Fabio Bicciato (c1b8)

PIERIN COME MICRO GENERATORE DI SEGNALI

13 September 2013

Introduzione

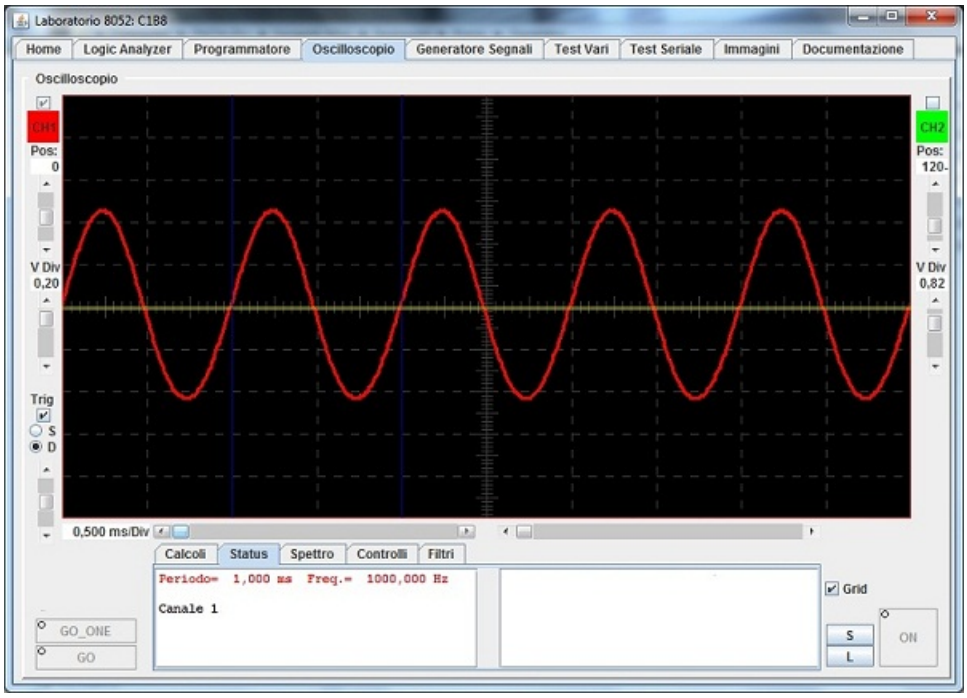
In questo secondo articolo della serie "Pierin come Micro strumentino" proviamo a realizzare un semplice generatore di segnali. Andremo ad implementare un generatore di onde sinusoidali, triangolari ed a dente di sega.

In questo articolo si farà uso di uno dei PWM interni al PIC come convertitore DAC e della tecnica DDS, implementata nel firmware, per generare le diverse forme d'onda. Vedremo come sia semplice implementare altre forme d'onda.

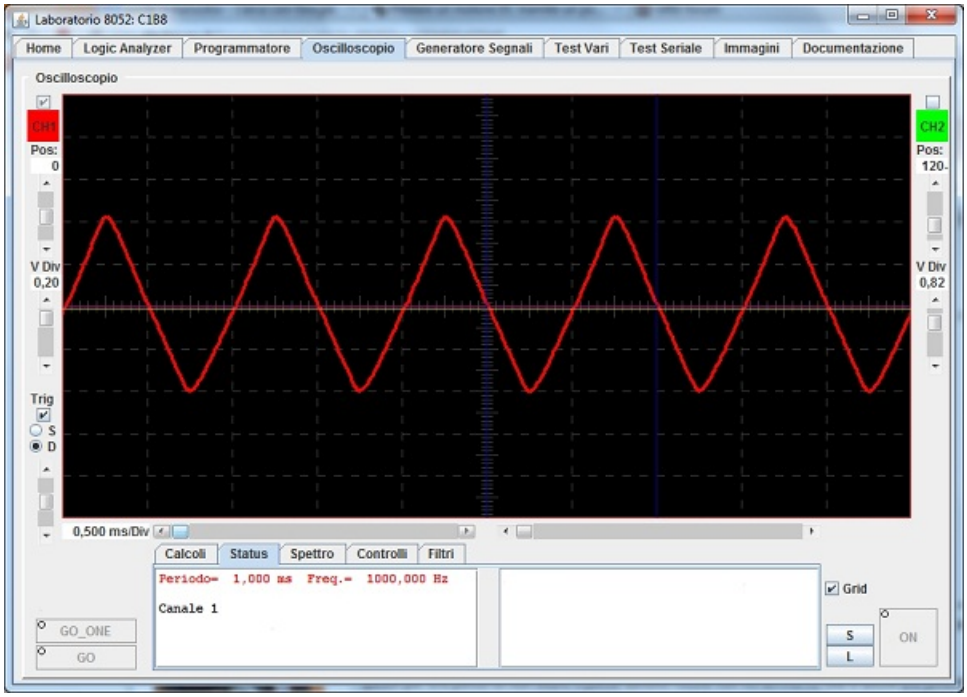
Anche in questo caso si è cercato di utilizzare solo la schedina Pierin PIC18 collegata ad un PC. In realtà si è reso necessario aggiungere un filtro Passa-Basso per la realizzazione del convertitore DAC.

Da PC è possibile selezionare la forma d'onda da generare e la frequenza del segnale stesso. Frequenza che può variare da 1 Hz a 20 kHz con risoluzione di 0,1 Hz. Non è possibile variare l'ampiezza in tensione del segnale generato. La scelta di non impiegare un DAC esterno, unita alla "scarsa" efficienza del filtro realizzato e alla bassa frequenza di campionamento, rendono tuttavia già a partire da 10 kHz di frequenza i segnali generati non molto "buoni".

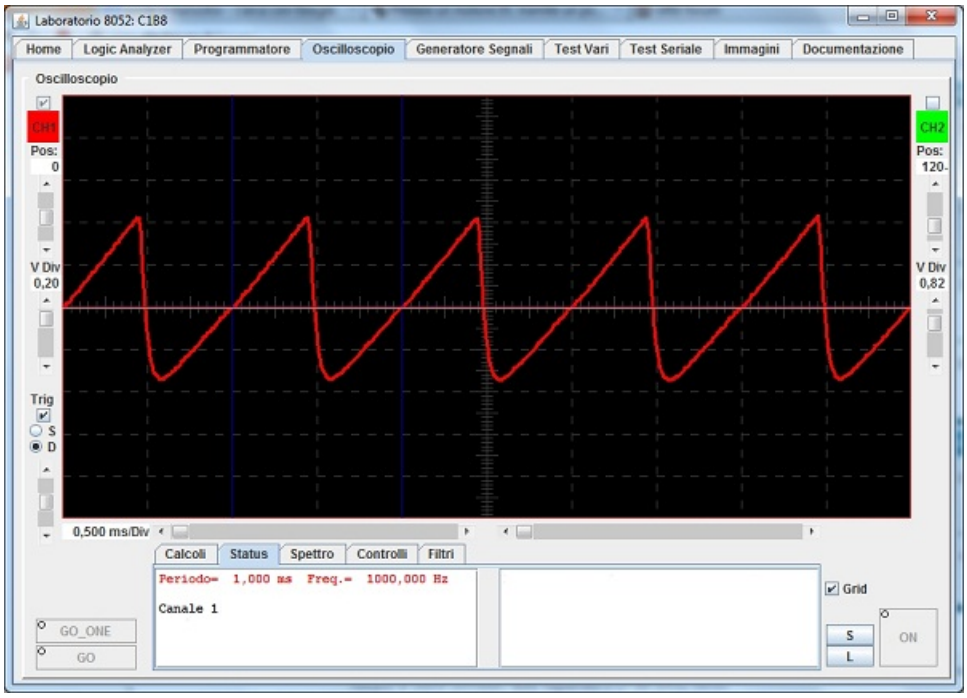
Ecco alcune immagini di esempio dei segnali generati.



Sinusoide 1kHz



Triangolo 1kHz



Sega 1kHz

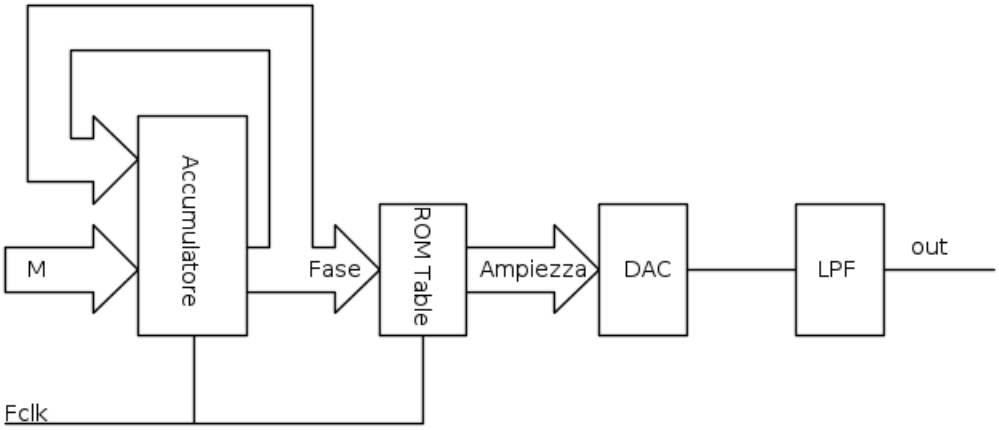
DDS: Direct Digital Synthesis

Il sistema utilizza la tecnica DDS per generare le varie frequenze in uscita. L'impiego della DDS consente una variazione delle frequenze del segnale molto rapida, con ottima precisione ed è realizzabile con poche righe di codice.

Una ottima presentazione della tecnica DDS la si può trovare in Questo ottimo articolo di brabus.

Riepilogo molto brevemente la tecnica stessa al fine solo di comprendere le parti di firmware scritte nel seguito.

Un sistema DDS può essere schematizzato come segue:



Accumulatore di fase

L'accumulatore di fase è un registro a N bit che viene incrementato, di un valore M, ad ogni ciclo di clock Fclk. Il valore in uscita all'accumulatore rappresenta il valore della fase del segnale che si desidera generare, tale valore dovrà essere convertito nel corrispondente valore di ampiezza.
Il valore di incremento M in ingresso all'accumulatore determina quindi lo step della fase e di conseguenza la frequenza del segnale di uscita. Semplicemente variando M è possibile ottenere valori di frequenza in uscita diversi.

La risoluzione di frequenza ottenibile in questo modo è dipendende dal numero di bit dell'accumulatore e dalla frequenza di incremento dello stesso (Fclk). Supponendo infatti di incrementare di un valore pari a 1 l'accumulatore ad ogni ciclo di clock, per un accumulatore di N bit, otterremo tutti i possibili valori di fase dopo:
$\frac{2^N}{F_{clk}} secondi$
ovvero con una risoluzione in frequenza pari a:
(1) $\frac{F_{clk}}{2^N} Hz$

da cui possiamo scrivere
(2) $F_{out} = M * \frac{F_{clk}}{2^N}$

E' facile dalla (2) determinare il valore di M da utilizzarsi per ottenere una qualsiasi frequenza Fout:
$M = F_{out} * \frac{2^N}{F_{clk}}$

Nel progetto è stato realizzato un accumulatore da 26 bit ed è stata utilizzata una frequenza di clock pari a circa 88235,2942 Hz, per una risoluzione massima

pari a 0,00131480834186077118 Hz. Il software PC consentirà però di selezionare frequenze a step di 0,1 Hz.

Convertitore Fase/Ampiezza (ROM Table)

Il secondo fondamentale componente di un sistema DDS è il convertitore Fase/Ampiezza. Questo componente riceve in input il valore di uscita all'accumulatore e lo converte nel corrispondente valore di ampiezza del segnale. E' stato realizzato come tabella contenente tutti i valori di un periodo completo del segnale da generare (ad esempio una sinusoide) memorizzata nella memoria programma del PIC. Avere tutti i valori di un periodo del segnale pre-calcolati in memoria consente un rapido accesso agli stessi (lookup table) e nessun calcolo matematico particolare.

Il convertitore Fase/Ampiezza potrebbe non ricevere in input tutti gli N bit dell'accumulatore ma solo una loro parte. Ad esempio con l'accumulatore del nostro progetto, a 26 bit (N=26), il numero di entrate della tabella sarebbe eccessivo per un PIC, in questi casi si utilizzano solo alcuni bit dell'accumulatore stesso (i più significativi).

L'uscita del convertitore Fase/Ampiezza, ancora digitale e di B bit, determina la risoluzione verticale del segnale.

Nel progetto è stata utilizzata una tabella di 1024 elementi (i 10 bit più significativi dell'accumulatore) con uscita ad 8 bit (1 byte) impegnando quindi 1kB di memoria per ogni forma d'onda da generare.

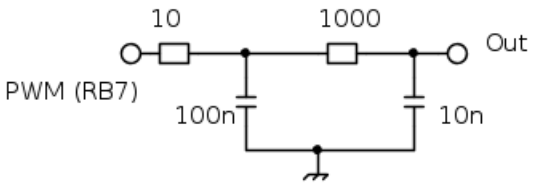
DAC e LPF

L'uscita digitale del convertitore Fase/Ampiezza viene convertita in analogico da un DAC. Nel progetto è stato utilizzato uno dei PWM interni al PIC, in combinazione con un filtro passa-basso (LPF), per limitare al minimo l'impiego di componenti aggiuntivi e consentire di verificare subito i risultati ottenuti da firmware. La scelta effettuata non è sicuramente la migliore, anche se la realizzazione di un DAC con PWM è un'ottima soluzione per segnali a bassa frequenza che consente di contenere i costi sia in componenti sia in spazio sullo stampato.
Per ottenere risultati migliori si potrebbe ad esempio inviare l'uscita del convertitore Fase/Ampiezza direttamente in uscita su una porta del PIC e realizzare un DAC con una rete R2R o impiegare un DAC esterno o ancora utilizzare un microcontrollore che integri al proprio interno un DAC.

La conversione da digitale ad analogico mediante PWM si ottiene modificando il valore del Duty Cycle del segnale ad onda quadra del PWM. Come descritto in questo documento, il valore della componente continua in uscita da un segnale PWM a frequenza costante è direttamente proporzionale al valore del Duty Cycle del segnale

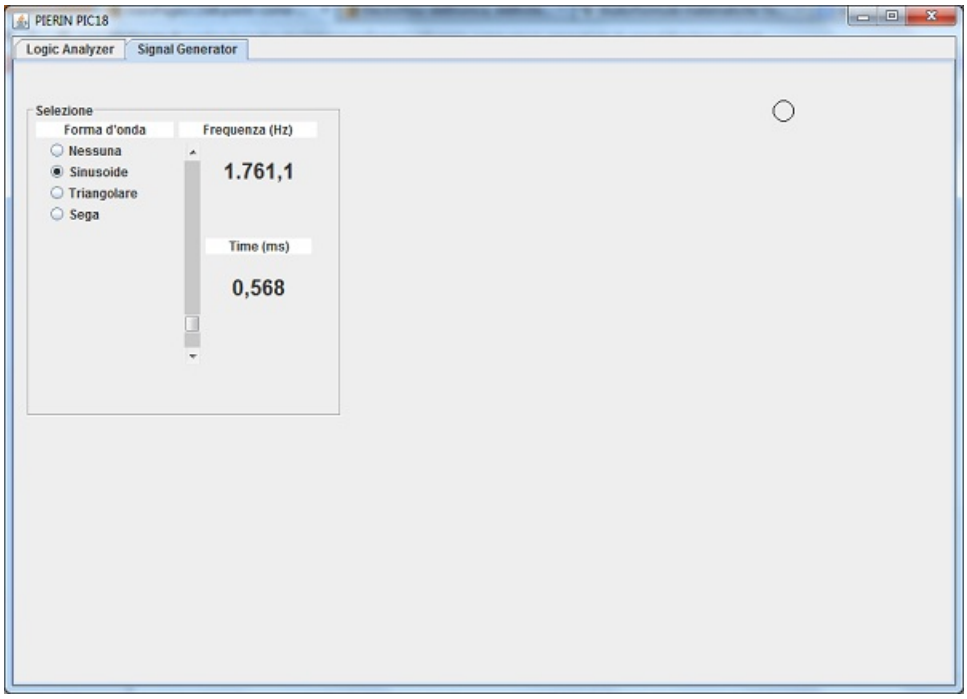
stesso. Un opportuno filtro in cascata consente di ridurre l'interferenza sul segnale dato dalla frequenza del PWM ed ottenere solo la componente continua proporzionale al valore del Duty Cycle impostato.

Nel progetto ho utilizzato il seguente filtro:



Software lato PC

Il software PC necessario alla selezione della forma d'onda desiderata e alla frequenza è stato integrato nel software del Logic Analyzer presentato in [questo articolo], al quale rimando per l'installazione. La nuova versione è scaricabile in allegato al presente documento.



PC.jpg

Come visibile dall'immagine l'interfaccia grafica è estremamente semplice.

Firmware PIC

Anche il firmware del PIC è stato integrato nel firmware del Logic Analyzer consentendo quindi di avere contemporaneamente nel PIC entrambi gli strumentini.

DAC

Per la realizzazione del DAC è stato utilizzato il modulo CCP7, uscita RB7 del Pierin, configurato come PWM utilizzando il Timer6 come temporizzatore. La frequenza del PWM scelta è la più alta possibile, per consentire al filtro LPF di meglio rimuoverla, ma tale per cui la risoluzione del Duty Cycle non scenda sotto gli 8 bit. Si è quindi utilizzato una frequenza pari a 176470,5882 Hz. La parte di codice che inizializza il PWM è la seguente:

```
        PR6 = 67;                // PWM7 =   5.66666us --> 176470,5882Hz
        T6CON = 0x04;                // Timer6 ON; 1/1  Prescaler

        CCPTMRS1 = 0x84;        // PWM4 <- TIMER2;   PWM5 <- TIMER4;    PWM7 <- TIMER6;
        CCPR7L = 0x20;                // DT PWM7 --> 50%
        CCP7CON = 0x0f;                // PWM Mode; DT PWM7
```

Fase/Ampiezza

Le tabelle per la conversione Fase/Ampiezza delle diverse forme d'onda sono state pre-calcolate con Excel, quindi salvate nella memoria programma del PIC in 3 array di char da 1024 elementi ciascuno: sineArray[], sawtoothArray[] e triangleArray[].
Per consentire una maggiore velocità di accesso ai valori stessi si è però preferito copiare la tabella della forma d'onda da utilizzare, sempre una per volta in funzione della richiesta del PC, in RAM.

```
void prepareGenBuffer(char type) {
        int p=0;
        waveFlag=0;

        switch (type) {

                case GEN_TYPE_SINE:
                        waveFlag=1;
                        for (p=0; p<BUFFER_SIZE; p++) {
                                setBufferChar(p, sineArray[p]);
                        }
                        break;
                case GEN_TYPE_TRIANGLE:
```

```
                        waveFlag=4;
                        for (p=0; p<BUFFER_SIZE; p++) {
                                setBufferChar(p, triangleArray[p]);
                        }
                        break;
                case GEN_TYPE_SAWTOOTH:
                        waveFlag=2;
                        for (p=0; p<BUFFER_SIZE; p++) {
                                setBufferChar(p, sawtoothArray[p]);
                        }
                        break;
                default:
                        for (p=0; p<BUFFER_SIZE; p++) {
                                setBufferChar(p, 128);
                        }
                        break;
        }
}
```

E' possibile generare un qualsiasi altro segnale periodico semplicemente introducendo la nuova tabella e prevedendone la copia in RAM ad opportuna richiesta del PC.
Ricordo che la tabella deve essere di 1024 elementi, in questi 1024 elementi deve essere codificato un ciclo completo del segnale e che i valori della tabella variano da 0 a 255 con il valore 128 corrispondente al valore centrale (equivalente ai 0V) in uscita al DAC.

Accumulatore di fase

Il valore utilizzato per l'incremento dell'accumulatore di fase viene calcolato ogni qual volta il PC invia la richiesta di cambio frequenza. Queste le istruzioni, molto semplici, per il calcolo:

```
/** CONSTANTS *****************************************************/
#define GEN_SAMPLE_RATE                         (float)88235.2942

#define GEN_TMR0_VALUE                                  145

// Accumulatore a 26bit
#define GEN_MAX_VALUE                                   0x04000000
#define GEN_FREQ_MULT                                   (float) GEN_MAX_VALUE/GEN_SAMPLE_R
```

```
        freq = *((float*)&ReceivedDataBuffer[2]);
        addValue = GEN_FREQ_MULT*freq;
```

Il "*cuore*" del sistema è realizzato all'interno della routine di interrupt per overflow del Timer0. Per avere una precisa frequenza di clock ho infatti utilizzato il Timer0 impostato in modo da generare overflow, con conseguente interrupt, ad una frequenza di 88235.2942 Hz circa. Questa frequenza è stata scelta in modo da poter essere la più alta possibile ma non superiore alla metà della frequenza del PWM: questo per consentire al PWM di presentare in uscita almeno 2 cicli con l'ultimo Duty Cycle impostato. Questa la routine di interrupt:

```
        void YourHighPriorityISRCode()
        {
_asm
                                // 6+10 = 16 Chiamata+Salvataggio
                                // 10+2 = 12 Ripristino+Ritorno

                                movff                WREG, WREGsave
                                movff                STATUS, STATUSsave
                                movff                FSR0H, FSRHsave
                                movff                FSR0L, FSRLsave
                                movff                BSR, BSRsave
                                btfsc                INTCON, 2, 0
                                BRA                 interrupt_tmr0

                                BRA                  interrupt_exit


interrupt_tmr0:

                                btfss                INTCON, 5, 0
                                BRA                  interrupt_exit

                                movlw                GEN_TMR0_VALUE
                                movwf                TMR0L, 0

                                movlb                accumulatore

                                bcf                 STATUS, 0, 0
                                movf                 addValue, 0, 1
                                addwfc                 accumulatore, 1, 1
                                movf                 addValue+1, 0, 1
```

```
                        addwfc          accumulatore+1, 1, 1
                        movf            addValue+2, 0, 1
                        addwfc          accumulatore+2, 1, 1
                        movf            addValue+3, 0, 1
                        addwfc          accumulatore+3, 1, 1

                        movlw           0x03
                        andwf           accumulatore+3, 1, 1
accumulatoreSend:
                        movf            accumulatore+2, 0, 1
                        movwf           FSR0L, 0
                        rlncf           accumulatore+3, 0, 1
                        iorlw           0x01
                        movwf           FSR0H, 0

                        bcf             ccp7conTemp, 5, 1
                        bcf             ccp7conTemp, 4, 1
                        bcf             STATUS, 0, 0

                        rrcf            INDF0, 0, 0
                        btfsc           STATUS, 0, 0
                        bsf             ccp7conTemp, 4, 1
                        bcf             STATUS, 0, 0
                        rrcf            WREG, 0, 0
                        btfsc           STATUS, 0, 0
                        bsf             ccp7conTemp, 5, 1

                        movlb           0xf
                        movff           ccp7conTemp, CCP7CON
                        movwf           CCPR7L, 1


interrupt_exit:
                        movff           BSRsave, BSR
                        movff           WREGsave, WREG
                        movff           STATUSsave, STATUS
                        movff           FSRHsave, FSR0H
                        movff           FSRLsave, FSR0L

                        bcf             INTCON, 2, 0

                        retfie          0
_endasm
```

```
}
```

Realizzata in assembler per consentire la maggiore velocità di esecuzione possibile, la routine prima di tutto si preoccupa di salvare i registri che andrà a modificare, quindi reimposta il timer0, incrementa l'accumulatore, recupera il dato dalla tabella di conversione Fase/Ampliezza ed imposta il Duty Cycle del PWM.
L'accumulatore è composto da 4 byte avendo scelto di implementare un accumulatore da 26 bit. Ho scelto un accumulatore a 26 bit per poter avere una risoluzione di almeno 0,1Hz e richiedere il minor numero di elaborazioni necessarie al fine di estrarre i 10bit più significativi necessari all'indirizzamento della tabella. 26 bit corrispondono infatti a 3 byte + 2 bit: i 10 bit si ottengono quindi utilizzando direttamente i 2 byte più significativi dei 4 che compongono l'accumulatore.

Download

Questi i link per il download di tutte le parti necessarie:

- Programma PC
- Sorgenti del Firmware
- HEX del Firmware

Ricordo che il firmware può essere trasferito al Pierin mediate il Bootloader. Per l'installazione e l'esecuzione del programma su PC si veda questo articolo.