



آزمایشگاه سخت افزار

فانکشن ژنراتور و اسیلوسکوپ نرم افزاری
گزارش نهایی

دانشکده مهندسی کامپیوتر
دانشگاه صنعتی شریف
نیم سال دوم ۱۰-۰۰

استاد:
جناب آقای دکتر اجلالی

اعضای تیم :
محمد مهدی جراحی - ۹۷۱۰۵۸۴۴
نگین جعفری - ۹۷۱۰۵۸۵۵
مهسا امامی - ۹۷۱۰۵۷۶۹



فهرست مطالب

۳	۱	بخش تجاری
۳	۱.۱	معرفی اجمالی
۴	۲.۱	مشخصات فنی
۵	۲	بخش فنی
۵	۱.۲	مزایای رقابتی
۶	۲.۲	ساختار فایل‌ها
۷	۳.۲	اجزای سیستم
۷	۱.۳.۲	معماری سخت‌افزار
۹	۲.۳.۲	نرم‌افزار روی برد سخت‌افزار
۱۱	۳.۳.۲	نرم‌افزار روی PC
۱۲	۴.۳.۲	نرم‌افزار ارتباط با آردوئینو
۱۴	۵.۳.۲	نرم‌افزار رسم نمودار
۱۹	۴.۲	طراحی جعبه
۲۰	۳	جمع‌بندی
۲۰	۱.۳	هزینه‌ی تولید محصول
۲۱	۲.۳	تست‌های انجام شده
۲۲	۳.۳	شکل نهایی محصول

فهرست تصاویر

۱	نمونه‌ای از اوپسیلوسکوپ و فانکشن ژنراتور
۲	پوشه‌بندی و ساختار فایل‌ها
۳	برد آردوینو
۴	ساختار برد آردوینو
۵	نحوه‌ی استفاده از یک ورودی و خروجی محصول
۶	استفاده از تمام ورودی‌ها و خروجی‌های محصول
۷	جعبه آردوینو
۸	کابل‌ها و گیره‌های سوسماری
۹	نمودار رسم شده از داده‌های آنالوگ دریافتی ورودی‌های ۱ و ۳ و ۶



۲۳ بخش ساخت افزاری محصول ۱۰

فهرست برنامه‌ها

۱۱ Software/R\main.py
۱۱ Software/R\requirements.txt



۱ بخش تجاری

۱.۱ معرفی اجمالی

یکی از ابزارهای بسیار پرکاربرد در مهندسی سخت‌افزار و مهندسی برق برای طراحی و کار کردن با مدارهای الکترونیکی، اوسیلوسکوپ و فانکشن ژنراتور است.



شکل ۱: نمونه‌ای از اوسیلوسکوپ و فانکشن ژنراتور

با توجه به گران بودن این ابزار، برای پروژه‌های ساده که دقت و سرعت‌های خیلی بالا نمی‌خواهیم، استفاده از این ابزار توجیه اقتصادی ندارد.

به همین دلیل، ما محصولی طراحی کردی‌ایم که به کمک یک برد آردوئینوی ارزان قیمت و اتصال آن به یک کامپیوتر، بتوانیم یک اوسیلوسکوپ و فانکشن ژنراتور ساده داشته باشیم.



۲.۱ مشخصات فنی

MICROCONTROLLER	ATmega2560
OPERATING VOLTAGE	5V
INPUT VOLTAGE (RECOMMENDED)	7-12V
INPUT VOLTAGE (LIMIT)	6-20V
USABLE INPUT PINS (ADC)	8 PINS (A0-A7 / D54-D61)
USABLE OUTPUT PINS (PWM)	8 PINS (D2-D9)
SAMPLE RATE	20Hz
DC CURRENT PER I/O PIN	20 mA
CLOCK SPEED	16 MHz
BOARD LENGTH	101.52 mm
BOARD WIDTH	53.3 mm
BOARD WEIGHT	37 g
BOX LENGTH	11 cm
BOX WIDTH	6 cm
BOX HEIGHT	2.7 cm

جدول ۱: مشخصات بخش سخت افزار

OS	WINDOWS, UNIX BASED (LINUX, macOS)
Python	VERSION 3.9
pyserial	VERSION 3.5
numpy	VERSION 1.22.3
matplotlib	VERSION 3.3.3

جدول ۲: مشخصات بخش نرم افزار



۲ بخش فنی

۱.۲ مزایای رقابتی

- اصلی‌ترین مزیت رقابتی محصول ما، توانایی ذخیره هر ۸ سیگنال به صورت جداگانه و تولید دوباره‌ی آن‌ها است. سایر محصولات مشابه موجود در بازار این قابلیت مهم و کاربردی را ارائه نمی‌دهند.
 - همان‌طور که در بخش تجاری گفته شد، محصولات مشابه حرفه‌ای، قیمت‌های بسیار بالایی دارند. حال آنکه محصول ما صرفاً به یک برد آردوانیوی نسبتاً ارزان قیمت احتیاج دارد.
 - یکی دیگر از مزایای محصول ما این است که نمودارها را روی لپتاپ نمایش می‌دهد؛ که هم باعث راحتی بیشتر و هم باعث رسم نمودارهای جذاب‌تر می‌شود.
 - مزیت دیگر محصول ما این است که قابلیت رسم و همچنین تولید ۸ سیگنال به صورت همزمان دارد.

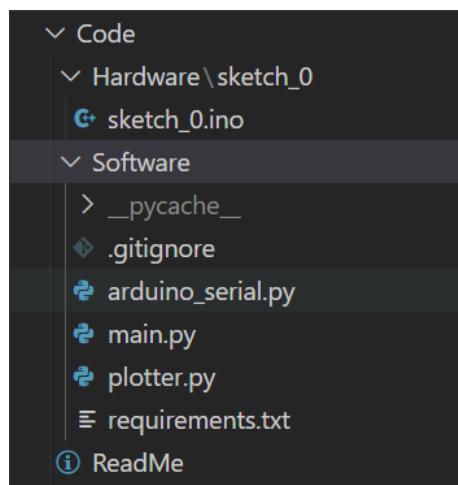


۲.۲ ساختار فایل‌ها

پروژه ما دو بخش اصلی دارد: بخش سخت‌افزاری و بخش نرم‌افزاری. جهت راحتی کار فایل‌های این دو بخش به صورت مجزا از هم و در پوشش‌های Hardware و Software به ترتیب قرار گرفته‌اند. فایل sketch_0.ino که کد مربوط به آردوینو می‌باشد در پوشش سخت‌افزار قرار می‌گیرد. در پوشه نرم‌افزار ۳ فایل اصلی داریم:

- main.py که نقطه شروع و اجرای برنامه هست و کد بخش‌های مختلف را بهم متصل می‌کند.
- arduino-serial.py که پس از شروع اجرای برنامه توابع موجود در آن اجرا می‌شوند و امکان خواندن و نوشتمن همزمان در آردوینو را فراهم می‌کنند.
- plotter.py که بعد از گرفتن ورودی‌ها توسط فایل قبل، مقادیر موجود را نمایش می‌دهد و امکاناتی دیگری از قبیل ذخیره ورودی‌ها و امکان فرستادن آن‌ها به آردوینو برای نمایش توسط LED ، انتخاب نمودارهای دلخواه و زوم بر روی نمودار دلخواه را فراهم می‌کند.
- requirements.txt که نام کتابخانه‌های مورد نیاز برای اجرای برنامه در آن موجود است و نیاز است تا قبل شروع اجرا آن‌ها را نصب کرد. برای نصب تنها کافی هست تا به محل این پوشش رفته و دستور زیر را در ترمینال اجرا کنیم.

```
1 pip install -r requirements.txt
```



شکل ۲: پوشش‌بندی و ساختار فایل‌ها



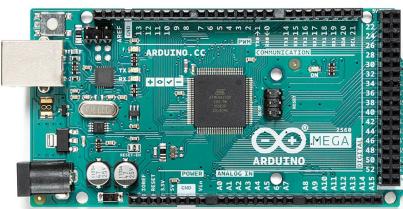
۳.۲ اجزای سیستم

همانطور که بالاتر نیز به آن اشاره شد، پروژه ما از دو بخش تشکیل می‌شود: بخش نمونه‌برداری (سخت‌افزاری) و بخش رسم نمودار (نرم‌افزاری).

در زیر به توضیح کارهای انجام‌شده‌ی سخت‌افزار و نرم‌افزار به صورت جداگانه می‌پردازیم.

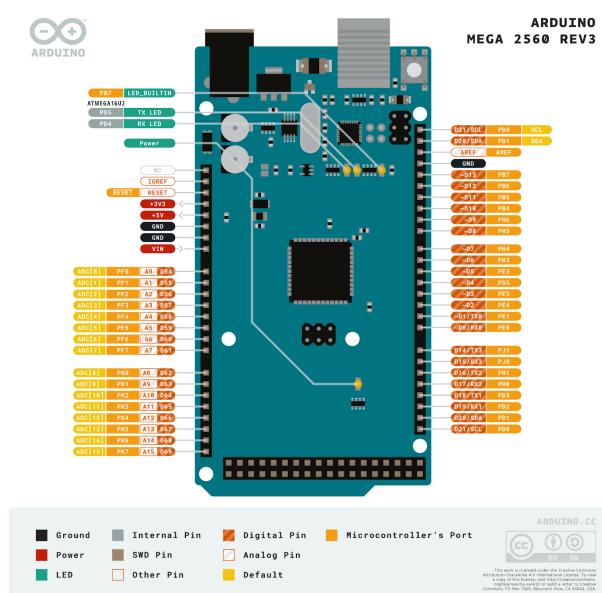
۱.۳.۲ معماری سخت‌افزار

ما از برد 2560 arduino mega استفاده می‌کنیم.



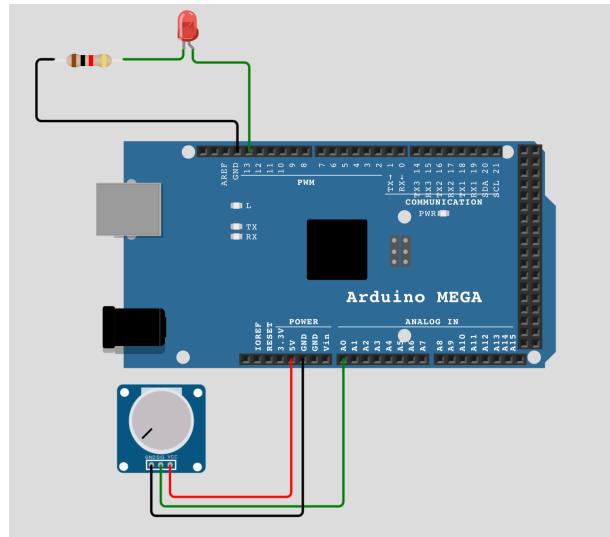
شکل ۳: برد آردوینو

دلیل استفاده از این برد این است که هم ارزان قیمت است و هم تعداد پورت‌های ADC و PWM لازم را دارد. این برد ۱۶ ورودی آنالوگ و ۱۵ خروجی PWM دارد که در تصویر زیر نیز قابل ملاحظه‌اند:

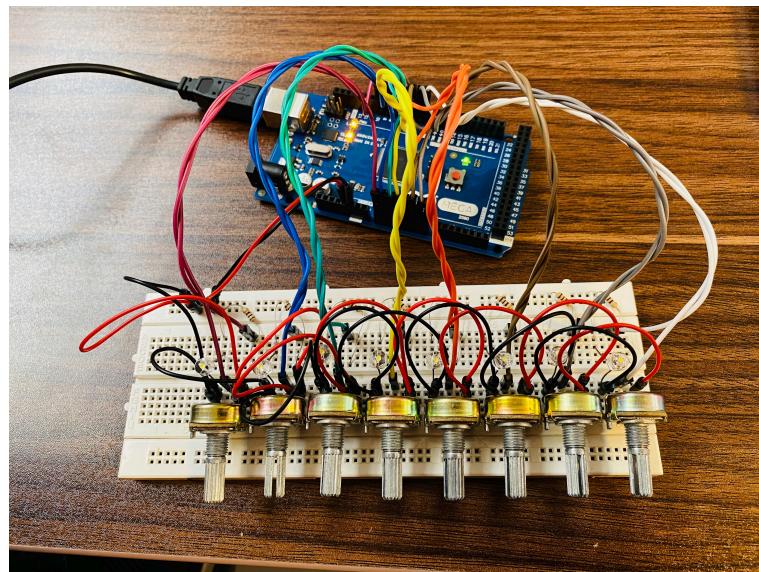


شکل ۴: ساختار برد آردوینو

در شکل زیر نحوه اتصال محصولمان به نقطه‌ای که می‌خواهیم از آن نمونه برداری کنیم و یا تولید سیگنال کنیم را مشاهده می‌کنید.



شکل ۵: نحوه استفاده از یک ورودی و خروجی محصول



شکل ۶: استفاده از تمام ورودی‌ها و خروجی‌های محصول



۲.۳.۲ نرم افزار روی برد سخت افزار

ایده‌ی اولیه‌ی ما این بود که به کمک وقفه‌ها، نمونه‌برداری انجام دهیم و نمونه‌ها را در یک بافر جمع کنیم و در بازه‌های زمانی مشخص، از طریق سریال ارسال کنیم. اما از آنجا که ما می‌خواستیم ۸ ورودی آنالوگ را به صورت همزمان بخوانیم، استفاده از وقفه‌ها معقول به نظر نیامد. زیرا وقفه روی یک ورودی می‌تواند اعمال شود و ما مجبور بودیم در هر بار صدا زده شدن تابع وقفه، تنظیمات را تغییر دهیم تا وقفه روی ورودی بعدی تنظیم شود. پس از مشورت با استاد، در نهایت به این نتیجه رسیدیم که از وقفه‌ها صرف نظر کنیم.

کد نهایی ما به این گونه است که در ابتدا چند متغیر گلوبال تعریف می‌کنیم. اولی تعداد پورت‌های ورودی و خروجی است. دومی لیست شماره پورت‌های PWM به ترتیب مورد استفاده است.

```
1 const int NUM = 8;
2 int pwm_pins[NUM] = {2, 3, 4, 5, 6, 7, 8, 9};
```

سپس تنظیمات اولیه سریال و پورت‌ها را انجام می‌دهیم.

```
1 void setup()
2 {
3     Serial.begin(115200);
4     for (int i = 0; i < NUM; i++)
5         pinMode(pwm_pins[i], OUTPUT);
6 }
```

سپس حلقه‌ی زیر را داریم که هر ۵۰ میلی‌ثانیه نمونه‌برداری کرده و ارسال می‌کند و نمونه‌های دریافتی را نیز روی پایه‌های PWM قرار می‌دهد. پس فرکانس نمونه‌برداری ما ۲۰ هرتز است.

```
1 void loop()
2 {
3     long int t1 = millis();
4
5     // Create analog inputs packet.
6     String out_packet = String("");
7     for (int i = 0; i < NUM; i++)
8     {
9         int val = analogRead(i);
10        out_packet += String(val) + "|";
11    }
12    Serial.println(out_packet);
13
14
15    if (Serial.available() > 40)
16    {
17        // Receive pwm outputs packet.
18        String in_packet = Serial.readStringUntil('\n');
19        for (int i = 0; i < NUM; i++)
20        {
21            int val = in_packet.substring(5 * i, 5 * i + 4).toInt();
```



```
22     analogWrite(pwm_pins[i], val / 4);
23     // analogRead values go from 0 to 1023, analogWrite
24     values from 0 to 255
25   }
26   long int t2 = millis();
27   delay(50 - (t2 - t1));
28 }
```

بخش اول حلقه، نمونه‌های اندازه‌گیری شده را در یک رشته قرار می‌دهد و ارسال می‌کند.

```
1 // Create analog inputs packet.
2 String out_packet = String("");
3 for (int i = 0; i < NUM; i++)
4 {
5     int val = analogRead(i);
6     out_packet += String(val) + "|";
7 }
8 Serial.println(out_packet);
```

بخش دوم نیز بسته دریافتی از کامپیوتر را به اجزایش می‌شکند و به عدد تبدیل می‌کند و سپس روی پایه‌های PWM قرار می‌دهد.

```
1 // Receive pwm outputs packet.
2 String in_packet = Serial.readStringUntil('\n');
3 for (int i = 0; i < NUM; i++)
4 {
5     int val = in_packet.substring(5 * i, 5 * i + 4).toInt();
6     analogWrite(pwm_pins[i], val / 4);
7     // analogRead values go from 0 to 1023, analogWrite
8     values from 0 to 255
9 }
```



۳.۳.۲ نرم افزار روی PC

برای پیاده سازی بخش نرم افزاری این سیستم، به دو بخش اصلی نیاز داریم:

- رابط بین نرم افزار و سخت افزار
- بخش نرم افزاری و نمایش نمودارها

علاوه بر این دو بخش که هر کدام در زیر به تفصیل توضیح داده خواهد شد، یک فایل تحت عنوان main.py وجود دارد که این دو بخش را در یک فایل لود می کند و به عنوان نقطه شروع عمل می کند و برای اجرای برنامه، کافی است این فایل اجرا شود. محتوای این فایل به صورت زیر می باشد:

```
1 from arduino_serial import *
2 from plotter import live_plotter
3
4 # a thread for reading data from arduino and showing saved
5     values on LED
6 start_read_adc_thread()
7 # plot live data
8 live_plotter()
```

جهت اجرای بخش نرم افزاری، به تعدادی کتابخانه نیاز داریم که به شرح زیر در فایل requirements.txt موجودند:

```
1 pyserial
2 matplotlib
```

کتابخانه pyserial برای دسترسی به پورت سریال استفاده می شود.

کتابخانه matplotlib نیز به طور کلی برای رسم نمودار در زبان برنامه نویسی پایتون مورد استفاده قرار می گیرد. به کمک این کتابخانه می توان نمودارهای ثابت، پویا و یا حتی تعاملی ایجاد کرد. در این پژوهه از این کتابخانه برای رسم نمودار زنده از داده ها استفاده خواهیم کرد.



۴.۳.۲ نرم افزار ارتباط با آردوبئینو

این بخش همانند رابطه بین نرم افزار و سخت افزار عمل می کند و می تواند بطور همزمان داده ها را از ورودی دریافت کرده و در خروجی نمایش دهد.

برای اجرای توابع مربوط به این بخش، نیاز داریم تا ابتدا کتابخانه های مربوطه را ایمپورت کنیم:

```
1 import queue
2 import threading
3 import serial
4 import random
5 import time
```

سپس باید متغیر هایی را که نیاز داریم را تعریف کنیم. این متغیرها به شرح زیر هستند:

- تعداد ورودی های و خروجی ها یا NUM
- فرکانس نمونه برداری یا SAMPLE-DELAY
- یک ریسه برای ورودی گرفتن و خروجی دادن همزمان تحت عنوان buffer-lock
- آرایه ای برای ذخیره داده های هر ورودی یا adc-inputs
- آرایه ای از صفحه های برای ذخیره داده های هر خروجی یا pwm-queues

متغیرهای بالا توسط قطعه کد زیر تعریف شده اند:

```
1 # number of channels
2 NUM = 8
3
4 # time between sampling intervals
5 SAMPLE_DELAY = 50 # milli
6
7 # lock for reading and writing data in their respective queues
8 buffer_lock = threading.Lock()
9
10 # arrays of inputs and pwm outputs for each channel
11 adc_inputs = [[] for _ in range(NUM)]
12 pwm_queues = [queue.Queue() for _ in range(NUM)]
```

پس از شروع اجرای برنامه، تابع start-read-adc-thread فراخوانی می شود و با فعال کردن تردی که بالاتر به آن اشاره شد، بطور همزمان داده ها را از ورودی خوانده و در خروجی نمایش می دهد.

```
1 # define a thread for continuously reading and writing data
2     from and to arduino
3 def start_read_adc_thread():
4     # infinite loop to read and write
5         def continuous_read_adc():
```



```
5     ser = serial.Serial(port="/dev/cu.usbmodem14201",
6                           baudrate=115200, timeout=1)
7     while True:
8         read_adc(ser)
9         write_adc(ser)
10
11    # start a thread running above function
12    threading.Thread(target=continuous_read_adc).start()
```

برای خواندن داده ها از تابع `read_adc` استفاده می شود که برای هر کدام از ۸ ورودی داده ها را با فرکانس مشخص شده از پورت سریال می خواند، رشته‌ی خوانده شده را به وسیله‌ی | تکه تکه کرده و در آرایه‌ی مربوط به هر کanal ذخیره می‌کند.

```
1 # read data from input port in arduino
2 def read_adc(ser: serial.SerialBase):
3     try:
4         packet = ser.readline().strip().decode("utf-8")
5         # seperate input data with |
6         samples = packet.split("|")
7         if len(samples) != (NUM + 1):
8             return
9         # get the buffer to save input values into
10        with buffer_lock:
11            for i in range(NUM):
12                if samples[i].isnumeric():
13                    adc_inputs[i].append(int(samples[i]))
14    except Exception as e:
15        print(e)
```

برای نمایش داده ها در خروجی نیز مقادیری که در `pwm-queues` ذخیره شده اند با | به هم وصل شده و در پورت سریال نوشته می شوند. در صورتی که درخواست مشاهده برای کanalی وجود نداشته باشد، عدد صفر را برای آن کanal در نظر می‌گیریم تا LED مربوط به آن خاموش باشد.

```
1 def write_adc(ser: serial.SerialBase):
2     packet = ""
3     # get the buffer to put output values into
4     with buffer_lock:
5         for i in range(NUM):
6             # if no value is available in the queue, so LED is
7             off
8             if pwm_queues[i].empty():
9                 val = 0
10            else:
11                val = pwm_queues[i].get()
12                packet += "{:4}|".format(val)
13
14    ser.write(str.encode("{}\n".format(packet)))
```



۵.۳.۲ نرم افزار رسم نمودار

همزمان با دریافت و ارسال داده ها از ورودی ها و خروجی ها، نمودار آن ها نیز رسم می شوند. این نمودار قابلیت های زیر را دارد:

- رسم نمودار زنده داده های هر ۸ ورودی
- زوم کردن بر روی نمودار دلخواه
- ذخیره داده های ورودی دلخواه با دکمه save
- نمایش مقادیر ذخیره شده با دکمه observe
- انتخاب ورودی های دلخواه برای نمایش جهت راحتی مقایسه

در این بخش نیز ابتدا توابع و ماثوله های مورد نیاز را ایمپورت می کنیم:

```
1 import matplotlib as mpl
2 import matplotlib.pyplot as plt
3 from matplotlib.animation import FuncAnimation
4 from arduino_serial import adc_inputs, pwm_queues, buffer_lock
5 , NUM
6 import numpy as np
7 from matplotlib.widgets import CheckButtons
8 import json, os
```

سپس فرکانس آپدیت نمودارها، رنگ نمودارها، آرایه های مربوطه برای ذخیره و مشاهده و فعال بودن یا نبودن را مقداردهی می کنیم و فقط نمودار مربوط به ورودی ۱ را در ابتدای کار فعال می کنیم تا نمایش داده شود.

```
1 # Time between graph frames in milli seconds.
2 INTERVAL = 200
3
4 # plots colors
5 colors = ["salmon", "grey", "yellow", "lightgreen", "orange",
6 "pink", "cyan", "plum"]
7
8 # initialization of buttons related to different channels
9 save_buttons = [] * NUM
10 transfer_buttons = [] * NUM
11 visibility_buttons = [] * NUM
12
13 # index of the element to save values after that
14 save_start_index = [0] * NUM
15
16 # a list of boolean values, True if the corresponding channel
17 is visible
```



```
16 visible_diagrams = [True] # only first diagram is visible in
   the beginning
17 visible_diagrams.extend([False] * (NUM - 1))
18
19 # initializing the figure to plot
20 fig = plt.figure(figsize=(10, 8))
21 axes = [None] * NUM
22 axes[0] = fig.add_subplot(label=str(0))
23 fig.tight_layout()
```

با شروع اجرای برنامه و صدا زده شدن تابع `live_plotter` ابتدا تابع `Animate` برای رسم نمودار زنده ورودی‌هایی که توسط کاربر فعلی شده اند و آپدیت آن‌ها با آمدن مقادیر جدید شروع به کار می‌کند. سپس برای هر کدام از ۸ ورودی و خروجی لیبل‌ها، محل قرارگیری آن‌ها، رنگ هر کدام و فایل مربوطه برای ذخیره داده‌ها مشخص می‌شود و پس از افزودن به لیست دکمه‌های مربوطه، فانکشنالیتی مربوط به آن‌ها نیز مشخص می‌شود.

```
1 # plot live data, define buttons of channels and their
   functionalities
2 def live_plotter():
3     ani = FuncAnimation(plt.gcf(), animate, interval=INTERVAL)
4
5     for i in range(NUM):
6         save_label = ["save{}".format(i + 1)]
7         observe_label = ["observe{}".format(i + 1)]
8         visible_label = ["visible{}".format(i + 1)]
9
10        # x position, y position, width and height
11        save_ax = plt.axes([np.linspace(0.05, 0.9, 8)[i],
12                           0.16, 0.1, 0.08], frame_on=False)
13        observe_ax = plt.axes(
14            [np.linspace(0.05, 0.9, 8)[i], 0.08, 0.1, 0.08],
15            frame_on=False
16        )
17        visible_ax = plt.axes([np.linspace(0.05, 0.9, 8)[i],
18                               0, 0.1, 0.08], frame_on=False)
19        save_buttons.append(CheckButtons(save_ax, save_label,
20                                         [False]))
21        transfer_buttons.append(CheckButtons(observe_ax,
22                                              observe_label,
23                                              [False]))
24        visibility_buttons.append(CheckButtons(visible_ax,
25                                     visible_label,
26                                     [True if i == 0 else False]))
27
28        # set colors for buttons
29        save_buttons[-1].rectangles[0].set_facecolor(colors[i])
30    transfer_buttons[-1].rectangles[0].set_facecolor(
```



```
23     colors[i])
24         visibility_buttons[-1].rectangles[0].set_facecolor(
25             colors[i])
26
27     # clear content of values file
28     save_file_name = "values_{}.txt".format(i + 1)
29     open(save_file_name, "w").close()
30
31     # buttons functionalities
32     save_buttons[i].on_clicked(save(i))
33     transfer_buttons[i].on_clicked(transfer_to_arduino(i))
34     visibility_buttons[i].on_clicked(set_visible(i))
35
36     # add some space between subplots and to the right of the
37     # rightmost ones
38     plt.subplots_adjust(bottom=0.24, hspace=1)
39     plt.show()
```

تابع animate که بالاتر به آن اشاره شد، برای هر کدام از ۸ ورودی در صورتی که فعال شده باشند، داده‌ها را از بافر ورودی می‌خواند و ۴۰۰ مقدار نهایی را در نمودار نمایش می‌دهد.

```
1 # animating each input data
2 def animate(_):
3     for i in range(NUM):
4         if visible_diagrams[i]:
5             with buffer_lock:
6                 input_copy = adc_inputs[i][-400:].copy()
7                 if len(input_copy) == 0:
8                     continue
9                 axes[i].cla()
10                axes[i].set_ylim(-100, 1100)
11                axes[i].set_xlim(0, 410)
12                axes[i].plot(input_copy, color=colors[i])
13                axes[i].title.set_text("{}th Input".format(i + 1))
```

یکی از امکانات نمودارها، قابلیت انتخاب ورودی دلخواه برای ذخیره داده‌های آن می‌باشد. تابع `Save` منظور می‌باشد و یک تابع wrapper است. عملکرد کلی این تابع به این صورت است که در صورتی که در صورتی که دکمه Save نمودار مربوطه فعال باشد، مقادیر جدید را به انتهای فایل اضافه می‌کند و در غیر این صورت تعداد ورودی‌های تا آن لحظه را ذخیره می‌کند تا بعداً مقادیر جدیدتر را با شروع از آن ایندکس بخواند.

```
1 # wrapper function for saving values of the input channels
2 def save(channel):
3     def func(label):
4         save_file_name = f"values_{int(channel) + 1}.txt"
5
6         # button pressed to start saving
```



```
7     if save_buttons[channel].get_status()[0]:
8         save_start_index[channel] = lenadc_inputs[channel]
9
10    # button pressed to stop saving values and write saved
11    # ones to the file
12    else:
13        vals = []
14        f = open(save_file_name, "r")
15        if os.stat(save_file_name).st_size != 0:
16            vals.extend(json.loads(f.read()))
17        f.close()
18        vals.extend(adc_inputs[channel][save_start_index[channel]:])
19        f = open(save_file_name, "w")
20        f.write(json.dumps(vals))
21        f.close()
22
23    return func
```

قابلیت بعدی، امکان مشاهده مقادیر ذخیره شده در خروجی است و به این صورت عمل می کند که با فشردن دکمه observe مقادیری را که در فایل مربوطه ذخیره شده اند در صفحه خروجی مربوطه قرار می دهد تا LED بسته به آن مقدار روشناییش تغییر کند و در غیر این صورت مقدار صفحه خروجی مربوطه پاک می شود تا مقادیری نمایش داده نشود و LED خاموش بماند.

```
1 # wrapper function to transfer saved values to arduino to show
2     them on the LEDs
3 def transfer_to_arduino(channel):
4     def func(label):
5         save_file_name = f"values_{int(channel) + 1}.txt"
6
7         # button pressed to start transferring to arduino
8         if transfer_buttons[channel].get_status()[0]:
9             vals = []
10            f = open(save_file_name, "r")
11            if os.stat(save_file_name).st_size != 0:
12                vals.extend(json.loads(f.read()))
13            f.close()
14            with buffer_lock:
15                pwm_queues[channel].queue.clear()
16                # send saved values to pwm_queues
17                [pwm_queues[channel].put(i) for i in vals]
18
19         # button pressed to stop transferring to arduino
20         else:
21             with buffer_lock:
```



```
21             # clear pwm_queues to show nothing on the LEDs
22             (turn them off)
23             pwm_queues[channel].queue.clear()
24
25         return func
```

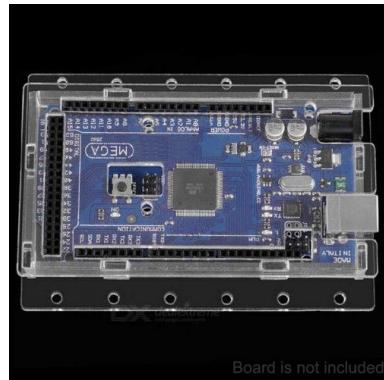
قابلیت آخر set-visible نام دارد و برای فعال کردن نمودار ورودی‌ها بکار می‌رود و در صورتی که دکمه فعال شود، نمودار آن به صفحه اضافه می‌شود (برحسب شماره ورودی‌ها که ورودی ۱ بالاتر از ورودی ۲ نمایش داده می‌شود و الی آخر) و در صورت غیرفعال شدن، دیگر نمودار مربوطه در صفحه نشان داده نمی‌شود و فقط سایر نمودارهای فعال نمایش داده می‌شوند.

```
1 # set the diagram of the channel to be visible
2 def set_visible(channel):
3     def func(label):
4         # button pressed to show diagram of the channel
5         if visibility_buttons[channel].get_status()[0]:
6             visible_diagrams[channel] = True
7             axes[channel] = fig.add_subplot(sum(
8                 visible_diagrams), 1, sum(visible_diagrams), label=str(
9                 channel))
10            for i in range(len(visible_diagrams)):
11                if visible_diagrams[i]:
12                    axes[i].change_geometry(sum(
13                     visible_diagrams), 1, sum(visible_diagrams[:i + 1]))
14
15            # button pressed to hide diagram of the channel
16        else:
17            visible_diagrams[channel] = False
18            fig.delaxes(axes[channel])
19            axes[channel] = None
20            for i in range(len(visible_diagrams)):
21                if visible_diagrams[i]:
22                    axes[i].change_geometry(sum(
23                     visible_diagrams), 1, sum(visible_diagrams[:i + 1]))
24
25        return func
```



۴.۲ طراحی جعبه

برای طراحی جعبه، از جعبه‌های آماده استفاده می‌کنیم. جنس این جعبه‌ها از پلکسی است. دلیل استفاده از جعبه‌های آماده این است که به دلیل تولید انبوه، قیمت بسیار مناسبی دارند. به عنوان مثال می‌توان از نمونه‌ی زیر استفاده کرد. آدرس خریداری تمام قطعات در بخش انتهایی گزارش آورده شده است.



شکل ۷: جعبه آردوئینو

سپس تعدادی کابل سوسناری می‌خریم و آن‌ها را نصف می‌کنیم. نصفه‌ی لخت را در پین‌های مدنظر در آردوئینو قرار می‌دهیم و از سمت گیره‌دار برای اتصالات خارجی استفاده می‌کنیم. در صورتی که سر لخت سیم‌ها در پین‌هدرهای آردوئینو وارد نشود، می‌توان از سیم‌های استخوانی که ضخامت کمتری دارند استفاده کرد.

گیره‌های سوسناری برای محصول ما بسیار مناسب هستند. زیرا قابلیت اتصال بسیار راحت به نقاط مدنظر ما را دارند.



Photo by CafeRobot

شکل ۸: کابل‌ها و گیره‌های سوسناری



۳ جمع‌بندی

۱.۳ هزینه‌ی تولید محصول

ردیف	نام سخت‌افزار	قیمت (به تومان)
۱	آردوینو مگا	۴۳۵۸۰۰
۲	عدد ولوم ۸	۳۳۶۰۰
۳	آداتور ۱۲ ولت ۲ آمپر	۵۳۸۰۰
۴	۲ عدد بردبورد ۸۳۰ حفره‌ای	۱۱۲۰۰۰
۵	سیم‌های بردبورد	۵۰۰۰۰
۶	۸ عدد LED و مقاومت	۸۰۰۰
۷	کابل و کانکتور ۴۰ تایی فلت و برد واسط	۶۵۰۰۰
۸	برش لیزری پلکسی و یا طراحی ساده با چوب	۵۰۰۰۰
۹	همه	۸۰۸۲۰۰

جدول ۳: هزینه‌ی برآورد شده‌ی سخت‌افزارهای لازم

ردیف	نام سخت‌افزار	قیمت (به تومان)
۱	آردوینو مگا	۶۲۸۰۰۰
۲	عدد ولوم ۸	۳۹۲۰۰
۳	۱ عدد بردبورد ۸۳۰ حفره‌ای	۵۶۰۰۰
۴	سیم‌های بردبورد	۵۰۰۰۰
۵	۸ عدد LED و مقاومت	۸۰۰۰
۶	جعبه آردوئینو	۳۶۱۰۰
۷	کابل و گیره سوسماری	۳۷۹۰۰
۸	همه	۸۵۵۲۰۰

جدول ۴: هزینه‌ی نهایی شده‌ی سخت‌افزارهای لازم

همانطور که در جداول بالا مشخص است، تخمین اولیه ما از قیمت محصول حدود ۸۰۰ هزار تومان بود، اما در نهایت هزینه‌ی نهایی حدود ۸۵۰ هزار تومان شد، که فاصله‌ی چندانی با تخمین اولیه ندارد.



۲.۳ تست‌های انجام‌شده

برای اینکه بتوانیم درستی عملکرد آردوینو و رسم نمودار را چک کنیم، توابعی معادل با توابعی که در فایل-arduino موجود هستند و وظیفه ارتباط با آردوینو را دارند، تعریف می‌کنیم. این تابع جدید که در کد ما تحت عنوان FAKE کامنت گذاری شده‌اند، به صورت زیر می‌باشند:

```
1 # create random inputs to plot
2 def read_adc():
3     time.sleep(SAMPLE_DELAY / 1000)
4     with buffer_lock:
5         for i in range(NUM):
6             speed_s[i] += random.randint(-3, 3)
7             speed_s[i] = min(speed_s[i], 6)
8             speed_s[i] = max(speed_s[i], -6)
9             position_s[i] += speed_s[i]
10            position_s[i] = min(position_s[i], 1024)
11            position_s[i] = max(position_s[i], 0)
12            adc_inputs[i].append(position_s[i])
13            pwm_queues[i].put(position_s[i])
```

تابع بالا وظیفه تولید ورودی را بر عهده دارد تا بتوان با استفاده از این مقادیر نمودار آن‌ها را رسم کرد.

```
1 # print saved values in the terminal, in case of having no
2 # input, prints 0
3 def write_adc():
4     with buffer_lock:
5         for i in range(NUM):
6             if pwm_queues[i].empty():
7                 val = 0
8             else:
9                 val = pwm_queues[i].get()
print('Output {}: {}'.format(i, val))
```

این تابع برای نمایش ورودی‌های ذخیره شده بکار می‌رود تا بتوان درستی عملکرد ذخیره و نمایش را مشاهده کرد.

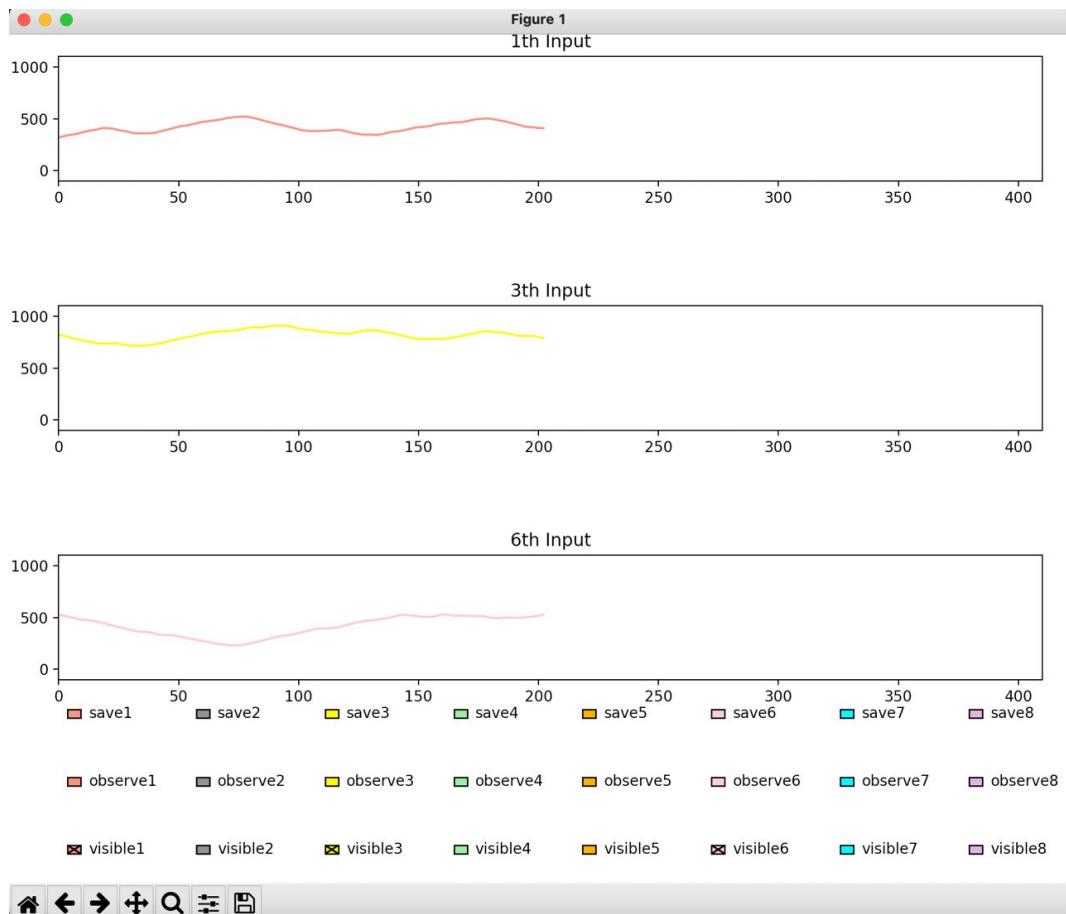
```
1 # define a thread for continuously reading and writing data
2 def start_read_adc_thread():
3     def continuous_read_adc():
4         while True:
5             read_adc()
6             write_adc()
7
8     threading.Thread(target=continuous_read_adc).start()
```

نهایتاً تابع بالا نیز، دو بخش قبل را در کنار هم قرار می‌دهم و توسط یک ترد و در حلقه بی‌نهایت، ورودی‌ها را تولید کرده و در خروجی چاپ می‌کند. از طریق نمودار رسم شده نیز درستی عملکرد سیستم سنجیده می‌شود.



۳.۳ شکل نهایی محصول

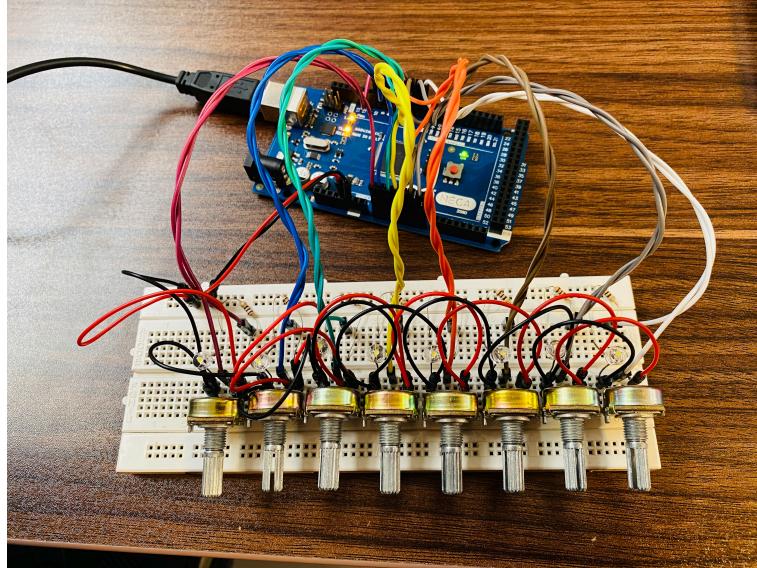
در تصویر زیر مثالی از خروجی اجرای محصول را مشاهده می‌کنید.



شکل ۹: نمودار رسم شده از داده‌های آنالوگ دریافتی ورودی‌های ۱ و ۳ و ۶



در شکل زیر بخش سخت‌افزاری محصول را مشاهده می‌کنید که برای آزمودن آن، از ۸ عدد ولوم و ۸ عدد دیود نوری استفاده کرده‌ایم.



شکل ۱۰: بخش سخت‌افزاری محصول