M049 - ESAME DI STATO ISTRUZIONE TECNICA
a.s. 2004/2005

CORSO DI NUOVO ORDINAMENTO

Indirizzo: ELETTRONICA E TELECOMUNICAZIONI

TEMA DI SISTEMI

Una scuola vuole monitorare la potenza elettrica continua di un pannello fotovoltaico per la generazione d'energia elettrica di cui è dotata.

Il pannello fotovoltaico può produrre una corrente massima di 3,3 Ampere e una tensione massima di 16,5 Volt. Questi valori massimi si riducono notevolmente a seconda della quantità dì luce solare che raggiunge gli elementi.

Per monitorare la potenza elettrica prodotta durante la giornata e nelle varie condizioni climatiche, si misurano tensione prodotta e la corrente prodotta.

Questi dati devono essere rilevati ogni 5 minuti e riversati in una memoria di tipo flash.

Una volta al giorno devono essere inviati ad un personal computer per produrre una statistica.

Per misurare la corrente si utilizza un sensore ad effetto Hall già parzialmente condizionato con uscita lineare in corrente, secondo la seguente proporzione:

• Se la corrente misurata è nulla (0 Ampere), in uscita la corrente vale 0 mA.

• Se la corrente misurata è 15 Ampere, in uscita la corrente è pari a 15 mA.

Le due grandezze da misurare devono essere convertite in tensioni comprese tra 0 e 3,3 Volt per essere adattate all'ingresso del convertitore analogico-digitale impiegato.

Il candidato, fatte le ipotesi aggiuntive ritenute opportune:

1) descriva lo schema a blocchi del sistema d'acquisizione dati per le grandezze elencate e le tecnologie impiegate;
2) progetti il condlzionamento dei segnali in uscita dal sensori;
3) descriva la struttura del sw di gestione della centralina di acquisizione;
4) codifichi in un linguaggio di sua scelta almeno una porzione significativa del sw di gestione

Durata massima della prova: 6 ore. E' consentito soltanto l'uso di manuali tecnici e calcolatrici non programmabili. Non è consentito lasciare l'Istituto prima che siano trascorse 3 ore dalla dettatura dei tema.

Soluzione:

Ipotesi aggiuntive:
a) Si pensa di limitare il numero di ore di luce ad un massimo di 15 ore (estate). In tal modo è possibile limitare il numero di acquisizioni che scendono da 24 * 60 / 5 = 288 a 15 * 60 / 5 = 180. In tal modo è possibile utilizzare solo 360 byte di memoria (2 byte per acquisizione a 8 bit di V ed I) in luogo di 576. Le acquisizioni saranno a 8 bit, in mancanza di specifica.
b) Allo scopo di non dover implementare anche un orologio si demanda al sistema PC ricevitore dei dati di correlare i dati ricevuti all'ora del giorno corrente. L'operatore metterà in funzione il sistema (o premerà il reset) sempre alla stessa ora convenuta.
c) Sempre il sistema di ricezione provvederà alla conversione fra numeri interi ricevuti da 0 a 255 in valori di tensione e corrente effettuando le dovute proporzioni relativamente ai 2 fondo scala di 16.5V e 3.3A

1) Descrizione dello schema elettrico / a blocchi = struttura dell'HW:
Utilizziamo il microcontrollore Microchip Pic18F2420 esaminato durante l'anno. Esso può funzionare con tensione di alimentazione da 2.2V fino a 5V. Di tale microcontrollore utilizzeremo due canali (Ch0 e Ch1 del convertitore AD a 10 bit di tipo SAR in esso contenuto; Ch0 allo scopo di acquisire un segnale proporzionale alla tensione fornita dal pannello e Ch1 allo scopo di acquisire un segnale proporzionale alla corrente fornita dal pannello. Il segnale da inviare a Ch0 sarà semplicemente ottenuto con un partitore di tensione in quanto la tensione del pannello non può ovviamente essere direttamente inviata all'ingresso del Ch0 perchè troppo elevata; dovrà quindi essere analogamente ridotta. Il segnale da inviare a Ch1 sarà invece ottenuto mediante il sensore di Hall, che come noto fornisce un segnale proporzionale alla corrente che scorre nel suo elemento sensibile. Il sensore è posto quindi in serie all'utilizzatore e la sua corrente di uscita convertita in tensione facendola passare in una resistenza. I due operazionali U3A e U3B servono semplicemente da buffer separatori, in quanto la giusta dinamica di ingresso 0-Vrif è ottenuta con il gruppo R1-RV1 per il Ch0 e RV2 per il Ch1. Si tratta quindi di dimensionare due semplici attenuatori di tensione mediante partitore resistivo. Per ogni acquisizione utilizzeremo solamente gli 8 bit più significativi.

Le temporizzazioni necessarie sono ottenute utilizzando un'altra periferica interna al microcontrollore, precisamente il timer0, allo scopo di ottenere l'attesa di 5 minuti fra una acquisizione e l'altra. Il timer è utilizzato anche per ottenere il timeout di rispedizione del pacchetto dati nel caso in cui il ricevitore segnali errore di ricezione.

Le coppie di valori acquisiti (1 byte per V e 1 byte per I) sono via via memorizzate, in forma di valori interi su un byte (da 0 a 255, con 255 = Vmax = 16.5V o Imax = 3.3A) nella EEPROM interna al microcontrollore; dopo 180 coppie di acquisizioni (360 in totale) tutti i dati vengono trasmessi impacchettati utilizzando la UART presente nel microcontrollore, e poi il ciclo ricomincia.

La UART è connessa alla linea seriale RS232 e tramite essa ad analogo dispositivo su un PC remoto, sul quale i dati vengono elaborati; poichè, allo scopo di garantire una sufficiente immunita ai disturbi, i segnali dello standard RS232 sono notevolmente diversi da quelli TTL sui terminali del microcontrollore (+15/-15 per 0 e 1 in luogo di 0 e +5), è necessario interporre uno stadio di conversione di livello, rappresentato dall'integrato MAX232 presente nello schema (convertitore di livello). Analogo stadio è presente anche all'interno del PC. Si utilizza in realtà la versione a 3.3V Max3232.

L'alimentazione del microcontrollore e del resto della circuiteria è fornita dal pannello stesso, tramite apposito stabilizzatore. La batteria mostrata nel circuito a valle dello stabilizzatore è una batteria tampone necessaria al funzionamento del circuito in condizione di scarsa illuminazione del pannello o di buio.

Il microcontrollore è utilizzato in configurazione con oscillatore al quarzo da 8MHz (#pragma OSC=HS), anche se date le limitate necessità del sistema si potrebbe benissimo utilizzare l'oscillatore RC interno da 1MHz (#pragma OSC=INTIO67), ritoccando opportunamente il baud rate, la selezione del clock dell'AD e l'impostazione del timer0 in modo da ottenere le stesse temporizzazioni.

Pertanto le periferiche del microcontrollore che vengono utilizzate e i pin ad esse associati sono:

1) convertitore AD, ch0 e ch1 (rispettivamente pin 2 e 3 dell'integrato pic18F2420 versione 28 pin)
2) UART segnali TxD e RxD (rispettivamente pin 17 e 18 dell'integrato pic18F2420 versione 28 pin)
3) Timer TMR0 per la creazione dei tempi di attesa necessari (gest. a polling o in versione migliorata a interrupt). E' utilizzato internamente all'integrato, senza uscite sui pin.
4) EEPROM per la memorizzazione dei dati. E' utilizzata internamente all'integrato, senza uscite sui pin.
5) Pin 10 e 9 per il quarzo esterno e pin 8 e 19 per Vss (massa) e 20 per Vdd (alimentazione).

2) Condizionamento dei segnali:
Condizionamento della tensione: occorre passare da 16.5V (tensione max del pannello) a 3.3V garantendo una certa regolazione; per via del successivo inseguitore di tensione non dobbiamo preoccuparci di dover fornire un segnale di corrente, pertanto questa regolazione può essere effettuata con un partitore di resistenza complessiva non troppo elevata, 12.7K in totale, di cui una parte fissa da 10K e una parte variabile da 2.7K che regoleremo per ottenere 3.3V quando la V del pannello è max.

Condizionamento della corrente: come da specifiche il sensore produce 15 mA per 15A di ingresso, quindi quando il pannello produce 3.3A il sensore produrrà 3.3 mA; pertanto il condizionamento può essere realizzato con trimmer da 1K in funzione di partitore variabile (3.3mA per 1K = 3.3V).

3) Struttura del SW:
Il software di gestione è composto da una routine principale (main()), da una routine accessoria che effettua le acquisizioni dal canale selezionato e da una routine che implementa i ritardi necessari alle temporizzazioni del sistema.

Il main() svolge i compiti seguenti:

a) inizializza le periferiche: AD: scelta tensioni di riferimento, frequenza di clock, tempo fra le acquisizioni, giustificazione dato acquisito nel registro ADRES (si lascia il default, a SX), (e inizializza il sistema di interrupt, nella seconda soluzione);
b) inizializza la UART: formato trasmissione, parità, dati, stop, baud rate;
c) inizializza il timer0 per ottenere una temporizzazione periodica di 1ms;
d) rimane in un ciclo senza fine dove per 288 volte (24 ore per 60 minuti diviso 5 minuti) :

-d1) aspetta 5 minuti;

-d2) se il numero di cicli di 5 minuti è < 180 effettua i passi seguenti, altrimenti passa al ciclo successivo solo aspettando
-d3) invoca la funzione per l'acquisizione dal Ch0 e memorizza nella EEPROM, il valore ottenuto, sfruttando le apposite funzioni di libreria;
-d4) ripete il passo d3 per il canale 1;

-d5) dopo 288 cicli rilegge i dati acquisiti (360 byte) dalla EEPROM e li invia uno alla volta sulla seriale al PC remoto.

La routine di acquisizione dati, Acq(ch) effettua i passi:

a) seleziona il canale ch passato come argomento e da il via all'acquisizione (impostando il bit SOC(GO);
il canale viene impostato negli appositi bit di ADCON0 che comandano il multiplexer del canale.
b) aspetta in polling la fine della conversione controllando il bit EOC(DONE);
c) legge il risultato della conversione dal registro ADRESH e lo restituisce alla funzione main attraverso il valore di ritorno della funzione.

Allo scopo di garantire la corretta trasmissione dei dati al punto d5 il pacchetto delle 180 doppie acquisizioni è inviato con il metodo seguente (detto **PAR** : positive acknowledgment and retransmission, è alla base del TCP/IP):

Il trasmettitore manda il pacchetto composto da: carattere di delimitazione del pacchetto ('S'), numero identificativo che cresce modulo 2 (0-1-0-1....) 360 byte di dati V e I e una somma di controllo (checksum) di tutti i byte; dopodichè si pone in attesa per un tempo T di timeout (ad esempio 1 secondo) che il ricevitore invii in risposta un carattere ACK di riconoscimento (Acknowledgment = riconoscimento = ASCII 6); se tale ACK arriva il trasmettitore spedisce il prossimo pacchetto (quando è pronto) incrementando l'ID modulo 2, altrimenti rispedisce il solito pacchetto con il solito ID. Il ricevitore mantiene anche lui un suo ID dei pacchetti ricevuti che incrementa modulo 2 ogni volta che un pacchetto è ricevuto correttamente; quando riceve un pacchetto controlla dapprima la checksum ricalcolandola dai dati ricevuti e confrontandola con quella ricevuta, se essa è sbagliata scarta il pacchetto come corrotto, non spedisce ACK e non incrementa il suo ID; se la checksum è OK controlla l'ID ricevuto; se esso è uguale al suo (che è quello atteso) il pacchetto è OK, spedisce ACK, incrementa il suo ID e usa il pacchetto; se l'ID ricevuto è diverso da quello atteso scarta il pacchetto come duplicato ma spedisce lo stesso ACK (per arrestare la rispedizione).

Formato Pacchetto:

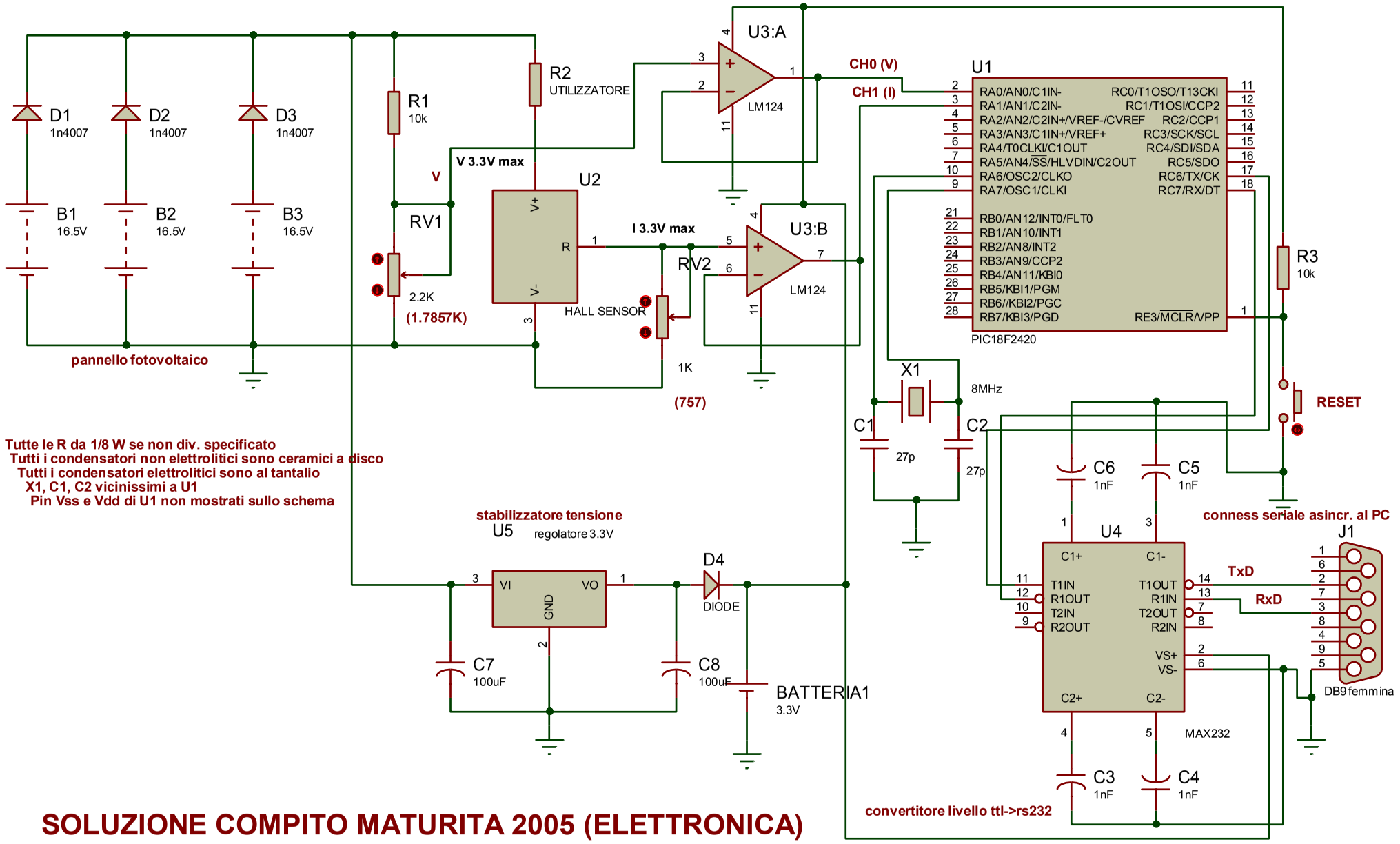
'S' : utilizzato come delimitatore di pacchetto. 1 byte
ID = ('0'/'1') : identificatore del pacchetto 1 byte
<360 dati delle acquisizioni dei canali codificati in esadecimale su due cifre hex = 720 byte>
<somma di controllo> (somma di 'S'+ID+tutti i 360 byte) codificata come 5 caratteri hex (('S'= 83)+ ('1'= 49) +360*255 = 91932 = 1671C hex max 5 caratteri)

In totale un pacchetto è lungo quindi 727 byte.

Si noti che vengono trasmessi solo caratteri ASCII da '0' a 'F' cioè le cifre esadecimali, e il carattere 'S'. pertanto possiamo utilizzare la funzione di libreria printf. Pacchetto tipico: S Id $hh_{(0)}$ hh hh .....$hh_{(359)}$ ccccc (dove ID è 0 o 1 e hh sono cifre hex)

Le due attese di 300000 ms (5 minuti) e di 1000 ms sono ottenute aspettando 3000000 e 1000 ms. L'attesa base di un 1ms è ottenuta utilizzando il timer0 con prescaler 1 e caricando dentro TMR0 il valore 64536; in tal modo dopo 1000 impulsi di clock TMR0 va in overflow, cioè passa da 65535 a 0 (essendo a 16 bit) e con il clock a 8MHz è passato 1 ms; posso accorgermene andando ad interrogare il bit TMR0IF di INTCON0; quando ciò accade la attesa è finita e allora ricarico il timer0 per un altro ms; la routine Attesa() effettua questo controllo per timer volte, essendo timer una variabile globale che viene impostata con il valore di ms desiderati e che ritorna 1 fintanto che timer è diverso da 0; pertanto la scrittura: timer = 1000; while(Attesa()) comando; esegue comando per 1000 ms.

E' riportata anche una soluzione per la implementazione della attesa base di 1 ms che utilizza la gestione ad interrupt del timer0; in tal caso il decremento della variabile timer non è più effettuato a polling all'interno della routine Attesa() dentro la funzione main() ma ad interruzione dalla routine di gestione dell'interrupt di timer0 che va abilitato a interrompere la CPU.



SOLUZIONE COMPITO MATURITA 2005 (ELETTRONICA)

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

// PROGRAMMA CONTROLLO CENTRALINA ACQUISIZIONE DATI PANNELLO FOTOVOLTAICO (in linguaggio di progetto C-like)

```
1.  blocco #include <file di intestazioni di libreria necessari>
2.  blocco #pragma <direttive di configurazione del microcontrollore necessarie>
3.
4.  blocco #define <definizioni di costanti numeriche e letterali necessarie>
5.
6.  blocco dichiarazione variabili globali necessarie esempio unsigned long int timer;

7.  unsigned char Aspetta(void);  // routine di attesa; torna 0 se timer è diventato 0, altrimenti torna 1.  prototipo funz accessoria
8.  unsigned char acq(unsigned char ch);  // prototipo ROUTINE DI ACQUISIZIONE; ingresso:canale, ritorno: valore acquisito

9.  void main()   // funzione principale
10. {
11.   <dichiarazione variabili LOCALI necessarie> // SE LA EEPROM NON BASTA, USO LA RAM: unsigned char Dati[360]; // tanto c'è la batteria
12.
13.  // INIZIALIZZAZIONE PERIFERICHE NECESSARIE (AD, SERIALE, TIMER, SISTEMA INTERRUZIONI)
14.
15.  while(1)  // ciclo senza fine ACQUISIZIONE-SPEDIZIONE
16.  {
17.   for(i = 0; i < 288; i++) // ACQUISIZIONE DATI:  (24 ore * 60 min / 5 min) = 288. Solo 180 sono però utili.
18.   {
19.    if(i < 180)  // 180 *  5 minuti: superate 15 ore di acquisizioni...va in fondo al ciclo: aspetta e basta per le rimanenti 9 ore....
20.    {
21.     <acquisizione canale 0>
22.     <scrittura del valore acquisito nella EEPROM agli indirizzi pari (cioè 2*i)  o in Dati[i]>
23.     <acquisizione canale 1>
24.     <scrittura del valore acquisito nella EEPROM agli indirizzi dispari (cioè 2*i+1), o in Dati[2*i+1]>
25.    }
26.    <Attesa 5 minuti, con Aspetta()>
27.   }
28.
29.   do    // SPEDIZIONE DATI COL METODO PAR  (se elimino il do while diventa trasmissione senza error recovery)
30.   {     // formato: S ID ch0 ch1...ch0 ch1 checksum  1+1+360+4 byte (la cks sta al più su 5 caratteri hex: 23DC2)
31.
32.           printf("S%01x", id);  // invio 'S' come delimitatore di pacchetto e  l'ID (id è 0 oppure 1 (un byte))
33.
34.           cks = 'S' + id; // inizializzo la checksum
35.
36.           for(i = 0; i < 360; i++) // per ogni byte acquisito:
37.           {
38.             lo rileggo dalla EEPROM (o da Dati[i])
39.             lo spedisco in hex 2 byte
40.             lo aggiungo alla checksum
41.           }
42.
43.           printf("%05x", cks);      // invio la checksum su 5 caratteri hex
44.
45.           aspetto per 1 secondo che il ricevitore mi mandi ACK
46.
47.   } while(non arriva ACK); // se ACK non arriva entro il timeout rispedisco  il solito pacchetto e il solito ID
48.
49.   se invece è arrivato ACK ed esco dal ciclo, incremento ID e procedo con un nuovo ciclo di acquisizioni...
50.  }
51. }
52.
53.  unsigned char Aspetta()  // torna 0 se timer è diventato 0, altrimenti torna 1.
54.  {
55.     if (INTCONbits.TMR0IF == 1)          // controllo se  TMR0 ha avuto il rollover (o overflow che dir si voglia)...è passato un millisecondo
56.     {
57.       INTCONbits.TMR0IF = 0;             // se si azzero il flag di overflow del timer hw
58.       if(timer) timer--;  else return(0);      // decremento timer, se non è 0, atrimenti ritorno 0 per segnalare fine dell'attesa
59.       TMR0H = PH; TMR0L = PL;            // ricarico il timer con il valore per 1000 impulsi
60.     }
61.    return(1);   // segnalo che l'attesa non è finita
62.  }
63.
64.  unsigned char acq(unsigned char ch)  // ROUTINE DI ACQUISIZIONE   // QUESTO POTREBBE ESSERE IL SEGMENTO DI CODICE
65.  {                                                                 // CHE IL TESTO RICHIEDE, ALLA PEGGIO
66.    ADCON0 = 0x03 | (ch << 2);  // SOC:   0x02 è la maschera per il bit GO in ADCON0...seleziono il canale, abilito l'AD
67.
68.    while(ADCON0 & 0x02); // attesa EOC
69.
70.   return(ADRESH);    //  ritorno solo gli 8 bit piu significativi
71.  }
72.
```

```
// PROGRAMMA CONTROLLO CENTRALINA ACQUISIZIONE DATI PANNELLO FOTOVOLTAICO
#include <p18cxxx.h> // for TRISA and PORTA declarations
#include <stdio.h>   // printf, puts etc
#include <usart.h>   // funzioni UART
#include <eep.h>     // funzioni EEPROM

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

#define ACK 6 // codice ASCII carattere ACK

// impostazione  USART: no tx & rx interrupt, UART mode, 8N1, no disable, high BR
#define PROTOCOL USART_TX_INT_OFF & USART_RX_INT_OFF & USART_ASYNCH_MODE & USART_EIGHT_BIT & USART_CONT_RX & USART_BRGH_HIGH
#define BRD  51 // valore di baud rate divisor dalla formula BRD = Fclock/16*baud_rate – 1 : 8MHz/16*9600-1 = 51
#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 long int timer;
unsigned char acq(unsigned char ch);  // prototipo ROUTINE DI ACQUISIZIONE; ingresso:canale, ritorno: valore acquisito
unsigned char Aspetta(void);  // torna 0 se timer è diventato 0, altrimenti torna 1.

void main()
{
 unsigned int i, val, cks, ch, id; // variabili necessarie

 // INIZIALIZZAZIONE PERIFERICHE NECESSARIE

 OpenUSART(PROTOCOL, BRD);   // INIZIALIZZAZIONE UART: baud rate = (Fclock / 16) - 1

 ADCON1 = 0x0d; // + e - vrif da alimentazione, ch 0, ch1 porta RA0 e RA1 ingressi analogici
 T0CON = 0x82;  // set up timer0 - prescaler 1:8 -> a 8 MHz --> 1us x count
 TMR0H = PH;    TMR0L = PL;    // setup period  high byte low byte 1 TOF = 1 ms

 while(1)  // ciclo senza fine ACQUISIZIONE-SPEDIZIONE
 {
  for(i = 0; i < 288; i++) // ACQUISIZIONE DATI: 24 ore * 60 min / 5 min = 288 cicli; solo 180 utili.
  {
   if(i < 180) continue;  // 180 *  5 minuti: superate 15 ore di acquisizioni...va in fondo al ciclo: aspetta e basta per le rimanenti 9 ore....
   {
    Write_b_eep(2*i, Acq(0));          // acquisisco e memorizzo l'acquisizione da Ch0;  posti 0-2-4-6...(pari); o senza moltipl: i << 1;
    Busy_eep();                        // aspetto fine scrittura in EEPROM
    Write_b_eep(2*i+1, Acq(1));        // acquisisco e memorizzo l'acquisizione da Ch0; posti 1-3-4-7...(dispari); o senza molt. I << 1 + 1;
    Busy_eep();                        // aspetto fine scrittura in EEPROM
   }
   timer = 300000; while(Aspetta());  // Attesa 5 minuti (300 mila ms)
  }

  do                    // SPEDIZIONE DATI COL METODO PAR  (se elimino il do while diventa trasmissione senza error recovery)
  {                     // formato: S ID ch0 ch1...ch0 ch1 checksum  1+1+720+5 byte (la cks sta al più su 5 caratteri hex: 23DC2)
            printf("S%01x", id);  // invio 'S' come delimitatore di pacchetto e  l'ID (id è 0 oppure 1 (un byte))

            cks = 'S' + id; // inizializzo la checksum

            for(i = 0; i < 360; i++) // per ogni byte acquisito:
            {
                    val = Read_b_eep(i); // lo rileggo dalla EEPROM
                    printf("%02x", val); // lo spedisco in hex 2 byte
                    cks += val;          // lo aggiungo alla checksum
            }
            printf("%04x", cks);      // invio la checksum

            timer = 1000; while(Aspetta())  if(DataRdyUSART()) ch = getcUSART(); // aspetto ACK dalla UART.per 1 s..

  } while(ch != ACK); // se ACK non arriva entro il timeout rispedisco  il solito pacchetto e il solito ID

  id = id + 1; id = id % 2; // se arriva ACK esco dal ciclo, incremento ID e procedo con 576 nuovi cicli.
 }
}

unsigned char Aspetta()  // torna 0 se timer è diventato 0, altrimenti torna 1.
{
   if (INTCONbits.TMR0IF == 1)          // controllo se  TMR0 ha avuto il rollover (o overflow che dir si voglia)...è passato un millisecondo
   {
     INTCONbits.TMR0IF = 0;             // &= 0xfb; se si azzero il flag di overflow del timer hw
     if(timer) timer--;  else return(0);      // decremento timer, se non è 0, atrimenti ritorno 0 per segnalare fine dell'attesa
     TMR0H = PH; TMR0L = PL;            // ricarico il timer con il valore per 1000 impulsi
   }
  return(1);   // segnalo che l'attesa non è finita
}

unsigned char acq(unsigned char ch)  // ROUTINE DI ACQUISIZIONE   // QUESTO POTREBBE ESSERE IL SEGMENTO DI CODICE
{                                                                 // CHE IL TESTO RICHIEDE, ALLA PEGGIO
  ADCON0 = 0x03 | (ch << 2);              // SOC:   0x02 è la maschera per il bit GO in ADCON0...seleziono il canale, abilito l'AD
  while(ADCON0 & 0x02);                   // attesa EOC
 return(ADRESH);                          //  ritorno solo gli 8 bit piu significativi
}
```

```
// PROGRAMMA CONTROLLO CENTRALINA ACQUISIZIONE DATI PANNELLO FOTOVOLTAICO (versione a INTERRUPT)
#include <p18cxxx.h> // for TRISA and PORTA declarations
#include <stdio.h>   // printf, puts etc
#include <usart.h>   // funzioni UART
#include <eep.h>     // funzioni EEPROM

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

unsigned long int timer; // contatore degli overflow di timer, globale, visibile al main e alla routine di interruzione

#define ACK 6 // codice ASCII carattere ACK
// impostazione  USART: no tx & rx interrupt, UART mode, 8N1, no disable, high BR
#define PROTOCOL USART_TX_INT_OFF & USART_RX_INT_OFF & USART_ASYNCH_MODE & USART_EIGHT_BIT & USART_CONT_RX
    & USART_BRGH_HIGH
#define BRD  51 // valore di baud rate divisor dalla formula BRD = Fclock/16*baud_rate – 1 : 8MHz/16*9600-1 = 51

#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 acq(unsigned char ch);  // prototipo ROUTINE DI ACQUISIZIONE; ingresso:canale, ritorno: valore acquisito

void main()
{
 unsigned int i, val, cks, ch, id; // variabili necessarie

 // INIZIALIZZAZIONE PERIFERICHE NECESSARIE

 OpenUSART(PROTOCOL, BRD);

 ADCON1 = 0x0d; // + e - vrif da alimentazione, ch 0, ch1 porta RA0 e RA1 ingressi analogici
 T0CON = 0x82;  // set up timer0 - prescaler 1:8 -> a 8 MHz --> 1us x count
 TMR0H = PH;    TMR0L = PL;    // setup period  high byte low byte 1 TOF = 1 ms
 INTCON = 0xa0; // abilito tutti le interruzioni e quelle del TMR0

 while(1)  // ciclo senza fine ACQUISIZIONE-SPEDIZIONE
 {
  for(i = 0; i < 288; i++) // ACQUISIZIONE DATI: 24 ore * 60min / 5min  = 288. Solo 180 utili.
  {
   if(i < 180) continue;  // 180 *  5 minuti: superate 15 ore di acquisizioni...va in fondo al ciclo: aspetta e basta per le rimanenti 9 ore....
   {
    Write_b_eep(2*i, Acq(0));            // acquisisco e memorizzo l'acquisizione da Ch0;  posti 0-2-4-6...(pari); o senza moltipl: i << 1;
    Busy_eep();                          // aspetto fine scrittura in EEPROM
    Write_b_eep(2*i+1, Acq(1));          // acquisisco e memorizzo l'acquisizione da Ch0; posti 1-3-4-7...(dispari); o senza molt. I << 1 + 1;
    Busy_eep();                          // aspetto fine scrittura in EEPROM
   }
   timer = 300000;    while(timer);     // imposto timer a 300000 ms = 5 minuti e aspetto che timer diventi 0
  }

  do       // SPEDIZIONE DATI COL METODO PAR  (se elimino il do while diventa trasmissione senza error recovery)
  {        // formato: S ID ch0 ch1...ch0 ch1 checksum  1+1+360+5 byte (la cks sta al più su 5 caratteri hex: 23DC2)

            <SPEDIZIONE PACCHETTO COME PER IL CODICE PAGINA PRECEDENTE>

            timer = 1000; // inizializzo timer a 1000 (1s)
            while(timer)  if(DataRdyUSART()) ch = getcUSART(); // aspetto ACK dalla UART.per 1 s..

   } while(ch != ACK); // se ACK non arriva entro il timeout rispedisco  il solito pacchetto e il solito ID

   id = id + 1; id = id % 2; // se arriva ACK esco dal ciclo, incremento ID e procedo con 576 nuove acquisizioni...
  }
}

unsigned char acq(unsigned char ch) { <ROUTINE ACQUISIZIONE COME PER IL CODICE PAGINA PRECEDENTE> }

// Routine di interruzione di timer0 per l'attesa di 1 ms.

#pragma interrupt GestioneTimer0    // GestioneTimer0 è una routine di interrupt

void GestioneTimer0 ()              // questa è la routine di interrupt; va in esecuzione ogni ms
{
    if (INTCONbits.TMR0IF == 1)      // controllo se l'interruzione viene veramente da TMR0
    {
      if(timer) timer--;             // se si faccio il lavoro...(decremento timer, se non è 0)
      INTCONbits.TMR0IF = 0;         // &= 0xfb; devo segnalare che la interruzione è stata servita
      TMR0H = PH; TMR0L = PL;        // ricarico il timer con il valore per 1000 impulsi
    }
}

#pragma code  GestioneTimerJmp0 = 0x08 // carico il vettore di interruzione a 0x08 (interruzioni ad alta priorità
void   GestioneTimerJmp0 { GestioneTimer(); }
```