# *Materiale per attività aggiuntive*

## *Extra 1: Acquisizione di segnali da scheda didattica*

Ripetere le misure eseguite utilizzando l'oscilloscopio nella prima parte, con l'utilizzo del **Modulo DAQ** e di Matlab. Replicare quanto visto la volta scorsa con i seguenti accorgimenti:

1) Collegamenti diretti fili tra DAQ e scheda: consigliato collegare la scheda al modulo DAQ utilizzare le clip per assicurare il corretto contatto elettrico.

2) La configurazione della misurazione questa volta sarà differenziale. Collegare il cavo che si collegherà al punto di misura a uno degli ingressi analogici e la terra della scheda didattica a un'altro ingresso analogico.

Il campo ch.InputType sarà 'Differential'

**PUNTI DI MISURA 12 e 13:**

12 sinusoide a frequenza 1.1 kHz; l'ampiezza è regolabile agendo sul potenziometro posto vicino al punto di misura.

13 sinusoide simile alla precedente; è possibile variarne lo sfasamento rispetto a questa agendo sul potenziometro posto vicino al punto di misura.

Sfruttare questi segnali come "**test**" per apprendere l'acquisizione dei segnali dalla scheda al modulo DAQ.

Misurare periodo, calcolare il valore efficace, notate differenze rispetto alle misure effettuate con l'oscilloscopio?

**PUNTO DI MISURA 11:**

11 segnale sinusoidale con rumore; frequenza 1.1 kHz, ampiezza 1.4 V picco-picco.

Valutare come cambia l'acquisizione del segnale variando la frequenza di sampling.

Eseguire un confronto tra l'SNR calcolato sull'oscilloscopio precedentemente (partendo dal valore della fascia visualizzata con la modalità di persistenza, Vp-p circa 6*dev.standard rumore) e il valore di SNR che si può calcolare con la funzione di Matlab

snr(data)

dove data sarà il vettore contenente i dati acquisiti da modulo DAQ.

**PUNTO DI MISURA 1,2, 6, 9 e 10:**

1 onda quadra, frequenza fondamentale 500 kHz, ampiezza 3.7 V picco-picco.

2 onda quadra uguale alla precedente, ritardata di una frazione di periodo.

6 impulso periodico di durata 325 ns, periodo 6.4 ms

9 e 10 impulsi "ON DEMAND", durate circa 60 us

è possibile osservare questi segnali sfruttando le modalità a tempo finito regolando un tempo di osservazione lungo? Perchè? Che limiti riscontrate?

**PUNTI DI MISURA 3,4,5**

Valutare per ciascuno se esistono delle condizioni di Tw e s.rate che consentano l'acquisizione dei segnali e confrontare i risultati con quelli dell'oscilloscopio.

## *Extra 2: Acquisizione con modulo daq in modalita' background*

In base alle limitazioni riscontrate analizzando l'utilizzo dei moduli DAQ in foreground, provare ad implementare:

- ➔ Un'acquisizione tempo finito in background
- ➔ Un'acquisizione continua in background
- ➔ Un'acquisizione in background che si interrompa quando viene superato un limite di tensione.

**MATERIALE ADATTATO DA SITO UFFICIALE MATHWORKS, PER ULTERIORI INFO LINK UTILI:**

https://www.mathworks.com/help/daq/acquire-data-in-the-background.html

https://it.mathworks.com/help/matlab/matlab_oop/learning-to-use-events-and-listeners.html?searchHighlight=Events%20and%20Listeners%20%E2%80%94%20Concepts&s_tid=doc_srchtitle

al termine confrontare i risultati con i file contenuti nella cartella compressa sulla pagina Moodle: **Esempi_background_DAQ**

# a. Acquisire Dati in Background

A background acquisition depends on events and listeners to allow your code to access data as the hardware acquires it and to react to any errors as they occur. For more information, see Events and Listeners — Concepts in the MATLAB Object-Oriented Programming documentation.

Use events to acquire data in the background. In this example, you acquire data from an NI 9205 device with ID cDAQ1Mod1 using a listener and a `DataAvailable` event.

Listeners execute a callback function when notified that the event has occurred.
Use `Session.addlistener` to create a listener object that executes your callback function.

Create an NI session object and an analog input 'Voltage' channel on cDAQ1Mod1:

```
s = daq.createSession('ni');

addAnalogInputChannel(s,'cDAQ1Mod1', 'ai0', 'Voltage');
```

Add the listener for the `DataAvailable` event and assign it to the variable `lh`:

```
lh = addlistener(s,'DataAvailable', @plotData);
```

For more information on events, see Events and Listeners — Concepts in the MATLAB Object-Oriented Programming documentation.

Create a simple callback function to plot the acquired data and save it as `plotData.m` in your working directory:

```
function plotData(src,event)

    plot(event.TimeStamps, event.Data)

end
```

Here, `src` is the session object for the listener and `event` is a `daq.DataAvailableInfo` object containing the data and associated timing information.

Acquire the data and see the plot update while MATLAB® is running:

```
startBackground(s);
```

When the operation is complete, delete the listener:

```
delete (lh)
```

## a. Generare un grafico live dei dati acquisiti in background

- Usare `daq.createSession` per creare una sessione
- Valutare le caratteristiche del segnale osservato all'oscilloscopio ed impostare dei valori sensati di s.rate e s.DurationInSeconds.

```
s = daq.createSession('ni');
addAnalogInputChannel(s,'ni', ai0, 'Voltage');
s.Rate = ----;
s.DurationInSeconds = ---- ;
```

Acquisendo i dati in background, è possible fornire le direttive per gestire i dati man mano che questi sono acquisiti il tutto utilizzando gli elementi del "listener" e gli "events".

Un evento **DataAvailable** avviene quando una specifica quantità di dati è disponibile per la sessione.Il listener risponde a quell'evento e inizia una funzione specifica.

Usare **`addlistener`** per aggiungere una funzione anonima alla sessione. Questa funzione viene richiamata ogni volt ache l'evento DataAvailable si verifica e plotta i dati rispetto al tempo. Quando viene aggiunto un listener, viene inviato un handle al listener.

Salva l'handle nella variabile lh e cancellala più tardi.

```
lh = addlistener(s,'DataAvailable', @(src,event))

plot(event.TimeStamps, event.Data));
```

Di default il listener viene richiamato **10 volte al secondo** ma può essere anche modificata la frequenza di richiama a seconda di quanto si vule che sia aggiornata la traccia. Per farlo bisogna modificare la proprietà `NotifyWhenDataAvailableExceeds`. Il listener verrà richiamato ogni volt ache il numero di punti supera ilv alore impostato. Per esempio se lo impostiamo a 2000, ipotizzando che il rate impostato 2000 sample/s,, vorrà dire che lo richiamerà 1 volta al secondo.

```
s.NotifyWhenDataAvailableExceeds = ----; (in base alla vostra scelta del s.rate)
```

Dopo che avete aggiunto il listener, dare inizio all'acquisizione in backgound.

```
s.startBackground();
```

Non ci sono altri calcoli da eseguire e l'acquisizione viene fatta procedure per il tempo di osservazione stabilito, fino a che l'acquisizione sia terminate. Chiamando la funzione wait senza argomenti il programma imposta il tempo di atesa come infinito e viene interrotta una volta che viene terminata l'acquisizione.

```
s.wait()
```

Cancellare il listener in modo che non venga compilato durante la prossima acquisizione.

```
delete(lh)
```

## b. Catturare un Singolo Evento nei dati in ingresso

In alcuni casi non c'è un tempo prefissato di osservazione o un certo numero di scans da acquisire.

Può accadere che sia necessario acquisire in continuo oppure in continuo solo fino a che una specifica condizione viene incontrata.

***Opzione 1: Acquisizione continua***

Resettare la frequenza a cui il listener viene richiamato alla condizione di default (10 times/sec) Impostare: `s.IsNotifyWhenDataAvailableExceedsAuto = 1;`

```
lh = addlistener(s,'DataAvailable', @(src,event)

plot(event.TimeStamps, event.Data));
```

Configura la sessione per acquisire in modo continuo.

```
s.IsContinuous = 1;
```

```
s.startBackground()
```

Per interrompere l'acquisizione manualmente una volta che si visualizza l'evento

**stop()**

***Opzione 2: Acquisizione continua fino al superamento di una certa condizione***

Esempio supponiamo di acquisire finche il segnale supera 1V (regolare questa ampiezza in base alla forma del segnale visualizzata sull'oscilloscopio).

Resettare la frequenza a cui il listener viene richiamato alla condizione di default (10 times/sec) Impostare:
`s.IsNotifyWhenDataAvailableExceedsAuto = 1;`

Configura un nuovo listener per processare i dati in ingresso:

```
lh = addlistener(s,'DataAvailable', @stopWhenExceedOneV);
```

***stopWhenExceedOneV è una funzione che va salvata in un file separato.***

```
Type('stopWhenExceedOneV.m')
```

```
function stopWhenExceedOneV(src, event)
    if any(event.Data > 1.0)
        disp('Event listener: Detected voltage exceeds 1, stopping acquisition')
        % Continuous acquisitions need to be stopped explicitly.
        src.stop()
        plot(event.TimeStamps, event.Data)
    else
        disp('Event listener: Continuing to acquire')
    end
end
```

Configura la sessione per acquisire in modo continuo. Il listener individua l'evento >1V e interrompe l'acquisizione.

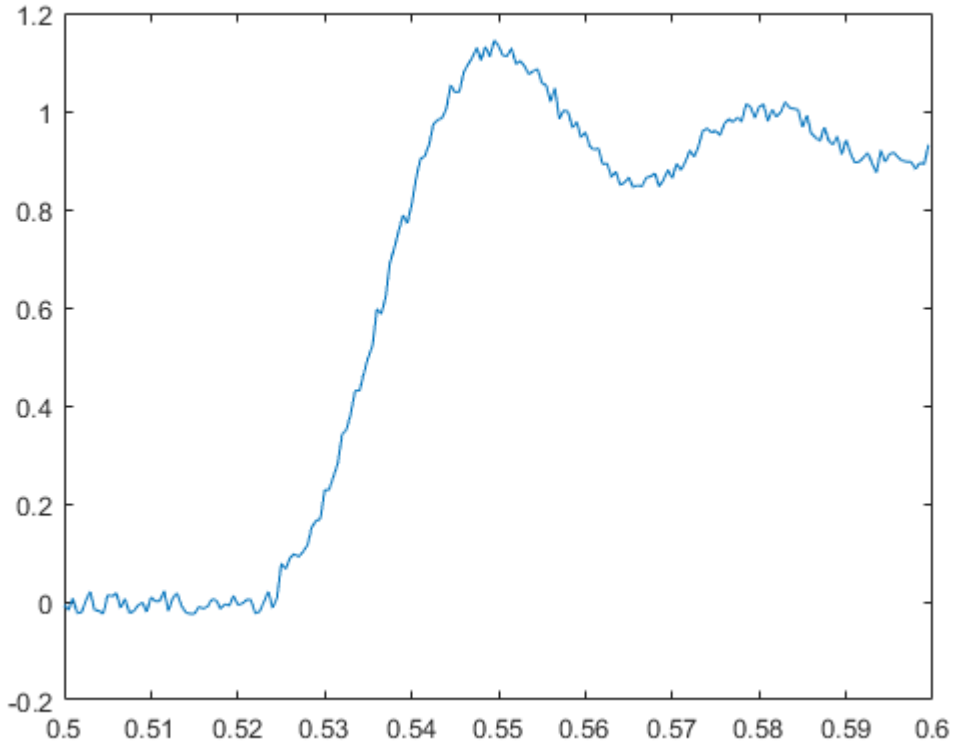
```
s.IsContinuous = 1;
s.startBackground()
```

Usare `pause` in a loop per monitorare I dati acquisiti durante l'acquisizione. L'output verrà unito all'output dell'evento listener.

```
while s.IsRunning
    pause(0.5)
    fprintf('While loop: Scans acquired = %d\n', s.ScansAcquired)
end

fprintf('Acquisition has terminated with %d scans acquired\n', s.ScansAcquired);
```

```
Event listener: Continuing to acquire
Event listener: Continuing to acquire
Event listener: Continuing to acquire
Event listener: Continuing to acquire
Event listener: Continuing to acquire
While loop: Scans acquired = 1007
Event listener: Detected voltage exceeds 1, stopping acquisition
While loop: Scans acquired = 1207
Acquisition has terminated with 1207 scans acquired
```


1.2
1
0.8
0.6
0.4
0.2
0
-0.2
0.5
0.51
0.52
0.53
0.54
0.55
0.56
0.57
0.58
0.59
0.6