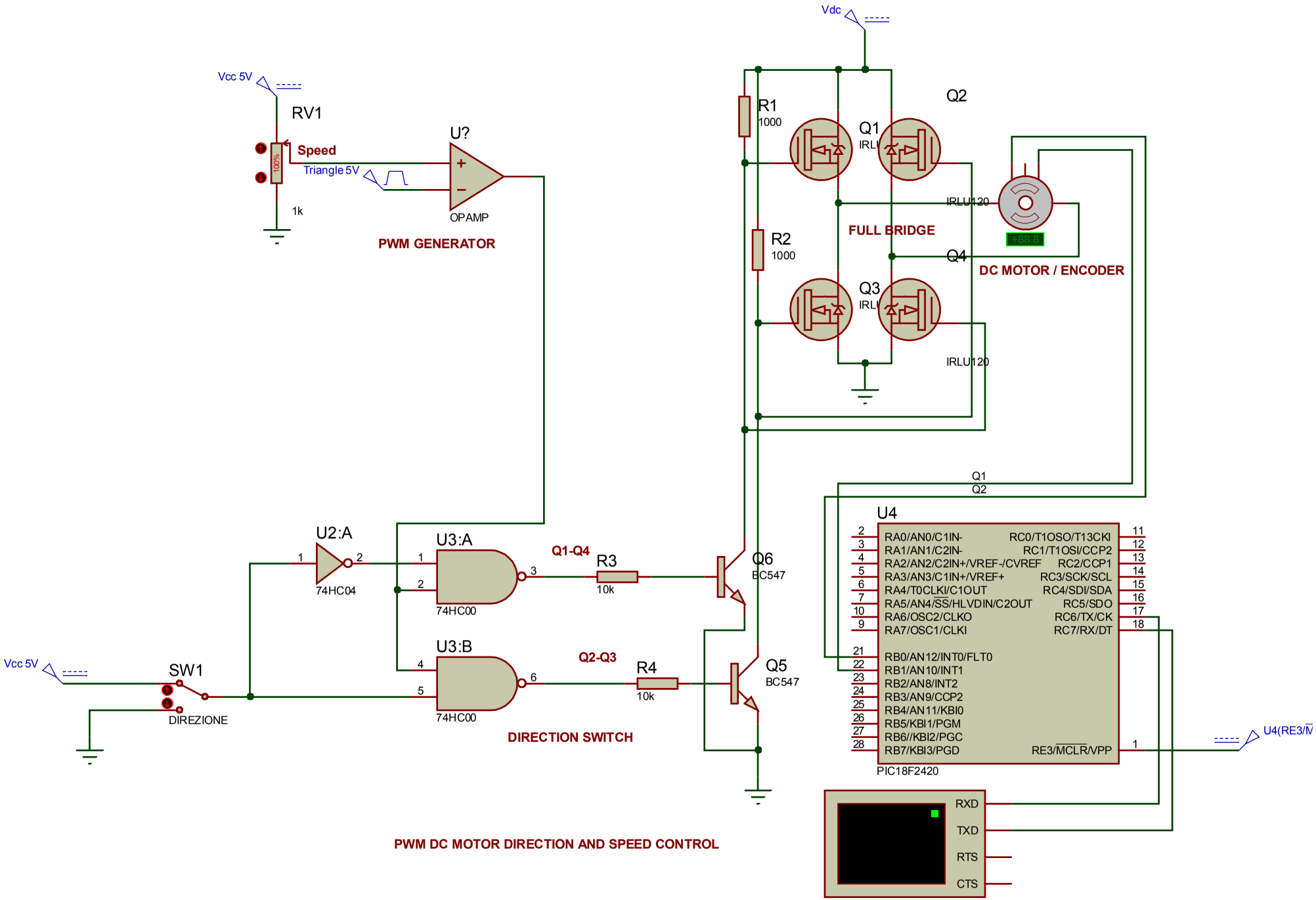
```
1   #include <p18cxxx.h>   /* for TRISA and PORTA declarations */
2   #include <stdio.h>
3   #include <usart.h>
4
5   #pragma config OSC = HS
6
7   // no TX interrupt no RX interrupt UART mode 8 bit No parity 1 stop  non disabilita dopo 1 RX aud rate alto
8   #define PROTOCOL  USART_TX_INT_OFF & USART_RX_INT_OFF & USART_ASYNCH_MODE & USART_EIGHT_BIT &
    USART_CONT_RX & USART_BRGH_HIGH
9
10  unsigned int ch = 0, val[2];
11
12  void main()
13  {
14   unsigned char buff[41];
15
16   INTCONbits.GIE = 1;    // abilito globalmente le interruzioni
17   INTCONbits.PEIE = 1;   // abilito le interruzioni da periferica
18   PIE1bits.ADIE = 1;     // abilito le interruzioni da AD
19
20   // configure USART ATTENZIONE: INIZIALIZZAZIONE UART:
21   OpenUSART(PROTOCOL, 100); /* baud rate = (Fclock / 1024) - 1: 100->9600 a 16MHz    */
22
23   puts("Ciao sono AnalogToLCDParallelDisplay!!\x0d\x0a");
24
25   ADCON1 = 0x0d; // + e - vrif da alimentazione, ch 0, ch1 porta A.0 ingressi analogici
26
27   while(1)
28   {
29    ADCON0 = 0x03;  // avvio l'acquisizione per il primo canale...
30
31    if(ch == 2)
32    {
33     sprintf(buff, "ch0: %04u  ch1: %04u\x0d\x0a", val[0], val[1]); //ch0: xxxx  ch1: yyyy     // preparo stringa in buff
34                                                                    // 12345678912345678912 20 caratteri esatti!!
35     putsUSART(buff);  // stampo su seriale puts fa casino!!
36
37     ch = 0;
38    }
39
40  // QUI POSSO FARE ALTRE COSE, ANCHE STARE IN LOOP, LE ACQUISIZIONI PROCEDONO...
41
42    while(!DataRdyUSART()); ReadUSART(); // aspetto pressione carattere per nuova acquisizione
43   }
44  }
45
46  // Impostazione interrupt system...DEVE ESSERE TUTTO DI SEGUITO ESATTAMENTE COSI.
47
48  #pragma interrupt GestioneInterruzione // questa riga segnala che GestioneInterruzione è una routine di
49                                         // Interrupt e salva nello stack indirizzo di ritorno e PSW
50
51  void GestioneInterruzione()          // questa è la routine di interrupt vera e propria...
52  {
53   unsigned int hb, lb;
54
55    if (PIR1bits.ADIF)       // controllo se l'interruzione viene veramente dall'AD
56    {
57     // leggo il risultato che è giustificato a sinistra
58     lb = ADRESL;  hb = ADRESH; // leggo byte basso e poi leggo byte alto (solo i due bit alti sono signif.)
59
60     val[ch] = ((hb << 8) | lb) >> 6; // assemblo in un intero a 10 bit giustif. a destra
61
62     if(ch < 2)  // se era l'ultimo canale non lo faccio...
63     {
64      ch++;                          // incremento il canale
65
66      ADCON0 = 0x03 | (ch << 2);       // faccio partire l'acquisizione sul canale successivo...
67     }
68     PIR1bits.ADIF = 0;  // resetto il flag di interrupt...
69    }
70  }
71
72  #pragma code GestioneInterruzioneJmp = 0x08   // questa riga serve per caricare il vettore di interruzione
73                                                // a 0x08 con l'indirizzo di partenza di GestioneInterruzioneJmp
74  void GestioneInterruzioneJmp() { GestioneInterruzione(); }
```

LO SCHEMA E' QUELLO PER AD1

// microcontroller DC motor speed and direction detection.c

```
1   #include <p18cxxx.h>   /* for TRISA and PORTA declarations */
2   #include <stdio.h>
3   #include <usart.h>
4
5   #pragma config OSC = INTIO67   // oscillatore interno a 1MHz
6
7   // no TX interrupt no RX interrupt UART mode 8 bit No parity 1 stop  non disabilita dopo 1 RX aud rate alto
8   #define PROTOCOL  USART_TX_INT_OFF & USART_RX_INT_OFF & USART_ASYNCH_MODE &
    USART_EIGHT_BIT & USART_CONT_RX & USART_BRGH_HIGH
9
10  #define PH 0xff // valori per contare 256 impulsi: il contatore è up modulo
11  #define PL 0x00 // 65536: ci metto 65289 (0xff00) così gli rimangono 256 impulsi per l'overflow
12
13  volatile unsigned int direction = 0, pulsesec = 0, timer = 0; // variabili globale
14
15  void main()
16  {
17   INTCON = 0x00;
18   ADCON1 = 15; // no AD sulle porte
19   CMCON  = 7; // no comparatore sulla porta
20   TRISB = 0xff; // porta B in ingresso
21
22   T0CON = 0x88;    //  set up timer0 16 bit - internal clock = Fosc/4 -NO prescaler 1:1 --> 250KHz --> 4us x count
23   TMR0H = PH;       TMR0L = PL;      // // setup period high byte and low bytes: 1 TOF = 1 ms
24
25   RCONbits.IPEN = 1;         /* enable interrupt priority levels */
26   INTCONbits.INT0IE = 1;     /* enable INT0 interrupt */
27   INTCONbits.TMR0IE = 1;
28   INTCONbits.GIEH = 1;       /* enable all high priority interrupts */
29
30   OpenUSART(PROTOCOL, 12); // ATTENZIONE: INIZIALIZZAZIONE UART:
31
32    while(1)
33    {
34              timer = 1000; while(timer); // aspetta 1 s
35              if(direction == 0) puts("direzione oraria "); else puts("direzione ANTIoraria ");
36              printf("impulsi/s: %u \x0d\x0a", pulsesec); pulsesec = 0;
37    }
38  }
39
40  // Impostazione interrupt system
41
42  // Routine di interruzione
43  #pragma interrupt GestoreInterruzione
44
45  void GestoreInterruzione ()               // questa è la routine di interrupt
46  {
47      if (INTCONbits.INT0F)              // controllo se l'interruzione viene veramente da INT0
48      {
49         INTCONbits.INT0F = 0; // devo segnalare che la interruzione è stata servita
50         if(PORTBbits.RB1) direction = 1; else direction = 0; // se Q1 arriva prima di Q2...orario
51         pulsesec++;
52      }
53
54     if (INTCONbits.TMR0IF)             // controllo se l'interruzione viene veramente da TMR0
55     {
56        if(timer) timer--;              // se si faccio il lavoro...(decremento timer, se non è 0)
57        INTCONbits.TMR0IF = 0;          // devo segnale che la interruzione è stata servita
58        TMR0H = PH; TMR0L = PL;         // ricarico il timer con il valore per 256 impulsi
59     }
60  }
61
62  #pragma code GestoreInterruzioneJmp = 0x08   // queste righe caricano  il vettore di interruzione a 0x08
63  void GestoreInterruzioneJmp() { GestoreInterruzione(); }
```

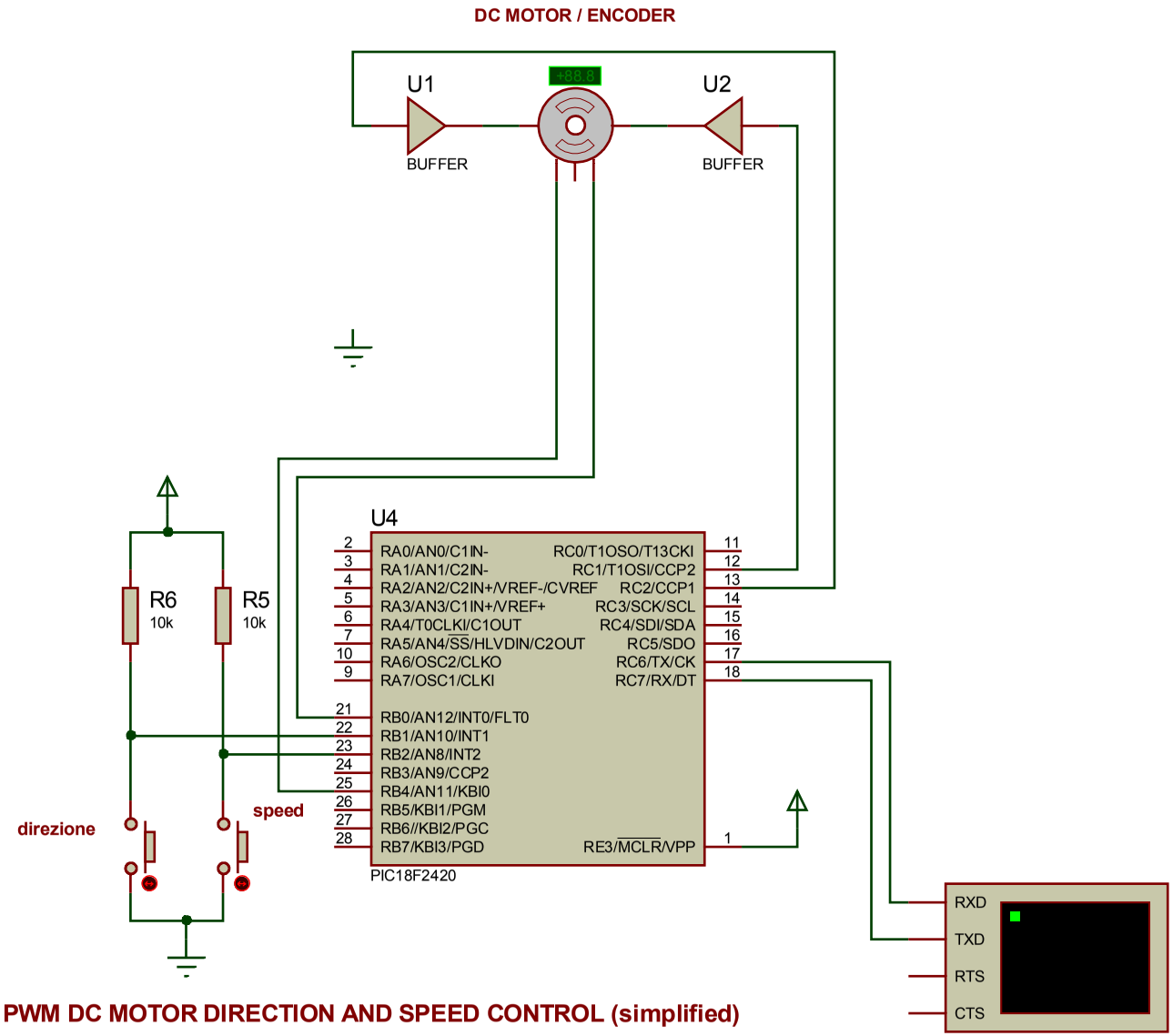

Vdc
Vcc 5V
RV1
Speed
100%
1k
U1
Triangle 5V
PWM
COMPI
PWM GENERATOR
R1
1000
Q1
IRLU
Q2
IRLU120
FULL BRIDGE
R2
1000
Q3
IRLU
Q4
IRLU120
DC MOTOR / ENCODER
A
B
C
D
U4:A
D
S
Q
CLK
R
Q
74HC74
PWM DC MOTOR CONTROL
U2:A
74HC04
U3:A
74HC00
Q1-Q4
R3
10k
Q6
BC547
U3:B
74HC00
Q2-Q3
R4
10k
Q5
BC547
Vcc 5V
SW1
DIREZIONE
DIRECTION SWITCH
D1
LED-BIRG
R5
330

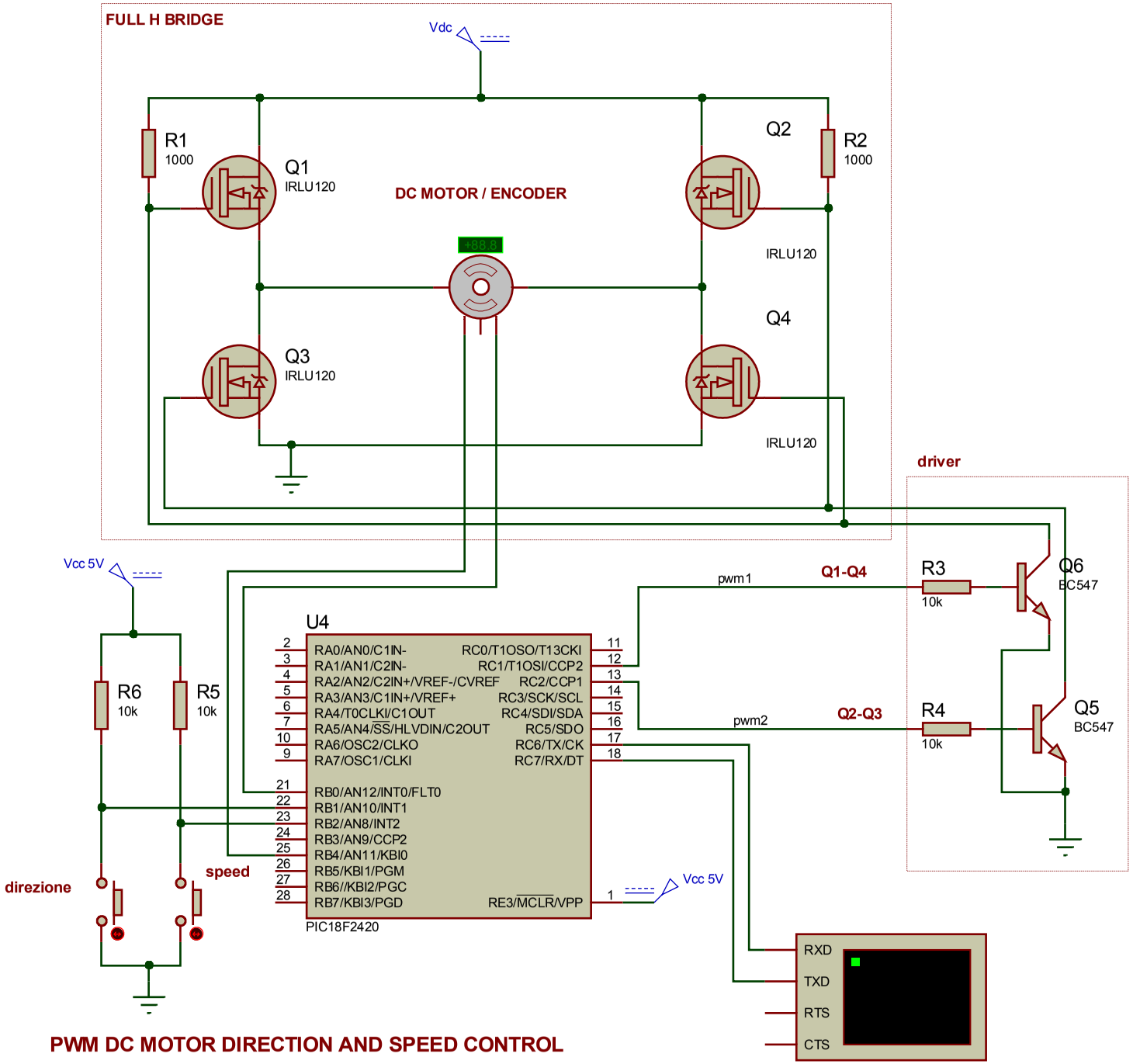

Vcc 5V
RV1
Speed
100%
1k
Triangle 5V
U?
OPAMP
PWM GENERATOR
Vdc
R1
1000
Q1
IRLU
Q2
IRLU120
FULL BRIDGE
R2
1000
Q4
Q3
IRLU
IRLU120
DC MOTOR / ENCODER
Q1
Q2
U4
RA0/AN0/C1IN-
RA1/AN1/C2IN-
RA2/AN2/C2IN+/VREF-/CVREF
RA3/AN3/C1IN+/VREF+
RA4/T0CLKI/C1OUT
RA5/AN4/SS/HLVDIN/C2OUT
RA6/OSC2/CLKO
RA7/OSC1/CLKI
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
U4(RE3/M
RXD
TXD
RTS
CTS
U2:A
74HC04
U3:A
74HC00
Q1-Q4
R3
10k
Q6
BC547
U3:B
74HC00
Q2-Q3
R4
10k
Q5
BC547
Vcc 5V
SW1
DIREZIONE
DIRECTION SWITCH
PWM DC MOTOR DIRECTION AND SPEED CONTROL

// microcontroller DC motor speed and direction control (open loop).c

```
1   #include <p18cxxx.h>   /* for TRISA and PORTA declarations */
2   #include <stdio.h>
3   #include <usart.h>
4
5   #pragma config OSC = INTIO67   // oscillatore interno a 1MHz
6
7   // no TX interrupt no RX interrupt UART mode 8 bit No parity 1 stop  non disabilita dopo 1 RX aud rate alto
8   #define PROTOCOL  USART_TX_INT_OFF & USART_RX_INT_OFF & USART_ASYNCH_MODE & USART_EIGHT_BIT &
    USART_CONT_RX & USART_BRGH_HIGH
9
10  #define PH 0xff // valori per contare 256 impulsi: il contatore è up modulo
11  #define PL 0x00 // 65536: ci metto 65289 (0xff00) così gli rimangono 256 impulsi per l'overflow
12
13  #define SPEED_INCREMENT 10
14  #define MAX_SPEED       110
15
16  volatile unsigned int direction = 0, pulsesec = 0, timer = 0, speed = 0, slope = 0; // variabili globali
17
18  void main()
19  {
20   INTCON = 0x00; ADCON1 = 15; /* no AD sulle porte */ CMCON  = 7; // no comparatore sulla porta
21   TRISA = 0xfe; TRISB = 0xff; // porta A bit 0 uscita (controllo) porta B in ingresso
22
23   T0CON = 0x88;    //  set up timer0 16 bit - internal clock = Fosc/4 -NO prescaler 1:1 --> 250KHz --> 4us x count
24   TMR0H = PH; TMR0L = PL;      // setup period high and low bytes 1 TOF = 1 ms
25
26   TRISCbits.TRISC2 = 0;   /* uscita su RC2 */ TRISCbits.TRISC1 = 0;   // uscita su RC1
27
28   LATCbits.LATC2 = 0; LATCbits.LATC1 = 0;
29
30   PR2 = MAX_SPEED;     // periodo
31
32   CCPR1L  = 0x00;  CCPR2L  = 0x00; // duty a 8 bit (256 livelli) a 0 inizialmente
33
34   T2CON = 0x04;    // timer2 prescaler = 1, abilitato
35
36   RCONbits.IPEN = 1;         /* enables interrupt priority levels */
37   INTCONbits.INT0IE = 1; INTCON3bits.INT1IE = 1;   INTCON3bits.INT2IE = 1; /* enables INT0, INT1, INT2 interrupt */
38   INTCONbits.TMR0IE = 1; // enables TMR0 interrupts
39   INTCONbits.GIEH = 1;       /* enable all high priority interrupts */
40
41   OpenUSART(PROTOCOL, 12); // ATTENZIONE: INIZIALIZZAZIONE UART:
42
43   PORTA = 0x01; CCP1CON = 0x00; CCP2CON = 0x0C; // CCP1 PWM mode on; CCP2 PWM mode off
44
45   while(1)
46   {
47              timer = 1000; while(timer);  // aspetta 1 s
48              if(direction == 0) puts("direzione oraria "); else puts("direzione ANTIoraria ");
49              printf("impulsi/s: %u speed %u \x0d\x0a", pulsesec, speed);    pulsesec = 0;
50   }
51  }
52
53
54  // Impostazione interrupt system
55
56  // Routine di interruzione
57  #pragma interrupt GestoreInterruzione
58
59  void GestoreInterruzione ()   // questa è la routine di interrupt
60  {
61     if (INTCONbits.INT0F)     // controllo se l'interruzione viene veramente da INT0
62     {
63        if(PORTBbits.RB4) direction = 1; else direction = 0; // se Q1 arriva prima di Q2...orario
64        pulsesec++;           // incremento il numero di impulsi al secondo
65        INTCONbits.INT0F = 0;  // devo segnalare che la interruzione è stata servita
66     }
67
68     if (INTCON3bits.INT2IF)     // controllo se l'interruzione viene veramente da INT2
69     {
70                        if(slope == 0) { speed += SPEED_INCREMENT; if(speed == MAX_SPEED) slope = 1; }
71                        if(slope == 1) { speed -= SPEED_INCREMENT; if(speed == 0) slope = 0; }
72                        CCPR1L = speed;             CCPR2L = speed;  // aggiorno il duty cycle...
73         INTCON3bits.INT2IF = 0; // devo segnalare che la interruzione è stata servita
74     }
```

```
75
76      if (INTCON3bits.INT1IF)     // controllo se l'interruzione viene veramente da INT1
77      {
78                              LATA ^= 0x01; CCP1CON ^= 0x0C; CCP2CON ^= 0x0C;
79          INTCON3bits.INT1IF = 0; // devo segnalare che la interruzione è stata servita
80      }
81
82     if (INTCONbits.TMR0IF)              // controllo se l'interruzione viene veramente da TMR0
83     {
84        if(timer) timer--;              // se si faccio il lavoro...(decremento timer, se non è 0)
85        TMR0H = PH; TMR0L = PL;         // ricarico il timer con il valore per 256 impulsi
86        INTCONbits.TMR0IF = 0;          // devo segnale che la interruzione è stata servita
87     }
88  }
89
90  #pragma code GestoreInterruzioneJmp = 0x08   // queste righe servono per caricare  il vettore di interruzione a 0x08
91
92  void GestoreInterruzioneJmp() { GestoreInterruzione(); }
```


DC MOTOR / ENCODER
U1
U2
BUFFER
BUFFER
U4
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
R6
10k
R5
10k
direzione
speed
RXD
TXD
RTS
CTS
PWM DC MOTOR DIRECTION AND SPEED CONTROL (simplified)


FULL H BRIDGE
Vdc
R1
1000
Q1
IRLU120
DC MOTOR / ENCODER
Q2
R2
1000
IRLU120
Q3
IRLU120
Q4
IRLU120
driver
Vcc 5V
R6
10k
R5
10k
U4
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
RC0/T1OSO/T13CKI
RC1/T1OSI/CCP2
RC2/CCP1
RC3/SCK/SCL
RC4/SDI/SDA
RC5/SDO
RC6/TX/CK
RC7/RX/DT
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
pwm1
Q1-Q4
R3
10k
Q6
BC547
pwm2
Q2-Q3
R4
10k
Q5
BC547
direzione
speed
Vcc 5V
RXD
TXD
RTS
CTS
PWM DC MOTOR DIRECTION AND SPEED CONTROL

// microcontroller DC motor speed and direction control (closed loop PID).c

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

#pragma config OSC = INTIO67   // oscillatore interno a 1MHz

// no TX interrupt no RX interrupt UART mode 8 bit No parity 1 stop  non disabilita dopo 1 RX aud rate alto
#define PROTOCOL  USART_TX_INT_OFF & USART_RX_INT_OFF & USART_ASYNCH_MODE & USART_EIGHT_BIT & USART_CONT_RX & USART_BRGH_HIGH
#define PH 0xff // valori per contare 256 impulsi: il contatore è up modulo
#define PL 0x00 // 65536: ci metto 65289 (0xff00) così gli rimangono 256 impulsi per l'overflow

#define Kp 1                  // guadagno stadio proporzionale, essenziale
#define Ki 0.05               // guadagno stadio integrativo    (precisione, storia dell'errore, più l'errore dura più forzo)
#define Kd 0.001              // guadagno stadio derivativo     (prontezza, più l'errore è rapido, più forzo)

#define SPEED_INCREMENT 10
#define MAX_SPEED       256

volatile unsigned int direction = 0, pulsesec = 0, timer = 0, speed = 0, slope = 0; // variabili globali

void main()
{
 float errore, errore_integrato = 0, errore_derivato, vecchio_errore = 0;

 int duty;

 INTCON = 0x00; ADCON1 = 15; /* no AD sulle porte */ CMCON  = 7; // no comparatore sulla porta
 TRISA = 0xfe; TRISB = 0xff; // porta A bit 0 uscita (controllo) porta B in ingresso

 T0CON = 0x88;    //  set up timer0 16 bit - internal clock = Fosc/4 -NO prescaler 1:1 --> 250KHz --> 4us x count
 TMR0H = PH; TMR0L = PL;      // setup period high and low bytes 1 TOF = 1 ms

 TRISCbits.TRISC2 = 0;   /* uscita su RC2 */ TRISCbits.TRISC1 = 0;   // uscita su RC1

 LATCbits.LATC2 = 0; LATCbits.LATC1 = 0;

 PR2 = MAX_SPEED-1;     // periodo

 CCPR1L  = 0x00;  CCPR2L  = 0x00; // duty a 8 bit (256 livelli) a 0 inizialmente

 T2CON = 0x04;    // timer2 prescaler = 1, abilitato

 RCONbits.IPEN = 1;         /* enables interrupt priority levels */
 INTCONbits.INT0IE = 1; INTCON3bits.INT1IE = 1;   INTCON3bits.INT2IE = 1; /* enables INT0, INT1, INT2 interrupt */
 INTCONbits.TMR0IE = 1; // enables TMR0 interrupts
 INTCONbits.GIEH = 1;       /* enable all high priority interrupts */

 OpenUSART(PROTOCOL, 12); // ATTENZIONE: INIZIALIZZAZIONE UART:

 PORTA = 0x01; CCP1CON = 0x00; CCP2CON = 0x0C; // CCP1 PWM mode on; CCP2 PWM mode off

 while(1)
 {
          timer = 1000; while(timer);  // aspetta 1 s

          errore = (float)speed - (float)pulsesec / 6; // encoder 256 impulsi/giro, motore max 6 giri/s -> 1536 impulsi al giro
                                                       // errore % * 256 ovvero in termine di impulsi/giro (e anche di duty per 100%)
          errore_integrato += errore;                  // errore integrale approssimato con la somma (diviso per T che è 1 s)
          errore_derivato = errore - vecchio_errore;      // rapporto inrementale (il dt è 1 s)
          vecchio_errore = errore;                        // aggiorno vecchio errore per il prox giro...

          duty = Kp * errore + Ki * errore_integrato + Kd * errore_derivato; // pesante calcolo in float, possibilmente evitare...

          if(duty < 0) duty = 0;                       // protezione overflow dell'errore
          if(duty >= MAX_SPEED) duty = 0;

          CCPR1L = duty;  CCPR2L = duty;  // aggiorno il duty cycle...

          if(direction == 0) puts("direzione oraria "); else puts("direzione ANTIoraria ");
          printf("impulsi/s: %u speed %u duty %u\x0d\x0a", pulsesec, speed, duty);          pulsesec = 0;
 }
}

// Impostazione interrupt system
```

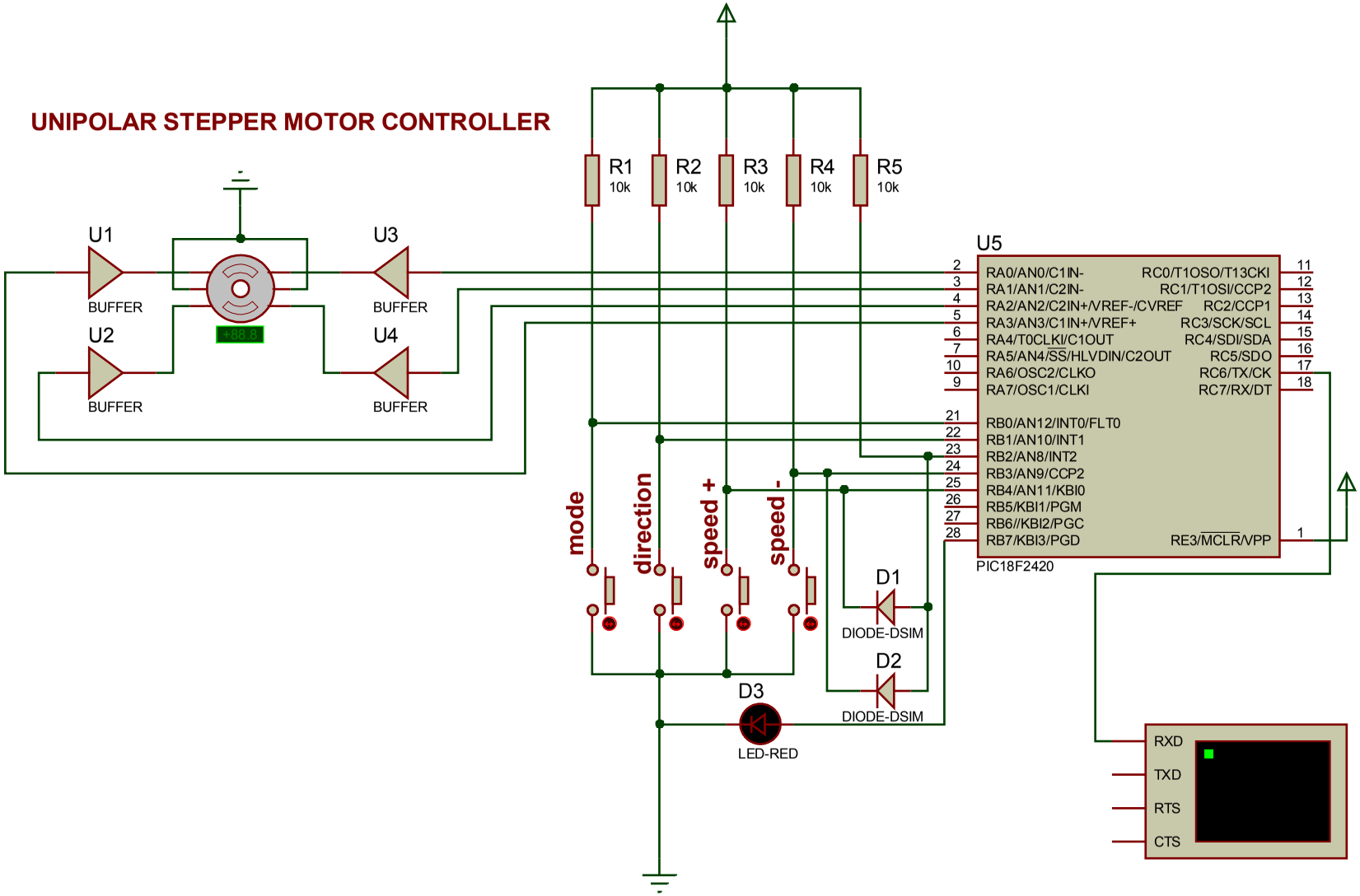
```
74
75  // Routine di interruzione
76  #pragma interrupt GestoreInterruzione
77
78  void GestoreInterruzione ()   // questa è la routine di interrupt
79  {
80     if (INTCONbits.INT0F)     // controllo se l'interruzione viene veramente da INT0
81     {
82        if(PORTBbits.RB4) direction = 1; else direction = 0; // se Q1 arriva prima di Q2...orario
83        pulsesec++;            // incremento il numero di impulsi al secondo
84        INTCONbits.INT0F = 0;  // devo segnalare che la interruzione è stata servita
85     }
86
87     if (INTCON3bits.INT2IF)     // controllo se l'interruzione viene veramente da INT2
88     {
89         if(slope == 0) { speed += SPEED_INCREMENT; if(speed >= MAX_SPEED) slope = 1; }
90         if(slope == 1) { speed -= SPEED_INCREMENT; if(speed == 0) slope = 0; }
91         INTCON3bits.INT2IF = 0; // devo segnalare che la interruzione è stata servita
92     }
93
94     if (INTCON3bits.INT1IF)     // controllo se l'interruzione viene veramente da INT1
95     {
96         LATA ^= 0x01; CCP1CON ^= 0x0C; CCP2CON ^= 0x0C;
97         INTCON3bits.INT1IF = 0; // devo segnalare che la interruzione è stata servita
98     }
99
100    if (INTCONbits.TMR0IF)              // controllo se l'interruzione viene veramente da TMR0
101    {
102       if(timer) timer--;              // se si faccio il lavoro...(decremento timer, se non è 0)
103       TMR0H = PH; TMR0L = PL;         // ricarico il timer con il valore per 256 impulsi
104       INTCONbits.TMR0IF = 0;          // devo segnale che la interruzione è stata servita
105    }
106 }
107
108 #pragma code GestoreInterruzioneJmp = 0x08   // queste righe servono per caricare  il vettore di interruzione a 0x08
109
110 void GestoreInterruzioneJmp() { GestoreInterruzione(); }
```

IL codice della gestione interruzioni è simile a quello della soluzione open loop...semplicemente l'aggiornamento del duty cycle in CCPx è stato portato nel main loop dopo l'algoritmo PID.

// unipolar stepper motor controller.c

```
1   #include <p18cxxx.h>   /* for TRISA and PORTA declarations */
2   #include <stdio.h>
3   #include <usart.h>
4
5   #pragma config OSC = INTIO67   // oscillatore interno a 1MHz
6
7   // no TX interrupt no RX interrupt UART mode 8 bit No parity 1 stop  non disabilita dopo 1 RX aud rate alto
8   #define PROTOCOL  USART_TX_INT_OFF & USART_RX_INT_OFF & USART_ASYNCH_MODE &
    USART_EIGHT_BIT & USART_CONT_RX & USART_BRGH_HIGH
9   #define PH 0xff // valori per contare 256 impulsi: il contatore è up modulo
10  #define PL 0x00 // 65536: ci metto 65289 (0xff00) così gli rimangono 256 impulsi per l'overflow
11
12  volatile unsigned int direction = 0, period = 300, timer = 0, mode = 0; // variabili globali
13
14  void sleep(unsigned int t) { timer = t; while(timer); }
15
16  void main()
17  {
18   unsigned int period2; //  meta period (in modo half step)
19
20   INTCON = 0x00; ADCON1 = 15; /* no AD sulle porte */ CMCON  = 7; // no comparatore sulla porta
21   TRISA = 0xf0; TRISB = 0x7f; LATB = 0x00; // porta A bit 0-3 uscita porta B in ingresso tranne RB7
22
23   T0CON = 0x88;    //  set up timer0 16 bit - internal clock = Fosc/4 -NO prescaler 1:1 --> 250KHz --> 4us x count
24   TMR0H = PH; TMR0L = PL;      // setup period high and low bytes 1 TOF = 1 ms
25
26   //RCONbits.IPEN = 1;         /* enables interrupt priority levels */
27   INTCONbits.INT0IE = 1; INTCON3bits.INT1IE = 1;   INTCON3bits.INT2IE = 1; // INT0, INT1, INT2 interrupt  ON
28   INTCONbits.TMR0IE = 1; // enables TMR0 interrupts
29   INTCONbits.GIEH = 1;       /* enable all high priority interrupts */
30
31   INTCON2bits.INTEDG2 = 0; // INT2 fronte in discesa
32
33   OpenUSART(PROTOCOL, 12); // ATTENZIONE: INIZIALIZZAZIONE UART:
34
35   while(1)
36   {
37            printf("direction %u mode %u period %u\x0d\x0a", direction, mode, period);
38            switch(mode)
39            {
40                     case 0: switch(direction)
41                              {
42                                       case 0: sleep(period); LATA = 0x01;      sleep(period); LATA = 0x02;
                                                 sleep(period); LATA = 0x04;      sleep(period); LATA = 0x08;
                                                 break;
43                                       case 1: sleep(period); LATA = 0x08;      sleep(period); LATA = 0x04;
                                                 sleep(period); LATA = 0x02;      sleep(period); LATA = 0x01;
44                              }
45                              break;
46
47                     case 1:  period2 = period >> 1;
48                              switch(direction)
49                              {
50                                       case 0: sleep(period2); LATA = 0x01;    sleep(period2); LATA = 0x03;
                                                 sleep(period2); LATA = 0x02;    sleep(period2); LATA = 0x06;
51                                               sleep(period2); LATA = 0x04;    sleep(period2); LATA = 0x0c;
                                                 sleep(period2); LATA = 0x08;    sleep(period2); LATA = 0x09;
                                                 break;
52                                       case 1: sleep(period2); LATA = 0x09;    sleep(period2); LATA = 0x08;
                                                 sleep(period2); LATA = 0x0c;    sleep(period2); LATA = 0x04;
53                                               sleep(period2); LATA = 0x06;    sleep(period2); LATA = 0x02;
                                                 sleep(period2); LATA = 0x03;    sleep(period2); LATA = 0x01;
54                              }
55            }
56   }
57  }
```

```
58
59  // Impostazione interrupt system
60
61  // Routine di interruzione
62  #pragma interrupt GestoreInterruzione
63
64  void GestoreInterruzione ()   // questa è la routine di interrupt
65  {
66     if (INTCONbits.INT0F)     // controllo se l'interruzione viene veramente da INT0
67     {
68               if(mode) mode = 0; else mode = 1;
69               INTCONbits.INT0F = 0;  // devo segnalare che la interruzione è stata servita
70     }
71
72     if (INTCON3bits.INT2IF)     // controllo se l'interruzione viene veramente da INT2
73     {
74            if(PORTBbits.RB4 == 0) if(period) period -= 10;
75            if(PORTBbits.RB3 == 0) period += 10;
76            INTCON3bits.INT2IF = 0; // devo segnalare che la interruzione è stata servita
77            LATB ^= 0x80;
78     }
79
80     if (INTCON3bits.INT1IF)     // controllo se l'interruzione viene veramente da INT1
81     {
82            if(direction) direction = 0; else direction = 1;
83            INTCON3bits.INT1IF = 0; // devo segnalare che la interruzione è stata servita
84     }
85
86    if (INTCONbits.TMR0IF)              // controllo se l'interruzione viene veramente da TMR0
87    {
88       if(timer) timer--;              // se si faccio il lavoro...(decremento timer, se non è 0)
89       TMR0H = PH; TMR0L = PL;         // ricarico il timer con il valore per 256 impulsi
90       INTCONbits.TMR0IF = 0;          // devo segnalare che la interruzione è stata servita
91    }
92  }
93
94  #pragma code GestoreInterruzioneJmp = 0x08   // queste righe servono per caricare  il vettore di interruzione a
    0x08
95
96  void GestoreInterruzioneJmp() { GestoreInterruzione(); }
```


UNIPOLAR STEPPER MOTOR CONTROLLER
R1
10k
R2
10k
R3
10k
R4
10k
R5
10k
U1
BUFFER
U2
BUFFER
U3
BUFFER
U4
BUFFER
+88.8
U5
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
mode
direction
speed +
speed -
D1
DIODE-DSIM
D2
DIODE-DSIM
D3
LED-RED
RXD
TXD
RTS
CTS