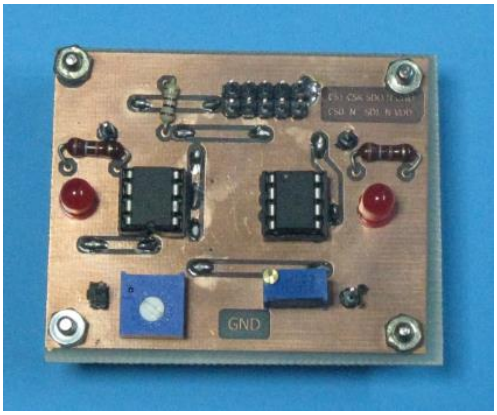
PROTOCOLLO SERIALE: SPI

BOARD PARTITORE DI TENSIONE PROGRAMMABILE



1. COME SI USA

La board consta di quattro trimmer resistivi, due digitali e due analogici, che partizionano la tensione di alimentazione. I due trimmer analogici sono un singolo giro da 10 KΩ e un multigiro da 5KΩ, mentre i due trimmer digitali sono due integrati della Microchip, uno da 10 KΩ e uno da 5 KΩ. Il collegamento della board alla scheda di sviluppo è molto semplice: si devono collegare tutti gli otto pin più a sinistra alla porta C del microcontrollore (o alternativamente alla porta B), e i restanti due pin a VDD e GND secondo la legenda indicata sulla board. Il collegamento corretto è mostrato in Figura 1 per una scheda di sviluppo DAM (Digital Analog Microcontroller), e in Figura 2 per una scheda di sviluppo EasyPIC. Ogni trimmer, digitale o analogico che sia, è dotato di un pin su cui prelevare l'uscita, posizionati in modo intuitivo nella board.

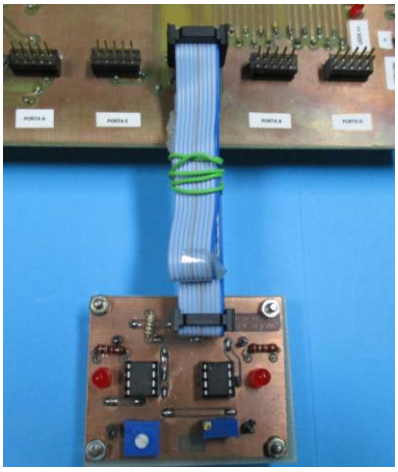


Figura 1. Esempio di collegamento a una scheda di sviluppo DAM

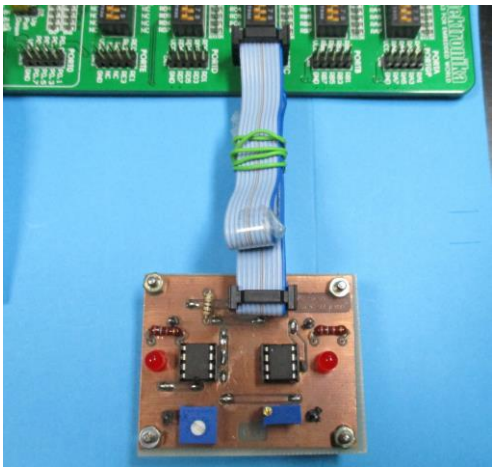


Figura 2. Esempio di collegamento a una scheda di sviluppo EasyPIC

Il controllo dei trimmer analogici è immediato; il controllo dei trimmer digitali avviene tramite protocollo seriale SPI. Un esempio di programma (main + alcune funzioni di libreria), che effettua un'impostazione del valore al POR dei dispositivi e ne effettua operazioni di Increment Wiper e decrement Wiper, per il microcontrollore PIC16F1789 è il seguente:

:MAIN

```
#include <pic16f1789.h>
#include <xc.h>
#include "xc8_header.h"
#include "libreria.h"


void main (void)
{
    configSPI ();

// impongo la posizione della resistenza NON VOLATILE del primo integrato all'accensione del dispositivo

    chipSelect0 = 0;            // abbasso la linea del chip select per dire che sto cominciando una trasmissione
    write16bitSPI(32, 0);       // 32 è il codice dell'istruzione che scrive in modo NON VOLATILE nella EEPROM, 0 è il valore che voglio scrivere
    chipSelect0 = 1;            // alzo la linea del chip select per dire che ho finito di trasmettere
    __delay_ms(400);

// impongo la posizione della resistenza NON VOLATILE del secondo integrato all'accensione del dispositivo

    chipSelect1 = 0;            // abbasso la linea del chip select per dire che sto cominciando una trasmissione
    write16bitSPI(32, 0);       // 32 è il codice dell'istruzione che scrive in modo NON VOLATILE nella EEPROM, 0 è il valore che voglio scrivere
    chipSelect1 = 1;            // alzo la linea del chip select per dire che ho finito di trasmettere
    __delay_ms(400);
```

```
  // effettuo due cicli basati su INCREMENT WIPER per entrambi i device,
  // il primo incrementa il wiper fino a fondo scala,
  // il secondo fa DECREMENT WIPER fino a fondo scala

    while(1){

        for(int i = 0; i<= 255; i++){

            chipSelect0 = 0;          // abbasso la linea del chip select per dire che sto cominciando una trasmissione
            write8bitSPI(4);          // 4 è il codice della istruzione INCREMENT WIPER
            chipSelect0 = 1;          // alzo la linea del chip select per dire che ho finito di trasmettere

            chipSelect1 = 0;          // abbasso la linea del chip select per dire che sto cominciando una trasmissione
            write8bitSPI(4);          // 4 è il codice della istruzione INCREMENT WIPER
            chipSelect1 = 1;          // alzo la linea del chip select per dire che ho finito di trasmettere
        }

        for(int i = 0; i<= 255; i++){
            chipSelect0 = 0;          // abbasso la linea del chip select per dire che sto cominciando una trasmissione
            write8bitSPI(8);          // 8 è il codice della istruzione DECREMENT WIPER
            chipSelect0 = 1;          // alzo la linea del chip select per dire che ho finito di trasmettere

            chipSelect1 = 0;          // abbasso la linea del chip select per dire che sto cominciando una trasmissione
            write8bitSPI(8);          // 8 è il codice della istruzione DECREMENT WIPER
            chipSelect1 = 1;          // alzo la linea del chip select per dire che ho finito di trasmettere
        }
    }
}
```

FUNZIONI DI LIBRERIA:

```
void configSPI (){

//----------CONFIGURAZIONE PIN-----------------

    #define chipSelect0 PORTCbits.RC0
    #define chipSelect1 PORTCbits.RC1

    TRISCbits.TRISC0 = 0;   // RC0 PER SS (o chip select) del primo integrto (SPI) (l'ho scelto io, non e' necessario che sia RC0)
    TRISCbits.TRISC1 = 0;   // RC1 PER SS (o chip select) del secondo integrato (SPI)
    TRISCbits.TRISC3 = 0;   // RC3 PER SCK (SPI)
    TRISCbits.TRISC5 = 0;   // RC5 PER SDO (SPI)
    TRISCbits.TRISC4 = 1;   // RC4 PER SDI (SPI) - NOT USED NOW

    ANSELC = 0;

    chipSelect0 = 1;      // il chip select deve stare sempre alto quando non si deve comunicare nulla
    chipSelect1 = 1;      // il chip select deve stare sempre alto quando non si deve comunicare nulla

//----------CONFIGURAZIONE MODULO MSSP-----------------

    SSPCON1bits.SSPEN = 1;   // abilita SSP (Syncronous Serial Port)
    SSPCON1bits.SSPOV = 0;   // azzera il lag dell'overflow
    SSPCON1bits.SSPM3 = 0;
    SSPCON1bits.SSPM2 = 0;   // questi 4 bit decidono in che modalita SSP si trova
    SSPCON1bits.SSPM1 = 0;   // (0000 indica modalita SPI MASTER con la divisione per il clock Fosc/4)
    SSPCON1bits.SSPM0 = 0;

    SSPSTATbits.BF = 0;      // azzero il flag che dice che il buffer e' pieno
    SSPSTATbits.SMP = 1;     // definisce quando il master deve campionare i dati (0 in mezzo alla trasmissione, 1 alla fine della trasmissione)

    SSPSTATbits.CKE = 1;     // controla quando il dato e' trasmesso rispetto al ck
    SSPCON1bits.CKP = 1;     // definsce come e' il segnale di clock in IDLE (1 e' alto in idle)
                             // questi 2 bit definiscono l' SPI MODE

}
void write16bitSPI (int MSB, int LSB){

    SSPBUF = MSB;              // scrivo nel registro buffer che si occupera' di passare il contenuto allo shift register
                               // 32 e' istruzione di write NV wiper
    while (SSPSTATbits.BF == 0){    // finche non ha finito la trasmissione non fare niente

    }

    SSPSTATbits.BF = 0;         // azzero il flag

    SSPBUF = LSB;                  //e' il dato che voglio scrivere in NV wiper (con 1 viene Rwb da 39.21 ohm)

    while (SSPSTATbits.BF == 0){    // finche non ha finito la trasmissione non fare niente
        __delay_ms(1);
    }

     __delay_ms(400);
     __delay_ms(400);

    SSPSTATbits.BF = 0;         // azzero il flag

void write8bitSPI (int BYTE){

    SSPBUF = BYTE;                // scrivo nel registro buffer che si occupera' di passare il contenuto allo shift register
                                  // 4 effetua un Increment Wiper

    while (SSPSTATbits.BF == 0){    // finche non ha finito la trasmissione non fare niente
       __delay_ms(1);
    }

    __delay_ms(100);

    SSPSTATbits.BF = 0;         // azzero il flag

}
```

Per il dettaglio sul programma vedi Paragrafo 4

2.1. DESCRIZIONE: HARDWARE

La board Partitore di tensione programmabile è composta dai seguenti **componenti**:

- 1 resistore da 100 Ω;
- 2 LED rossi e 2 resistori da 180 Ω;
- 2 integrati MCP4161-103/502;
- 1 trimmer da 10 KΩ;
- 1 trimmer da 5 KΩ multigiro.

La board presenta 10 **pin di ingresso** di tipo strip-line:

- 8 pin per l'implementazione del protocollo SPI, non realmente tutti usati (i pin 2-6-7 non sono usati);
- 2 pin per VDD e GND.

Lo schematico della basetta e il suo layout (privato dei piani di massa) sono mostrati rispettivamente in Figura 3 e Figura 4.

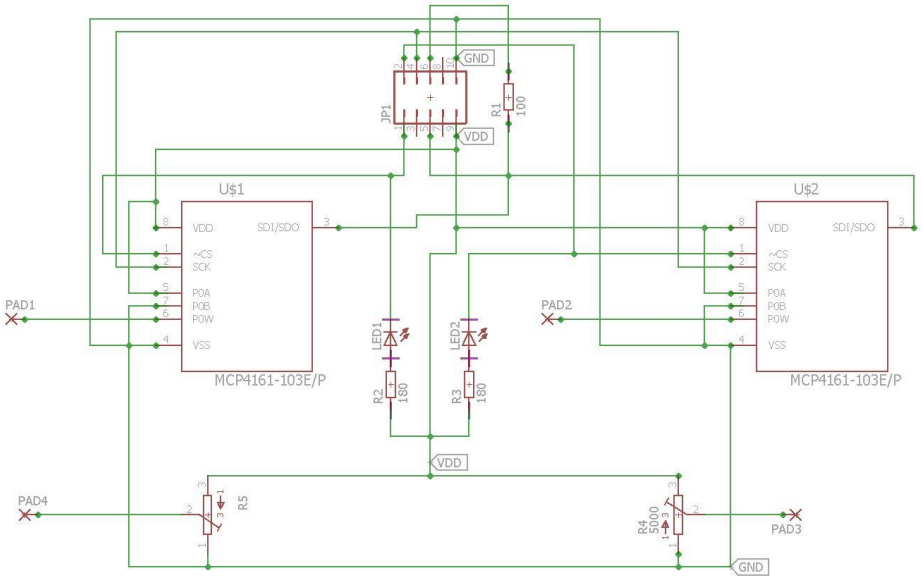


Figura 3. Schematic

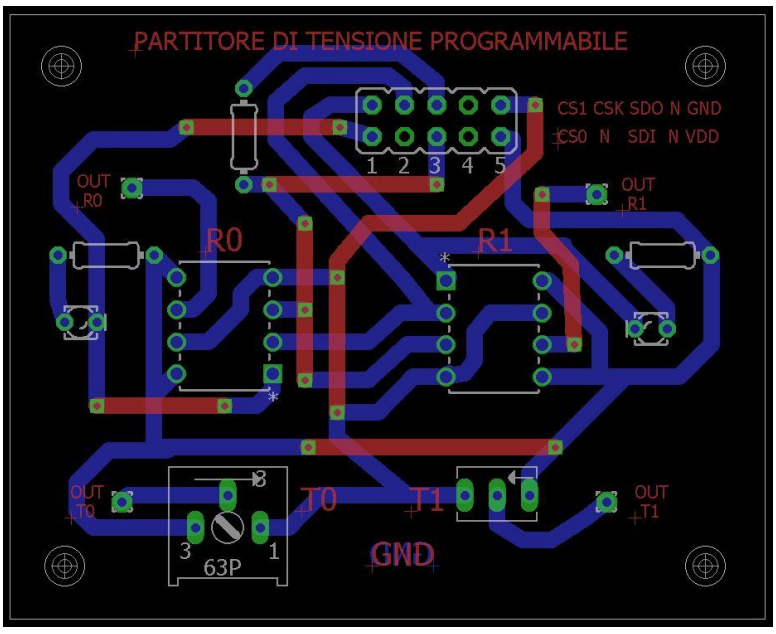


Figura 4. Layout (senza piani di massa): piste e componenti, Top (rosso), Bottom (blu)

2.2. DESCRIZIONE: TOP LAYER E BOTTOM LAYER

La board è stata progettata in dual-layer per riuscire a mantenere contenute le dimensioni. Il top layer (Figura 5) è stato utilizzato per la collocazione dei componenti, mentre le piste si trovano per lo più sul bottom layer (Figura 6). Su tutta la board il **piano di massa** sta alla tensione di riferimento GND; nel caso di un collegamento a coccodrillo è conveniente usare gli angoli della basetta (in particolare quelli a destra), perché altrimenti è probabile far contatto con alcune piste poste sul bottom layer.

I **connettori** strip-line (indicati in blu in Figura 5) sono tutti di tipo maschio e posti sulla parte superiore della board. I primi 8 pin sono quelli che vanno collegati alla porta del microcontrollore, i restanti due sono i pin di VDD e GND; i pin 2-6-7 non sono utilizzati. Il pin 5 è collegato, secondo lo standard SPI previsto dall'integrato della Microchip (per SPI vedi paragrafo 3.2), a un resistore da 100 Ω (indicato in giallo in Figura 5).

Le **uscite** di ogni trimmer sono indicate in Figura 5 con lo stesso colore con cui sono state indicate le rispettive resistenze variabili.

I **trimmer digitali** usati sono gli integrati MCP4161-103 e MCP4161-502 (indicati rispettivamente in arancione e rosso in Figura 5) che differiscono tra loro solo per la resistenza di fondo scala e conseguentemente per la minima risoluzione in termini di resistenza ($\frac{R_{fondo\ scala}}{256}$) che riescono a fornire: il -103 ha una resistenza di fondo scala da 10 KΩ e una minima risoluzione pari a 39 Ω, mentre il -502 ha una resistenza di fondo scala di 5 KΩ e una minima risoluzione pari a 19 Ω; entrambi sono dei dispositivi digitali a 8 bit. Gli integrati usati sono comunque posti su uno zoccolo saldato sulla board, per cui è possibile sostituirli in caso di guasto, o nel caso in cui si voglia un valore di resistenza non ottenibile con quelli presenti di default. Bisogna notare che, per facilitare i collegamenti sulla board, è stato necessario invertire la posizione di un integrato rispetto all'altro (le posizioni dei due integrati sono mostrate dalle frecce in Figura 5). I classici tre pin che caratterizzano un trimmer sono gli ultimi tre pin in basso a destra degli integrati (secondo il datasheet), mentre i restanti pin sono 2 di alimentazione, 3 per la comunicazione SPI.

I valori dei **trimmer analogici** utilizzati rispecchiano i valori utilizzati per i trimmer digitali, quindi il trimmer a sinistra (mostrato in verde scuro in Figura 1) è di 10 KΩ, come l'integrato di sinistra, mentre quello di destra (mostrato in verde chiaro in Figura 1) è di 5 KΩ, come l'integrato di destra. Il trimmer di sinistra è un singolo giro, mentre il trimmer di destra è un multigiro.

Nella board sono stati montati anche due **LED** (indicati in viola in Figura 5), con rispettive resistenze, per mostrare quando il microcontrollore sta dialogando con un dispositivo o con l'altro: i LED hanno infatti un terminale a VDD tramite la resistenza e un terminale collegato al rispettivo chip select, cosicché quando un chip select va basso (il CS è attivo basso), il LED si accende.

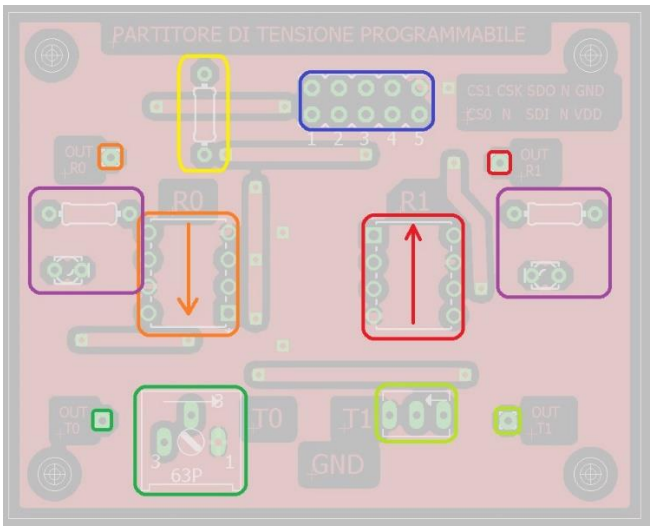


Figura 5. Top Layer

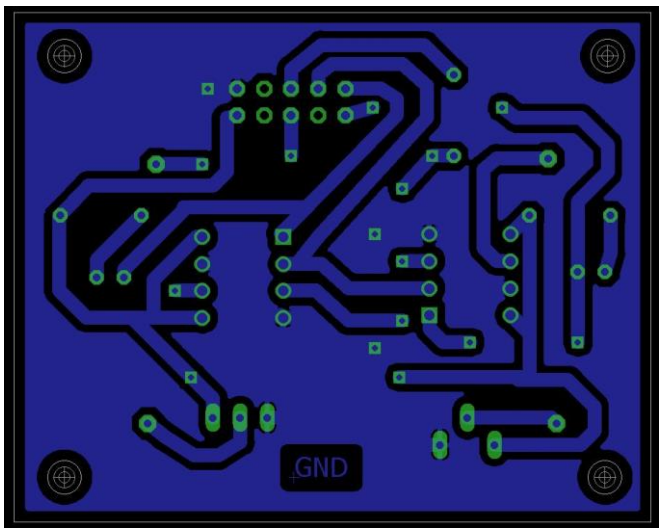


Figura 6. Bottom Layer (Mirrored)

3.1. COME FUNZIONA: LA BOARD

La board ha lo scopo di:

- mostrare l'analogia fra un resistore programmabile via software e un trimmer analogico: ciò che contraddistingue i due (dal punto di vista dell'utilizzatore) è per lo più il controllo del cursore, che da una parte è digitale, dall'altra è manuale (analogica);
- mostrare l'implementazione del protocollo seriale SPI.

3.2. COME FUNZIONA: MCP4161

La piedinatura del dispositivo è mostrata in Figura 7. Il dispositivo basa il proprio funzionamento su due elementi: la resistenza R_s che è la minima resistenza che si riesce a realizzare, e il Wiper che è il cosiddetto cursore che seleziona ora uno ora l'altro valore di resistenza desiderato dall'array resistivo (vedi Figura 8).

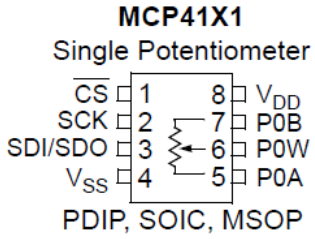


Figura 7. Piedinatura MCP4161

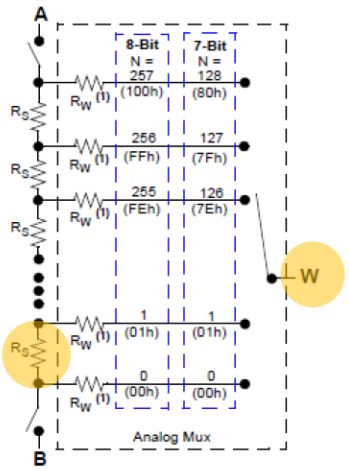


Figura 8. Elementi base del MCP4161

Tutte le istruzioni che vengono passate al dispositivo sono dirette per lo più al Wiper che cambia valore in relazione agli argomenti ricevuti. Le istruzioni possono essere di quattro tipi (riassunti brevemente nella Tabella 1 e molto più in dettaglio nella Figura 9). Il protocollo di comunicazione è SPI (vedi il prossimo paragrafo).

C1:C0 Bit States	Command	# of Bits	Operates on Volatile/ Non-Volatile memory
11	Read Data	16-Bits	Both
00	Write Data	16-Bits	Both
01	Increment (1)	8-Bits	Volatile Only
10	Decrement (1)	8-Bits	Volatile Only

Tabella 1. Tipi di istruzioni

Address		Command	Data (10-bits) (1)	SPI String (Binary)	
Value	Function			MOSI (SDI pin)	MISO (SDO pin) (2)
00h	Volatile Wiper 0	Write Data	nn nnnn nnnn	0000 00nn nnnn nnnn	1111 1111 1111 1111
		Read Data	nn nnnn nnnn	0000 11nn nnnn nnnn	1111 111n nnnn nnnn
		Increment Wiper	—	0000 0100	1111 1111
		Decrement Wiper	—	0000 1000	1111 1111
01h	Volatile Wiper 1	Write Data	nn nnnn nnnn	0001 00nn nnnn nnnn	1111 1111 1111 1111
		Read Data	nn nnnn nnnn	0001 11nn nnnn nnnn	1111 111n nnnn nnnn
		Increment Wiper	—	0001 0100	1111 1111
		Decrement Wiper	—	0001 1000	1111 1111
02h	NV Wiper 0	Write Data	nn nnnn nnnn	0010 00nn nnnn nnnn	1111 1111 1111 1111
		Read Data	nn nnnn nnnn	0010 11nn nnnn nnnn	1111 111n nnnn nnnn
		HV Inc. (WL0 DIS) (3)	—	0010 0100	1111 1111
		HV Dec. (WL0 EN) (4)	—	0010 1000	1111 1111
03h	NV Wiper 1	Write Data	nn nnnn nnnn	0011 00nn nnnn nnnn	1111 1111 1111 1111
		Read Data	nn nnnn nnnn	0011 11nn nnnn nnnn	1111 111n nnnn nnnn
		HV Inc. (WL1 DIS) (3)	—	0011 0100	1111 1111
		HV Dec. (WL1 EN) (4)	—	0011 1000	1111 1111
04h (5)	Volatile TCON Register	Write Data	nn nnnn nnnn	0100 00nn nnnn nnnn	1111 1111 1111 1111
		Read Data	nn nnnn nnnn	0100 11nn nnnn nnnn	1111 111n nnnn nnnn
05h (5)	Status Register	Read Data	nn nnnn nnnn	0101 11nn nnnn nnnn	1111 111n nnnn nnnn
06h (5)	Data EEPROM	Write Data	nn nnnn nnnn	0110 00nn nnnn nnnn	1111 1111 1111 1111
		Read Data	nn nnnn nnnn	0110 11nn nnnn nnnn	1111 111n nnnn nnnn
07h (5)	Data EEPROM	Write Data	nn nnnn nnnn	0111 00nn nnnn nnnn	1111 1111 1111 1111
		Read Data	nn nnnn nnnn	0111 11nn nnnn nnnn	1111 111n nnnn nnnn
08h (5)	Data EEPROM	Write Data	nn nnnn nnnn	1000 00nn nnnn nnnn	1111 1111 1111 1111
		Read Data	nn nnnn nnnn	1000 11nn nnnn nnnn	1111 111n nnnn nnnn
09h (5)	Data EEPROM	Write Data	nn nnnn nnnn	1001 00nn nnnn nnnn	1111 1111 1111 1111
		Read Data	nn nnnn nnnn	1001 11nn nnnn nnnn	1111 111n nnnn nnnn
0Ah (5)	Data EEPROM	Write Data	nn nnnn nnnn	1010 00nn nnnn nnnn	1111 1111 1111 1111
		Read Data	nn nnnn nnnn	1010 11nn nnnn nnnn	1111 111n nnnn nnnn
0Bh (5)	Data EEPROM	Write Data	nn nnnn nnnn	1011 00nn nnnn nnnn	1111 1111 1111 1111
		Read Data	nn nnnn nnnn	1011 11nn nnnn nnnn	1111 111n nnnn nnnn
0Ch (5)	Data EEPROM	Write Data	nn nnnn nnnn	1100 00nn nnnn nnnn	1111 1111 1111 1111
		Read Data	nn nnnn nnnn	1100 11nn nnnn nnnn	1111 111n nnnn nnnn
0Dh (5)	Data EEPROM	Write Data	nn nnnn nnnn	1101 00nn nnnn nnnn	1111 1111 1111 1111
		Read Data	nn nnnn nnnn	1101 11nn nnnn nnnn	1111 111n nnnn nnnn
0Eh (5)	Data EEPROM	Write Data	nn nnnn nnnn	1110 00nn nnnn nnnn	1111 1111 1111 1111
		Read Data	nn nnnn nnnn	1110 11nn nnnn nnnn	1111 111n nnnn nnnn
0Fh	Data EEPROM	Write Data	nn nnnn nnnn	1111 00nn nnnn nnnn	1111 1111 1111 1111
		Read Data	nn nnnn nnnn	1111 11nn nnnn nnnn	1111 111n nnnn nnnn
		HV Inc. (WP DIS) (3)	—	1111 0100	1111 1111
		HV Dec. (WP EN) (4)	—	1111 1000	1111 1111

Figura 9. Tutte le istruzioni che accetta MCP4161

3.3. COME FUNZIONA: SPI

SPI (Serial Peripheral Interface) è un protocollo seriale usato in ambito industriale. Basa la propria architettura sul concetto di master e slave, ed è frequente la situazione in cui più slave sono collegati ad un solo master (in questo circuito ci sono due slave collegati allo stesso master).

Il protocollo consta di quattro pin:

- CS o SS (Chip Select o Slave Select), tramite il quale il master seleziona lo slave con il quale vuole dialogare in quel preciso istante (è quindi una sorta di segnale di "enable" per la comunicazione con uno slave); è tipicamente attivo basso ma può esserci qualche situazione che non segue lo standard;
- SCLK (Serial CloCK), tramite il quale il master invia a tutti gli slave il clock di comunicazione; ATTENZIONE: non si tratta necessariamente del clock interno del dispositivo, ma solo del clock tramite il quale la comunicazione avviene;
- SDI (Serial Data Input), sul quale transitano i dati che vanno dal dispositivo al microcontrollore;
- SDO (Serial Data Output), sul quale transitano i dati che vanno dal microcontrollore al dispositivo.

Nel caso del dispositivo MCP4161 i pin SDI e SDO sono cortocircuitati internamente, per cui, per effettuare un corretto passaggio dei dati, è stato necessario introdurre un resistore (100 Ω) come indicato nel datasheet dell'integrato (Figura 10).

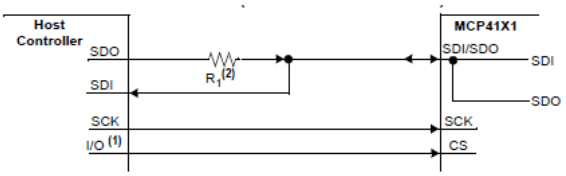
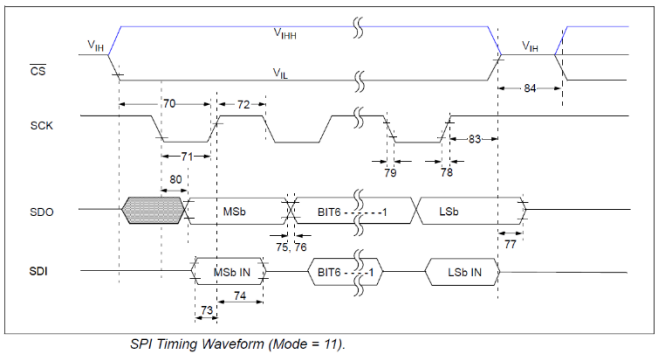


Figura 10. Collegamenti fra microcontrollore e MCP4161

Esistono quattro modalità di funzionamento del protocollo SPI, e differiscono fra loro in base alla temporizzazione dei dati e alla loro sincronizzazione con il segnale SCLK; il nome delle quattro modalità deriva dai due bit di configurazione del controller master che servono per settarle (CKE e CKP); il dispositivo (MCP4161) supporta sia la modalità "00" che la modalità "11" (che sono le due modalità più diffuse), e in particolare è stata scelta la seconda fra le due. Per il dettaglio sulla differenza fra le modalità "00" e "11" si veda la Figura 11.



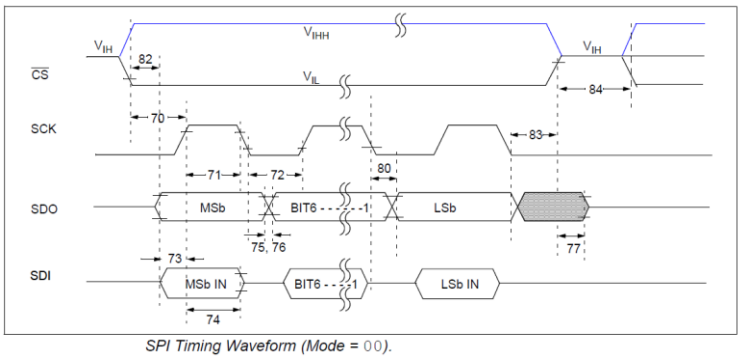


Figura 11. Differenza tra Mode = 00 e Mode = 11.

Per la corretta implementazione del protocollo si può ricorrere alla scansione manuale dei vari tempi che esso prevede, oppure utilizzare il modulo dedicato del microcontrollore MSSP (Master Synchronous Serial Port). Poiché la prima tecnica è ormai poco usata e anche molto difficile da implementare, è stata scelta la seconda. L'MSSP basa il proprio funzionamento su un buffer e uno shift register; settando correttamente i registri dell'MSSP, ci si deve

solamente preoccupare di inserire il dato sul buffer e aspettare che l'MSSP lo immetta, in modo seriale e con le corrette temporizzazioni, sul bus, facendo uso dello shift register sopra citato. Ci sono diversi flag che rendono intuitiva e semplice l'interfaccia con il modulo. Per maggiori dettagli riguardo la programmazione del modulo MSSP vedi paragrafo 4.

Per questa board tutti gli slave sono stati collegati al medesimo segnale SCLK, al medesimo segnale SDI e al medesimo segnale SDO, mentre sono stati utilizzati tanti pin di CS quanti sono gli slave (due). Esiste un'altra configurazione molto avanzata di SPI, che ha lo scopo di ridurre drasticamente il numero di pin utilizzati, chiamata Daisy Chain, che però non è oggetto di questa trattazione.

4. IL PROGRAMMA PIU' IN DETTAGLIO

Il programma proposto al paragrafo 1 fa variare periodicamente la resistenza di ciascun integrato, effettuando a ogni iterazione del ciclo un incremento del Wiper che, una volta arrivato al valore di fondo scala, scende nuovamente fino a raggiungere il valore di partenza; prima del ciclo su citato viene anche effettuata una variazione del valore al POR (Power On Reset) dei dispositivi, per vedere il cambiamento è quindi necessario dare al circuito un reset.

Le funzioni necessarie al funzionamento di SPI sono tre, si trovano nel file di libreria "libreria.h" e vengono gestite tutte all'interno del main.
La funzione **configSPI** serve a configurare sia i pin di uscita e di ingresso del microcontrollore, sia a configurare il modulo MSSP di cui si è discusso precedentemente. Per maggiori informazioni sui singoli registri dell'MSSP consultare il datasheet del PIC16F1789.
La funzione **write16bitSPI** manda una istruzione a 16 bit tramite il pin SDO del microcontrollore; essa accetta due parametri che non sono altro che i due pacchetti da 8 bit che compongono l'istruzione da 16 bit che deve essere inviata. Si noti che inizialmente essa immette il dato nel buffer (SSPBUF), e successivamente aspetta che il trasferimento sia finito controllando il bit BF (Buffer Full, se 0 il buffer non è vuoto e il dato non è stato ancora completamente inviato). Dopodiché si resetta il bit BF e si invia nuovamente allo stesso modo l'altra metà della parola a 16bit. Dopo aver aspettato un tempo sufficiente a far vedere lampeggiare il LED la funzione termina.
La funzione **write8bitSPI** manda una istruzione a 8 bit tramite il pin SDO del microcontrollore; essa funziona allo stesso modo della precedente funzione ma invia un solo pacchetto dati a 8 bit anzichè due.
Il **main** quindi inizialmente chiama la configSPI e setta il valore al POR dei dispositivi (32 è il codice istruzione 8 bit di write NON VOLATILE value, 0 è il valore 8 bit che voglio scrivere), e dopodiché si inizia un ciclo per effettuare una operazione di INCREMENT WIPER (codice istruzione da 8 bit: 4) su entrambi i dispositivi fino al valore di fondo scala; una volta raggiunto quest'ultimo si effettua lo stesso ciclo ma con operazioni di DECREMENT WIPER fino ad arrivare al valore di partenza. Si noti che prima di ogni operazione di scrittura sui dispositivi è necessario abilitare questi ultimi tramite il proprio Chip Select.