

Relazioni Laboratorio Elettronica

Giorgio Pizzati

Novembre 2019

Indice

1	Microcontrollore 8051	2
1.1	Introduzione al microcontrollore 8051	2
1.1.1	Configurazione del micro	2
1.2	Relazione led	3
1.2.1	Led in assembly in polling	3
1.2.2	Led in assembly con interrupt	4
1.2.3	Led in C	5
1.3	Relazione Seriale	7
1.3.1	Seriale in C	8
1.3.2	Seriale in assembly	11
2	ARM	12
2.1	Introduzione ai micro ARM e CORTEX	12
2.1.1	Configurazione del micro	12
2.2	Seriale	12
2.3	Timer7 e Led	16
2.4	I2C, termometro e display	16

Capitolo 1

Microcontrollore 8051

1.1 Introduzione al microcontrollore 8051

Il microcontrollore 8051, il primo dei due usati in questo corso ha una struttura di tipo Harvard(come tutti i microcontrollori).

- memoria programmi distinta memoria dati, alcuni anche doppio bus
- 8bit
- interrupt non nested
- assembly

1.1.1 Configurazione del micro

Qui di seguito viene riportata la configurazione generica eseguita all'inizio di ogni esperienza con l'8051.

Il primo passo per configurare il microcontrollore è disattivare il watchdog. Per fare ciò viene usato il Configuration Wizard. All'interno di questo software viene creato un file assembly che viene incluso nel progetto del'8051 ed eseguito all'inizio del main mettendo in funzione tutte le periferiche interessate. Un ulteriore step da fare in fase di configurazione è quello di indicare al micro una sorgente esterna per il clock. La sorgente che useremo è un cristallo oscillatore che ha una frequenza di 22.1184 MHz. Questo passaggio è necessario per garantire al micro di avere una frequenza di clock più precisa in quanto quella interna è di (2 ± 0.02) MHz. Una volta completata la configurazione nel Wizard, si esporta un file .asm che viene incluso nel progetto e nel file contenente il programma(e.g. "led.a51") viene dichiarata una funzione esterna Init_Device che fa riferimento a quella scritta dal Wizard.

1.2 Relazione led

1.2.1 Led in assembly in polling

In questa esperienza si è programmato il microcontrollore per accendere e spegnere il led. Il codice è stato scritto in assembly utilizzando il timer in polling.

Per la realizzazione è necessario impostare la porta P1.6 che controlla l'accensione e lo spegnimento del led, in configurazione push-pull ed impostare il Timer0. Per realizzare i due cicli su due registri distinti sono stati usati R0 per i cicli che durano un overflow del timer ed R1 per i 10 cicli in cui viene acceso e spento il led. Dopo aver configurato il micro, nel main, inizia un loop che dura 10 cicli, all'interno dei quali:

- si accende il led
- si chiama la funzione timer, all'interno della quale il micro starà per un tempo τ
- si spegne il led
- si chiama di nuovo timer
- si ritorna all'inizio del loop

Listing 1.1: Main del programma

```
MOV R0, #10
REPEAT:
SETB P1.6 ;accendo il led
LCALL TIMER
CLR P1.6 ;spengo il led
LCALL TIMER
DJNZ R0, REPEAT
```

La funzione timer è implementata come una serie di cicli(50), durante i quali:

- si aspetta un overflow del timer in un while loop
- si resettano le variabili del timer
- si controlla se sono passati 50 cicli

Listing 1.2: Funzione timer

```
TIMER:
;faccio trascorrere un po' di tempo
MOV R1, #50 ;voglio che venga eseguito due volte
FOR:
SETB TRO ;faccio partire il cronometro
WHILE:
```

```

        JNB TFO, WHILE ;
CLR TRO ; FERMO IL TIMER
CLR TFO ; AZZERO IL FLAG DI OVERFLOW
SETB TRO ; faccio ripartire il timer
DJNZ R1, FOR ; loop eseguito 50 volte?
CLR TRO ; fermo il timer
CLR TFO ; azzerò il flag di overflow
RET

```

1.2.2 Led in assembly con interrupt

La gestione dell'interrupt è il fulcro di questa esperienza. Innanzitutto quindi bisogna abilitarlo nel configuration wizard. È stato poi necessario capire come gestirlo. È stato necessario fare un code segment a 0x000B dove punta l'interrupt del timer0, ovvero la porzione di codice che viene eseguita quando scatta l'interrupt. Visto che lo spazio riservato in 0x000B permette di eseguire solo un'istruzione è necessario fare un JMP ad un'area di memoria programmi in cui è possibile svolgere più istruzioni. In quest'area viene gestito completamente l'interrupt dalle funzioni sotto elencate.

Listing 1.3: Funzione timer

```

TIMER :
    CLR TRO ; viene fermato il timer
    CLR TFO ; si pulisce la flag di overflow
    DJNZ R0, NULLA ; si decrementa R0, salta se=0
    JMP SCELTA ; questo jump viene eseguito se R0=0

NULLA :
    SETB TRO ; faccio ripartire il micro
    RETI ; si esce dall'interrupt

```

Scelta è una semplice funzione che determina cosa fare, se spegnere oppure accendere il led.

Listing 1.4: Funzione scelta

```

SCELTA :
    ; SE R1 = 1 ALLORA ACCENDO, SE = 0 SPENGO
    MOV A, R1 ; si sposta nell'acc. R1 per fare l'if
    JZ ACCENSIONE
    JNZ SPEGNIMENTO

```

Mentre le funzioni accensione e spegnimento sono le seguenti

Listing 1.5: Funzioni accensione e spegnimento

```

ACCENSIONE :
    SETB P1.6 ; si accende il led

```

```

MOV R0, #255 ; si imposta R0 a 255
MOV R1, #1 ; si imposta lo status a R1
SETB TR0; faccio ripartire il micro
RETI

```

SPEGNIMENTO:

```

CLR P1.6; si spegne il led
MOV R0, #255
MOV R1, #0
SETB TR0; faccio ripartire il micro
RETI

```

R1 quindi in questo caso è un registro di stato, tiene in memoria lo stato del led, se è acceso o spento.

Ciò che viene eseguito nel main è solamente la configurazione del micro, viene messo in R0 l'indirizzo di 255 e in R1 quello di 0.

1.2.3 Led in C

Scopo di quest'esperienza è di apprendere l'interazione tra il programma in C e quello in assembly. La parte di configurazione del micro infatti rimane scritta in asm mentre il codice principale è scritto in C. L'interazione in realtà si limita a dichiarare nel file .c una funzione esterna Init Device definita nel file .asm. Il programma in questo caso, invece di limitarsi ad accendere e spegnere il led, permette di regolarne la luminosità tramite il bottone disposto sulla scheda di sviluppo. Dunque le novità del codice rispetto a quelli precedenti sono

- codice in C
- gestione della luminosità del led
- gestione dell'interrupt del bottone

Quindi andando in ordine il codice in C: volendo operare solo su alcuni bit dei registri di controllo è stato necessario definirli a partire dalle variabili dichiarate nel file C8051f020.h in questo modo:

```

sbit P1_6 = P1 ^ 6;
sbit P3_7 = P3 ^ 7;

```

che permette di selezionare solo i bit che ci interessano.

Per quanto riguarda la luminosità, che in realtà non è regolabile sul micro, viene usato un'escamotage modificando la frequenza con cui viene acceso e spento il led. All'occhio umano il cambio di frequenza di accensione appare come una differente luminosità in quanto la frequenza è paragonabile con il refresh rate dell'occhio.

La gestione dell'interrupt del bottone viene eseguita in modo molto simile a quella del led, tranne per il fatto che il flag dell'interrupt non è bit addressable

quindi per resettarlo è necessario usare una maschera e per il numero di interrupt assegnato. Come ulteriore differenza dall'asm infatti si ha che per gestire l'interrupt è necessario definire funzioni il cui nome è seguito da "interrupt" e da un numero che identifica quale interrupt si sta gestendo.

Loops identifica quanti cicli di overflow del timer si aspettano prima di cambiare lo stato del led, ovvero identifica la frequenza di accensione e spegnimento. Questo controllo viene effettuato all'interno dell'interrupt del timer.

Listing 1.6: Interrupt timer

```
void interruzione_timer(void) interrupt 1{
    //fermo
    TR0=0;
    TF0=0;
    loops--;
    if (loops<=0){
        //devo cambiare lo stato del led
        if(led_status){
            //accendo
            P1_6=1;
            led_status=0;
            loops=loops_on;
        }
        else{
            //spengo
            P1_6=0;
            led_status=1;
            loops=n_loops-loops_on;
            //faccio ripartire
        }
    }
    TR0=1;
}
```

Dunque si è definita la luminosità come un int che può essere 0, 25,50,75,100. La variabile lumstatus identifica in che stato si è tra i 5 possibili. Essa viene modificata nell'interrupt del bottone, quindi ogni volta che viene schiacciato.

Listing 1.7: Interrupt pulsante

```
void interruzione_pulsante(void) interrupt 19{
    TR0=0;
    TF0=0;
    //cambio status
    if(lum_status<4){
        lum_status++;
    }
    else{
```

```

        lum_status=0;
    }
    luminosity=lum_status*25;
    loops_on=n_loops*luminosity/100;
    //resetto tutto
    loops=loops_on;
    led_status=0;
    P1_6=1;
    //reset del flag dell'interrupt esterna 7
    P3IF &= ~0x80;
    TR0=1;//faccio ripartire il timer
}

```

Infine è necessario settare le variabili globali e quelle nel main

Listing 1.8: Variabili globali

```

#define n_loops 5
char idata stack1[16];
int luminosity = 0;
int loops_on;
int loops; //lo stato iniziale=1
int led_status=0;
int lum_status=0;//indica le opzioni(0,25, 50, 75,100)

```

Listing 1.9: main

```

SP = (char) (&stack1);
Init_Device();
loops_on=n_loops*luminosity/100;
loops=loops_on;

```

1.3 Relazione Seriale

La comunicazione UART è l'oggetto di questa esperienza. Si è messo in comunicazione il micro con il computer, tramite un cavo che collega la usb del computer con i due pin (più il pin GND) del micro, uno per la trasmissione e uno per la ricezione.

Prima di trasmettere e ricevere un carattere è necessario innanzitutto configurare la UART, con il BAUD RATE (9600 bps) corretto, e con il timer corretto e infine abilitare l'interrupt.

Per trasmettere è necessario mettere in SBUF0 il carattere che si vuole mandare, il micro provvederà a trasmetterlo e a trasmissione conclusa genererà un'interrupt settando la flag TI (trasmissione completata). Per quanto riguarda la ricezione invece, il programma entra in interrupt con la flag RI (ricezione completata) dopo aver ricevuto il bit di stop, e mette a disposizione il carattere ricevuto nello SBUF0.

1.3.1 Seriale in C

In una prima fase del programma il micro trasmetteva un messaggio di benvenuto, nella seconda fase entrava in modalità eco, quindi ad ogni carattere che gli si mandava lui trasmetteva lo stesso carattere. Infine nell'ultima fase memorizzava una serie di caratteri (massimo 10) e dopo un certo carattere("#") ritrasmetteva la stessa serie. Per realizzare ciò è stata implementata la funzione scelta che a seconda del valore di status(0,1 o 2) svolge compiti diversi. Nella prima trasmette il messaggio a cui punta *puntatore* (nella prima fase è il messaggio di benvenuto ma successivamente diventa il messaggio trasmesso dall'utente al computer), nella seconda viene implementata la modalità eco e nella terza si riceve una serie di caratteri che viene memorizzata in un vettore. Se il messaggio mandato dall'utente è più lungo di 10 caratteri viene trasmesso un messaggio di errore.

Listing 1.10: Funzione scelta

```
void scelta(void){
    //controllo lo status
    switch(status){
        case(0):
            //trasmetto il messaggio
            i++;
            if (i<lenght_da_trasmettere){
                SBUF0=*(puntatore+i);
            }
            else{
                //ho finito di trasmettere entro in ricezione
                status++;
                i=0;
                REN0=1;
            }
            break;
        case(1):
            key=SBUF0; //ricevo, leggo SBUF0
            RIO=0;
            if(key=='#'){
                i=2;
                status++;
            }
            else{
                if(!loaded){
                    loaded=1;
                    SBUF0=key;
                }
            }
    }
}
```

```

        break;
    case(2):
        RIO=0;
        if(i<ML+3){
            key=SBUF0;
            //aggiungo il carattere letto al vettore
            msg_ricevuto[i]=key;
            i++;
            if(key=='#'){
                REN0=0; //smetto di ricevere
                status=0;
                puntatore=msg_ricevuto;
                msg_ricevuto[i-1]=CR;
                msg_ricevuto[i]=LF;
                lenght_da_trasmettere=i+1;
                i=0;
                SBUF0=*puntatore; //trasmetto messaggio
            }
        }
        else{
            REN0=0;
            i=0;
            status=0;
            puntatore=msg_errore; //trasmetto errore
            lenght_da_trasmettere=lenght_errore;
            SBUF0=*puntatore;
        }
        break;
    }
}

```

La funzione scelta viene chiamata solo dall'interrupt della UART, a prescindere che si sia finito di trasmettere o di ricevere.

Listing 1.11: Interrupt UART

```

void UART0() interrupt 4{
    if(TIO==1){
        //ho appena finito di trasmettere
        scelta();
        //dice che ha caricato l'ultimo carattere
        loaded=0;
        //resettare flag
        TIO=0;
        return;
    }
    else if(RIO==1){

```

```

        //ho appena finito di ricevere
        scelta();
        return;
    }
    else{
        //ho un problema
        return;
    }
}

```

Listing 1.12: Variabili globali

```

#define CR 13
#define LF 10
#define ML 10
uchar key;
uchar idata msg_trsm[]={ 'C', 'i', 'a', 'o', CR, LF};
uchar lenght_da_trasmettere = 6;
uchar idata msg_errore[]={ CR, LF, 'E', 'r',
'r', 'o', 'r', 'e', CR, LF};
uchar lenght_errore=10;
uchar i=0;
uchar * puntatore;
uchar idata msg_ricevuto[ML+4];
uchar status=0;
uchar loaded=0;

```

Il tutto inizia trasmettendo il primo carattere del messaggio di benvenuto, una volta che la trasmissione è stata completata si passa al carattere successivo e così via fino alla fine. Al termine di queste istruzioni il programma entra in un ciclo infinito ed esce solo quando l'utente trasmette dei caratteri, entrando nell'interrupt e quindi nella funzione scelta.

Listing 1.13: main

```

SP=(char)&stack;
Init_Device();
//inizia trasmissione msg di benvenuto
puntatore = msg_trsm;
SBUF0=*puntatore;
msg_ricevuto[0]=CR;
msg_ricevuto[1]=LF;
while(1);

```

1.3.2 Seriale in assembly

Questo semplice programma, è una piccola variante di quello precedente in cui si vuole imparare a gestire un vettore in Assembly. Infatti ciò che si definisce è un vettore nella memoria programmi, questo perché il vettore non verrà modificato ma solo letto, contenente il messaggio di benvenuto. L'indirizzo del vettore viene poi passato, subito dopo la configurazione del micro, al Data Pointer. Viene successivamente implementata una funzione NEXT che salva nell'Accumulator il carattere successivo in modo tale che sia accessibile dal main del programma in c. Dunque nel programma in C basterà chiamare questa funzione e prendere il carattere contenuto in A e trasmetterlo.

Listing 1.14: Vettore in Assembly

```

VETTORE SEGMENT CODE
STACK SEGMENT IDATA
NEXT SEGMENT CODE
RSEG STACK
STACK2: DS 10H
RSEG VETTORE
VETT: DB 'Ciao',00H
RSEG NEXT
N:  MOV C,A,@A+DPTR
    RET
RSEG CODICE
START:
    LCALL Init_Device
    MOV DPTR, #VETT
    RET
END
```

Listing 1.15: codice in C

```

NEXT();
*puntatore =(uchar)ACC;
```

Capitolo 2

ARM

2.1 Introduzione ai micro ARM e CORTEX

2.1.1 Configurazione del micro

2.2 Seriale

L'esperienza della seriale è del tutto analoga a quella con l'8051, il programma è diviso in 3 fasi:

1. Nella prima fase si trasmette un messaggio di benvenuto al computer
2. Nella seconda fase il micro entra in modalità eco, ritrasmettendo indietro tutti i caratteri che gli si mandano
3. Infine riceve una serie di caratteri memorizzandoli e poi rimandando il messaggio ricevuto

Per la USART è stata usata la USART2 del micro abilitandola dal STM32Cube e se ne è abilitato l'interrupt nel NVIC(Nested Vector Interrupt Controller), questo perché la USART2 è già implementata nell'STLink quindi la comunicazione avviene tramite il cavo mini-USB.

Un'ulteriore differenza con l'8051 è che il buffer per la trasmissione e per la ricezione sono distinti(il che probabilmente rende più chiaro anche il codice) e sono rispettivamente *TDR* e *RDR*.

Si è creato un file `_condiviso.h` come header sia per il file `main.c` sia per il file che gestisce gli interrupt(`stm32f0xx_it.c`). Esso contiene alcune definizioni importanti per entrambi i codici e permette il dialogo tra i due file, come ad esempio *puntatore* che viene usata da entrambi.

Listing 2.1: Header dei files

```
#define CR 13
#define LF 10
```

```

#define ML 10
extern unsigned char * puntatore;
extern unsigned char msg_ricevuto[ML+4];
extern int lenght_da_trasmettere;
extern unsigned char messaggio_benvenuto[7];
extern int lenght;
extern unsigned char msg_errore[10];
extern int lenght_errore;
extern int status;
extern int i;
extern int loaded;

```

Quelle riportate di seguito sono le inizializzazioni delle variabili precedenti nel main.

Listing 2.2: Inizializzazione variabili

```

unsigned char * puntatore;
unsigned char msg_ricevuto[ML+4];
int lenght_da_trasmettere=6;
unsigned char messaggio_benvenuto[]={ 'C', 'i', 'a',
, 'o', CR, LF};
unsigned char;
msg_errore[]={CR, LF, 'E', 'r', 'r', 'o', 'r', 'e', CR, LF};
int lenght_errore=10;
int status = 0;
int loaded =0;
int i=0;

```

Il vero codice inizia dopo che è avvenuta la configurazione del micro, abilitando l'interrupt di trasmissione e ricezione della USART2. Successivamente si passa il primo carattere da trasmettere al registro TDR.¹

Listing 2.3: main

```

//reset del interrupt control register
//al bit transmission completed
USART2->ICR |= USART_ICR_TCCF;
//abilito l'interrupt per la trasmissione
USART2->CR1 |= USART_CR1_TCIE;
//abilito l'interrupt per la ricezione
USART2->CR1 |= USART_CR1_RXNEIE;
//inizio a trasmettere il messaggio di benvenuto
msg_ricevuto[0]=CR;
msg_ricevuto[1]=LF;
puntatore= messaggio_benvenuto;

```

¹È necessario fare il reset del bit di transmission completed perché quando vengono abilitati gli interrupt il micro entra in interrupt e se questo bit fosse 1 lo interpreterebbe come primo carattere trasmesso

```
USART2->TDR=*puntatore;
```

Quando il micro finisce di trasmettere entra nell'interrupt con il bit di TC ad 1, a questo punto è necessario fare qualcosa e sarà la funzione scelta a determinare cosa.

Listing 2.4: gestione dell'interrupt

```
if((USART2->ISR & USART_ISR_TC) == USART_ISR_TC){  
    //resetto il flag di trasmissione completata  
    USART2->ICR |= USART_ICR_TCCF;  
    //allora ho appena finito di trasmettere  
    scelta();  
    loaded=0;  
    return;  
}  
else if((USART2->ISR&USART_ISR_RXNE)==USART_ISR_RXNE){  
    //allora ho appena finito di ricevere  
    scelta();  
    return;  
}
```

La funzione scelta è il cuore di questo programma. È l'evoluzione della stessa implementata per la seriale in C 1.10.

Listing 2.5: Funzione scelta

```
void scelta(){  
    unsigned char carattere;  
    switch(status){  
        case 0:  
            //inserisco nel buffer di trasm il carattere  
            i++;  
            if(i<lenght_da_trasmettere){  
                USART2->TDR=*(puntatore+i);  
            }  
            else{  
                status=1;  
                i=0;  
            }  
            break;  
        case 1:  
            //do  
            //inserisco nel buffer di trasmissione  
            //il carattere trovato nel buffer in ricezione  
            carattere = (char)USART2->RDR;  
            if(carattere=='#'){  
                i=2;  
                status=2;  
            }  
        }  
    }
```

```

    }
    else{
        if(!loaded){
            loaded=1;
            USART2->TDR=carattere;
        }
    }
    break;
case 2:
    //do

    carattere=USART2->RDR;
    if(i<ML+3){
        //aggiungo il carattere letto al vettore
        msg_ricevuto[i]=carattere;
        i++;
        if(carattere=='#'){
            //smetto di ricevere
            status=0;
            puntatore=msg_ricevuto;
            msg_ricevuto[i-1]=CR;
            msg_ricevuto[i]=LF;
            lenght_da_trasmettere=i+1;
            i=0;
            //trasmetto il messaggio ricevuto
            USART2->TDR=*puntatore;
        }
    }
    else{
        status=0;
        //trasmetto il messaggio di errore
        puntatore=msg_errore;
        lenght_da_trasmettere=lenght_errore;
        i=0;
        USART2->TDR=*puntatore;
    }

    break;
}
}

```


2.3 Timer7 e Led

2.4 I2C, termometro e display

Listing 2.6: Comunicazione seriale ARM