POLITECNICO DI BARI

DIPARTIMENTO DI INGEGNERIA ELETTRICA E DELL'INFORMAZIONE

MASTER DEGREE IN
COMPUTER SCIENCE ENGINEERING

MEASUREMENT AND DATA ACQUISITION SYSTEMS

TARATURA
CELLA DI CARICO

Docente:
DI NISIO Attilio

Studenti:

FIORE Marco (mat. 579930)
STASOLLA Michele (mat. 579597)
TINTI Guido Tommaso (mat. 580375)

ANNO ACCADEMICO 2019-2020

Sommario

Obiettivo

Obiettivo del progetto è eseguire una corretta taratura di una cella di carico a doppia flessione, in modo da poterla utilizzare per leggere pesi e studiarne il funzionamento. Tra le funzioni da implementare si trovano:

- Taratura ed equilibratura della cella di carico.
- Lettura di pesi noti e costruzione della retta di taratura.
- Comparazione di valori letti con diversi guadagni.
- Lettura di pesi non noti e conversione della stessa in grammi.

Strumenti utilizzati

Tra gli strumenti utilizzati per il progetto si distinguono:

- Componenti hardware
  - Arduino
  - Amplificatore HX711
  - Cella di carico
- Componenti software
  - LabVIEW
  - Sketch Arduino

Arduino

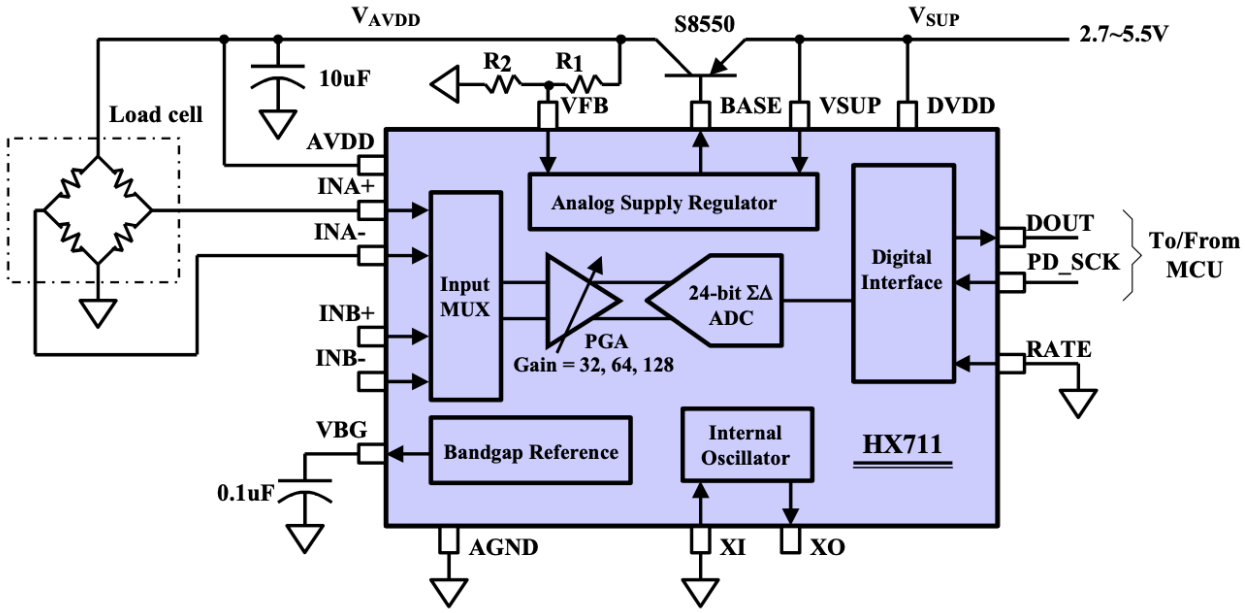
La scheda utilizzata nel progetto è Arduino UNO rev.3, basata sul microcontrollore a 8 bit Atmel Atmega 328.

Microcontroller	*ATmega328*
Operating Voltage	*5V*
Input Voltage (recommended)	*7-12V*
Input Voltage (limit)	*6-20V*
Digital I/O Pins	*14*
PWM Digital I/O Pins	*6*
Analog Input Pins	*6*
DC Current per I/O Pin	*40 mA*
DC Current for 3.3V Pin	*50 mA*
Flash Memory	*32 KB*
Flash Memory for Bootloader	*0.5 KB*
SRAM	*2 KB*
EEPROM	*1 KB*
Clock Speed	*16 MHz*
Lenght	*68.6 mm*
Width	*53.4 mm*
Weight	*25 g*

Amplificatore HX711

L'amplificatore utilizzato è l'HX711, dotato di un convertitore analogico-digitale a 24 bit ideale per acquisire dati da un sensore a ponte.[1]

Dispone di due canali di ingresso, rispettivamente A e B selezionabili in base al guadagno impostato; in particolare sul canale A si può impostare un guadagno di 128, rappresentante una tensione di ingresso differenziale pari ± 20mV, oppure di 64, indicante una tensione di ingresso differenziale di ± 40mV. Il canale B invece ha un guadagno fisso di 32, pari a ± 80mV.



L'acquisizione dei dati avviene mediante i pin PD_SCK e DOUT, necessari sia per la selezione del guadagno (e di conseguenza del canale di ingresso) sia per il recupero dati; l'output dei 24 bit di dati è espresso in complemento a 2.

Cella di carico

Una cella di carico è un trasduttore, ossia un componente elettronico che misura la forza applicata da un oggetto, grazie a un segnale elettrico che varia al variare della deformazione che la forza in questione produce sul componente stesso. La cella di carico rilascia un segnale proporzionale alla forza misurata che viene poi tradotto in un valore numerico.

Le celle di carico che utilizzano estensimetri sono costituite da quattro di questi elementi disposti in una configurazione a ponte di Wheatstone, posizionati su un elemento elastico. L'elemento elastico in genere è in acciaio o alluminio, quindi molto robusto, ma ha anche una minima elasticità: subisce una leggera deformazione sotto carico, ma è in grado di tornare alla posizione di partenza se non più sollecitato.

Gli estensimetri sono saldamente collegati all'elemento elastico, diventando solidali ai movimenti dello stesso.

[1] Appendice A: HX711

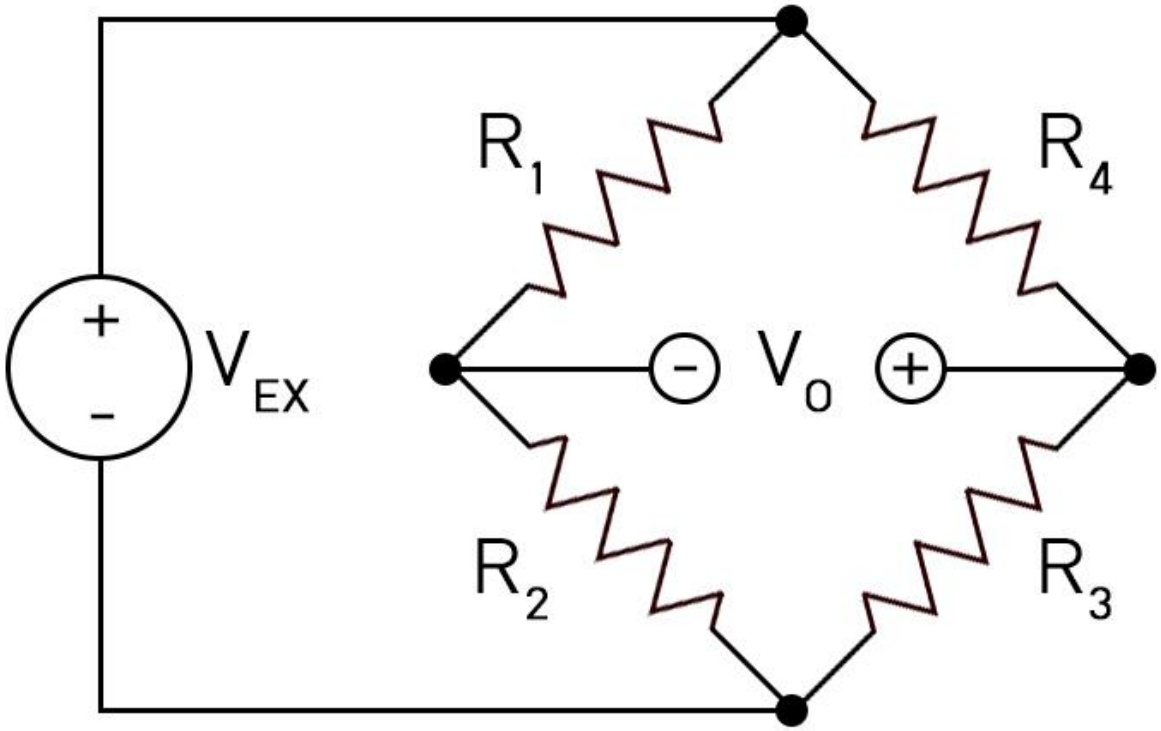
Configurazione a ponte di Wheatstone

I quattro estensimetri presenti sulla cella di carico sono collegati ad anello e la griglia di misurazione della forza misurata è allineata di conseguenza. Questa configurazione è detta a ponte di Wheatstone, il quale si compone di un generatore di tensione che alimenta due rami resistivi posti in parallelo: il primo è composto da un resistore campione in serie cassetta di resistori (resistenza variabile tramite opportune manopole) di elevata precisione. Il secondo ramo è composto da un resistore campione in serie alla resistenza incognita. Si pone un galvanometro (dispositivo che traduce una corrente elettrica in un momento magnetico) a zero centrale, tra i due resistori del primo ramo e i due del secondo ramo.

Alimentando quindi il circuito si nota che il galvanometro segnala il passaggio di una corrente elettrica. Si varia quindi il valore della cassetta di resistenze fino a quando il galvanometro non indica più il passaggio di una corrente. In questa situazione il valore della resistenza elettrica del resistore incognito è calcolabile.

R1 e *R3* sono due resistori di valore fisso e noto, mentre il resistore *R2* è variabile. Per effettuare la misura si fa variare il resistore *R2* fino ad ottenere il punto di equilibrio, cioè fino a che il galvanometro misurerà passaggio di corrente nullo.

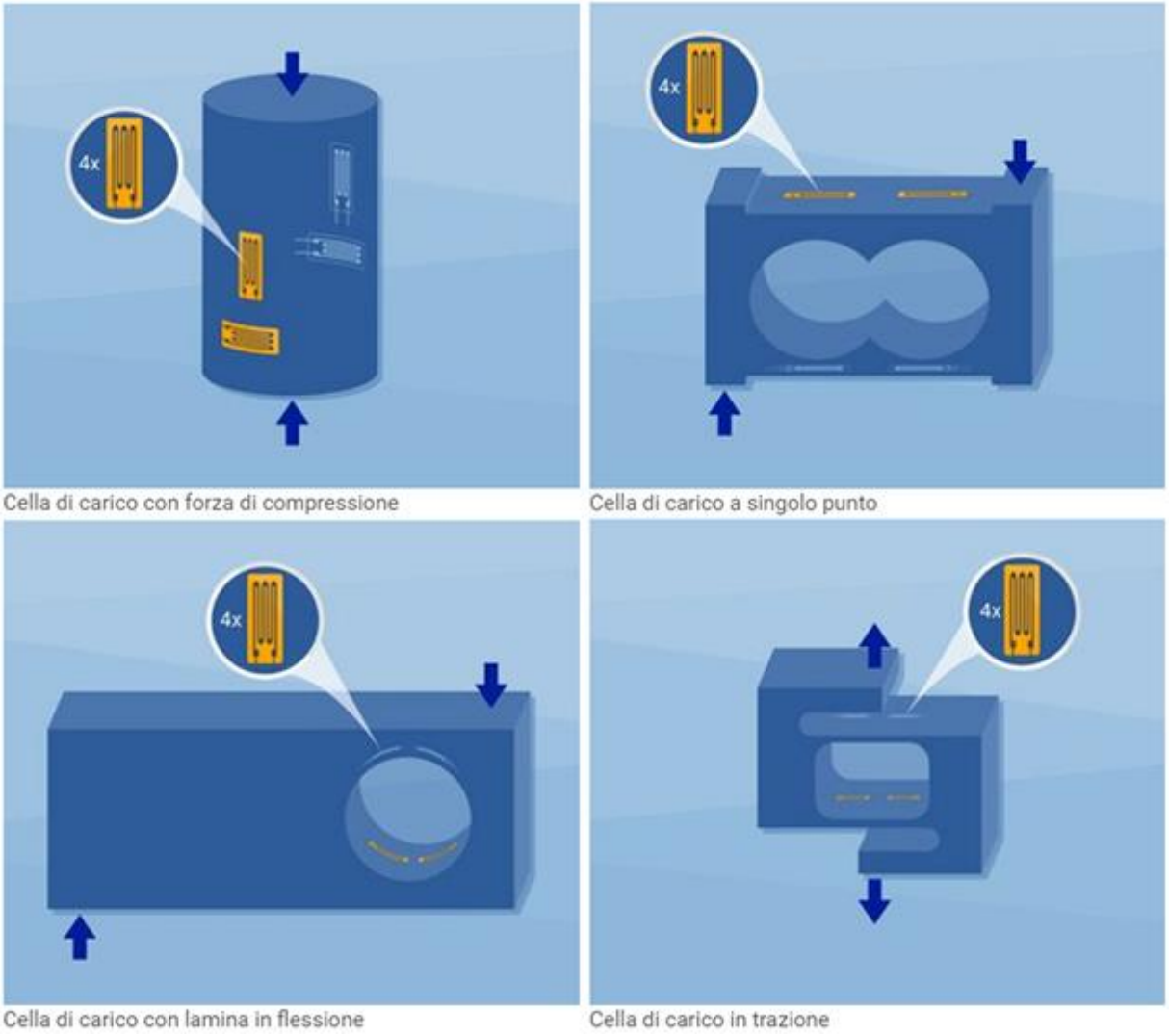
Quando il ponte è costruito in modo che *R1* sia uguale a *R3* ed *Rx* risulta essere uguale a *R2* solitamente in condizione di equilibrio. In condizione di equilibrio è sempre vero che $Rx = \frac{R2 * R3}{R1}$, quindi noti *R1*, *R2* e *R3* è possibile determinare *Rx*.



Tipologie di celle di carico

Esistono diversi tipi di celle di carico per diverse applicazioni. Tra quelle più usate si trovano:

- **Cella di carico a singolo punto**: viene collocata una cella di carico sotto una piattaforma caricata con un peso dall'alto.
- **Cella di carico con lamina in flessione**: più celle di carico sono posizionate sotto una struttura di acciaio e sono caricate con un peso dall'alto.
- **Cella di carico con forza di compressione**: più celle di carico ad alta capacità sono posizionate sotto una struttura di acciaio caricata con un peso dall'alto.
- **Cella di carico in trazione**: viene sospeso un peso da una o più celle di carico.



Nel corso del progetto sono state utilizzate due celle: a singolo punto e mini-cella di carico.[2]

[2] Appendice A: datasheet Cella di carico

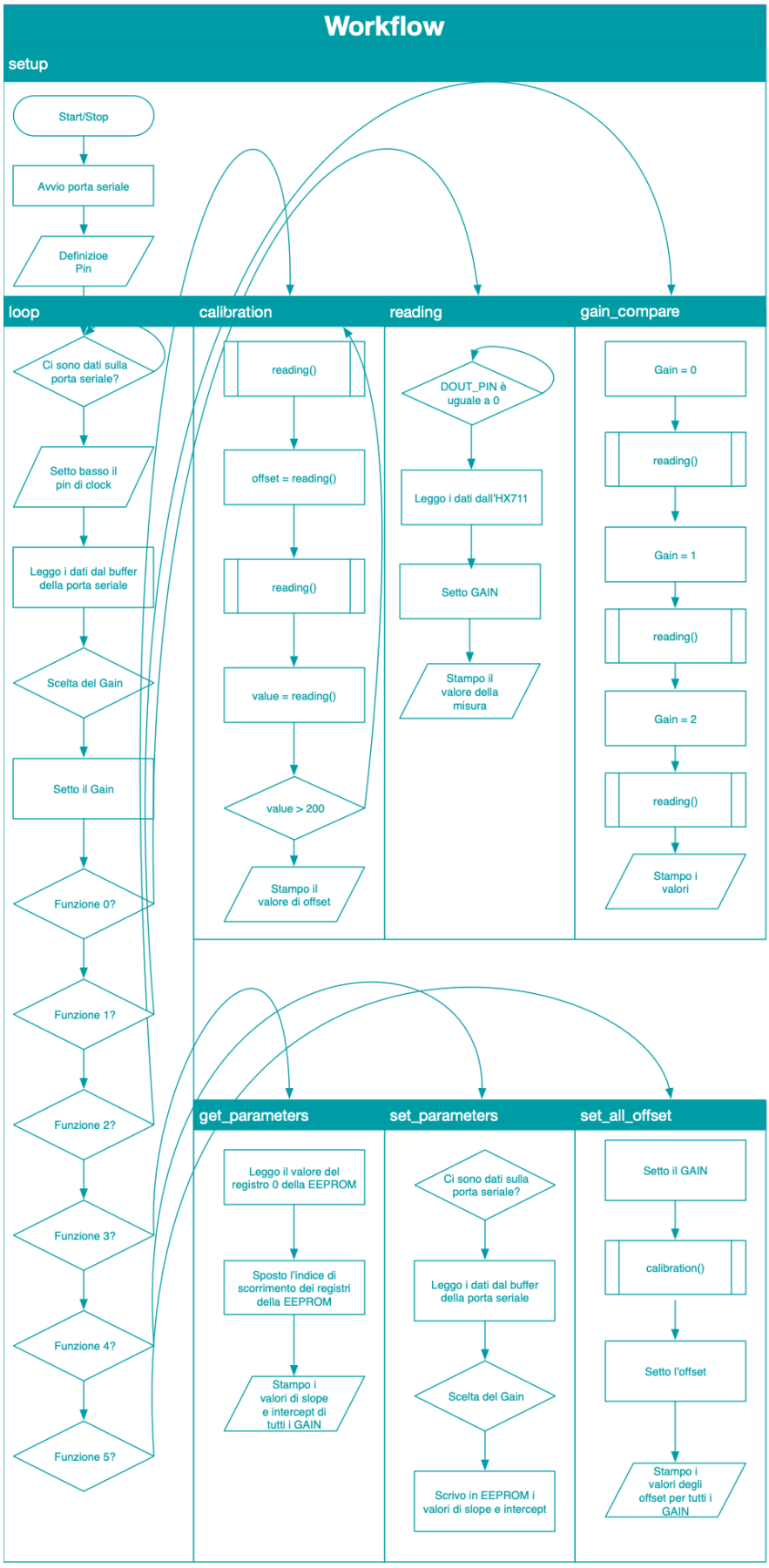
Descrizione del progetto

Il progetto sfrutta la taratura di una cella di carico per analizzare nel complesso il funzionamento del trasduttore. In particolare, tramite una comunicazione tra Arduino e LabVIEW, permette di eseguire tre funzioni principali:

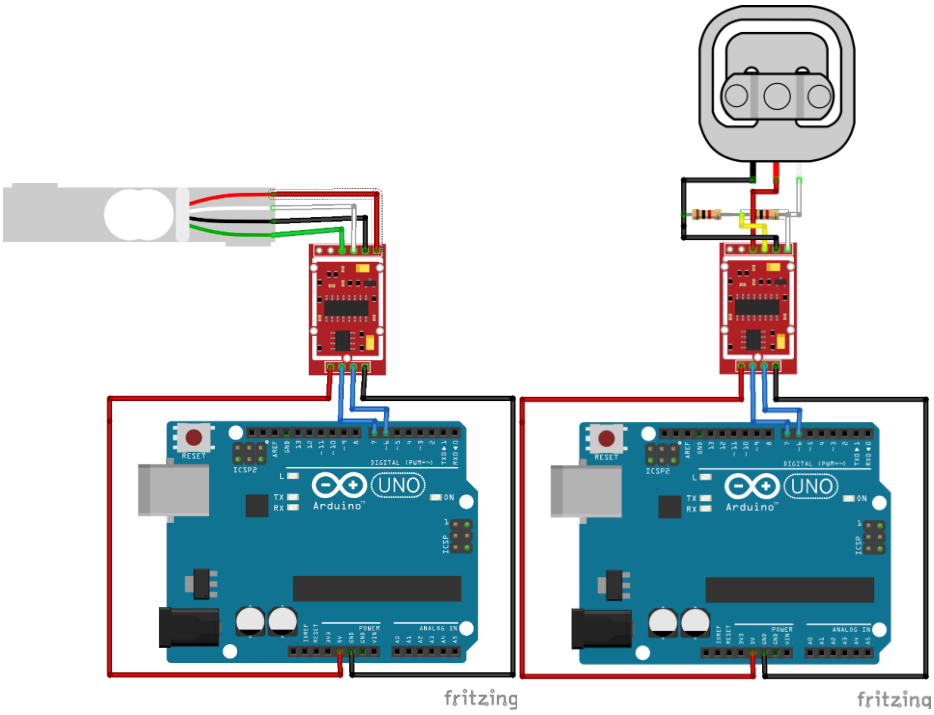
- Lettura dei dati: acquisizione di dati grezzi dalla cella, calcolo di parametri e ingegnerizzazione del dato.
- Comparazione dei guadagni: lettura di dati grezzi dalla cella variando il guadagno dell'amplificatore.
- Equilibratura e taratura: taratura della cella di carico con relativa impostazione dei parametri della retta di taratura: pendenza e intercetta.

In aggiunta alle funzioni principali, il programma prevede:

- Salvataggio dei dati acquisiti su file esterni, utilizzabili con software di calcolo.
- Salvataggio su EEPROM Arduino dei parametri principali di taratura, in modo da poterli utilizzare successivamente senza bisogno di nuova taratura.


Workflow
setup
Start/Stop
Avvio porta seriale
Definizioe Pin
loop
Ci sono dati sulla porta seriale?
Setto basso il pin di clock
Leggo i dati dal buffer della porta seriale
Scelta del Gain
Setto il Gain
Funzione 0?
Funzione 1?
Funzione 2?
Funzione 3?
Funzione 4?
Funzione 5?
calibration
reading()
offset = reading()
reading()
value = reading()
value > 200
Stampo il valore di offset
reading
DOUT_PIN è uguale a 0
Leggo i dati dall'HX711
Setto GAIN
Stampo il valore della misura
gain_compare
Gain = 0
reading()
Gain = 1
reading()
Gain = 2
reading()
Stampo i valori
get_parameters
Leggo il valore del registro 0 della EEPROM
Sposto l'indice di scorrimento dei registri della EEPROM
Stampo i valori di slope e intercept di tutti i GAIN
set_parameters
Ci sono dati sulla porta seriale?
Leggo i dati dal buffer della porta seriale
Scelta del Gain
Scrivo in EEPROM i valori di slope e intercept
set_all_offset
Setto il GAIN
calibration()
Setto l'offset
Stampo i valori degli offset per tutti i GAIN

Schema circuitale



Arduino e l'amplificatore HX711 sono collegati tramite i pin D6 e D7 di Arduino. In particolare,

- 5V => VCC
- D6 => SCK
- D7 => DT
- GND => GND

L'amplificatore comunica con la cella di carico a singolo punto con i seguenti collegamenti:

- E+ => Cavo rosso
- E- => Cavo nero
- A- => Cavo bianco
- A+ => Cavo verde

Per la mini-cella di carico sono necessari due resistori da 1kΩ per completare il ponte di Wheatstone:

- E+ => Cavo bianco
- E- => Cavo nero
- A- => Completamento del ponte di Wheatstone
- A+ => Cavo rosso

Arduino

Lo sketch Arduino[3] è stato progettato per l'acquisizione dei dati dalla cella di carico; per fare ciò non è stata utilizzata la libreria dell'amplificatore HX711 disponibile per questo tipo di applicazioni, ma si è proceduto all'implementazione delle funzioni direttamente nello sketch.

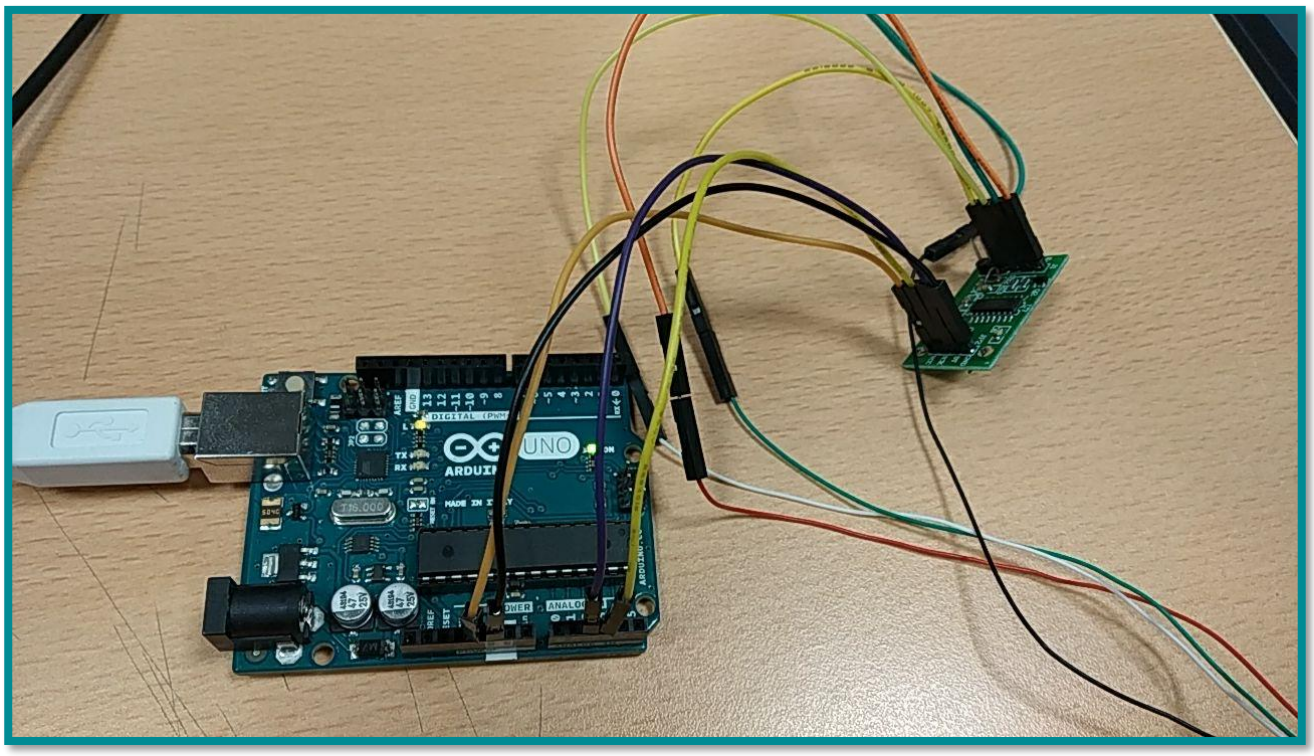
Dopo la definizione dei PIN di collegamento dell'amplificatore e il settaggio della velocità di comunicazione della porta COM, si è proceduto all'implementazione del menu di funzionamento, raggiungibile mediante sequenza numerica scritta sul buffer della porta COM.

In particolare, la sequenza numerica prevede la scelta del tipo di guadagno e successivamente la scelta della funzione da utilizzare.

Nella funzione *loop* è stata implementata la verifica della presenza di nuovi elementi sul buffer della porta COM: nel caso in cui ci siano dati, viene settato basso il PIN di clock (PD_SCK) dell'amplificatore HX711; successivamente viene effettuato il *parse* dei valori presenti sul buffer, mediante i quali si settano GAIN e la funzione.

Il valore di GAIN può acquisire i valori:

- 32
- 64
- 128



[3]Appendice B: codice Arduino

Le funzioni implementate sono:

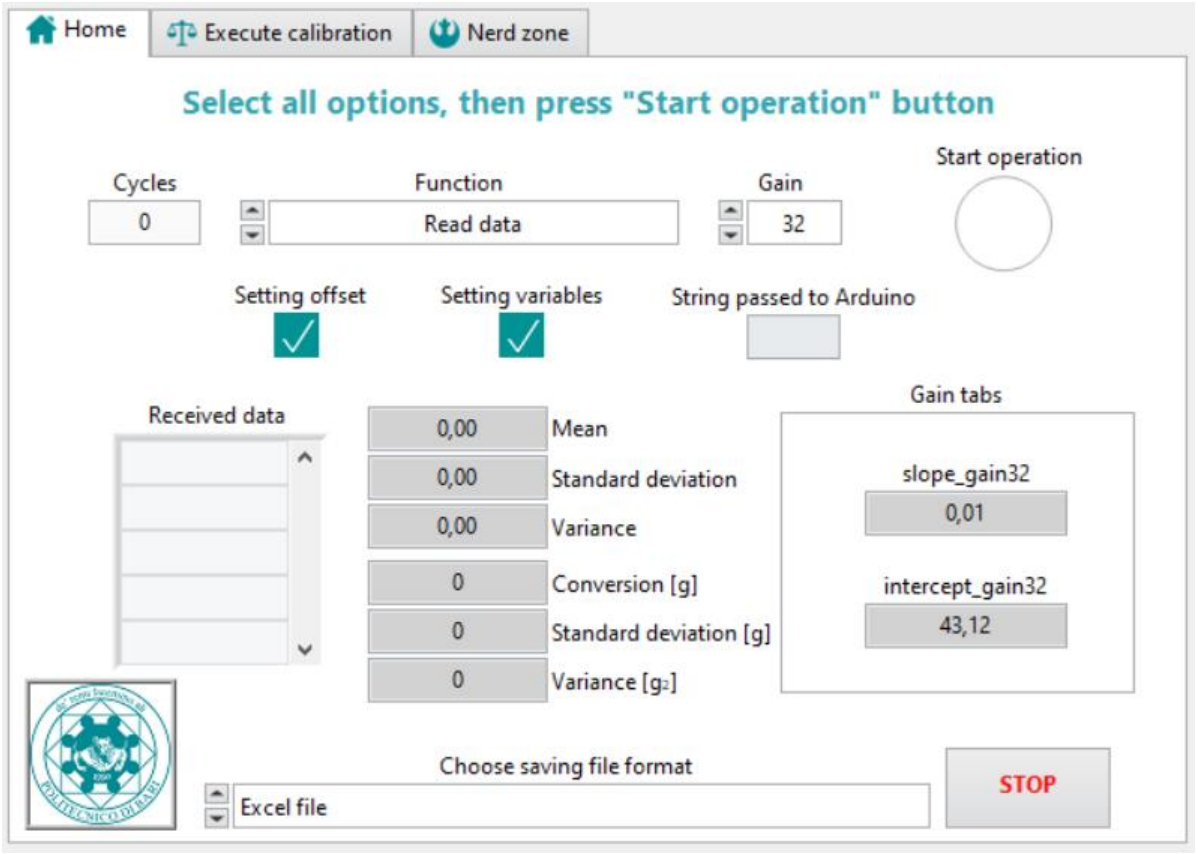
- Lettura (*reading*)
    - La funzione di lettura inizializza un array vuoto nel quale, dopo aver verificato che i dati siano pronti alla lettura mediante lo stato *LOW* del pin *DOUT,* si vanno a inserire i dati provenienti dall'amplificatore mediante la funzione *shiftin,* la quale sposta un byte di dati un bit alla volta, in questo caso con politica ***M**ost**S**ignificant**B**it**First.*** L'operazione è ripetuta fino al riempimento delle tre celle dell'array.
      Si ricompone infine il numero, applicando a tutti i componenti l'operatore << che effettua lo shift dei bit della quantità desiderata. Al numero ottenuto si sottrae l'offset calcolato mediante equilibratura.
- Confronto dei GAIN (*compare_gain*)
    - La funzione effettua la comparazione dei GAIN, effettuando dieci misurazioni e restituendo la media di queste per ognuno dei GAIN.
- Calibrazione (*calibration*)
    - La funzione di calibrazione in Arduino si occupa di effettuare l'equilibratura, acquisendo il primo valore e usandolo come offset per le successive. Per ottimizzare l'acquisizione dell'offset si è pensato di effettuare una verifica sul valore restituito dalla differenza tra offset e misura, verificando che questa non sia maggiore di 200 in valore assoluto.
- Recupero dei dati dalla EEPROM (*get_parameters*):
    - La funzione prevede di acquisire dai registri EEPROM di Arduino tutti i valori di taratura scritti in precedenza, partendo dal primo valore e settando il registro successivo come la somma del registro precedente più la lunghezza del valore appena letto.
- Salvataggio dei valori di taratura nei registri EEPROM (*set_parameters*):
    - All'interno di questa funzione è presente una nuova sezione di acquisizione dati dalla porta seriale, il cui protocollo prevede di acquisire per primo il valore del GAIN, poi il valore di *slope* e *intercept* della retta individuata mediante LabVIEW.
- Equilibratura iniziale (*set_all_offset*)
    - Questa funzione, a differenza di *calibration,* permette di fornire all'avvio dell'applicativo LabVIEW i valori di offset aggiornati per effettuare le misure, cosi da sfruttare le tarature precedentemente realizzate e salvate in EEPROM.

LabVIEW

Front panel

L'interfaccia grafica su LabVIEW è stata studiata per risultare di facile utilizzo e ben organizzata. In particolare, si compone di tre tab principali:

- Home
- Execute calibration
- Nerd zone



In Home si possono impostare i principali parametri necessari per il corretto funzionamento del programma come numero di cicli, funzione da utilizzare e guadagno.

- Ring per scegliere la funzione da eseguire, mappata secondo la seguente regola:
  - 0 – Read Data
  - 1 – Compare Different Gains
  - 2 – Execute Calibration
- Ring per scegliere il guadagno, mappato secondo la seguente regola:
  - 0 – guadagno a 32
  - 1 – guadagno a 64
  - 2 – guadagno a 128

Il tab mostra informazioni aggiuntive di configurazione iniziale – impostazione degli offset su Arduino e restituzione dei parametri per le conversioni – e altri riferimenti utili in fase di lettura dei dati. È inoltre possibile selezionare il formato del file di salvataggio dei dati – xls, tsv o txt.

Il tab Execute calibration mostra:

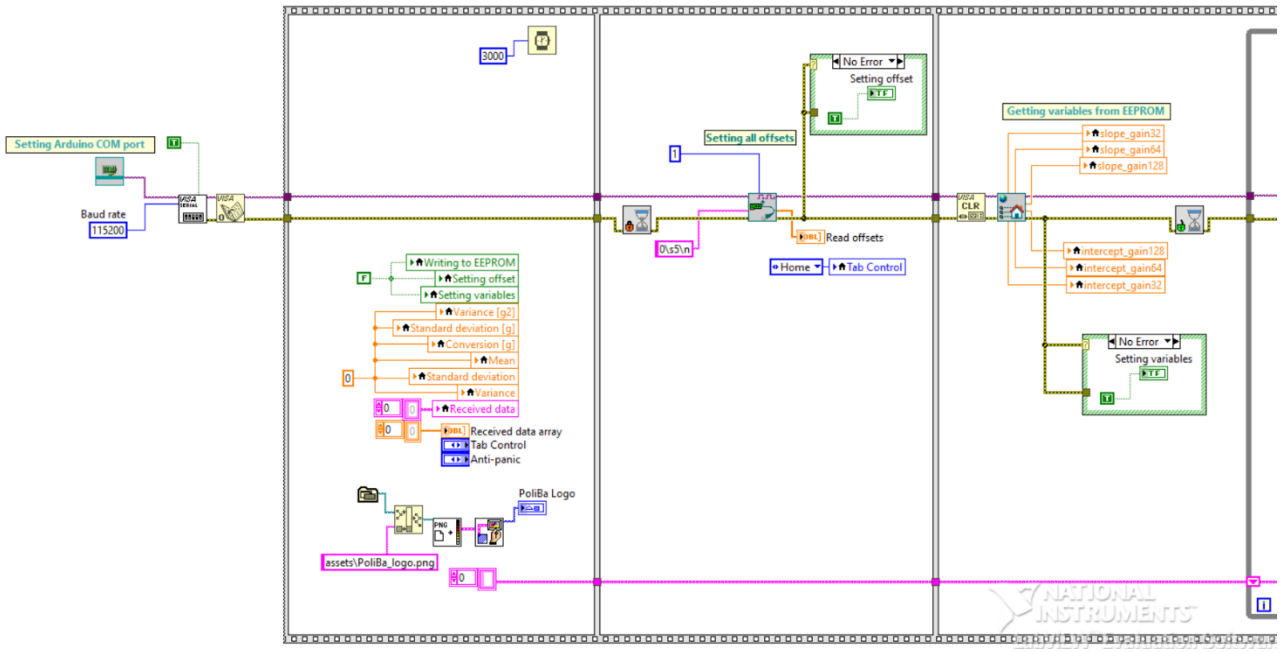
- Il rumore calcolato in base al guadagno impostato.
- L'array degli errori assoluti.
- L'array degli errori relativi.
- Un checkbox che mostra il corretto aggiornamento dei dati nell'EEPROM di Arduino.
- I dati dei pesi noti letti in fase di taratura.
- Un grafico che mostra la retta di taratura, basata sulle letture dei pesi noti effettuate.

Il tab Nerd zone mostra informazioni aggiuntive come gli offset letti in fase di collegamento con Arduino, i vari dati ricevuti durante l'intera esecuzione e un array di debug contenente i comandi inviati ad Arduino, insieme al comando di stop.

Block Diagram

Il diagramma a blocchi è composto da vari elementi e sub vi. Quando il VI viene eseguito, appare un popup che chiede all'utente di selezionare la porta COM cui è collegato Arduino. Una volta stabilita la connessione – con baud rate 115200 – l'esecuzione entra in una sequenza temporale formata da:

- Inizializzazione delle variabili utilizzate e tempo di attesa di tre secondi.
- Impostazione su Arduino dei tre offset – uno per ogni gain – con conseguente accensione del relativo checkbox nel front panel in caso di successo.
- Recupero delle variabili relative ai tre guadagni dall'EEPROM di Arduino con conseguente accensione del relativo checkbox nel front panel in caso di successo.
- Ingresso nel loop di esecuzione principale.

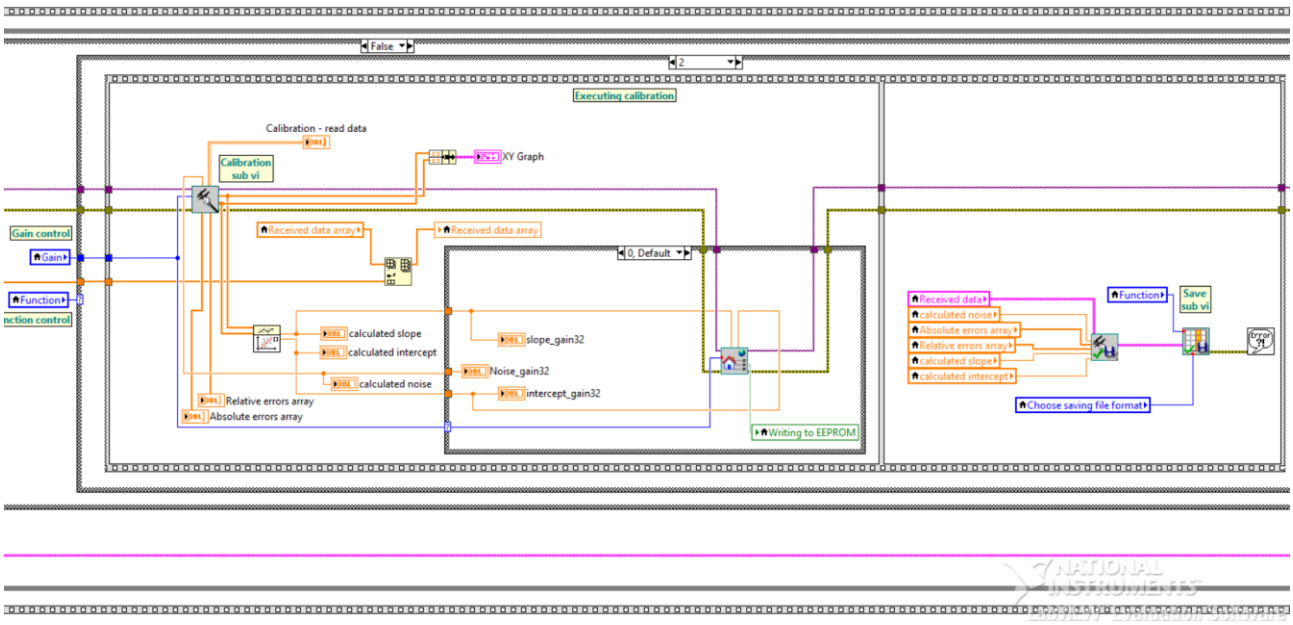


Il loop principale consente l'esecuzione dell'intero programma. Inizialmente è presente un loop secondario in cui il programma cicla indefinitamente – con attese di 500 ms per non intasare il buffer – fino alla pressione del pulsante Start operation o del pulsante di stop. Il loop secondario imposta quindi funzione e guadagno scelti.

Funzione e guadagno vengono fusi in una stringa correttamente formattata da inviare ad Arduino; contestualmente, viene aggiornato l'array di debug e impostato il numero di cicli da eseguire. Il sub VI di comunicazione con Arduino restituisce un array contenente i dati ricevuti dal microcontrollore. In base alla funzione scelta, è possibile entrare in un case:

- Lettura dei dati: i dati letti vengono utilizzati per calcolare media, deviazione standard e varianza. Grazie a pendenza e intercetta della retta – ottenuti in precedenza e selezionati in base al guadagno impostato – è possibile convertire la media delle letture effettuate in grammi. I vari dati vengono infine salvati nell'array principale, mostrati a video nel front panel e salvati in un file.
- Comparazione dei gain: richiama la relativa funzione su Arduino, che restituisce le medie delle letture effettuate con i vari guadagni. Le tre letture vengono mostrate nel front panel e salvate in un file di log.
- Taratura: inizialmente viene richiamato il sub VI relativo alla taratura, che restituisce vari valori:
    - Offset impostato e nuovo valore letto.
    - Array contenente i pesi noti impostati.
    - Array contenente le misure effettuate con i pesi noti impostati.
    - Array degli errori assoluti.
    - Array degli errori relativi.
    - Rumore ottenuto sulle letture.

  Gli array contenenti pesi noti e relative misure sono utilizzati per creare la retta di taratura mostrata nel front panel e per calcolare pendenza e intercetta. Questi ultimi due valori, insieme al rumore, vengono salvati nelle relative variabili in base al guadagno impostato. Pendenza e intercetta sono inoltre salvate nell'EEPROM di Arduino grazie al relativo sub VI, accendendo il relativo checkbox nel front panel in caso di successo. Tutti i dati calcolati sono infine salvati in un file di log.



Il programma termina con la chiusura della connessione con Arduino e un Simple Error Handler per la cattura di eventuali errori ottenuti nelle varie fasi.

Sub VI

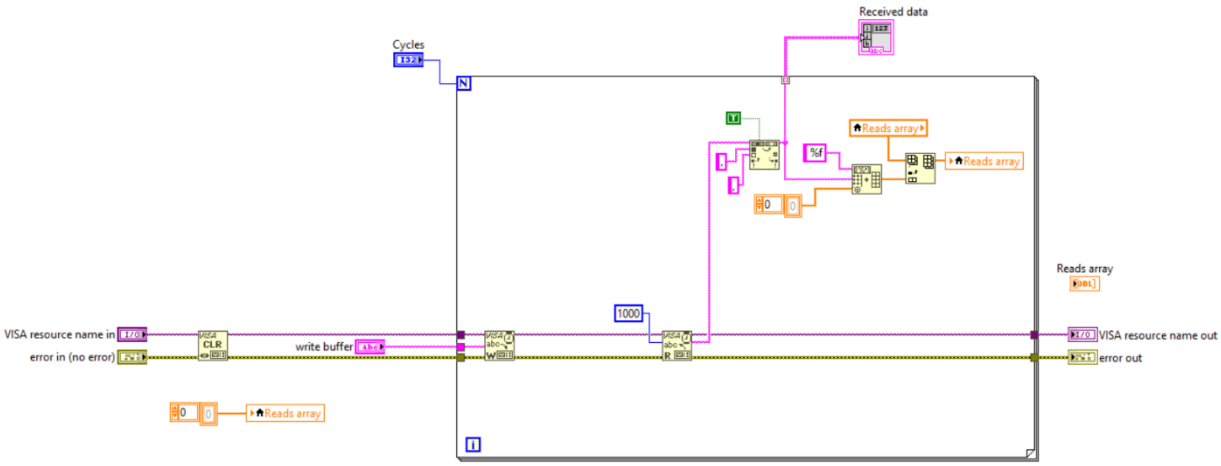
Seguendo l'ordine di esecuzione del programma, si incontrano vari sub VI, descritti di seguito.

Sub_vi_arduino_com_port_popup

Semplice sub VI, mostrato come finestra di dialogo, che permette di impostare la porta COM cui è collegato Arduino.

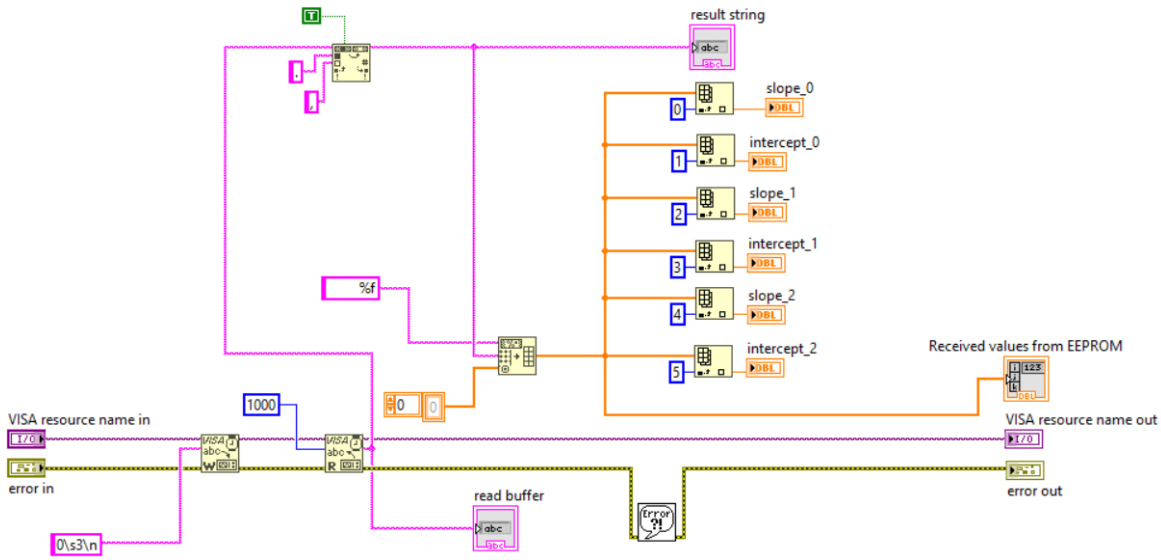
Sub_vi_arduino_communication

Sub VI che manda una stringa correttamente formattata ad Arduino per poi attendere una risposta, che viene raccolta, elaborata per trasformarla in array numerico e restituita. Inizialmente utilizzato per impostare i tre offset su Arduino, viene poi richiamato nella fase principale di comunicazione.



Sub_vi_get_linear_parameters

Sub VI necessario per l'acquisizione dei parametri di taratura dall'EEPROM di Arduino. I parametri restituiti da Arduino sono inseriti in una stringa, che deve essere modificata – convertendo il punto in virgola per i numeri double – e trasformata in array, usando il tabulatore come delimitatore. L'array viene poi scompattato nei sei parametri richiesti.

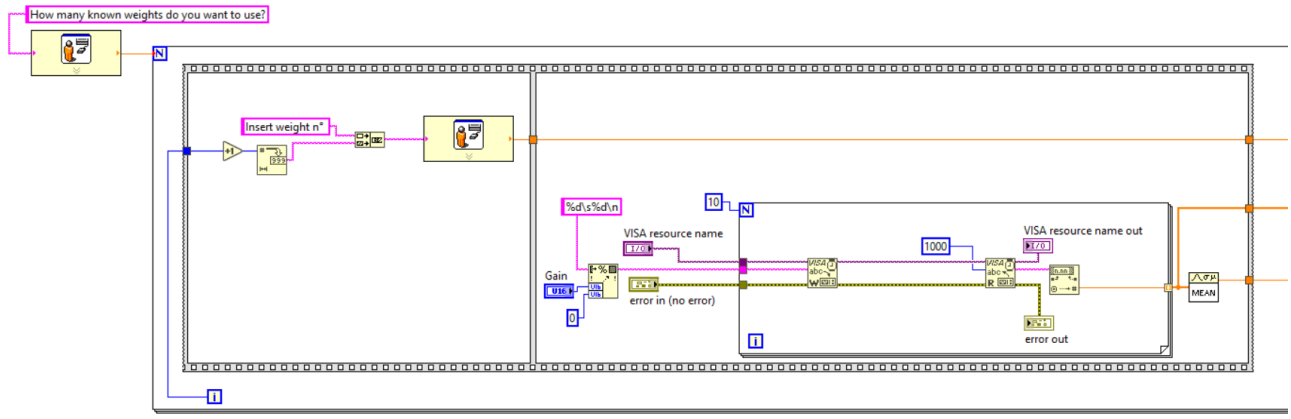


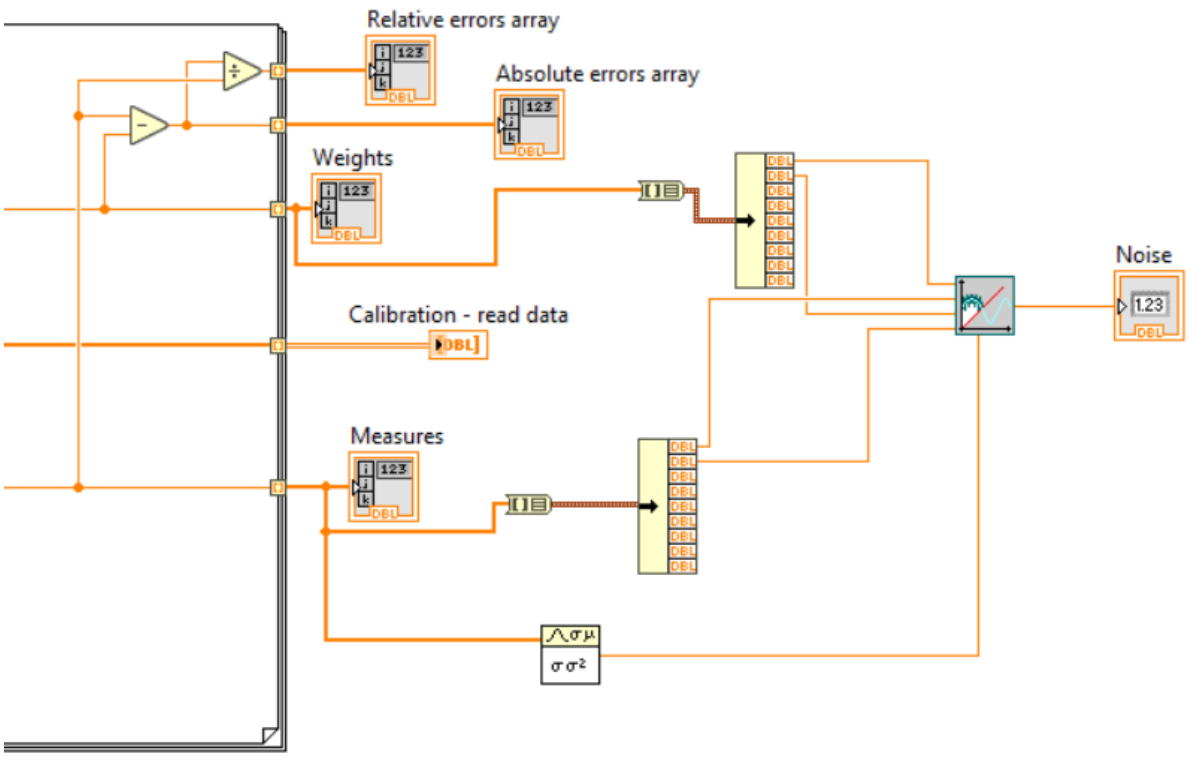
Sub_vi_prompt_calibration_weigths

Sub VI utilizzato per creare i messaggi di dialogo necessari a inserire il numero di pesi noti e i relativi pesi in grammi.

Sub_vi_calibration

Sub VI chiamato in fase di taratura; permette di scegliere quanti pesi noti utilizzare per la creazione dei parametri e della retta di taratura e di segnalare il peso effettivo in grammi, tramite il sub VI descritto in precedenza. Per ogni peso noto vengono effettuate dieci letture e restituita la media. Pesi noti e medie sono collezionati in due array. Sono inoltre qui calcolati i due vettori di errori assoluti ed errori relativi. Per il calcolo del rumore è utilizzato un ulteriore sub VI.





Sub_vi_noise

Sub VI con cinque input: due pesi noti (x2 e x1), due misure (y2 e y1) e deviazione standard (σ). L'output conferito riguarda il rumore, calcolato come segue.

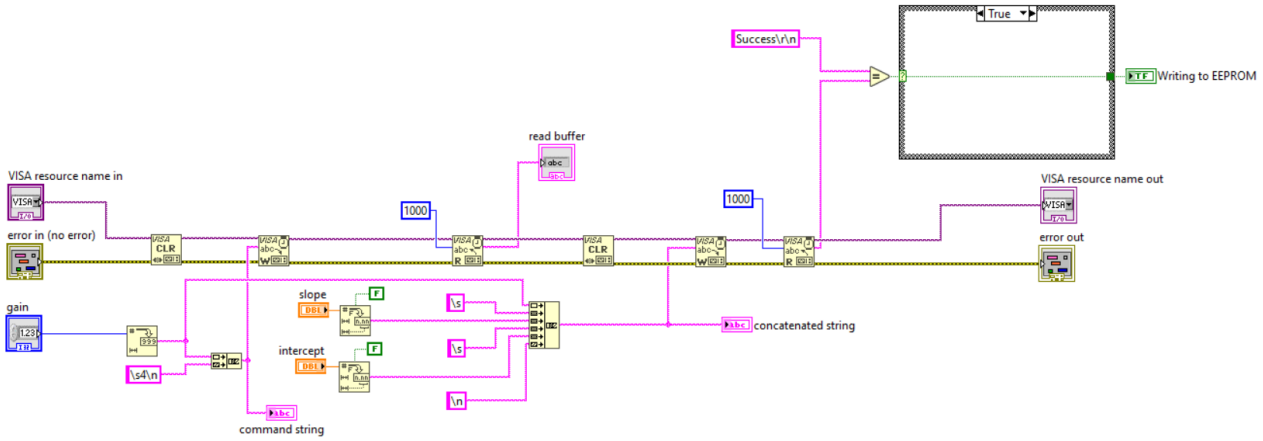
$$Noise = \sigma * \frac{x_2 - x_1}{y_2 - y_1}$$

Sub_vi_convert_measures

Sub VI che converte i valori di media, deviazione standard e varianza, ottenuti in fase di lettura, in grammi, sfruttando la pendenza e l'offset calcolati.

Sub_vi_set_linear_parameters

Sub VI necessario per salvare pendenza e intercetta relative al guadagno impostato nell'EEPROM di Arduino. In caso di successo, il relativo checkbox presente nel tab Execute calibration viene attivato.

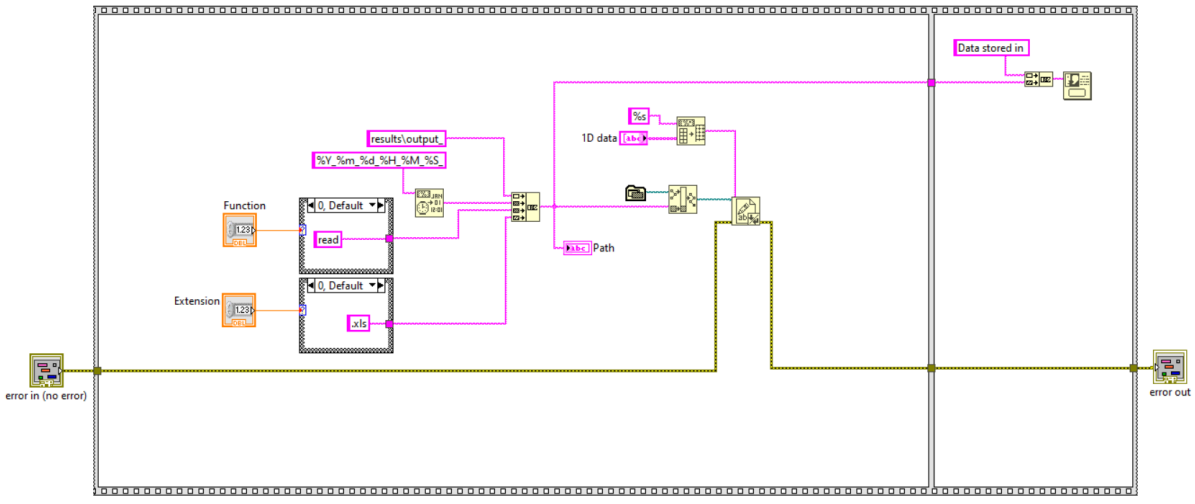


Sub_vi_saving_string_read e Sub_vi_saving_string_calibration

Sub VI utilizzati per creare la stringa correttamente formattata da salvare nei relativi file di log.

Sub_vi_write_file

Il sub VI crea il nome del file da scrivere – diverso in base alla funzione utilizzata – con indicazione dell'orario di salvataggio per evitare collisioni con file dello stesso nome. La stringa – ottenuta con uno dei due precedenti sub VI – è convertita in spreadsheet string e salvata in file. Infine, viene mostrato un popup che segnala il path relativo utilizzato e il nome del file.



Analisi dei risultati

Ottenute varie letture di pesi noti, è possibile calcolare la media delle letture e relativo rate.

Peso noto [g]	Guadagno [bit]	Taratura lettura	Taratura offset	Misura 1	Misura 2	Misura 3	Media misure	Rate	
5	128	-9871	153	2094	2171	2125	2130,00	0,002347	
10	128	-9871	153	4187	4322	4287	4265,33	0,002344	
20	128	-9871	153	8627	8539	8626	8597,33	0,002326	
20	128	-9871	153	8591	8673	8730	8664,67	0,002308	
50	128	-9871	153	21449	21413	21619	21493,67	0,002326	
100	128	-9871	153	43102	43094	43133	43109,67	0,002320	Media: 0,002328727
5	64	-8026	148	1098	1096	1201	1131,67	0,004418	
10	64	-8026	148	2155	2400	2221	2258,67	0,004427	
20	64	-8026	148	4307	4530	4332	4389,67	0,004556	
20	64	-8026	148	4305	4629	4383	4439,00	0,004506	
50	64	-8026	148	10709	10791	10831	10777,00	0,004640	
100	64	-8026	148	21681	21587	21587	21618,33	0,004626	Media: 0,004528757
5	32	?	?	29	47	114	63,33	0,078947	
10	32	?	?	191	-163	86	38,00	0,263158	
20	32	?	?	18	94	501	204,33	0,097879	
20	32	?	?	42	84	151	92,33	0,216606	
50	32	?	?	111	-120	101	30,67	1,630435	
100	32	?	?	-62	52	-229	-79,67	-1,255230	Media: 0,17196595

È possibile notare come, aumentando il numero di cicli, migliora la lettura del peso posto sulla cella di carico. La varianza, di conseguenza, diminuisce. La seguente tabella mostra una comparativa tra cinque diversi pesi noti – 5g, 10g, 20g, 50g, 100g – di cui si riportano le ultime tre – di cento – letture oltre a media, varianza, deviazione standard e conversione della media in grammi.

Peso [g]	Lettura 1	Lettura 2	Lettura 3	Media	Deviazione standard	Varianza	Conversione [g]	Differenza [g]
5	2302	2010	2328	2197,560	91,817	8430,411	5,077	-0,077
10	4269	4309	4257	5297,050	2205,673	4864993,785	12,432	-2,432
20	8679	8745	8691	8276,660	1252,377	1568449,277	19,366	0,634
50	21556	8679	21646	16104,580	6360,044	40450164,327	37,583	12,417
100	43070	43055	43027	43021,100	92,912	8632,616	100,090	-0,090

La differenza maggiore tra peso noto e lettura in grammi si nota con il peso di 50 g, in cui si ha una discordanza del 25%. Nel caso di 5 g e 100 g, invece, i valori letti sono molto vicini ai valori effettivi, con discordanze, rispettivamente, di 1,5% e 0,09%.

Confronto con mini-cella di carico

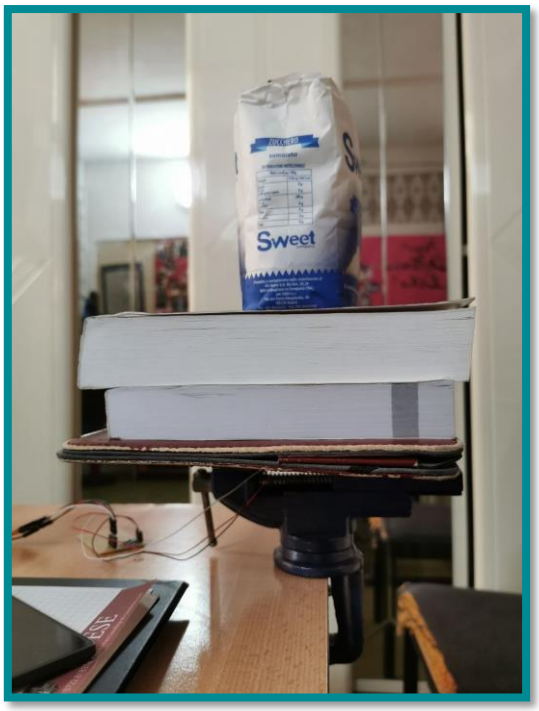
A causa dell'emergenza sanitaria Covid-19, è stato impossibile proseguire con le prove sperimentali in laboratorio. I file di log raccolti prima dell'emergenza sono incompleti poiché relativi a una precedente versione del programma, in cui deviazione standard e varianza non erano ingegnerizzati, quindi inutilizzabili.

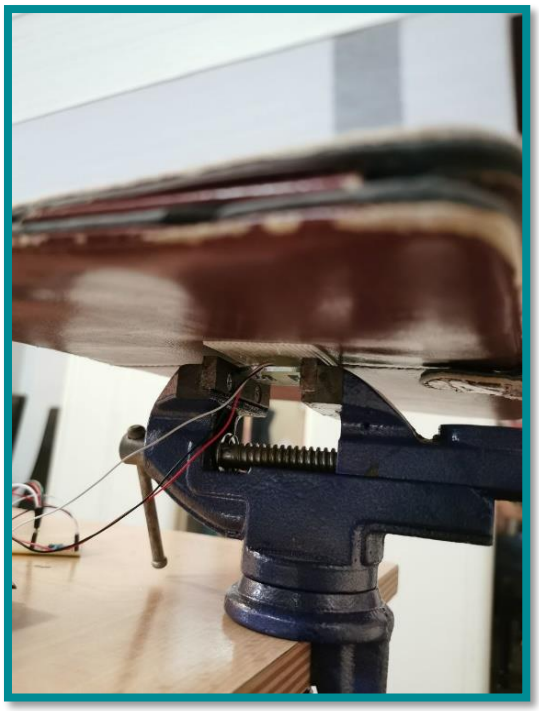
La seconda parte dei test è stata effettuata con una mini-cella di carico, dotata di una precisione nettamente minore rispetto alla cella a singolo punto usata in laboratorio. Per questo motivo, le letture proposte in questo paragrafo presentano un'incertezza alta.

Guadagno	Peso $[g]$	Lettura 1	Lettura 2	Lettura 3	Media $[g]$	Deviazione standard $[g]$	Varianza $[g^2]$
32	1000	2300	2394	2041	2113,861	20668,084	-1435069,672
32	2300	1645	2187	1667	2087,424	20527,299	-1534861,895
32	3300	2277	1582	1781	2987,597	20690,696	-1419348,935
64	1000	-23594	-23621	-23770	1007,534	-15,219	-1815,958
64	2300	-54104	-54129	-54179	2300,096	-14,038	-1361,274
64	3300	-76206	-76688	-76727	3279,100	-24,777	-7911,785
128	1000	-46548	-46430	-46353	997,413	0,794	-664,520
128	2300	-106653	-106570	-106176	2283,325	0,275	-860,683
128	3300	-153375	-153164	-153474	3290,833	-0,921	-1408,434

Analizzando la deviazione standard, è possibile determinare il comportamento del trasduttore quando sono applicati pesi diversi. In particolare, con l'aumentare del guadagno aumenta la precisione del dato letto, arrivando a una deviazione standard minima nel caso di lettura del peso di 2300 g con guadagno 128. Con lo stesso guadagno impostato, all'aumentare del peso applicato sulla mini-cella di carico aumenta la varianza.

Il guadagno impostato a 32 riporta letture meno coerenti tra loro: si nota come la deviazione standard e la varianza siano esageratamente alte in questa modalità di lettura dei dati.





Considerazioni finali

L'utilizzo combinato di Arduino e LabVIEW si è dimostrato particolarmente efficace per tarare una cella di carico, poiché permette di affiancare la possibilità di Arduino di collegarsi a diversi trasduttori e leggerne dati alla possibilità di LabVIEW di effettuare calcoli complessi unitamente a un'interfaccia grafica personalizzabile e appagante.

È possibile apportare ulteriori migliorie al progetto, come il calcolo delle incertezze di tipo A e B o la visualizzazione dei pesi letti sulla retta di taratura.

Appendice A: datasheet

Di seguito sono riportati i datasheet dell'amplificatore HX711 e della cella di carico a singolo punto.

HX711

Parameter	Notes	MIN	TYP	MAX	UNIT
Full scale differential input range	V(inp)-V(inn)		±0.5 (AVDD/GAIN)		V
Common mode input		AGND +1.2		AVDD - 1.3	V
Output data rate	Internal Oscillator, RATE = 0		10		Hz
	Internal Oscillator, RATE = DVDD		80		
	Crystal or external clock, RATE = 0		f_{clk}/1,105,920		
	Crystal or external clock, RATE = DVDD		f_{clk}/138,240		
Output data coding	2's complement	800000		7FFFFF	HEX
Output settling time (1)	RATE = 0		400		ms
	RATE = DVDD		50		
Input offset drift	Gain = 128		0.2		mV
	Gain = 64		0.4		
Input noise	Gain = 128，RATE = 0		50		nV(rms)
	Gain = 128，RATE = DVDD		90		
Temperature drift	Input offset（Gain = 128）		±6		nV/℃
	Gain（Gain = 128）		±5		ppm/℃
Input common mode rejection	Gain = 128，RATE = 0		100		dB
Power supply rejection	Gain = 128，RATE = 0		100		dB
Reference bypass（VBG)			1.25		V
Crystal or external clock frequency		1	110.592	20	MHz
Power supply voltage	DVDD	2.6		5.5	V
	AVDD，VSUP	2.6		5.5	
Analog supply current (including regulator)	Normal		1400		μA
	Power down		0.3		
Digital supply current	Normal		100		μA
	Power down		0.2		

Cella di carico

SPECIFICATIONS		
Capacity	kg	2,3,5,10,20,50kg
Safe overload	%FS	120
Ultimate overload	%FS	150
Rated output	mV/V	1.0 ± 0.1
Excitation voltage	Vdc	3~10
Combined error	%FS	± 0.05
Zero balance	%FS	± 0.1
Non-linearity	%FS	± 0.05
Hysteresis	%FS	± 0.05
Repeatability	%FS	± 0.05
Creep	%FS/3min	± 0.1
Input resistance	Ω	1000 ± 10
Output resistance	Ω	1090 ± 10
Insulation resistance	MΩ	≥ 2000
Operating temperature range	℃	-10 ~ +55
Compensated temperature range	℃	-10 ~ +40
Temperature coefficient of SPAN	%FS/10℃	± 0.05
Temperature coefficient of ZERO	%FS/10℃	± 0.05
Electrical connection	Cable	4 color wire, Ø 0.8 × 200 mm
	excitation(+):Red signal(+):Green excitation(-):Black signal(-):White	

Appendice B: codice Arduino

```
//*****************************************************************//
 //*********************     Arduino sketch for MDAS project     ***************//
  //***********************       Fiore, Stasolla, Tinti        *********************//
//*****************************************************************//
#include <EEPROM.h>
// Pin for HX711
const int DOUT_PIN = 6;          // DATA OUT Pin
const int PD_SCK = 7;            // CLOCK Pin
uint8_t GAIN = 0;                // Variable for GAIN read
long offset;                     // Variable for offset read for first equilibration
long value = 0;                  // Variable for result read
bool flag = 1;                   // Flag for writing the calibration values in EEPROM
bool flag_for_setting_all_offset = 1;    // Flag for initial balancing
long g_128 = 0;                  // Offset with GAIN 128
long g_64 = 0;                   // Offset with GAIN 64
long g_32 = 0;                   // Offset with GAIN 32
long read_32 = 0;                // Read with GAIN 32
long read_64 = 0;                // Read with GAIN 64
long read_128 = 0;               // Read with GAIN 128

void setup() {
 Serial.begin(115200);                // Serial port @115200
 pinMode(PD_SCK, OUTPUT);             // Setting pin for HX711
 pinMode(DOUT_PIN, INPUT);            // Setting pin for HX711
}

void loop() {
 if (Serial.available()) {            // Check data on serial port
  digitalWrite(PD_SCK, LOW);          // Set LOW Clock PIN of HX711
  int g = Serial.parseInt();          // Parsing for GAIN value
  int op = Serial.parseInt();         // Parsing for function value
  if (Serial.read() == '\n') {        // Verify termination character
   switch (g) {
    case 0:
     GAIN = 2;                        // GAIN 32
     break;
    case 1:
     GAIN = 3;                        // GAIN 64
     break;
    case 2:                           // GAIN 128
     GAIN = 1;
     break;
   }
```

```
// Switch for function
    switch (op) {
     case 0:
      reading();                          // Function for reading data
      Serial.println(value);
      break;
     case 1:
      compare_gain();                     // Function for GAIN comparison
      Serial.print(String(read_32) + ("\t") + String(read_64) + ("\t") + String(read_128) +
("\n"));
      break;
     case 2:
      flag_for_setting_all_offset = 1;    // Flag for activating the calibration
      calibration();                      // Calibration function
      flag_for_setting_all_offset = 0;
      if (GAIN == 1) {
       g_128 = offset;
      } else if (GAIN == 2) {
       g_32 = offset;
      } else {
       g_64 = offset;
      };
      Serial.print(String(offset) + ("\t") + String(value) + ("\n"));
      break;
     case 3:
      get_parameters();                   // Function for returning the calibration values
      break;
     case 4:
      flag = 1;
      Serial.println("Setting parameters");
      set_parameters();                   // Function for saving value in EEPROM
      break;
     case 5:
      set_all_offset();                   // Function for first equilibration
      Serial.print(String(g_32) + ("\t") + String(g_64) + ("\t") + String(g_128) + ("\n"));
      break;
    }
   }
  }
}

// Function for returning the calibration values
void get_parameters() {
  int address = 0;                        // Address for returning the calibration values
  double slope1;                          // Slope value for GAIN 32
  double intercept1;                      // Intercept for GAIN 32
  double slope2;                          // Slope value for GAIN 64
```

```
  double intercept2;                    // Intercept for GAIN 64
  double slope3;                        // Slope value for GAIN 128
  double intercept3;                    // Intercept for GAIN 128
  EEPROM.get(address, slope1);          // Putting in EEPROM starting from 0
  address += sizeof(slope1);            // Setting the new address
  EEPROM.get(address, intercept1);
  address += sizeof(intercept1);
  EEPROM.get(address, slope2);
  address += sizeof(slope2);
  EEPROM.get(address, intercept2);
  address += sizeof(intercept2);
  EEPROM.get(address, slope3);
  address += sizeof(slope3);
  EEPROM.get(address, intercept3);
  address += sizeof(intercept3);
  Serial.print(String(slope1) + ("\t") + String(intercept1) + ("\t") + String(slope2) + ("\t") +
String(intercept2) + ("\t") + String(slope3) + ("\t") + String(intercept3) + ("\n"));
  if (address == 512) {                 // If address == 512, start over
    address = 0;
  }
}

void set_parameters() {
  int address;
  while (flag == 1) {
    if (Serial.available()) {           // Check data presence on serial port
      int g = Serial.parseInt();        // Reading gain value on serial port
      float slope = Serial.parseFloat();// Reading slope value on serial port
      float intercept = Serial.parseFloat();// Reading intercept value on serial port
      if (Serial.read() == '\n') {
        switch (g) {
          case 0:                       // gain 32
            address = 0;
            EEPROM.put(address, slope);
            address += sizeof(slope);
            EEPROM.put(address, intercept);
            flag = 0;
            Serial.println("Success");
            break;
          case 1:                       // gain 64
            address = 8;
            EEPROM.put(address, slope);
            address += sizeof(slope);
            EEPROM.put(address, intercept);
            flag = 0;
            Serial.println("Success");
            break;
```

```
        case 2:                                   // gain 128
          address = 16;
          EEPROM.put(address, slope);
          address += sizeof(slope);
          EEPROM.put(address, intercept);
          flag = 0;
          Serial.println("Success");
          break;
       }
      }
     }
    }
   }

// Function for reading data from load cell
long reading() {
  long read = 0;                          // Variable for read value
  uint8_t data[3] = { 0 };                // Structure for read data saving
  uint8_t filler = 0x00;
  while (digitalRead(DOUT_PIN)) {         // If pin is high, wait for available value
    delay(100);
  }
  // 24 clock pulses with sum for GAIN definition
  data[2] = shiftIn(DOUT_PIN, PD_SCK, MSBFIRST);
  data[1] = shiftIn(DOUT_PIN, PD_SCK, MSBFIRST);
  data[0] = shiftIn(DOUT_PIN, PD_SCK, MSBFIRST);
  // Select channel and gain factor
  for (unsigned int i = 0; i < GAIN; i++) {
    digitalWrite(PD_SCK, HIGH);
    digitalWrite(PD_SCK, LOW);
  }

  // Padding 32 bit
  if (data[2] & 0x80) {
    filler = 0xFF;
  } else {
    filler = 0x00;
  }

  // 32 bit number, obtained by bit shift
  read = ( static_cast<unsigned long>(filler) << 24
         | static_cast<unsigned long>(data[2]) << 16
         | static_cast<unsigned long>(data[1]) << 8
         | static_cast<unsigned long>(data[0]) );
```

```
if (flag_for_setting_all_offset == 1) {
   value = read - offset;
 } else {
   // Calibration with offset subtraction
   if (GAIN ==  1) {
     value = read - g_128;
   } else if (GAIN == 2) {
     value = read - g_32;
   } else {
     value = read - g_64;
   }
 }
 return (value);
}

void calibration() {
 offset = 0;                    // Initialize the variable containing the offset
 offset = reading();            // Reads to populate the offset variable
 reading();                     // Take the second reading to check the condition
 while (abs(value) > 200) {     // To get a better calibration,
                                // cycle until the differences is less than 200
   calibration();
 }
}

// Function to set the GAIN offset
void set_all_offset() {
 // GAIN a 128
 GAIN = 1;
 calibration();
 g_128 = offset;
 delay(500);
 // GAIN a 64
 GAIN = 3;
 calibration();
 g_64 = offset;
 delay(500);
 // GAIN a 32
 GAIN = 2;
 calibration();
 g_32 = offset;
 // Disables the control on the flag concerning the setting of the offset
 flag_for_setting_all_offset = 0;
}
```

```
// Function for comparing GAIN
void compare_gain() {
  long read = 0;
  int i;
  // GAIN 128
  GAIN = 1;
  for (i = 0; i < 10; i++) {                // Average over ten samples
    read = read + reading();
  }
  read_128 = read / 10;
  delay(500);

  read = 0;
  // GAIN 64
  GAIN = 3;
  for (i = 0; i < 10; i++) {                // Average over ten samples
    read = read + reading();
  }
  read_64 = read / 10;
  delay(500);


// GAIN 32
  GAIN = 2;
  read = 0;
  for (i = 0; i < 10; i++) {                // Average over ten samples
    read = read + reading();
  }
  read_32 = read / 10;
}
```