



## آزمایشگاه سخت افزار

فانکشن ژنراتور و اسیلوسکوپ نرم افزاری  
گزارش سوم

دانشکده مهندسی کامپیوتر  
دانشگاه صنعتی شریف  
نیم سال دوم ۰۱ - ۰۰

استاد:  
جناب آقای دکتر اجلالی

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



## فهرست مطالب

۱	مراحل انجام بخش سوم پروژه	۲
۱.۱	سخت افزار	۲
۱.۱.۱	مدار سخت افزار	۲
۲.۱.۱	نرم افزار ریخته شده روی سخت افزار	۲
۲.۱	نرم افزار روی PC	۴
۱.۲.۱	نرم افزار ارتباط با آردوینو	۴
۲.۲.۱	نرم افزار رسم نمودار	۷

## فهرست تصاویر

۱	مدار با ۸ کانال ورودی و خروجی	۲
۲	نمودار رسم شده از داده های آنالوگ دریافتی	۱۱

## فهرست برنامه ها

۳	Hardware/R۳/sketch ۰.ino	۳
۴	Software/R۳/arduino_serial.py	۴
۷	Software/R۳/plotter.py	۷

## ۱ مراحل انجام بخش سوم پروژه

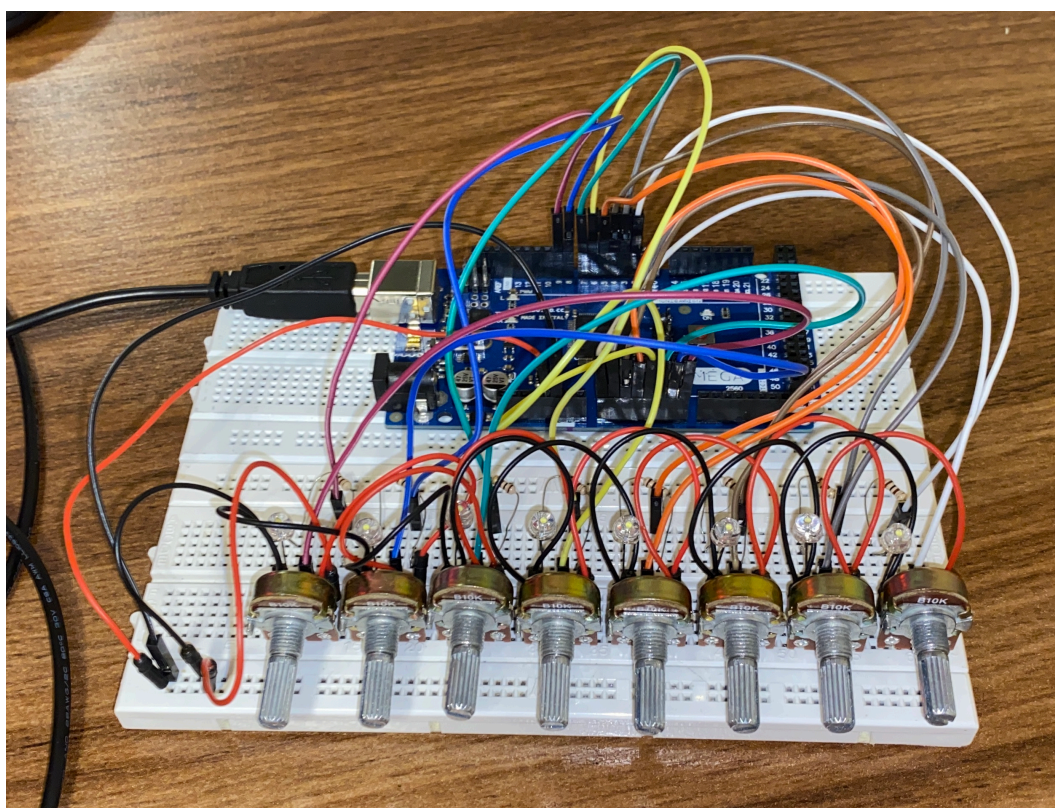
پروژه ما از دو بخش تشکیل می‌شود: بخش نمونه‌برداری (سخت‌افزاری) و بخش رسم نمودار (نرم‌افزاری). در زیر به توضیح کارهای انجام‌شده‌ی سخت‌افزار و نرم‌افزار به صورت جداگانه می‌پردازیم.

### ۱.۱ سخت‌افزار

سخت‌افزار ما از یک برد آردوینو به عنوان میکروکنترلر، مداری در کنار آردوینو، و کد ریخته‌شده روی آردوینو تشکیل شده‌است.

#### ۱.۱.۱ مدار سخت‌افزار

مدار بخش قبل را ۸ برابر تکرار کردیم.



شکل ۱: مدار با ۸ کانال ورودی و خروجی

#### ۲.۱.۱ نرم‌افزار ریخته‌شده روی سخت‌افزار

کد را به گونه‌ای تغییر دادیم که توان ارسال و دریافت همزمان ۸ ورودی و خروجی را داشته باشد. کد به شرح زیر است.



```
1 const int NUM = 8;
2 int pwm_pins[NUM] = {2, 3, 4, 5, 6, 7, 8, 9};
3
4 void setup()
5 {
6     Serial.begin(115200);
7     for (int i = 0; i < NUM; i++)
8         pinMode(pwm_pins[i], OUTPUT);
9 }
10
11 void loop()
12 {
13     long int t1 = millis();
14
15     String out_packet = String("");
16     for (int i = 0; i < NUM; i++)
17     {
18         int val = analogRead(i);
19         out_packet += String(val) + "|";
20     }
21     Serial.println(out_packet);
22
23     if (Serial.available() > 40)
24     {
25         String in_packet = Serial.readStringUntil('\n');
26         for (int i = 0; i < NUM; i++)
27         {
28             int val = in_packet.substring(5 * i, 5 * i + 4).toInt();
29             analogWrite(pwm_pins[i], val / 4);
30             // analogRead values go from 0 to 1023, analogWrite
31             // values from 0 to 255
32         }
33     }
34     long int t2 = millis();
35     delay(50 - (t2 - t1)); // TODO: use clock
36 }
```

طریقه‌ی ارسال پیام از آردوینو به نرم افزار این گونه است که مقادیر خوانده شده از ورودی آنالوگ با | به هم چسبانده شده و در پورت سریال چاپ می شوند.

مقادیر سیو شده که درخواست مشاهده‌ی آن‌ها ارسال می شود نیز به همان صورت هستند و برای تبدیل آن‌ها به عدد از زیررشته‌ای از آنچه در پورت سریال چاپ شده استفاده می کنیم.



## ۲.۱ نرم افزار روی PC

## ۱.۲.۱ نرم افزار ارتباط با آردوینو

کد را به گونه ای تغییر دادیم که توان ارسال و دریافت همزمان ۸ ورودی و خروجی را داشته باشد.  
کد به شرح زیر است.

```
1 import queue
2 import threading
3 import serial
4 import random
5 import time
6
7 NUM = 8
8 SAMPLE_DELAY = 50 # milli
9 buffer_lock = threading.Lock()
10 adc_inputs = [[] for _ in range(NUM)]
11 pwm_queues = [queue.Queue() for _ in range(NUM)]
12
13
14 ##### REAL #####
15 def read_adc(ser: serial.SerialBase):
16     try:
17         packet = ser.readline().strip().decode("utf-8")
18         samples = packet.split('|')
19         if len(samples) != (NUM + 1):
20             return
21         with buffer_lock:
22             for i in range(NUM):
23                 if samples[i].isnumeric():
24                     adc_inputs[i].append(int(samples[i]))
25     except Exception as e:
26         print(e)
27
28
29 def write_adc(ser: serial.SerialBase):
30     packet = ''
31     with buffer_lock:
32         for i in range(NUM):
33             if pwm_queues[i].empty():
34                 val = 0
35             else:
36                 val = pwm_queues[i].get()
37             packet += '{:4}|'.format(val)
```



```
38     ser.write(str.encode('{}\n'.format(packet)))
39
40
41 def start_read_adc_thread():
42     def continuous_read_adc():
43         ser = serial.Serial(port='/dev/cu.usbmodem14201',
44                               baudrate=115200, timeout=1)
45         while True:
46             read_adc(ser)
47             write_adc(ser)
48
49     threading.Thread(target=continuous_read_adc).start()
50
51 ##### FAKE #####
52 # position_s = [random.randint(100, 900) for _ in range(NUM)]
53 # speed_s = [random.randint(-5, +5) for _ in range(NUM)]
54
55
56 # def read_adc():
57 #     time.sleep(SAMPLE_DELAY / 1000)
58 #     with buffer_lock:
59 #         for i in range(NUM):
60 #             speed_s[i] += random.randint(-3, 3)
61 #             speed_s[i] = min(speed_s[i], 6)
62 #             speed_s[i] = max(speed_s[i], -6)
63 #             position_s[i] += speed_s[i]
64 #             position_s[i] = min(position_s[i], 1024)
65 #             position_s[i] = max(position_s[i], 0)
66 #             adc_inputs[i].append(position_s[i])
67 #             pwm_queues[i].put(position_s[i])
68
69
70 # def write_adc():
71 #     with buffer_lock:
72 #         for i in range(NUM):
73 #             if pwm_queues[i].empty():
74 #                 val = 0
75 #             else:
76 #                 val = pwm_queues[i].get()
77 #             print('Output {}: {}'.format(i, val))
78
79
80 # def start_read_adc_thread():
81 #     def continuous_read_adc():
```



```
82 #         while True:
83 #             read_adc()
84 #             write_adc()
85
86 #     threading.Thread(target=continuous_read_adc).start()
```

در تابع `adc read` آنچه در پورت سریال چاپ شده را می‌خوانیم، رشته‌ی خوانده شده را به وسیله‌ی `|` تکه تکه می‌کنیم و در آرایه‌ی مربوط به هر کانال ذخیره می‌کنیم.

در تابع `adc write` نیز اعداد ذخیره شده را با `|` به هم وصل می‌کنیم و در پورت سریال می‌نویسیم. در صورتی که درخواست مشاهده برای کانالی وجود نداشته باشد، عدد صفر را برای آن کانال در نظر می‌گیریم تا `LED` مربوط به آن خاموش باشد.



## ۲.۲.۱ نرم افزار رسم نمودار

در این بخش نرم افزار را طوری تغییر دادیم تا قابلیت رسم نمودار زنده برای هر ۸ کانال را داشته باشد. همچنین بتواند برای هر یک از این ۸ کانال در صورت فشار دادن دکمه‌ی save مربوط به آن‌ها داده‌ها را ذخیره کند. همچنین در صورت فشار دادن دکمه‌ی observe مربوط به هر کانال داده‌های مربوط به آن را بر روی LED مختص آن کانال نمایش دهد.

کد به شرح زیر است.

```
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
9
10 NUM = 4 # TODO: performance is not good enough for 8 charts
11 INTERVAL = 200 # Time between graph frames in milli seconds.
12
13 fig, axes = plt.subplots(2, 4, figsize=(15, 6))
14 fig.tight_layout()
15 colors = ["red", "black", "yellow", "green", "orange", "pink",
16     "cyan", "purple"]
17
18 save_buttons = [] * NUM
19 transfer_buttons = [] * NUM
20 save_start_index = [0] * NUM
21
22 def save(channel):
23     def func(label):
24         save_file_name = f"values_{int(channel) + 1}.txt"
25         # print("-----",
26         save_file_name, "-----")
27         if save_buttons[channel].get_status()[0]:
28             save_start_index[channel] = len(adc_inputs[channel])
29     ]
30     else:
31         vals = []
32         f = open(save_file_name, "r")
33         if os.stat(save_file_name).st_size != 0:
34             vals.extend(json.loads(f.read()))
35         f.close()
```





```
33         vals.extend adc_inputs[channel][save_start_index[
channel]:])
34         f = open(save_file_name, "w")
35         f.write(json.dumps(vals))
36         f.close()
37
38     return func
39
40
41 def transfer_to_arduino(channel):
42     def func(label):
43         save_file_name = f"values_{int(channel) + 1}.txt"
44         if transfer_buttons[channel].get_status()[0]:
45             vals = []
46             f = open(save_file_name, "r")
47             if os.stat(save_file_name).st_size != 0:
48                 vals.extend(json.loads(f.read()))
49             f.close()
50             with buffer_lock:
51                 pwm_queues[channel].queue.clear()
52                 [pwm_queues[channel].put(i) for i in vals]
53         else:
54             with buffer_lock:
55                 pwm_queues[channel].queue.clear()
56
57     return func
58
59
60 # plot live data
61 def live_plotter():
62     # set buttons x and y
63     save_xy = [
64         (0.185, 0.75),
65         (0.43, 0.75),
66         (0.675, 0.75),
67         (0.92, 0.75),
68         (0.185, 0.25),
69         (0.43, 0.25),
70         (0.675, 0.25),
71         (0.92, 0.25),
72     ]
73     observe_xy = [
74         (0.185, 0.7),
75         (0.43, 0.7),
76         (0.675, 0.7),
```



```
77         (0.92, 0.7),
78         (0.185, 0.2),
79         (0.43, 0.2),
80         (0.675, 0.2),
81         (0.92, 0.2),
82     ]
83
84     ani = FuncAnimation(plt.gcf(), animate, interval=INTERVAL)
85
86     for i in range(NUM):
87         save_label = ["save"]
88         observe_label = ["observe"]
89
90         # x position, y position, width and height
91         save_ax = plt.axes([save_xy[i][0], save_xy[i][1], 0.1,
100         0.1], frame_on=False)
92         observe_ax = plt.axes(
93             [observe_xy[i][0], observe_xy[i][1], 0.1, 0.1],
100         frame_on=False
101         )
94
95         save_buttons.append(CheckButtons(save_ax, save_label,
100         [False]))
96         transfer_buttons.append(CheckButtons(observe_ax,
100         observe_label, [False]))
97
98         # clear content of values file
99         save_file_name = "values_{}.txt".format(i + 1)
100         open(save_file_name, "w").close()
101
102         # buttons functionalities
103         save_buttons[i].on_clicked(save(i))
104         transfer_buttons[i].on_clicked(transfer_to_arduino(i))
105
106         # add some space between subplots and to the right of the
107         # rightmost ones
108         plt.subplots_adjust(wspace=0.6, right=0.92)
109         plt.show()
110
111     # animating each input data
112     def animate(_):
113         for i in range(NUM):
114             with buffer_lock:
115                 # input_copy = adc_inputs[i].copy()
```



```

117         input_copy = adc_inputs[i][-400:].copy()
118         if len(input_copy) == 0:
119             continue
120         axes[i // 4, i % 4].cla()
121         axes[i // 4, i % 4].set_ylim(-100, 1100)
122         # axes[i // 4, i % 4].set_xlim(
123         #     0, np.power(np.e, int(np.log(len(input_copy)))) +
124         1)
125         # )
126         axes[i // 4, i % 4].set_xlim(0, 410)
127         axes[i // 4, i % 4].plot(input_copy, color=colors[i])
128         axes[i // 4, i % 4].title.set_text("{}th Input".format
129         (i + 1))

```

به منظور نمایش اطلاعات هر ۸ کانال صفحه را به ۸ قسمت چهار در دو تقسیم کردیم. در هر یک از این قسمت‌ها نمودار زنده‌ی داده‌های آن کانال به همراه دکمه‌های مربوط به سیو و مشاهده‌ی آن کانال قرار دارد.

دکمه‌های سیو و ذخیره عملکردی کاملاً مشابه عملکرد دکمه‌های سیو و مشاهده‌ی فاز قبل دارند، لذا نیازی به توضیح مجدد آن‌ها نیست. از آنجایی که عملکرد این دکمه‌ها کاملاً یکسان است و صرفاً برای کانال‌های مختلف این کار را انجام می‌دهند از دکوراتور پایتون برای ساختن تابع‌هایی یکسان با ورودی‌های مختلف استفاده کردیم.

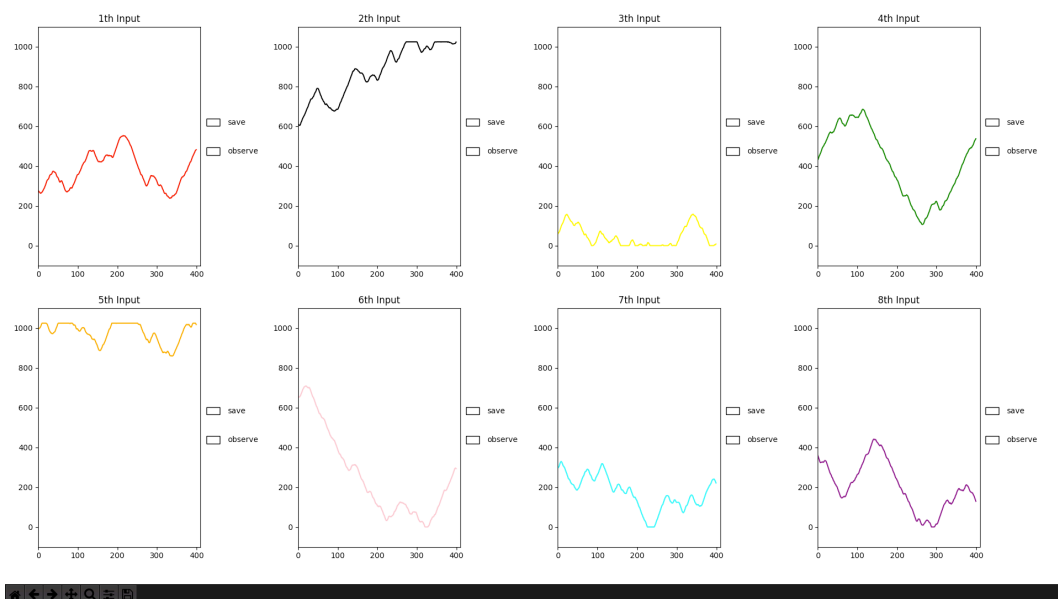
داده‌های مربوط به هر کانال را در مثلاً فایلی به نام ۲ - values ذخیره می‌کنیم که این مثال بیانگر داده‌های ذخیره‌شده‌ی کانال دوم هستند.

برای کانال‌های مختلف از رنگ‌های مختلف برای کشیدن نمودارشان استفاده می‌کنیم. به این ترتیب که نمودار اول قرمز، نمودار دوم سیاه، نمودار سوم زرد، نمودار چهارم سبز، نمودار پنجم نارنجی، نمودار ششم صورتی، نمودار هفتم آبی و نمودار هشتم بنفش است.

در تابع `plotter live` ابتدا مختصات مربوط به هر دکمه‌ی سیو و باتن را در آرایه‌ای مشخص می‌کنیم. سپس دکمه‌ها را مقداردهی می‌کنیم و آن‌ها را در آرایه‌ای ذخیره می‌کنیم، در انتها از خروجی توابع `save` و `transfer` برای مشخص کردن عملکرد دکمه‌ها استفاده می‌کنیم.

تغییر آخری که در این بخش نسبت به فاز قبل دادیم این است که برای شلوغ نشدن نمودارها فقط ۴۰۰ داده‌ی آخر هر کانال را نمایش می‌دهیم.

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



شکل ۲: نمودار رسم شده از داده‌های آنالوگ دریافتی

فیلم مربوط به اجرای این فاز را می‌توانید در اینجا مشاهده کنید.