آزمایشگاه سخت‌افزار

فانکشن ژنراتور و اسیلوسکوپ نرم افزاری

گزارش چهارم

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

۱ مراحل انجام بخش چهارم پروژه ۲
۱.۱ سخت‌افزار ۲
۲.۱ نرم‌افزار روی PC ۳
۱.۲.۱ نرم‌افزار ارتباط با آردوئینو ۳
۲.۲.۱ نرم‌افزار رسم نمودار ۳

فهرست تصاویر

۱ نمودار رسم شده از داده‌های آنالوگ دریافتی ورودی‌های ۱ و ۳ و ۶ . . . . . ۷

فهرست برنامه‌ها

Software/R۴/plotter.py ۳

۱ مراحل انجام بخش چهارم 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

در این فاز بخش سخت افزاری همانند فاز قبل می باشد و تغییری نکرده است.

۲.۱ نرم‌افزار روی PC

۱.۲.۱ نرم‌افزار ارتباط با آردوئینو

در فاز چهارم پروژه نحوه ارتباط با آدوینو تغییری نکرده و به همان صورت قبل باقی مانده است؛ فلذا مجددا به آن نمی‌پردازیم.

۲.۲.۱ نرم‌افزار رسم نمودار

در فاز قبلی پروژه قابلیت رسم نمودار زنده هر ۸ ورودی، زوم کردن بر روی نمودار دلخواه و ذخیره داده‌های آن‌ها با دکمه save و نمایش آن‌ها در خروجی با دکمه observe را داشت. در این فاز باید بتوانیم از بین نمودارهای موجود موارد دلخواه را انتخاب کنیم تا بتوانیم به‌راحتی بین آن‌ها مقایسه کنیم.
کد این بخش 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

INTERVAL = 200  # Time between graph frames in milli seconds.

colors = ["salmon", "grey", "yellow", "lightgreen", "orange", "pink", "cyan", "plum"]

save_buttons = [] * NUM
transfer_buttons = [] * NUM
visibility_buttons = [] * NUM
save_start_index = [0] * NUM
visible_diagrams = [True]
visible_diagrams.extend([False] * (NUM - 1))
# fig, axes = plt.subplots(sum(visible_diagrams), figsize=(10, 8))
fig = plt.figure(figsize=(10, 8))
axes = [None] * NUM
axes[0] = fig.add_subplot(label=str(0))
fig.tight_layout()


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


def set_visible(channel):
    def func(label):
        if visibility_buttons[channel].get_status()[0]:
            visible_diagrams[channel] = True
```

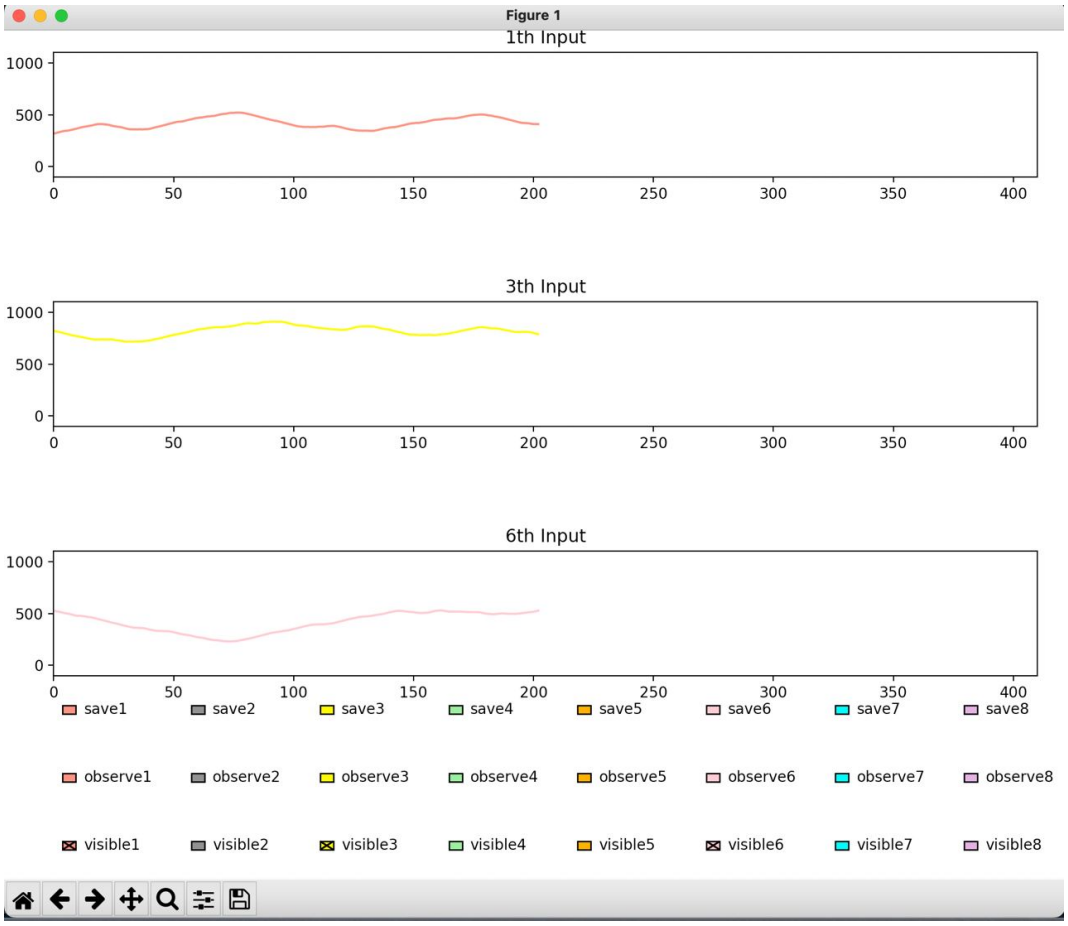
```
            axes[channel] = fig.add_subplot(sum(visible_diagrams), 1, sum(visible_diagrams), label=str(channel))
            for i in range(len(visible_diagrams)):
                if visible_diagrams[i]:
                    axes[i].change_geometry(sum(visible_diagrams), 1, sum(visible_diagrams[:i + 1]))
        else:
            visible_diagrams[channel] = False
            fig.delaxes(axes[channel])
            axes[channel] = None
            for i in range(len(visible_diagrams)):
                if visible_diagrams[i]:
                    axes[i].change_geometry(sum(visible_diagrams), 1, sum(visible_diagrams[:i + 1]))

    return func


# plot live data
def live_plotter():
    ani = FuncAnimation(plt.gcf(), animate, interval=INTERVAL)

    for i in range(NUM):
        save_label = ["save{}".format(i + 1)]
        observe_label = ["observe{}".format(i + 1)]
        visible_label = ["visible{}".format(i + 1)]

        # x position, y position, width and height
        save_ax = plt.axes([np.linspace(0.05, 0.9, 8)[i], 0.16, 0.1, 0.08], frame_on=False)
        observe_ax = plt.axes(
            [np.linspace(0.05, 0.9, 8)[i], 0.08, 0.1, 0.08], frame_on=False
        )
        visible_ax = plt.axes([np.linspace(0.05, 0.9, 8)[i], 0, 0.1, 0.08], frame_on=False)
        save_buttons.append(CheckButtons(save_ax, save_label, [False]))
        transfer_buttons.append(CheckButtons(observe_ax, observe_label, [False]))
        visibility_buttons.append(CheckButtons(visible_ax, visible_label, [True if i == 0 else False]))

        #set colors for buttons
```

```
104         save_buttons[-1].rectangles[0].set_facecolor(colors[i
    ])
105         transfer_buttons[-1].rectangles[0].set_facecolor(
    colors[i])
106         visibility_buttons[-1].rectangles[0].set_facecolor(
    colors[i])
107
108         # clear content of values file
109         save_file_name = "values_{}.txt".format(i + 1)
110         open(save_file_name, "w").close()
111
112         # buttons functionalities
113         save_buttons[i].on_clicked(save(i))
114         transfer_buttons[i].on_clicked(transfer_to_arduino(i))
115         visibility_buttons[i].on_clicked(set_visible(i))
116
117     # add some space between subplots and to the right of the
    rightmost ones
118     plt.subplots_adjust(bottom=0.24, hspace=1)
119     plt.show()
120
121
122 # animating each input data
123 def animate(_):
124     for i in range(NUM):
125         if visible_diagrams[i]:
126             with buffer_lock:
127                 input_copy = adc_inputs[i][-400:].copy()
128             if len(input_copy) == 0:
129                 continue
130             axes[i].cla()
131             axes[i].set_ylim(-100, 1100)
132             axes[i].set_xlim(0, 410)
133             axes[i].plot(input_copy, color=colors[i])
134             axes[i].title.set_text("{}th Input".format(i + 1))
```

در ابتدا در خطوط ۱۱ تا ۲۳ آرایه های مربوطه برای ذخیره و مشاهده و فعال بودن یا نبودن را مقداردهی می‌کنیم و فقط نمودار مربوط به ورودی ۱ را در ابتدای کار فعال می کنیم تا نمایش داده شود.
در خط ۸۵ با شروع اجرای برنامه و صدا زده شدن تابع live-plotter ابتدا تابع Animate همانند فازهای قبل برای رسم نمودار زنده ورودی‌هایی که توسط کاربر فعال شده اند و آپدیت آن‌ها با آمدن مقادیرجدید 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 می شود و پس از افزودن به لیست دکمه‌های مربوطه، فانکشنالیتی مربوط به آن‌ها نیز مشخص می‌شود.
از بین توابع مربوط به دکمه‌های مختلف، دکمه‌های save و transfer همانند فاز قبل هستند و تغییری نداشته‌اند.

تابع جدیدی که در خط ۶۵ موجود است و set-visible نام دارد برای فعال کردن نمودار ورودی‌ها بکار می‌رود و در صورتی که دکمه فعال شود، نمودار آن به صفحه اضافه می شود(برحسب شماره ورودی‌ها که ورودی ۱ بالاتر از ورودی ۲ نمایش داده می شود و الی آخر) و در صورت غیرفعال شدن، دیگر نمودار مربوطه در صفحه نشان داده نمی شود و فقط سایر نمودارهای فعال نمایش داده می‌شوند.

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



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