Davide Oldani (Galaxi93)

IMPARIAMO CON IL PIERIN - IL BUS SPI

20 July 2013

Premesse

Lo scopo di questo articolo è quello di riuscire a interfacciare un DAC della Maxim (MAX541) al PIERIN PIC18 via SPI. Con il programma di esempio, il PIERIN potrà generare 4 forme d'onda a 1KHz selezionabili mediante il tasto PL1. I due led indicano quale forma d'onda è stata selezionata.

Protocollo di comunicazione

Nella comunicazione SPI (Serial Peripheral Interface) sono presenti un unico master e un certo numero di dispositivi secondari detti slave connessi al Master mediante un bus a 4 fili su ci viaggiano i segnali necessari alla comunicazione SPI trai i vari dispositivi. L' SPI usa i seguenti segnali per serializzare lo scambio di dato con un altro dispositivo:

SS - Questo segnale è conosciuto come "Slave Select ". Quando passa a livello basso, il dispositivo slave si mette in attesa pronto ad ascoltare il segnali di clock e i dati.

SCK - Clock seriale generato dal master che controlla l'invio e la ricezione dei dati

SDO - Linea di uscita seriale dei dati di uscita da un dispositivo all'altro.

SDI - Linea di ingresso seriale dei dati in ingresso ad un dispositivo SPI

Lo scambio di dati è scandito da un segnale di temporizzazione detto clock presente sulla linea denominata con SCK. Il clock è generato dal MASTER che controlla quando i dati possono cambiare in uscita e quando possono essere letti. La sincronizzazione con il segnale di clock SCK dei dati in uscita sul bus SDO avviene in corrispondenza dei fronti di salita o di discesa del clock.

I registri di configurazione

Dato che la comunicazione SPI è generata dalla periferica MSSP la stessa che genera l'I2C, i registri sono gli stessi, ma i bit che li compongono hanno significato differente. La configurazione del SPI è molto più semplice e si basa su solo due registri:

REGISTER 20-5: SSPxSTAT: MSSPx STATUS REGISTER (I²C™ MODE) (1, ACCESS FC7h; 2, F73h)

R/W-1	R/W-1	R-1	R-1	R-1	R-1	R-1	R-1
SMP	CKE	D/$\overline{A}$	P(1)	S(1)	R/$\overline{W}$(2,3)	UA	BF
bit 7							bit 0

SSPxSTAT

Bit	Funzione
SMP	Configura se i dati in ingresso devono essere campionati a metà bit o alla fine
CKE	Indica se la trasmissione deve incominciare subito prima del segnale di clock o subito dopo
D/A	Usato solo in modalità I2C
P	Usato solo in modalità I2C
S	Usato solo in modalità I2C
R/W	Usato solo in modalità I2C
UA	Usato solo in modalità I2C
BF	Bit di sola lettura che indica se il registro SSPxBUF è pieno o vuoto

REGISTER 20-6: SSPxCON1: MSSPx CONTROL REGISTER 1 (I²C MODE) (1, ACCESS FC6h; 2, F73h)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
WCOL	SSPOV	SSPEN(1)	CKP	SSPM3(2)	SSPM2(2)	SSPM1(2)	SSPM0(2)
bit 7							bit 0

SSPxCON1

Bit	Funzione
WCOL	Serve per le operazioni di rilevamento delle collisioni tra le trasmissioni
SSPOV	Indica se è stato ricevuto un byte mentre il registro SPPxBUF era pieno
SSPEN	Attiva le porte SCKx, SDOx, SDIx
CKP	Configura se lo stato di riposo è a livello logico alto o basso
SSPM<3:0>	Sono 4bit che configurano la modalità del modulo SPI e la sua frequenza

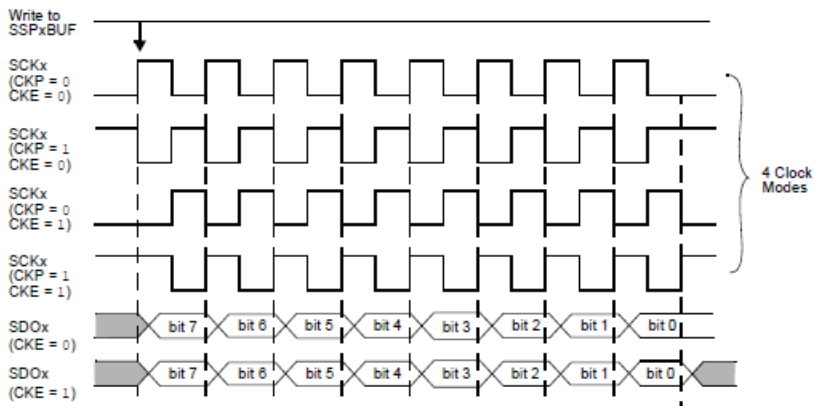
Il programma

Il programma sviluppato si compone di 4 file: header.h, configuration_bits.c, funzioni.c, main.c. La configurazione del modulo MSSP risulta essere molto semplice perché oltre ad esserci pochi bit da settare, il DAC in questione utilizza solo una linea

(solo la SDO). Quindi non è necessario configurare i bit che regolano la gestione dei dati in ingresso.

```
//Pin SPI
    TRISC7=0;
    TRISB4=0;
    PORTBbits.RB4=0;
    //Chip select
    TRISB0=0;
    //Inizializza il modulo MSSP in modalità SPI master: invio dati da condizione
    di riposo, condizione di riposo a livello basso e frequenza pari a Fosc/8
    SSP1STAT=0b01000000;
    SSP1CON1=0b00101010;
```

L'unica cosa che necessita di attenzione è la configurazione dei bit CKE e CKP. Come è possibile vedere dall'immagine sottostante, la comunicazione cambia molto in base a come vengono configurati. Per capire come configurarli basta leggere il datasheet del MAX541 a pagina 8, oppure confrontare le due figure che si fa prima :).



PIC

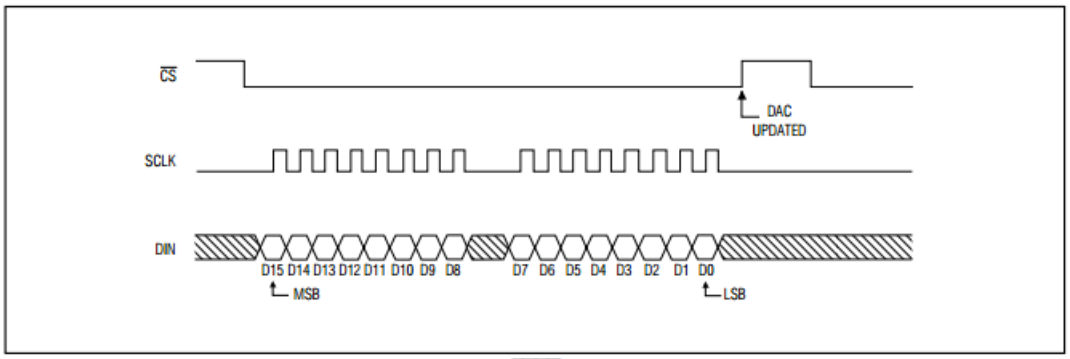
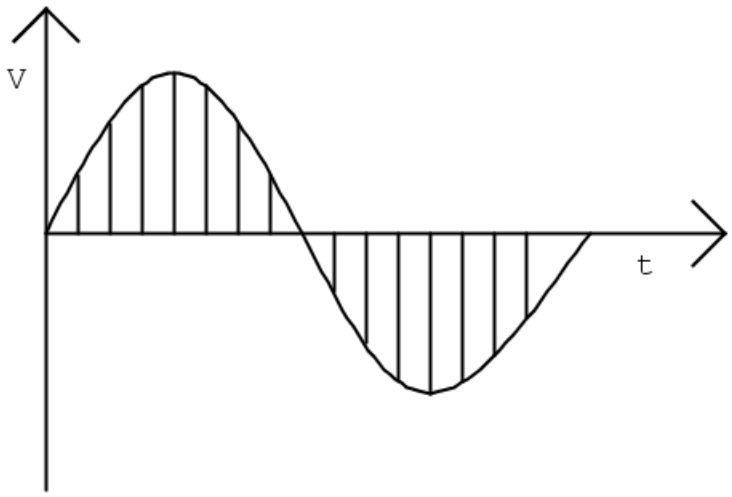


Figure 3a. MAX541/MAX542 3-Wire Interface Timing Diagram (LDAC = DGND for MAX542)

DAC

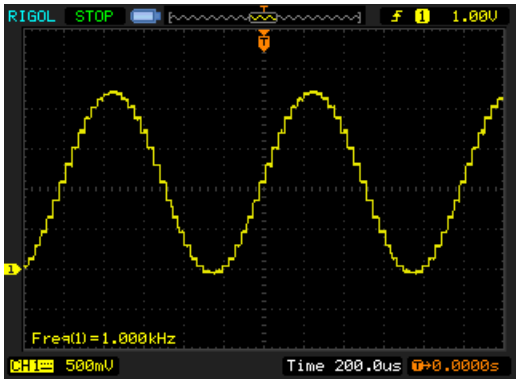
Confrontandole si deduce che la configurazione corretta è la seguente: CKE=1 e CKP=0 poiché il segnale di clock inizia a metà del primo bit inviato, passando da uno livello logico basso ad uno alto. Il tutto è comunque spiegato a parole sempre nel datasheet del convertitore.

Il resto del programma è abbastanza semplice ed è comunque ben commentato. Vale la pena però spendere qualche parola sulla generazione dei vari segnali. Infatti per poter generare le forme d'onda, ho fatto uso di "tabelle" in cui sono presenti i valori da 0 (0V) a 65535 (2,5V) che il segnale deve assume nei vari istanti. Queste tabelle non sono altro che degli array di valori da 32 posizioni. In pratica, prendendo la funzione seno nell'intervallo 0-2π, è come se la dividessi con 32 rette verticali e misurassi l'ampiezza che la curva assume in corrispondenza di ogni retta. Ecco un immagine che esemplifica quanto detto:



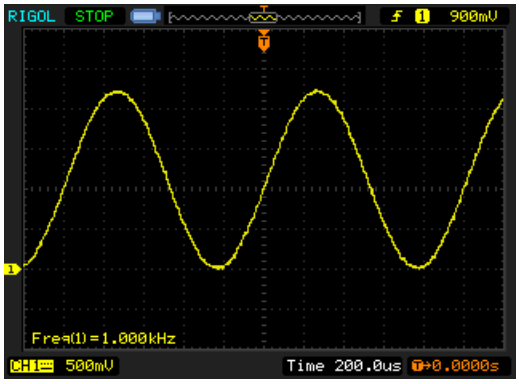
ONDA

Questa però è un' approssimazione della curva, perché il passaggio tra un valore e un altro della mia tabella avviene in modo simultaneo, e non gradualmente come la funzione originaria (senx). Infatti questo è il risultato in uscita dal nostro DAC:

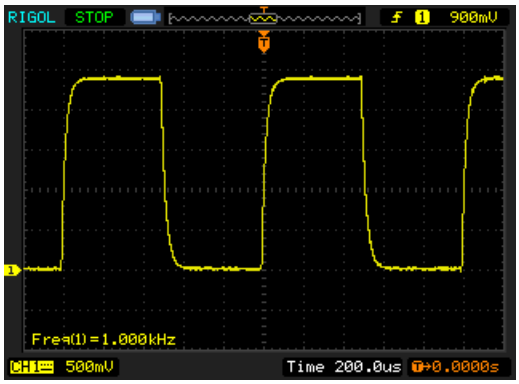


SIN grezza

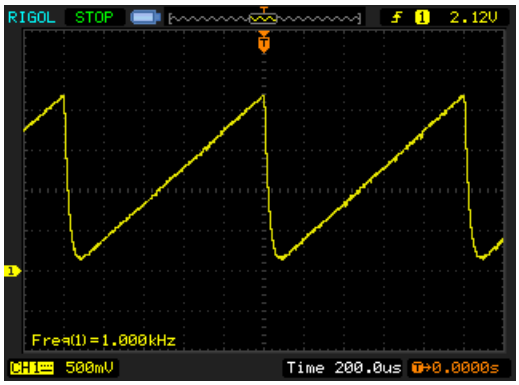
Non è proprio il massimo... Per ottenere risultati migliori si dovrebbe aumentare la risoluzione della tabella usata ovvero il numero di valori che contiene. Ad ogni modo si ottengono buoni risultati applicando un filtro passa basso in uscita dal DAC che mi tagli le frequenze superiori a quella generata. Ecco le varie onde generate e presenti dopo il filtro:



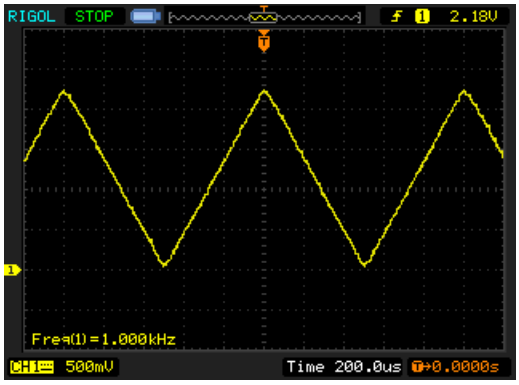
Sinusoide



Quadra



Dente di Sega



Triangolare

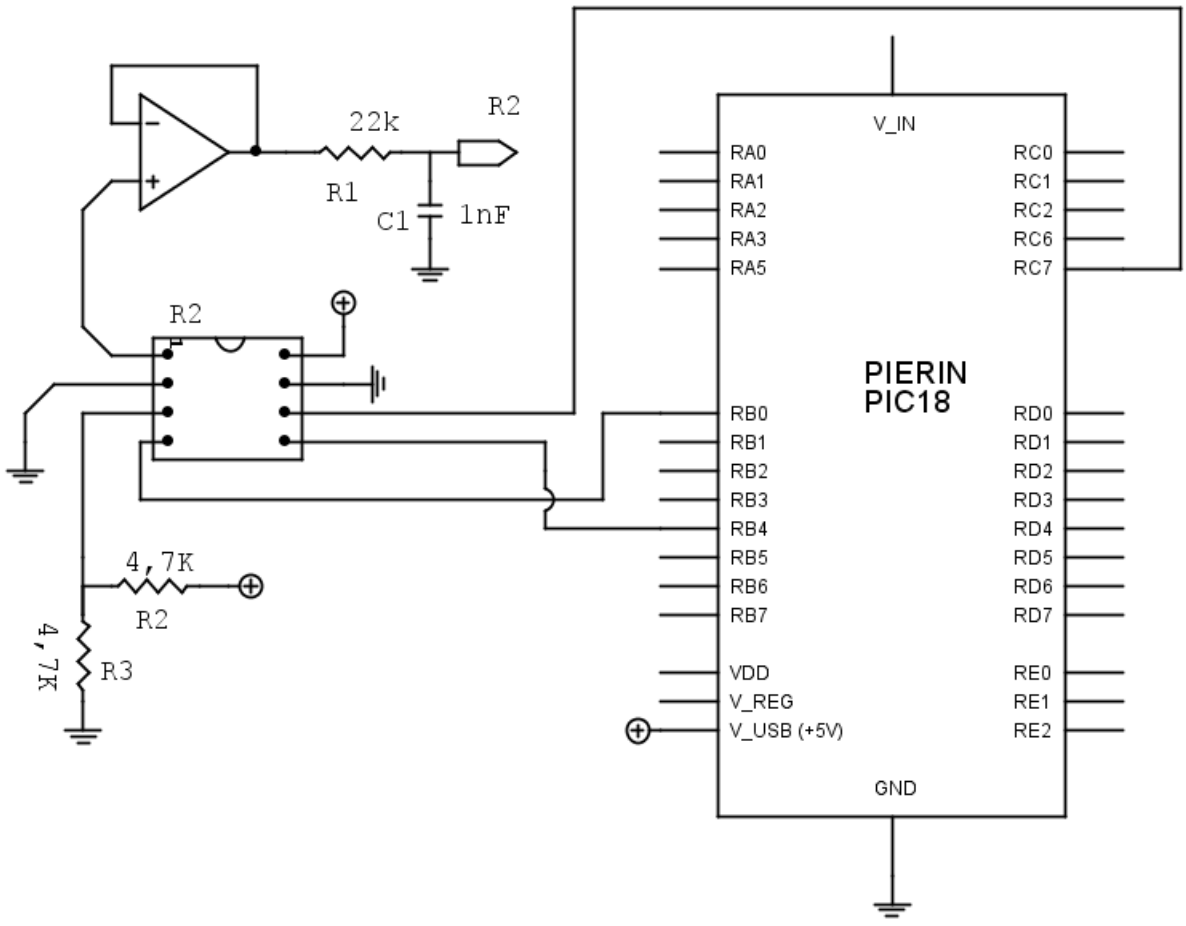
I valori presenti nelle tabelle vengono passati al DAC mediante questa funzione:

```
//Setta la tensione in uscita del DAC
void imposta_DAC(int valore)
{
    DAC=0;                      //Indica allo slave che i dati in arrivo sono indirizza
    SSP1BUF=valore>>8;          //Invio  dei bit più significativi
```

```
    while(!SSP1IF);             //Attesa e reset    S
    SP1IF=0;
    SSP1BUF=valore & 0x00FF;    //Invio bit meno significativi
    while(!SSP1IF);             //Attesa e reset
    SSP1IF=0;
    DAC=1;                      //Fine comunicazione
}
```

Per ottenere una frequenza in uscita, devo passare questi valori a intervalli regolari. In particolare, dato che il periodo di ogni funzione (tranne l'onda quadra) è composto da 32 valori, per ottenere una frequenza di 1KHz devo richiamare la funzione imposta_DAC(int valore) a una velocità 32 volte superiore. Per ottenere ciò mi avvalgo dell'uso del timer 2, che è un timer a comparatore, molto comodo, il cui funzionamento e la configurazione è ben spiegata in questo articolo.

Schema elettrico



Schema

Il datasheet del convertitore consiglia di mettere sull'alimentazione dello stesso due condensatori: uno elettrolitico da 10uF e uno poliestere/ceramico da 10nF.

N.B:I due dispositivi sono alimentati a tensioni differenti, il PIC a 3,3V e il DAC a 5V. Nonostante ciò, è possibile interfacciarli direttamente senza l'uso di nessun traslatore logico. Questo perchè i pin del PIC sono 5V-tolerant e gli ingressi del MAX541 considerano come stato logico alto, una tensione superiore a 2,4V.

Conclusioni

Qui potete scaricare il programma completo. Se avete domande o ho commesso qualche errore, non esitate a commentare l'articolo. Buona sperimentazione!