Fabio Bicciato (c1b8)

PIERIN COME MICRO LOGIC ANALYZER

24 July 2013

Introduzione

Con questo articolo vorrei iniziare a proporre delle soluzioni per il Pierin PIC18 che trasformino la schedina, ormai in possesso di tutti, in un piccolo, direi microscopico, strumento da laboratorio. Nessuna pretesa nei risultati, alla fine si tratta solo di sperimentare codice ma con un fine che non sia puramente teorico. Questo primo articolo si pone come risultato la realizzazione di un piccolo Logic Analyzer.

Scelte

L'idea di base è quella di utilizzare il Pierin, un PC con qualsiasi sistema operativo, collegato al Pierin attraverso la USB, e nulla o pochissimo materiale aggiuntivo (solo qualche resistenza e/o condensato). Da qui le scelte obbligate (o quasi) quali:

- Il Pierin verrà visto dal PC come un dispositivo HID. Questo consente l'utilizzo dello stesso su ogni sistema operativo senza alcun driver aggiuntivo da installare.
- Il software lato PC sarà realizzato in Java (unico multi-piattaforma che io sappia utilizzare).
- Il firmware per il Pierin sarà sviluppato in modo da essere caricato sulla schedina attraverso il bootloader, non rendendosi necessario quindi un programmatore.
- E' stata scelta la porta D del micro in quanto tale porta è 5V tollerant.

Caratteristiche del Logic Analyzer

Le caratteristiche dello strumentino sono le seguenti:

- 4 Canali di input
- Buffer di 1024 acquisizioni
- Frequenza di campionamento massima di 1 MHz
- Possibilità di definire logiche di trigger che coinvolgono tutti i canali
- Posizione del trigger selezionabile tra inizio buffer (vengono memorizzate 50 acquisizioni pre-trigger), centro buffer (512 acquisizioni pre-trigger) o alla fine del buffer (quasi tutto il buffer contiene campioni pre-trigger)

La scelta di consentire la definizione di logiche trigger su qualsiasi canale, o combinazione di canali, e la scelta di gestire un buffer pre-trigger, hanno limitato la frequenza di campionamento a 1 MHz come frequenza massima.
Gestendo il trigger in un solo canale e non gestendo un buffer pre-trigger si potrebbe arrivare ad una frequenza di campionamento di 2 MHz senza difficoltà.
La decisione di utilizzare la porta D del micro per rilevare i segnali impone, essendo i pin da RD4 a RD7 utilizzati nel Pierin, il limite di 4 canali in input, sarebbe comunque possibile salire ad 8 canali (l'intera porta) senza modifiche al firmware del PIC.

Software lato PC

Cominciamo con l'installazione e la presentazione del software lato PC del nostro analizzatore.

Installazione

Innanzitutto dovete preoccuparvi di aver installato nel PC una versione di Java 1.6 o successiva (per scaricare o aggiornare Java: Java Download).

Dopo aver effettuato il download del software PC, è sufficiente scompattare il tutto in una qualsiasi directory.

Esecuzione del programma

Per mandare in esecuzione il software del Logic Analyzer è necessario eseguire il seguente comando:

java -cp hidapi-1.1.jar; Pierin

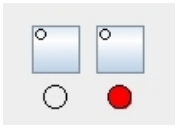
In Linux ho dovuto eseguire il comando come super user (ovvero **sudo java -cp hidapi-1.1.jar; Pierin**) in quanto non avevo altrimenti accesso in lettura/scrittura alle porte USB.

Il programma si presenta in questo modo:

Acquisizione_1.jpg

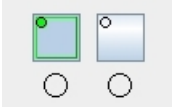
Il **LED contrassegnato dal numero (1)** indica lo stato di connessione della scheda Pierin al PC. Se il LED è acceso come in figura allora il Pierin è connesso e riconosciuto dal programma, in caso contrario il LED e tutti gli altri controlli saranno spenti/disabilitati.

I **Pulsanti contrassegnati dal (2) ed i LED (3)** rappresentano rispettivamente lo stato dei led e dei pulsanti del Pierin. Attraverso i pulsanti (2) è possibile accendere/spegnere i led della scheda, la pressione di uno dei pulsanti sulla scheda accende il corrispondente led (3).



PulsantePremuto.jpg

(Esempio di pulsante RD5 premuto sulla scheda Pierin).

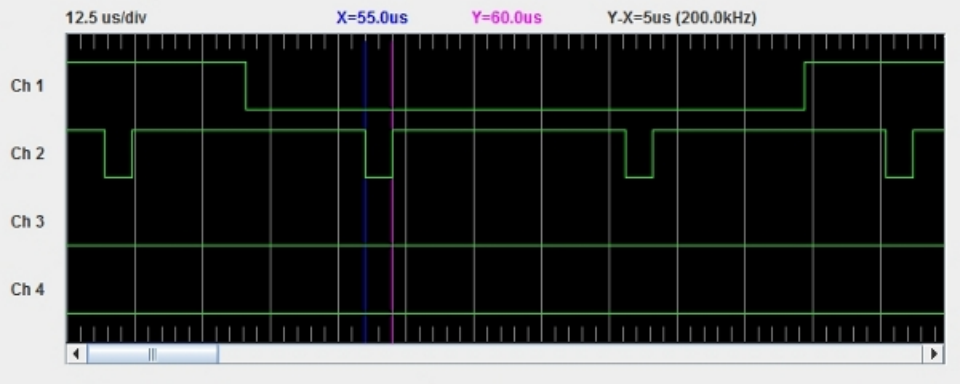


LedAcceso.jpg

(Esempio di accensione del led RD6)

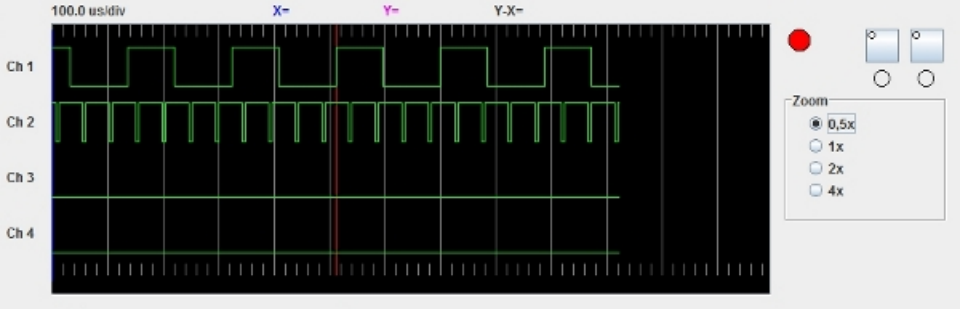
L'area contrassegnata come (**PANNELLO**) contiene la rappresentazione grafica dei segnali campionati in ingresso. Oltre alla griglia ed ai segnali campionati, sul Pannello trova posto una linea rossa verticale che indica il punto dove si è verificato l'evento trigger.
In questa area è inoltre possibile cliccare con il tasto sinistro del mouse (punto che sarà rappresentato da una linea verticale blu) e con il tasto sinistro del mouse (punto rappresentato da una linea verticale viola) per misurare l'intervallo di tempo che intercorre tra i due punti evidenziati dai click.
La zona (4), nella parte superiore del Pannello, riporta alcune informazioni necessarie a meglio interpretare i dati acquisiti.

- us/Div è il tempo rappresentato da una divisione del pannello, questo tempo è funzione della frequenza di campionamento e dello zoom impostato
- X=xxx, rappresentato in colore blu, indica il punto in us dove si è cliccato con il tasto sinistro del mouse.
- Y=yyy, rappresentato in colore viola, indica il punto in us dove si è cliccato con il tasto destro del mouse.
- X-Y=, rappresentato in colore nero, indica il delay time tra i punti X ed Y visti in precedenza. Nel seguente esempio si è voluto misurare la durata della parte bassa del segnale presente sul canale CH 2 (RD1). Tale durata è risultata essere di 5 us (pari a 200kHz).



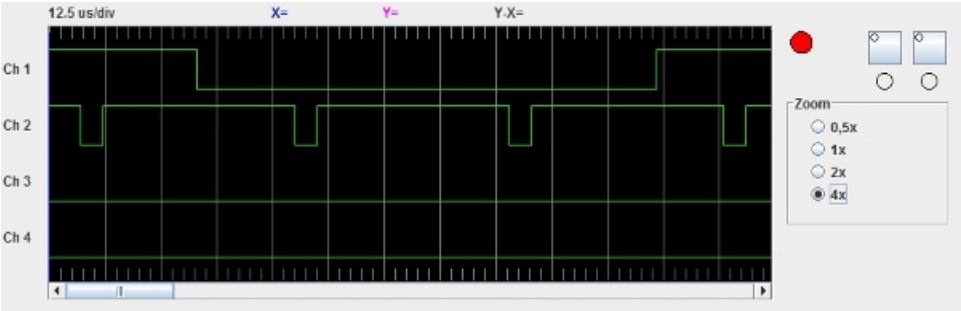
Intervallo.jpg

La zona (5) consente di selezionare lo zoom di visualizzazione dei dati nel Pannello.



Zoom05.jpg

(Esempio di zoom 0,5x)



Zoom4.jpg

(Esempio di zoom 4x).

Attraverso il **pannello Trigger (6)** è possibile indicare in quali stati debbano trovarsi i 4 canali in ingresso per determinare l'evento trigger. Le condizioni devono essere vere tutte contemporaneamente (sono verificate in AND). Per ogni canale è possibile indicare i seguenti valori:

- **_ - None** = Il canale non viene considerato al fine del trigger, Può assumere un qualsisi stato.
- **1 - Livello Alto** = Perchè scatti il trigger il canale deve trovarsi a livello alto (1 logico).
- **0 - Livello Basso** = Perchè scatti il trigger il canale deve trovarsi a livello basso (0 logico).
- **/ - Basso->Alto** = Perchè scatti il trigger nel canale deve verificarsi una transizioni del segnale da basso ad alto.
- **\ - Alto->Basso** = Perchè scatti il trigger nel canale deve verificarsi una transizioni del segnale da alto a basso.

Come già detto tutte le condizioni devono verificarsi contemporaneamente perchè scatti il trigger. Vediamo qualche esempio.

Trigger1.jpg

Impostando i trigger come in figura si indica che il trigger scatta nel momento in cui il canale 1 ha una transizione da livello Basso a livello Alto e contemporaneamente il canale 2 si trova in stato Alto. I canali 3 e 4 vengono acquisiti ma non influenzano il trigger.

Trigger2.jpg

In questo secondo esempio il trigger scatterà quando il canale 1 ha una transizione da Alto a Basso e contemporaneamente il canale 2 è Alto.

Almeno in un canale deve essere richiesta una trasizione da basso ad alto o viceversa. Attraverso il **pannello Acquisizione (7)** si indica la frequenza di acquisizione, il numero di campioni da acquisire pre e post trigger. Si può inoltre dare il via all'acquisizione.
I valori della frequenza di campionamento sono: 1 MHz, 500 kHz, 250 kHz, 100 kHz, 50 kHz, 25 kHz, 10 kHz e 5kHz.
I valori per il posizionamento del trigger sono:

- **Inizio buffer**. Saranno memorizzati 50 campioni di segnale prima del verificarsi le condizioni del trigger e 973 campionni successivi al trigger.
- **Fine buffer**. Saranno memorizzati 1000 campioni di segnale prima del verificarsi le condizioni del trigger e 23 campioni successivi.
- **Centro buffer**. Sarnno memorizzati rispettivamente 512 e 511 campioni prima e dopo il verificarsi del trigger.

E' possibile in questo modo analizzare i segnali sia dopo che immediatamente prima l'evento trigger.
Premendo sul pulsante Start si darà inizio all'acquisizione dei segnali. Il Pierin

attenderà l'evento trigger continuando a memorizzare i segnali in ingresso, solo al verificarsi del trigger saranno tornati i dati ed il controllo al programma PC. Se non dovesse verificarsi l'evento trigger è possibile interrompere l'acquisizione premendo sul pulsante RD5 presente sul Pierin. In questo caso sarà interrotta la procedura di acquisizione, il controllo torna al PC ma non saranno rappresentati dati nel Pannello.

Firmware per il Pierin

Come detto prima il Pierin si configura come dispositivo HID, la parte del firmware che realizza un dispositivo HID non è altri che l'esempio fornito dalla Microchip (Realizzazione di un HID Custom) ripulito delle parti non necessarie, ovvero di tutto tranne la gestione della USB.
L'esempio è fornito in C18 per MPLab ed ho mantenuto gli stessi strumenti di sviluppo.
Una volta ripulto al codice della Microchip ho inserito la vera e propia parte di acquisizione dei segnali che è stata realizzata in assembly. Questo per avere maggiore controllo dei tempi di esecuzione del programma. Per raggiungere una frequenza di campionamento di 1 MHz si hanno infatti a disposizione 12 cicli macchina, considerando che il PIC viene fatto lavorare con un clock a 48 MHz. Non volevo rischiare di lasciare al compilatore C l'onere di ottimizzare (e se poi non lo facesse adeguatamente?) inoltre devo essere sicuro che tra una lettura della porta D e la successiva trascorra sempre esattamente lo stesso tempo.
Brevemente alcune considerazioni e soluzioni adottate.

Il Buffer

Per consentire la memorizzazione di campioni pre e post trigger il codice deve acquisire in continuazione i segnali presenti sulla porta D, memorizzarli e verificare la condizione di trigger. Se si fosse deciso di non memorizzare la situazione pre-trigger sarebbe stato sufficiente leggere i dati presenti sulla porta D e verificare la condizione di trigger, senza memorizzarli e quindi senza impegnare memoria.
Non sapendo quanta memoria pre-trigger sarà necessaria è stato implementato il buffer da 1024 campioni come un buffer circolare nel quale il programma continua a salvare quanto acquisito. Il primo campione acquisito sarà salvato nella locazione 1 del buffer, il secondo nella 2, il 1024 campione alla locazione 1024, il 1025 campione nuovamente nella locazione 1, il 1026 campione nella 2 ecc. In questo modo è sufficiente sapere quanti campioni acquisire dopo l'evento trigger per essere certi di aver acquisito correttamente anche i campioni pre trigger.
Supponiamo infatti di voler acquisire 50 campioni pre-trigger, quindi 974 (1024-50=974) campioni post trigger:

- Il programma comincia ad acquisire (dovrà acquisire almeno 50 campioni prima di cominciare a verificare il trigger) e salvare i dati

- Si verifica il trigger, supponiamo che la condizione di trigger si presenti dopo 800 campioni acquisiti. Avremo che le prime 800 locazioni contengono i campioni pre-trigger.
- Il programma continuerà ad acquisire altri 974 campioni e li salverà a partire dalla locazione 801 del buffer, fermando l'acquisizione alla locazione 750 del buffer stesso (essendo il buffer circolare saranno infatti acquisiti 224 campioni prima di arrivare a fine buffer, quindi altri 974-224=700 campioni per completare l'aqcuisizione).
- Trasferendo quindi 1024 campioni al PC a partire dalla locazione 751 del buffer si garantisce il trasferimento di 50 campioni pre trigger (campioni presenti dalla 751 alla 800) e 974 campioni post trigger.

Scritto in C, la parte di acquisizione post trigger, potrebbe essere qualche cosa di simile:

```
// si arriva con puntatore = alla locazione successiva al trigger
// si arriva con acquisitionNumbers=974;
while (acquisitionNumbers>0) {
    buffer[puntatore]=PORTD;
    puntatore++;
    if (puntatore>=1024)
        puntatore=0;
    acquisitionNumbers--
}
```

Che sembrerebbe non poter mai stare dentro ai 12 cicli macchina massimi da noi richiesti. In realtà in assembly la circolarità del buffer e le considerazioni appena fatte sono garantite dalle seguenti istruzioni:

```
LoopLettura:
        MOVF        PORTD, 0, 0                     // buffer[puntatore]=PORTD; punta
        MOVWF        POSTINC0, 0                     //
        BCF          FSR0H, 3, 0                     // if (puntatore>=1024) puntator
        BSF          FSR0H, 0, 0                     //

        DCFSNZ        acquisitionNumbers, 1, 1      // acquisitionNumbers--
        DECFSZ        acquisitionNumbers+1, 1, 1
        BRA        LoopLettura                      // while (acquisitionNumbers>0)
```

che sono eseguite sempre in esattamente 8 cicli macchina, con qualche nop aggiuntiva si arriva ai 12 cicli richiesti
Viene utilizzato il registro FSR0 come puntatore al buffer, inizializzato a 0x0100 in modo da puntare al banco 1 della RAM, spostando un valore nel registro POSTINC0 il PIC salva quel valore nella locazione RAM puntata da FSR0 ed incrementa FSR0.

Il puntatore si incrementerà ad ogni salvataggio fino ad arrivare al valore 0x0200, le due istruzioni BCF e BSF, spegnendo il bit 11 del puntatore e accendendo il bit 8 dello stesso trasformeranno il 0x0200 in 0x0300 che verrà a sua volta incrementato ad ogni salvataggio fino ad arrivare a 0x0400. Ancora una volta le due istruzioni sui bit 8 e 11 trasformeranno il puntatore da 0x0400 a 0x0500 e cosi fino a diventare 0x0800, momento in cui le BCF e BSF riporteranno il puntatore al valore 0x0100, garantendo così la circolarità del buffer senza alcuna if.
Ecco che il nostro buffer di 1024 locazioni verrà distribuito in 4 banchi RAM del PIC: Banco 1, Banco 3, Banco 5 e Banco 7.
Non sono riuscito a riservare RAM a partire dal banco 0, ed ecco perchè l'impiego dei soli banchi dispari e di soli 1024 byte di buffer. Potendo utilizzare i banchi dallo 0 si poteva arrivare ad eseguire quel codice in 7 cicli macchina e si potevano acquisire fino a 2048 campioni.

Verifica trigger

Anche la parte di acquisizione/verifica trigger deve essere eseguita al massimo in 12 cicli macchina. Come per la parte appena vista vengono impiegati 4 cicli macchina per leggere la porta D, salvare il valore, incrementare il puntatore al buffer e garantire la circolarità del buffer. Ci rimangono 8 cicli per verifcare la condizione di trigger sui 4 canali.
La cosa si rivolve molto semplicemente pre-calcolando tre valori: maschera di trigger, valore ingressi prima del trigger e valori ingressi al trigger.
La **maschera trigger** viene utilizzata per eseguire una AND logica sul valore appena letto dalla porta D, questa AND forza a '0', per il solo controllo trigger, tutti i canali per i quali non si vuole effettuare controlli di trigger (i canali con il valore _ None).
Il **valore ingressi prima del trigger** rappresenta il valore che devono avere gli ingressi immediatamente prima si verifichi il trigger. Fino a quando non viene incontrato questo valore sugli ingressi non potrà mai verificarsi il trigger.
Il **valore ingressi al trigger** rappresenta il valore che devono avere gli ingressi al verificarsi del trigger.
Esempio:

- Si vuole controllare il canale 1 con una transizione basso -> alto
- Si vuole che il canale 2 abbia valore 1
- non si vuole controllare i canali 3 e 4.

Si dovranno utilizzare:

- Maschera Trigger = 0x03 (solo canale 1 e 2)
- Valore prima del Trigger = 0x02 (canale 1=0, canale 2=1)
- Valore al Trigger = 0x03 (canale 1=1, canale 2=1)

La sequenza di controllo sarà la seguente:# Acquisisci il valore della PORTD e salva il valore letto

1. Maschera i segnali da non controllare
2. Verifica con **valore ingressi prima del trigger**
3. Se non uguale torna a 1)
4. Acquisisci il valore della PORTD e salva il valore letto
5. Maschera i segnali da non controllare
6. Se ancora uguale a **valore ingressi prima del trigger** torna a 5)
7. altrimenti se uguale a **valore ingressi al trigger** comincia ad acquisire
8. altrimenti torna a 1)

Non riporto per semplicità tutta la parte di codice relativa a quanto appena detto, ma solo i punti da 1 a 4:

```
Trigger_B:        MOVF        PORTD, 0, 0
                  MOVWF       POSTINC0, 0
                  BCF         FSR0H, 3, 0
                  BSF         FSR0H, 0, 0
                  ANDWF        triggerMask, 0, 1

                  CPFSEQ       triggerBefore, 1
                  BRA       Trigger_B
```

Come vedete si esegue il tutto in 8 cicli quando non si verifca la condizione necessaria, 9 se si verifica la condizione richiesta.
Aggiungendo qualche nop e qualche test sul pulsante di interruzione forzata si arriva esattamente ai 12 cicli necessari.

Il codice assembly al completo

```
                  LFSR        0, 0x100

                  TSTFSZ       acquisitionNumbers, 1
                  INCF        acquisitionNumbers+1, 1, 1

                  TSTFSZ       firstAcquisition, 1
                  INCF        firstAcquisition+1, 1, 1

Lettura_Before:
                  MOVF        PORTD, 0, 0
                  MOVWF        POSTINC0, 0
```

```
                BCF          FSR0H, 3, 0
                BSF          FSR0H, 0, 0

                DCFSNZ        firstAcquisition, 1, 1
                DECFSZ        firstAcquisition+1, 1, 1
                BRA        LoopLettura_Before

                nop
                BRA         Trigger_B_Start

LoopLettura_Before:
                BTFSC   conDelay, 0, 1
                RCALL         DELAY_ACQ
                BRA        Lettura_Before


Trigger_B:
                BTFSS         PORTD, 5, 0
                BRA        fine_forzata

Trigger_B_Start:
                BTFSC   conDelay, 0, 1
Trigger_B_0:
                RCALL         DELAY_ACQ

Trigger_B_1:
                MOVF         PORTD, 0, 0
                MOVWF         POSTINC0, 0
                BCF          FSR0H, 3, 0
                BSF          FSR0H, 0, 0
                ANDWF         triggerMask, 0, 1

                CPFSEQ        triggerBefore, 1
                BRA        Trigger_B


                BTFSC   conDelay, 0, 1
                BRA        TriggerR_A

                nop
Trigger_A:
                nop
                nop
```

```
                MOVF        PORTD, 0, 0
                MOVWF        POSTINC0, 0
                BCF          FSR0H, 3, 0
                BSF          FSR0H, 0, 0
                ANDWF         triggerMask, 0, 1

                CPFSEQ        triggerBefore, 1
                BRA        Trigger_A_2

                BTFSC        PORTD, 5, 0
                BRA        Trigger_A

                BRA        fine_forzata

Trigger_A_2:
                nop
                CPFSEQ        triggerAfter, 1
                BRA        Trigger_B_1
                nop


Lettura:
                MOVF        PORTD, 0, 0
                MOVWF        POSTINC0, 0
                BCF          FSR0H, 3, 0
                BSF          FSR0H, 0, 0

                DCFSNZ        acquisitionNumbers, 1, 1
                DECFSZ        acquisitionNumbers+1, 1, 1
                BRA        LoopLettura

                BRA         fine_acquisizione

LoopLettura:
                BTFSC   conDelay, 0, 1
                RCALL        DELAY_ACQ
                BRA        Lettura



TriggerR_A:
                nop
                RCALL        DELAY_ACQ
```

```
                MOVF        PORTD, 0, 0
                MOVWF        POSTINC0, 0
                BCF          FSR0H, 3, 0
                BSF          FSR0H, 0, 0
                ANDWF         triggerMask, 0, 1

                CPFSEQ        triggerBefore, 1
                BRA        TriggerR_A_2

                BTFSC         PORTD, 5, 0
                BRA        TriggerR_A

                BRA        fine_forzata

TriggerR_A_2:
                CPFSEQ        triggerAfter, 1
                BRA        Trigger_B_0
                RCALL         DELAY_ACQ_1
                BRA        Lettura


                // 4 - chiamata+ritorno
                // 1 - nop
                // 2 - caricamento variabile temp + nop
                // se temp==1
                //                tempTotale = (7+2+3)/12 = 1 us
                //
                // se temp>1
                //                 12*temp - tempo ciclo
                //                 tempoTotale = (7+3+2+12*(temp-1))/12 us
DELAY_ACQ:
                nop
DELAY_ACQ_1:
                MOVF        timeLoop, 0, 1
                MOVWF        tempLoop, 1
                DECFSZ        tempLoop, 1, 1
                BRA        DELAY_ACQ_2
                nop
                nop
                nop
                nop
                return        0
```

```
DELAY_ACQ_2:
                nop
                nop
                nop
DELAY_ACQ_3:
                nop
                nop
                nop
                nop
                nop
                nop
                nop
                nop
                nop
                DECFSZ        tempLoop, 1, 1
                BRA        DELAY_ACQ_3
                nop

                return        0

fine_forzata:

                BRA     fine_asm

fine_acquisizione:
                MOVF        FSR0L, 0, 0
                MOVWF        bufferPunt, 1
                RRNCF        FSR0H, 0, 0
                ANDLW        0x03
                MOVWF        bufferPunt+1, 1

fine_asm:


_endasm
```

Test

Il PIC del Pierin viene configurato all'avvio del logic analyzer per generare su RB4 e RB5 2 segnali PWM di frequenza e duty cycle diversi tra loro. E' possibile collegare

tali pin agli ingressi del logic analyzer, ad esempio alla RD0 e RD1 rispettivamente, per eseguire qualche test senza necessità di circuiteria esterna.

Download e sorgenti

Questi i download per scaricare l'eseguibile Java, i sorgenti e l'.hex del firmware per PIC. A chi lo desidera posso inviare anche i sorgenti Java.

- Sorgenti del Firmware per PIC
- File .hex da caricare nel Pierin attaverso il Bootloader
- Programma per il PC