آزمایشگاه سخت‌افزار

فانکشن ژنراتور و اسیلوسکوپ نرم افزاری

گزارش سوم

دانشکده مهندسی کامپیوتر
دانشگاه صنعتی شریف
نیم سال دوم ۰۱ـ۰۰

استاد:
جناب آقای دکتر اجلالی

اعضای تیم :
محمدمهدی‌جراحی ـ ۹۷۱۰۵۸۴۴
نگین‌جعفری ـ ۹۷۱۰۵۸۵۵
مهساامانی ـ ۹۷۱۰۵۷۶۹

فهرست مطالب

فهرست تصاویر

فهرست برنامه‌ها

۱ مراحل انجام بخش سوم پروژه

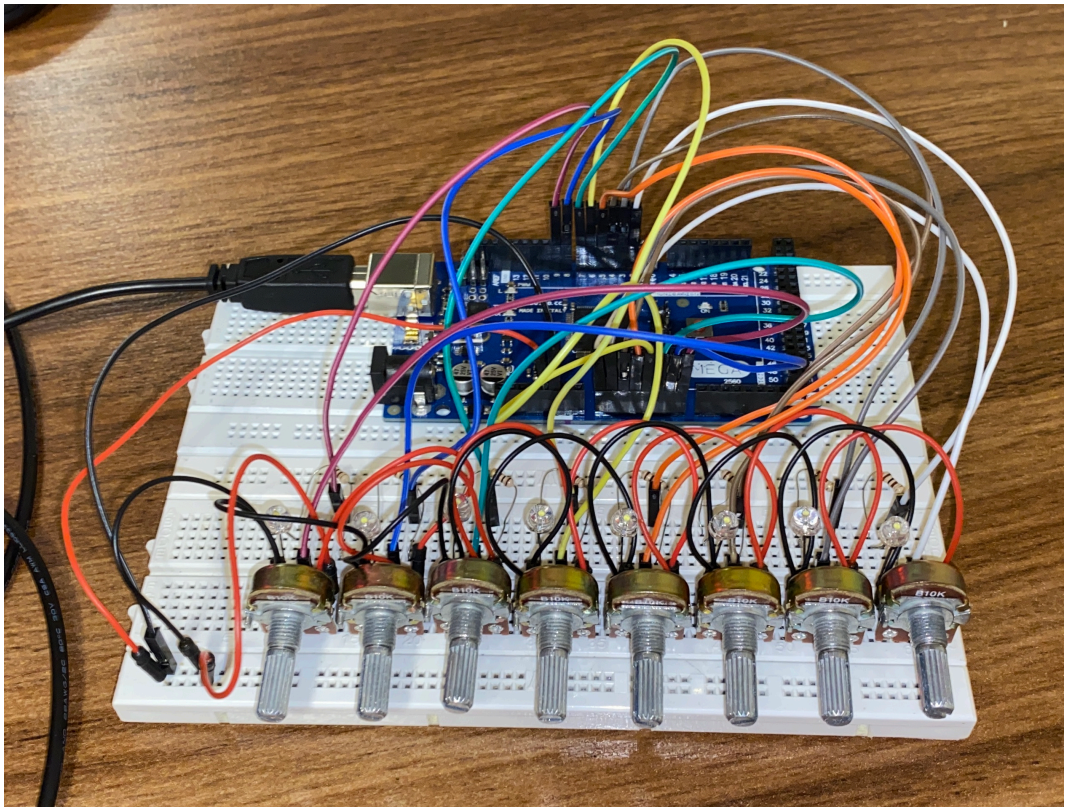
پروژه ما از دو بخش تشکیل می‌شود: بخش نمونه‌برداری (سخت‌افزاری) و بخش رسم نمودار (نرم‌افزاری).
در زیر به توضیح کارهای انجام‌شده‌ی سخت‌افزار و نرم‌افزار به صورت جداگانه می‌پردازیم.

۱.۱ سخت‌افزار

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

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

void setup()
{
  Serial.begin(115200);
  for (int i = 0; i < NUM; i++)
    pinMode(pwm_pins[i], OUTPUT);
}

void loop()
{
  long int t1 = millis();

  String out_packet = String("");
  for (int i = 0; i < NUM; i++)
  {
    int val = analogRead(i);
    out_packet += String(val) + "|";
  }
  Serial.println(out_packet);

  if (Serial.available() > 40)
  {
    String in_packet = Serial.readStringUntil('\n');
    for (int i = 0; i < NUM; i++)
    {
      int val = in_packet.substring(5 * i, 5 * i + 4).toInt();
      analogWrite(pwm_pins[i], val / 4);
      // analogRead values go from 0 to 1023, analogWrite values from 0 to 255
    }
  }
  long int t2 = millis();
  delay(50 - (t2 - t1)); // TODO: use clock
}
```

طریقه‌ی ارسال پیام از آردوئینو به نرم افزار این‌گونه است که مقادیر خوانده‌شده از ورودی آنالوگ با | به هم چسبانده شده و در پورت سریال چاپ می‌شوند.

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

۲.۱ نرم‌افزار روی PC

۱.۲.۱ نرم‌افزار ارتباط با آردوئینو

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

```
import queue
import threading
import serial
import random
import time

NUM = 8
SAMPLE_DELAY = 50  # milli
buffer_lock = threading.Lock()
adc_inputs = [[] for _ in range(NUM)]
pwm_queues = [queue.Queue() for _ in range(NUM)]


####  REAL  ####
def read_adc(ser: serial.SerialBase):
    try:
        packet = ser.readline().strip().decode("utf-8")
        samples = packet.split('|')
        if len(samples) != (NUM + 1):
            return
        with buffer_lock:
            for i in range(NUM):
                if samples[i].isnumeric():
                    adc_inputs[i].append(int(samples[i]))
    except Exception as e:
        print(e)


def write_adc(ser: serial.SerialBase):
    packet = ''
    with buffer_lock:
        for i in range(NUM):
            if pwm_queues[i].empty():
                val = 0
            else:
                val = pwm_queues[i].get()
            packet += '{:4}|'.format(val)
```

```
    ser.write(str.encode('{}\n'.format(packet)))


def start_read_adc_thread():
    def continuous_read_adc():
        ser = serial.Serial(port='/dev/cu.usbmodem14201', baudrate=115200, timeout=1)
        while True:
            read_adc(ser)
            write_adc(ser)

    threading.Thread(target=continuous_read_adc).start()


####  FAKE   ####
# position_s = [random.randint(100, 900) for _ in range(NUM)]
# speed_s = [random.randint(-5, +5) for _ in range(NUM)]


# def read_adc():
#     time.sleep(SAMPLE_DELAY / 1000)
#     with buffer_lock:
#         for i in range(NUM):
#             speed_s[i] += random.randint(-3, 3)
#             speed_s[i] = min(speed_s[i], 6)
#             speed_s[i] = max(speed_s[i], -6)
#             position_s[i] += speed_s[i]
#             position_s[i] = min(position_s[i], 1024)
#             position_s[i] = max(position_s[i], 0)
#             adc_inputs[i].append(position_s[i])
#             pwm_queues[i].put(position_s[i])


# def write_adc():
#     with buffer_lock:
#         for i in range(NUM):
#             if pwm_queues[i].empty():
#                 val = 0
#             else:
#                 val = pwm_queues[i].get()
#             print('Output {}: {}'.format(i, val))


# def start_read_adc_thread():
#     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 نرم‌افزار رسم نمودار

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

کد به شرح زیر است.

```
import matplotlib as mpl
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
from arduino_serial import adc_inputs, pwm_queues, buffer_lock, NUM
import numpy as np
from matplotlib.widgets import CheckButtons
import json, os

NUM = 4  # TODO: performance is not good enough for 8 charts
INTERVAL = 200  # Time between graph frames in milli seconds.

fig, axes = plt.subplots(2, 4, figsize=(15, 6))
fig.tight_layout()
colors = ["red", "black", "yellow", "green", "orange", "pink", "cyan", "purple"]

save_buttons = [] * NUM
transfer_buttons = [] * NUM
save_start_index = [0] * NUM


def save(channel):
    def func(label):
        save_file_name = f"values_{int(channel) + 1}.txt"
        # print("---------------------------------", save_file_name, "---------------------------------")
        if save_buttons[channel].get_status()[0]:
            save_start_index[channel] = len(adc_inputs[channel])
        else:
            vals = []
            f = open(save_file_name, "r")
            if os.stat(save_file_name).st_size != 0:
                vals.extend(json.loads(f.read()))
            f.close()
```

```
            vals.extend(adc_inputs[channel][save_start_index[
    channel]:])
            f = open(save_file_name, "w")
            f.write(json.dumps(vals))
            f.close()

    return func


def transfer_to_arduino(channel):
    def func(label):
        save_file_name = f"values_{int(channel) + 1}.txt"
        if transfer_buttons[channel].get_status()[0]:
            vals = []
            f = open(save_file_name, "r")
            if os.stat(save_file_name).st_size != 0:
                vals.extend(json.loads(f.read()))
            f.close()
            with buffer_lock:
                pwm_queues[channel].queue.clear()
                [pwm_queues[channel].put(i) for i in vals]
        else:
            with buffer_lock:
                pwm_queues[channel].queue.clear()

    return func


# plot live data
def live_plotter():
    # set buttons x and y
    save_xy = [
        (0.185, 0.75),
        (0.43, 0.75),
        (0.675, 0.75),
        (0.92, 0.75),
        (0.185, 0.25),
        (0.43, 0.25),
        (0.675, 0.25),
        (0.92, 0.25),
    ]
    observe_xy = [
        (0.185, 0.7),
        (0.43, 0.7),
        (0.675, 0.7),
```

```
        (0.92, 0.7),
        (0.185, 0.2),
        (0.43, 0.2),
        (0.675, 0.2),
        (0.92, 0.2),
    ]

    ani = FuncAnimation(plt.gcf(), animate, interval=INTERVAL)

    for i in range(NUM):
        save_label = ["save"]
        observe_label = ["observe"]

        # x position, y position, width and height
        save_ax = plt.axes([save_xy[i][0], save_xy[i][1], 0.1, 0.1], frame_on=False)
        observe_ax = plt.axes(
            [observe_xy[i][0], observe_xy[i][1], 0.1, 0.1], frame_on=False
        )

        save_buttons.append(CheckButtons(save_ax, save_label, [False]))
        transfer_buttons.append(CheckButtons(observe_ax, observe_label, [False]))

        # clear content of values file
        save_file_name = "values_{}.txt".format(i + 1)
        open(save_file_name, "w").close()

        # buttons functionalities
        save_buttons[i].on_clicked(save(i))
        transfer_buttons[i].on_clicked(transfer_to_arduino(i))

    # add some space between subplots and to the right of the rightmost ones
    plt.subplots_adjust(wspace=0.6, right=0.92)
    plt.show()


# animating each input data
def animate(_):
    for i in range(NUM):
        with buffer_lock:
            # input_copy = adc_inputs[i].copy()
```

```
            input_copy = adc_inputs[i][-400:].copy()
        if len(input_copy) == 0:
            continue
        axes[i // 4, i % 4].cla()
        axes[i // 4, i % 4].set_ylim(-100, 1100)
        # axes[i // 4, i % 4].set_xlim(
        #     0, np.power(np.e, int(np.log(len(input_copy))) + 1)
        # )
        axes[i // 4, i % 4].set_xlim(0, 410)
        axes[i // 4, i % 4].plot(input_copy, color=colors[i])
        axes[i // 4, i % 4].title.set_text("{}th Input".format(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 کاملا مشابه عملکرد تک دکمه‌های سیو و مشاهده‌ی فاز قبل دارند، لذا نیازی به توضیح مجدد آن‌ها نیست. از آنجایی که عملکرد این دکمه‌ها کاملا یکسان است و صرفا برای کانال‌های مختلف این کار را انجام می‌دهند از دکوراتور پایتون برای ساختن تابع‌هایی یکسان با ورودی‌های مختلف استفاده کردیم.

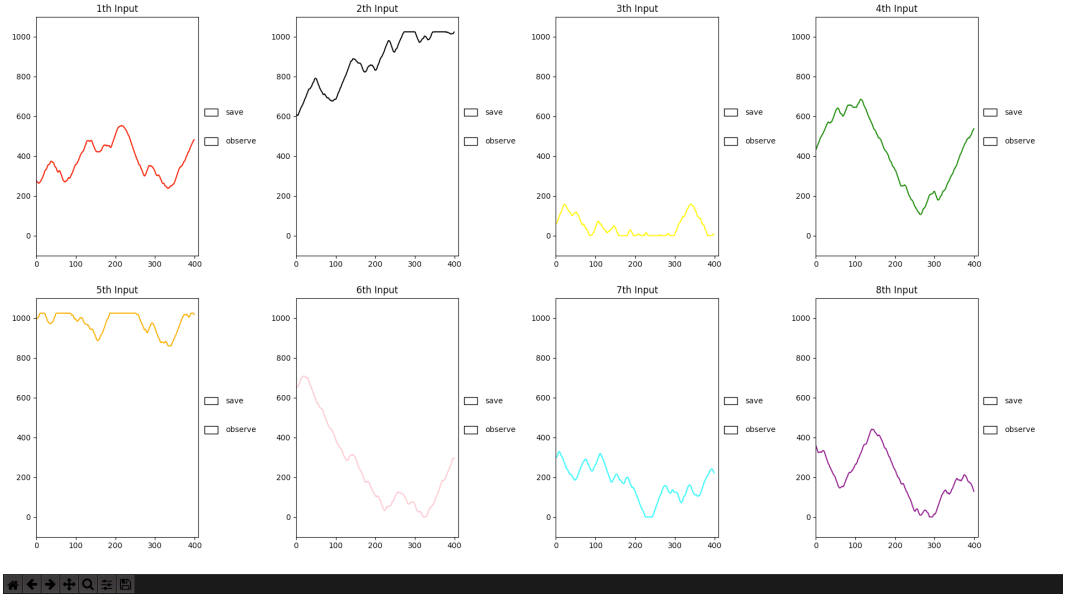
داده‌های مربوط به هر کانال را در مثلا فایلی به نام values ـ ۲ ذخیره‌می‌کنیم که این مثال بیانگر داده‌های ذخیره‌شده‌ی کانال دوم هستند.

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.