

6.115 Final Project - Bunker Knowledge

Mark Wright

April-May 2020

Contents

1	Introduction	2
2	Learning the TFT Board and emWin Graphics package	3
2.1	Circuit Construction	3
2.2	emWin Graphics Package	4
3	COVID-19 Data Retrieval	5
3.1	API Usage	5
3.2	Data Reformat	6
4	Serial Interface and MINMON style language with PSOC Big Board	6
4.1	Computer (Big Board Replacement)	7
4.2	PSOC 5 Big Board	8
5	Interfacing with the GUI	9
5.1	Creating Rectangles	10
5.2	Filling Rectangles	10
5.3	Graphing	12
6	Conclusion and Heartfelt Thanks	13
7	Appendix A	14
7.1	Data Retrieval Python Code	14
7.2	Serial/MINMON Python Code	27
7.3	Serial and Drawing PSOC C Code	32

1 Introduction

Before Reading!

[Youtube link to video of working product](#)

[Github repo containing all code if you want to reference](#)

In these uncertain times, misinformation can spread like wildfire. Any mistakes by administration in terms of loosening regulations or even by regular people who underestimate the severity of this pandemic can lead to loss of lives. To combat this misinformation crisis and inform the modern American citizen I present Bunker Knowledge: an easy way to retrieve accurate, up to date Coronavirus information for any state in the US, as well as the country and world as a whole, all safely quarantine inside your bunker. With the power of knowledge a few keystrokes away, you no longer have to listen to hearsay from your family and friends; You can think for yourself to advise your decision-making in quarantine and keep those you love safe and protected by staying informed.

This tool utilizes the python requests package to retrieve data from online API's. It then cleans the data and sends it to the PSoC for display, where a graphic library is used to display the received data in a nice manner. The software can also graph on the user's computer nice interactive graphs using the plotly library. I approached the two units separately at the beginning. First, I integrated the TFT screen with first PSoC stick and eventually the PSoC 5 big board and played with the graphics library to get it working. Then, I constructed the data retrieval system using the API's and created my own class system to store historical and current data for all states in the US, plus the country as a whole. I then integrated another API to get world data. After these were done, I wrote code to operate on the serial port and developed a MINMON style language to decide what kind of data you want displayed on the TFT screen. Options include current data, past dates, and graphs for each state in the US. After the data was able to be sent I developed a grid interface to display the data on the TFT screen with colors and formatting to make the results easy to parse and understand. To further the information available I finished by creating a tool to graph the data using the plotly Python package.

By starting with separate ideas and then combining them in a standardized way, I was able to create Bunker Knowledge to be efficient, accurate, and useful. Retrieving the whole dataset for the whole country takes at most 10 seconds, and any subsequent runs of the program used a cached version (for up to 1 hour) to load in the data in less than half a second. Command results are processed, sent, and displayed in less than a second, providing for a smooth user experience regardless of the amount of data you want to see. The data retrieval process has been verified through unit testing to ensure accurate data, and the graphs provide a secondary graphical interface that allows spotting of trends in Covid-19 data. Spotting these trends can help regular people stay informed and could even advise public officials for what still needs to be done to end this time of huddling in our bunkers during quarantine.

2 Learning the TFT Board and emWin Graphics package

Since a major component of the bunker knowledge process is the display to the TFT screen, I started with connecting the TFT board to my PC so I can know confidently the board works and learn about the process of drawing on the screen.

2.1 Circuit Construction

I began by following the process outlined in Sean Kent and Eric Ponce's tutorial ([Link](#)) of how to get a TFT screen to work the PSoC stick. I wanted to simply get the circuit working first so I just followed their exact schematic (recreated below with my resistor values) to initially connect the TFT board. I have worked with TFT boards in 6.08, an embedded systems class, so I have some understanding of how they work, but to continue I had to get a better understanding of the emWin graphics library.

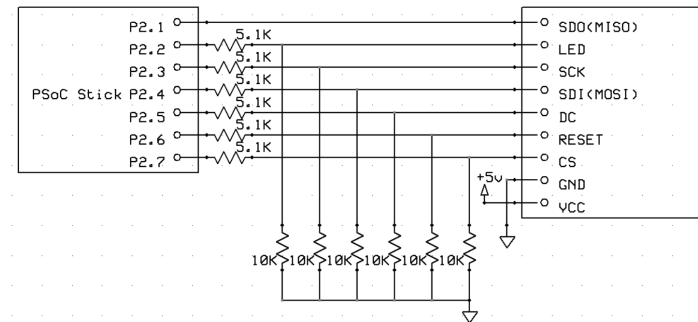


Figure 1: TFT Circuit Schematic

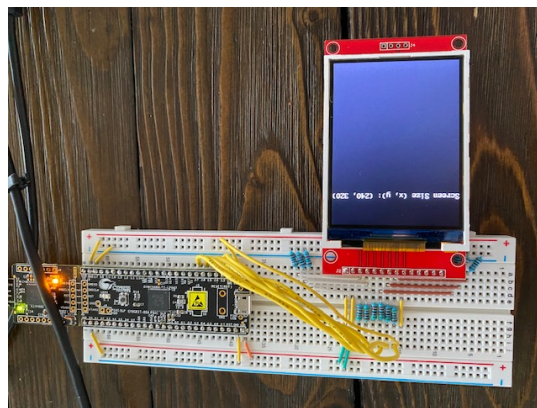


Figure 2: Working TFT Setup!

2.2 emWin Graphics Package

To get started learning this graphics package I figured I would just need to find the API documentation online, but when I found it it was a whopping 2969 pages!

[Link to emWin User Manual](#)

After reading through the 60 page table of contents I decided I would just need to work with the text API, rectangle API, and graph API for my purposes. I consulted this manual heavily during the design and implementation process.

After working with emWin for a while, I realized that it would be hard to make the big board interact with the stick in a way that would be both meaningful and easy enough to use. After consulting course staff, I decided to rewire the TFT screen to work with the big board. This was nice because I didn't need voltage dividers to lower the voltage since the big board runs on 3.3V logic. The updated design is even simpler than before and allows me to handle both the serial input and graphics drawing using the same powerful board. This overall is simpler for me as it would be inconvenient to have to send instructions to the stick since they would have to either go through the computer or very carefully through tx/rx lines since the two boards run on different logic voltages.

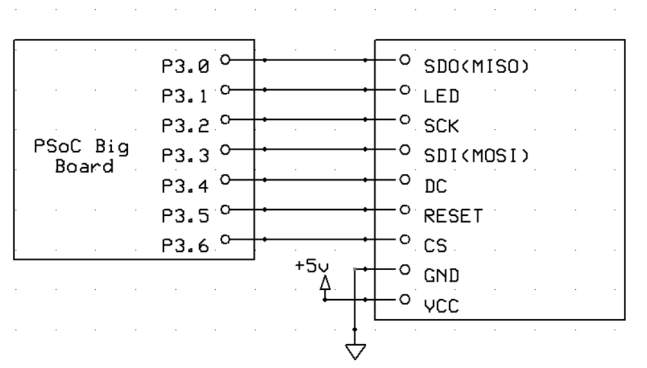


Figure 3: Circuit schematic for big board with TFT

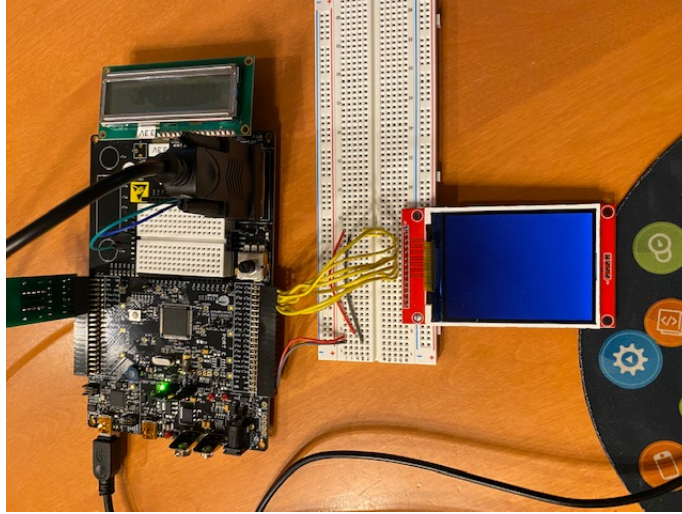


Figure 4: TFT Screen works with Big Board

With the hardware secure, I could confidently move on to obtaining the data, and sending it to the Big Board where it could be processed. Luckily, the hardware for this project turned out to be quite simple to connect, especially since the big board runs on 3.3V logic.

3 COVID-19 Data Retrieval

To begin the actual process of displaying coronavirus data, I needed to be able to retrieve the data and send it in some way. This ended up being more work than expected. There are lots of API's out there, but I was unable to find any where a single one had all the data I needed. I ended up settling for using two different API's, one to retrieve World data and one to retrieve us data. As the system works right now, data can be retrieved for any US state, the US, and the World. Expanding would be possible, but another API would need to be incorporated. I elaborate on the process with which I implemented the data retrieval below. All python code for data retrieval is in Appendix A.1

3.1 API Usage

The Covid-19 data retrieval step began with finding an API to retrieve coronavirus statistics of varying levels of granularity. This proved to be more difficult than expected as there are so many API's that have been created in the couple of months this virus has existed. I ended up going with a merging of two to receive world data versus state level data for the US. The data is retrieved in a raw JSON format using the appropriate calls from the API's using the requests package in Python.

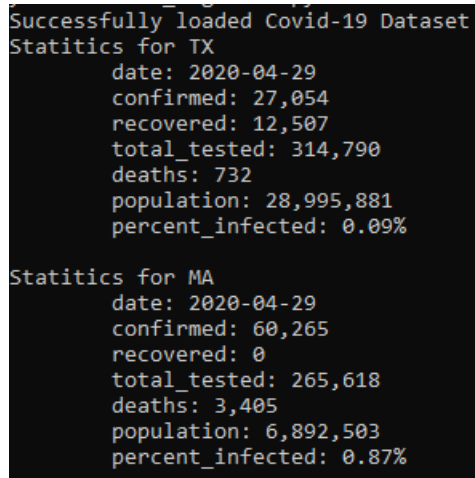
[State-Level Coronavirus API](#) ← This is the API I use for all my state and US call data, and right now I use a different one for the world.

[World-Level Coronavirus API](#) ← This is the API for my world data calls.

3.2 Data Reformat

The raw JSON data is then passed into my class system I created for parsing the JSON data. JSON's are extremely hard to work with in C as they have no dictionary type structure, so I would need to receive all the data, create a struct for dictionary types, and then parse the data into the struct. I figured this was too much work to do there when I could do it very simply in Python, so I opted to write python code to store the relevant data and send it to the PSoC when necessary.

The python classes encapsulate each region into its own node that holds the relevant information to that region and is able to display and send these to the big board for actual processing. The data itself is further encapsulated into its own node for each date, with historical data displayed as a list of these nodes. Additionally, since the requests can be large and take a decent amount of time, I store the data locally on my computer for up to an hour, after which I declare the data stale and load a new dataset in.



```
Successfully loaded Covid-19 Dataset
Statistics for TX
  date: 2020-04-29
  confirmed: 27,054
  recovered: 12,507
  total_tested: 314,790
  deaths: 732
  population: 28,995,881
  percent_infected: 0.09%

Statistics for MA
  date: 2020-04-29
  confirmed: 60,265
  recovered: 0
  total_tested: 265,618
  deaths: 3,405
  population: 6,892,503
  percent_infected: 0.87%
```

Figure 5: A photo of data retrieval working using the API I wrote using Python classes, serial to follow :)

4 Serial Interface and MINMON style language with PSOC Big Board

I communicate with my big board over a MINMON style serial interface. I mainly use the TX line on my computer's COM port to send data to the RX line of the big board. The commands are designed to be intuitive and error-safe, so users will not have to restart the whole program if they misspell or input a date or region in which no data exists. In the following sections, I will highlight what each end does for the data transfer process.

4.1 Computer (Big Board Replacement)

The computer acts how the big board would have had I done this on campus. When the script is run, the first thing that happens is the dataset is pulled using API calls and the data retrieval methods from the internet. The data is stored in memory on the computer, so that when the code is run there is minimal loading time if run previously. Afterwards, the user is greeted with a friendly message telling them that they have entered the program and to type help if they want a list of commands. All python serial interface code is in Appendix A.2. Appendix A.3 contains the code for serial PSoC as well as GUI on the PSoC.

```
C:\Users\mark6\Google Drive\College-MIT\S6Spring2020\6.115\FinalProject\BunkerKnowledge\SerialConnections>python start_bunker_knowledge.py
-Please wait for startup (could take up to 10 seconds)
Successfully loaded Covid-19 Dataset

*****
* Welcome to Bunker Knowledge, the Covid-19 Database Terminal! *
*****

-To get help, please type help

> help

Covid-19 Commands Help Guide

Definitions
  name - name of the area you're searching for eg. TX,
        can handle 2 letter postal codes, state names, us, and world
  month - number of the month
  day - number of the day

- In general, all commands will send over the serial port when executed,
  space separated and deployed as in description, number order
- Commands are sent with NO brackets

1. [name] - Sends the most current data for that region
2. [name] [month] [day] - Retrieve data for that region for the specified date
                        (not implemented for us and world!)
3. [name] graph - Displays in your browser a graph of historical Coronavirus
                  stats, and also sends the most current date for that region
4. exit - closes the terminal

>
```

Figure 6: Start of the MINMON interface, list of commands is simple and understandable.

Since the little language is so simple, the user should be fully able to understand how to operate the program after reading this small paragraph. The flow of data is pretty simple after that, if the user wants to exit or wants help they simply must type those words. Then, to send data you type the name of the state/region you want to send to the PSoC with potential modifiers. Inputting the region by itself will send the most up to date data for that region to the PSoC through the serial port. Users can also specify a month and day to get older data if they would like to see how the state has progressed over time. If the user wants very in-depth data on a particular region, they can type the region + graph at the end to display a graph of data over time for that region. The implementation of these displays is discussed in the GUI section, but typing the serial port sends the data needed one value at a time separated by a delimiter. After all data is sent, the command END is then sent, which is used to indicate no more data will be sent for the time being so it is safe to update the GUI.


```

> tx
Sent: ['Texas', '2020-05-11', 'Population', '28,995,881', 'Confirmed', '39,869', 'Total tested', '525,697',
'Recovered', '21,713', 'Deaths', '1,100', 'Active', '17,056', '% Infected', '0.14%', '% Tested', '1.81%',
'% Recovered', '54.46%', 'Death rate', '2.76%', '% Active', '42.78%', 'END']
> la
Sent: ['Louisiana', '2020-05-11', 'Population', '4,648,794', 'Confirmed', '31,815', 'Total tested', '220,83
0', 'Recovered', '22,608', 'Deaths', '2,308', 'Active', '6,899', '% Infected', '0.68%', '% Tested', '4.75%'
, '% Recovered', '71.06%', 'Death rate', '7.25%', '% Active', '21.68%', 'END']
> la 4 20
Sent: ['Louisiana', '2020-04-20', 'Population', '4,648,794', 'Confirmed', '24,523', 'Total tested', '136,11
1', 'Recovered', '0', 'Deaths', '1,328', 'Active', '23,195', '% Infected', '0.53%', '% Tested', '2.93%', '%
Recovered', '0.0%', 'Death rate', '5.42%', '% Active', '94.58%', 'END']
> exit

C:\Users\mark6\Google Drive\College-MIT\S6Spring2020\6.115\FinalProject\BunkerKnowledge\SerialConnections>

```

Figure 7: Sending a sequence of commands, will display each command in turn. A confirmation is provided of what strings are sent, with the delimiter being sent after each one of them.

4.2 PSOC 5 Big Board

On the PSoC's end it has an interrupt function for data retrieval character by character. I initially wanted to display them all on the GUI as each word came in, but I was missing characters because the interrupt routine was unable to run during GUI updates, so I had to figure out a storage solution for each word that I was transmitting. I planned to display each word in it's own individual rectangle that I drew out beforehand where the name and date would go the full width of the screen and the rest would be split into two equal sized rectangles on either side of the screen. All the rectangles would have the same height. The US data had the most data with 28 total rectangles required as it had a name, data, and 13 key-value pairs so I partitioned my screen to appropriately display all that data within rectangles to fit nicely on the screen. After creating the rectangles I simply had to have an array of size equal to the number of rectangles and store the sent words in the data. In the interrupt routine I check for a set delimiter (currently new line) to end each word. I also check at the end of each word if the word sent is END. If it is, then I know the data transmission has stopped, and the GUI can now be appropriately updated. The PSoC passes the data to the GUI control functions and afterwards proceeds to listen and wait for a new character. The overall flow of the software is fairly easy to follow, but I also made a graphic flowchart of the flow which I display on the next page.

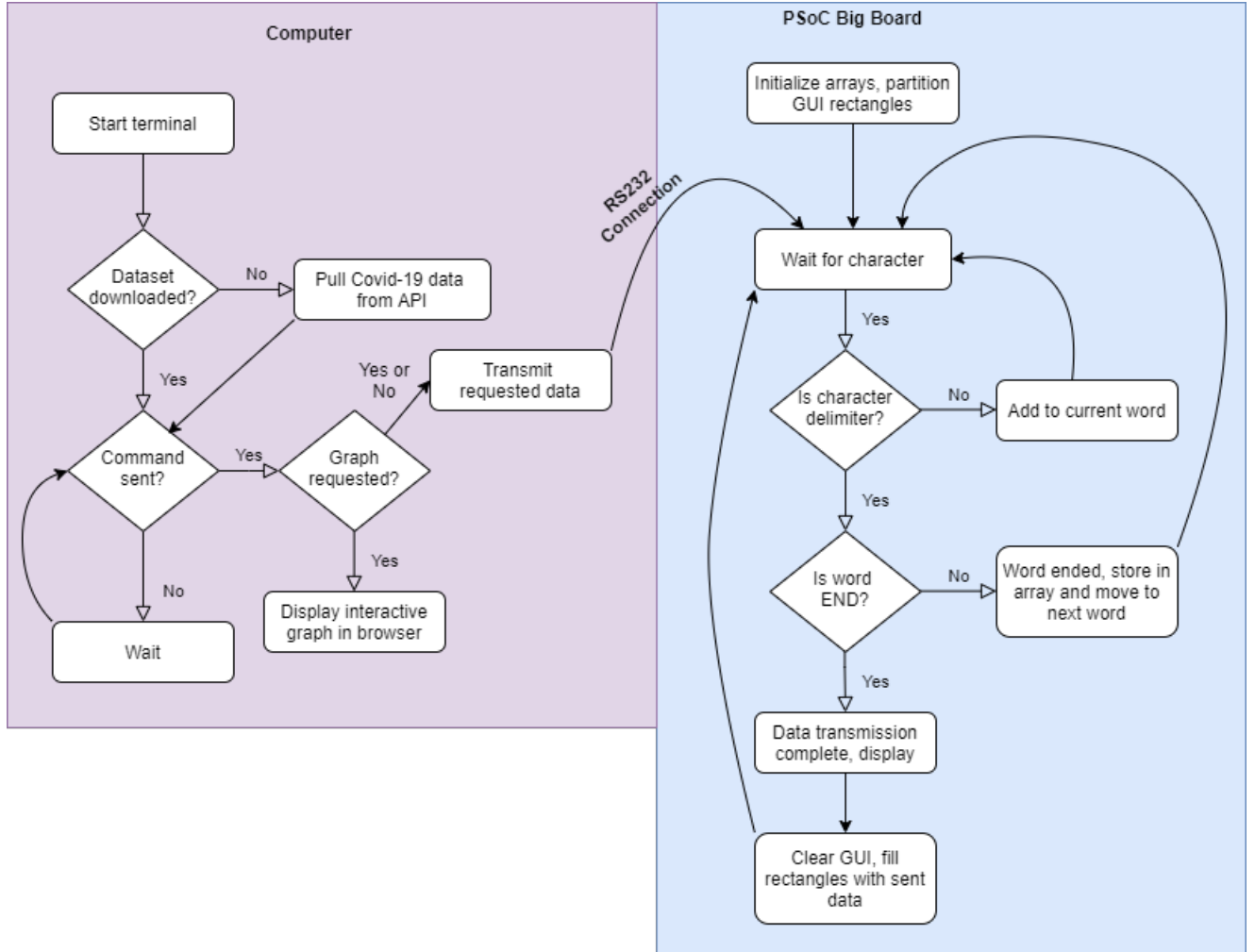


Figure 8: Full software flow, any purple blocks execute on the computer and blue blocks are executed on PSoC.

5 Interfacing with the GUI

Using the emWin platform the process was surprisingly intuitive to display the data on the screen once a full set of data has been sent. The graphs however, proved to be impossible without using the widget setup. I could not figure out how to get the Widget library to work on PSoC so I scrapped the idea of graphing on the PSoC. The graphs are still useful in my opinion since humans are visual beings so I decided to have the graphs display on my computer using the powerful plotly library. I interfaced with the GUI using an SPI Master block in PSoC creator that corresponds to the pins used in the TFT circuit schematic above. All GUI code is on the PSoC and shown in appendix A.3.

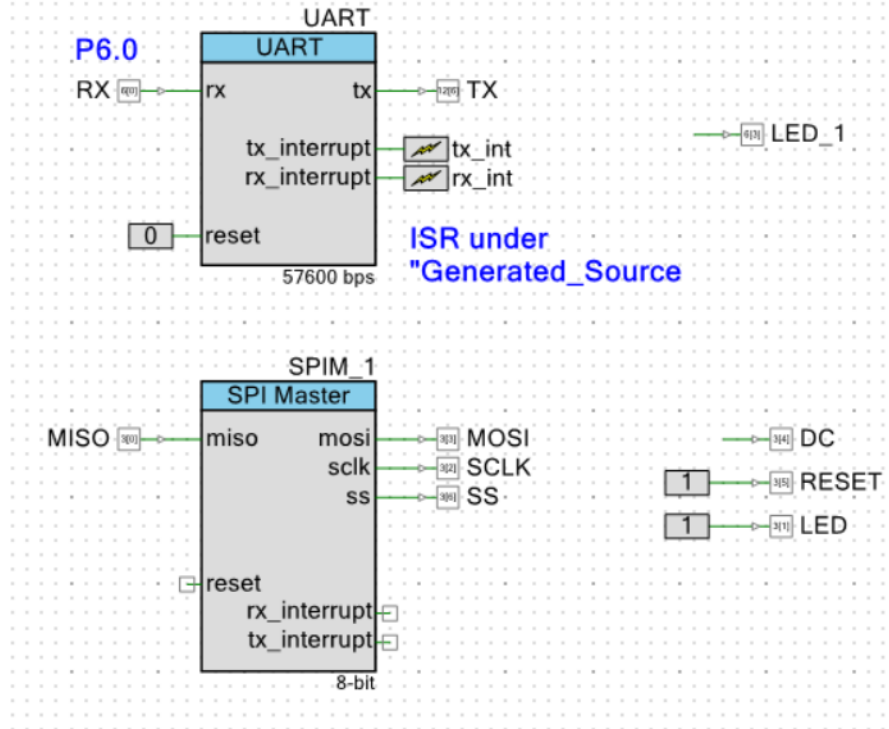


Figure 9: The PSoC Top level design. The UART module handles RS232 connections and generates interrupts for character retrieval. The SPI Master block powers the TFT, and the LED just indicates that the board is powered for me :-)

5.1 Creating Rectangles

Creating the rectangles was talked about above so I'll quickly rehash the process, I had a 10px margin and split the remaining 220x300 into equal height rectangles. The first two were the full width of the screen and the remaining were two columns down. The US had 28 necessary rectangles so I tuned the height to fit them. The documentation for the GUI_RECT struct proved very useful.

5.2 Filling Rectangles

Filling the rectangles was easy because of how I organized the data. When the printing function was called I had an array of all the data one word at a time, and an array of rectangles that corresponded one to one. Thus all I had to do was loop through the list and call the GUIDispStringInRectWrap function to display in the appropriate rectangle. I additionally added checks so that I could color the numbers in a way similar to the Johns Hopkins Dashboards colors. I used Red for confirmed, White for deaths, Green for recovered, and Yellow for active. I include pictures of my tft screen's display for different updates and different dates actually working on the next page.



Figure 10: Larger to actually show numbers, this is how the GUI looks with the rectangles and colors working, an update from the previous display. It takes less than a second to send and display new data.

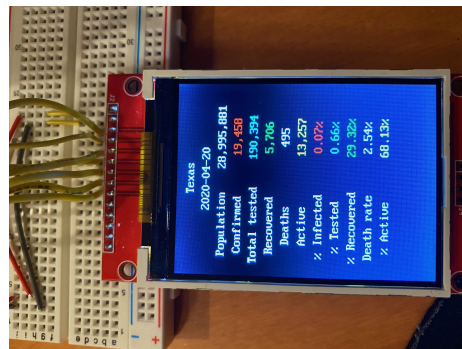


Figure 11: Display is still Texas but for a prior date, shows that command works, in this case TX 4 20 in terminal.

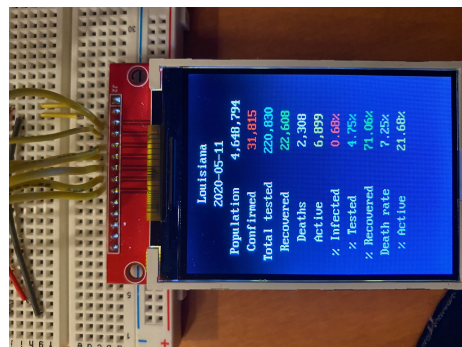


Figure 12: Display properly updates during a switch without artifacts thanks to a GUI Clear command call. This command would be LA

5.3 Graphing

In the emWin graphics library there is only one command for graphing without widgets: `GUI_DrawGraph`. This function can only handle 16 bit integers, which poses a challenge in itself since we use large numbers, and does not scale to fit the screen. It has no control parameters either so I tried to use the graph widget. This requires the use of a window manager that I couldn't figure out how to install on the PSoC. I still wanted the graphs so I instead opted to display them on the user's computer using plotly. Plotly provides powerful, interactive graphs that are able to be created in a short amount of time, especially since I had already performed all the data processing. Now when users type something such as "texas graph", a graph will be pulled up in their default web browser that users can play with and analyze to spot trends in Covid-19 cases. The graph has two sections, the left plots cumulative confirmed, recovered, deaths, and active case amounts as line plots. The interesting thing is that confirmed is the sum of the other three amounts so it's neat to notice trends. The right side is a bar graph that displays the number of new cases per day, and can show the "flattening of the curve" that's been so talked about if it's happening.

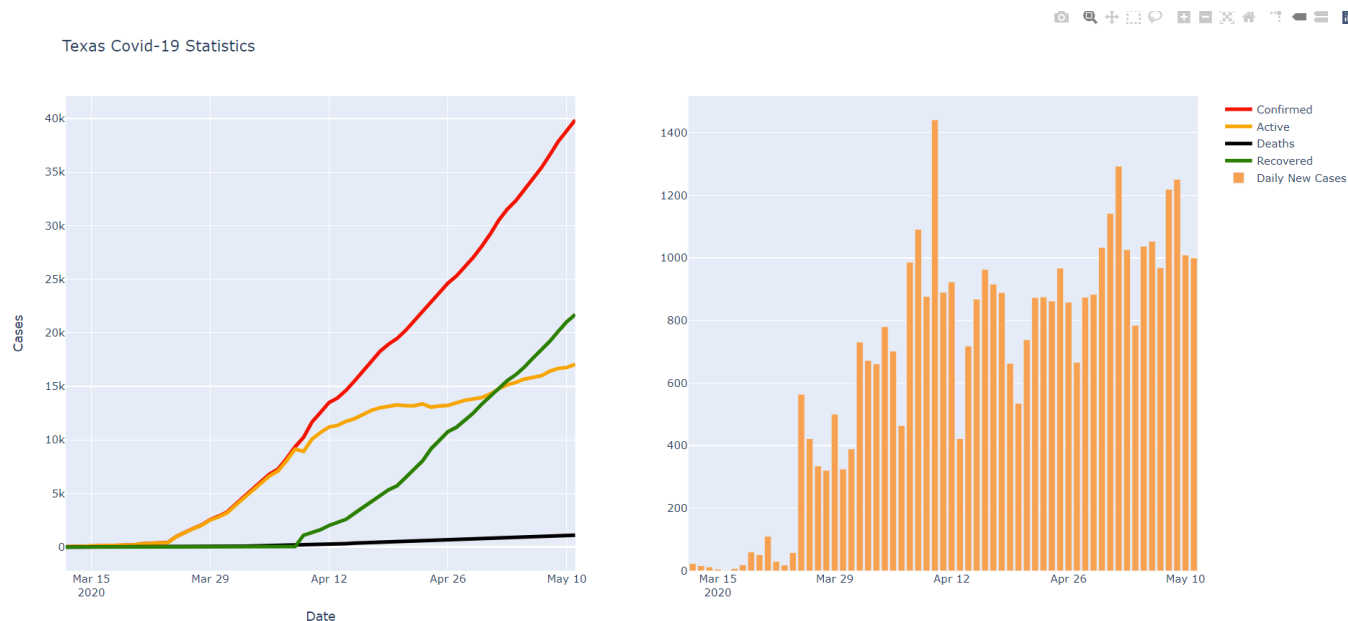


Figure 13: An example of the graphs generated by the data for TX. The command to get this is `tx graph`, and uses the same colors as the display (except the yellow looks orange). This graph is easily filterable by clicking on the legend in the right.

Despite the fact they are not on the PSoC, I think the graphs look really nice and add a lot to the project. Humans are very visual so having a tool like this is necessary and helpful, especially with the interactive ability of plotly. Interestingly, through this process I found that states are slow to report on weekends, which is why every 5 days you see a dip in the number of new cases.

6 Conclusion and Heartfelt Thanks

Overall I'm extremely happy with how Bunker Knowledge turned out. I think it could actually be really useful to inform administrations, colleges, and individuals during this hard time. The user interface is pleasant to the eye and allows the user to find only information relevant to them in a few simple commands. I wish everyone could have a tool like this in their lives, instead of just listening to what they hear on the news and social media, but that's for a different discussion. Bunker Knowledge is powerful, useful, and easy to operate. Again, the YouTube video in the intro shows off the full functionality, and the code can be found in either the GitHub link above as well as the following appendix.

I'd like to sincerely thank Professor Leeb and all of the course staff. In the words of Leeb, "Dudes, you rock! :-)". Y'all have been so helpful from your quick responses on Piazza to just general support in this weird transition we've all been going through. I hope to see you all in the fall, and Leeb can't wait to UROP with you this summer! THANK YOU! :-)

7 Appendix A

If full code wanted, please consult the Github link in the intro of document.

7.1 Data Retrieval Python Code

```
'''
retrieval.py

File used to retrieve coronavirus information using API's in its raw, JSON
format. Uses two different API's to do so.
'''

import requests
import json
import datetime

state_covid19_api_url = "https://covidtracking.com/api/v1/"

'''
Current API Options (API V1)
States Current Values - states/current.json
    to get one state: states/TX/current.json
States Historical Data - states/daily.json
States Information - states/info.json (Links to sources of info)
US Current Values - us/current.json
US Historical Data - us/daily.json
Counties- counties.json (not usable currently)
State Website Screenshots - states/screenshots.json (unused but available)
In the press - press.json
'''

def date_range(start, end, days_back):
    """
    Returns a list of Datetime objects for days between start and end, inclusive
    """
    r = (end + datetime.timedelta(days=1) - start).days
    return [start + datetime.timedelta(days=i) for i in range(r - days_back)]

def get_covid19_dates():
    """
    Returns a list of strings of covid-19 dates since the pandemic began on
    January 22, 2020
    """
```

```

start = datetime.date(2020, 1, 22)
end = datetime.date.today()
dateList = date_range(start, end, 1)
return [str(date) for date in dateList]

def make_request(url, headers=None):
    """
    Function to submit a request
        url - url to request from
        headers - headers of the request

    Returns JSON of response
    """
    if headers == None:
        headers = {}
    response = requests.request(
        "GET", url, headers=headers)
    j = json.loads(response.text)
    if type(j) == dict:
        j = [j]
    return j

def get_us_covid_data(level, time_length, state=None):
    """
    Function to request Covid data at the state level

    Parameters
        level - world, us, states
        time_length - daily, current
        state - Name of the state you want
                If none, will retrieve all state data

    Returns JSON of data
    """

    url = state_covid19_api_url
    url += level + "/"
    if state != None:
        url += state + "/"
    url += time_length + ".json"
    return make_request(url)

def get_world_data():
    """

```



```

Function to request Covid data at the world level

Parameters
    time_length - daily, current

Returns JSON of data
'''
# modify dates for either time string or not

url = "https://covid-19-coronavirus-statistics.p.rapidapi.com/v1/total"

# querystring = {"country": "Canada"}

headers = {
    'x-rapidapi-host': "covid-19-coronavirus-statistics.p.rapidapi.com",
    'x-rapidapi-key': "567026fb0fmshed791801b2ab42fp1072ecjsndf46ba69121e"
}

return make_request(url, headers)

def get_press_data():
    '''
    Function to request Covid data at the state level

    Returns JSON of press data
    '''

    url = state_covid19_api_url + "press.json"
    return make_request(url)

def get_covid19_data(level, time_length, state=None, press=False):
    '''
    Master function to request Covid data

    Parameters
        level - world, us, state
        time_length - daily, current
        state - Name of the state you want
                If none, will retrieve all state data
        press - true if you want the latest press releases

    Returns JSON of data
    '''
    level = level.lower()
    time_length = time_length.lower()

```

```

    if press:
        return get_press_data()
    elif level == "world":
        return get_world_data()
    if level == "state": # state is just easier to remember to type in
        level = "states"
    return get_us_covid_data(level, time_length, state)
'''
processing.py

```

This file does some simple data processing by just extracting the information from the JSON.

Performed here because C has no dictionary datatype and I wanted to just be able to send the info I needed to display. Data is collected in ~7 seconds and then is stored for up to one hour before being declared stale.

```

'''
from retrieval import *
import datetime
import pickle
import traceback
import os.path
import time

dataset_save_file = "../Covid19Data/Covid19_Dataset.txt"
last_saved_file = "../Covid19Data/Date_Modified.txt"
dataset_timeout = datetime.timedelta(hours=1)

states = ["AL", "AK", "AZ", "AR", "CA", "CO", "CT", "DC", "DE", "FL", "GA",
          "HI", "ID", "IL", "IN", "IA", "KS", "KY", "LA", "ME", "MD",
          "MA", "MI", "MN", "MS", "MO", "MT", "NE", "NV", "NH", "NJ",
          "NM", "NY", "NC", "ND", "OH", "OK", "OR", "PA", "RI", "SC",
          "SD", "TN", "TX", "UT", "VT", "VA", "WA", "WV", "WI", "WY"]

us_state_abbrev = {
    'Alabama': 'AL',
    'Alaska': 'AK',
    'American Samoa': 'AS',
    'Arizona': 'AZ',
    'Arkansas': 'AR',
    'California': 'CA',
    'Colorado': 'CO',
    'Connecticut': 'CT',
    'Delaware': 'DE',

```

'District of Columbia': 'DC',
'Florida': 'FL',
'Georgia': 'GA',
'Guam': 'GU',
'Hawaii': 'HI',
'Idaho': 'ID',
'Illinois': 'IL',
'Indiana': 'IN',
'Iowa': 'IA',
'Kansas': 'KS',
'Kentucky': 'KY',
'Louisiana': 'LA',
'Maine': 'ME',
'Maryland': 'MD',
'Massachusetts': 'MA',
'Michigan': 'MI',
'Minnesota': 'MN',
'Mississippi': 'MS',
'Missouri': 'MO',
'Montana': 'MT',
'Nebraska': 'NE',
'Nevada': 'NV',
'New Hampshire': 'NH',
'New Jersey': 'NJ',
'New Mexico': 'NM',
'New York': 'NY',
'North Carolina': 'NC',
'North Dakota': 'ND',
'Northern Mariana Islands': 'MP',
'Ohio': 'OH',
'Oklahoma': 'OK',
'Oregon': 'OR',
'Pennsylvania': 'PA',
'Puerto Rico': 'PR',
'Rhode Island': 'RI',
'South Carolina': 'SC',
'South Dakota': 'SD',
'Tennessee': 'TN',
'Texas': 'TX',
'Utah': 'UT',
'Vermont': 'VT',
'Virgin Islands': 'VI',
'Virginia': 'VA',
'Washington': 'WA',
'West Virginia': 'WV',
'Wisconsin': 'WI',
'Wyoming': 'WY',

```

        'USA': 'US',
        'World': 'WORLD'
    }

abbrev_to_name = {v: k for k, v in us_state_abbrev.items()}

populations = {"World": 7780953897,
               "US": 329579469,
               "AL": 4903185,
               "AK": 731545,
               "AZ": 7278717,
               "AR": 3017804,
               "CA": 39512223,
               "CO": 5758736,
               "CT": 3565287,
               "DC": 705749,
               "DE": 973764,
               "FL": 21477737,
               "GA": 10617423,
               "HI": 1415872,
               "ID": 1787065,
               "IL": 12671821,
               "IN": 6732219,
               "IA": 3155070,
               "KS": 2913314,
               "KY": 4467673,
               "LA": 4648794,
               "ME": 1344212,
               "MD": 6045680,
               "MA": 6892503,
               "MI": 9986857,
               "MN": 5639632,
               "MS": 2976149,
               "MO": 6137428,
               "MT": 1068778,
               "NE": 1934408,
               "NV": 3080156,
               "NH": 1359711,
               "NJ": 8882190,
               "NM": 2096829,
               "NY": 19453561,
               "NC": 10488084,
               "ND": 762062,
               "OH": 11689100,
               "OK": 3956971,
               "OR": 4217737,
               "PA": 12801989,

```

```

"PR": 3193694,
"RI": 1059361,
"SC": 5148714,
"SD": 884659,
"TN": 6829174,
"TX": 28995881,
"UT": 3205958,
"VT": 623989,
"VA": 8535519,
"WA": 7614893,
"WV": 1792147,
"WI": 5822434,
"WY": 578759}

```

```

class Covid19_Node():
    """
    Common class for Covid19 Nodes
    All have:
        self.name
        self.confirmed
        self.deaths
        self.recovered
        self.population
        self.date
        self.percent_infected
    """

    def __str__(self):
        # Gives a string representation of exactly what we need to send to PSOC
        delim = '\n'
        s = "{}{}".format(
            abbrev_to_name[self.__dict__["name"]], delim)
        s += "{}{}".format(self.__dict__["date"], delim)
        for key, value in self.__dict__.items():
            key = " ".join(key.capitalize().split("_"))
            if type(value) == int:
                s += "{}{}{:},{}{}".format(key, delim, value, delim)
            elif "Percent" in key or "rate" in key:
                if "rate" not in key:
                    key = key.split() # [percent, characteristic]
                    # % characteristic, to fit on screen
                    key = "% " + key[1].capitalize()
                s += "{}{}{}{}%{}{}".format(key, delim, value, delim)
            elif key == "Name" or key == "Date":
                continue
            else:

```

```

        s += "{}{}{}{}".format(key, delim, value, delim)
    s += "END".format(delim)
    return s

def __repr__(self):
    # just for nice display in dictionaries :-)
    return "{}-{}".format(self.__dict__["name"], self.__dict__["date"])

# All Node classes only have an init function where the datatypes are stored for
# later transmission.
class Covid19_World_Node(Covid19_Node):
    '''
    Class to represent one date status of the world in regards to coronavirus

    '''

    def __init__(self, c_json):
        self.name = "WORLD"
        self.date = str(datetime.date.today())
        self.population = populations["World"]
        self.confirmed = c_json["confirmed"]
        self.deaths = c_json["deaths"]
        self.recovered = c_json["recovered"]
        self.active = self.confirmed - self.deaths - self.recovered
        self.percent_infected = round(
            self.confirmed / self.population * 100, 2)
        self.percent_recovered = round(
            self.recovered / self.confirmed * 100, 2)
        self.death_rate = round(self.deaths / self.confirmed * 100, 2)
        self.percent_active = round(self.active / self.confirmed * 100, 2)

class Covid19_USA_Node(Covid19_Node):

    def __init__(self, c_json):
        self.name = "US"
        try:
            self.date = c_json["lastModified"][:10]
        except KeyError:
            self.date = c_json["dateChecked"][:10]
        self.population = populations[self.name]
        self.confirmed = c_json["positive"] if c_json[
            "positive"] != None else 0
        self.total_tested = c_json["totalTestResults"]
        self.recovered = c_json["recovered"] if c_json[
            "recovered"] != None else 0

```

```

# self.in_icu = c_json[
#     "inIcuCurrently"] if "inIcuCurrently" in c_json else 0
self.deaths = c_json["death"] if c_json["death"] != None else 0
self.active = self.confirmed - self.deaths - self.recovered
self.hospitalized = c_json[
    "hospitalizedCurrently"] if "hospitalizedCurrently" in c_json else 0
self.on_ventilator = c_json[
    "onVentilatorCurrently"] if "onVentilatorCurrently" in c_json else 0
self.percent_infected = round(
    self.confirmed / self.population * 100, 2)
self.percent_tested = round(
    self.total_tested / self.population * 100, 2)

if self.confirmed != 0:
    self.percent_recovered = round(
        self.recovered / self.confirmed * 100, 2)
    self.death_rate = round(self.deaths / self.confirmed * 100, 2)
    self.percent_active = round(self.active / self.confirmed * 100, 2)
else:
    self.percent_recovered = round(0, 2)
    self.death_rate = round(0, 2)
    self.percent_active = round(0, 2)

class Covid19_State_Node(Covid19_Node):

    def __init__(self, c_json):
        self.name = c_json["state"]
        self.date = c_json["dateChecked"][:10]
        self.population = populations[self.name]
        self.confirmed = c_json["positive"] if c_json[
            "positive"] != None else 0
        self.total_tested = c_json["totalTestResults"]
        self.recovered = c_json["recovered"] if c_json[
            "recovered"] != None else 0
        self.deaths = c_json["death"] if c_json["death"] != None else 0
        self.active = self.confirmed - self.deaths - self.recovered

        self.percent_infected = round(
            self.confirmed / self.population * 100, 2)
        self.percent_tested = round(
            self.total_tested / self.population * 100, 2)

        if self.confirmed != 0:
            self.percent_recovered = round(
                self.recovered / self.confirmed * 100, 2)
            self.death_rate = round(self.deaths / self.confirmed * 100, 2)

```



```

        self.percent_active = round(self.active / self.confirmed * 100, 2)
    else:
        self.percent_recovered = round(0, 2)
        self.death_rate = round(0, 2)
        self.percent_active = round(0, 2)

class Covid19_Press():
    # Not used currently, but available if wanted.

    def __init__(self):
        press_list = get_covid19_data("", "", press=True)
        self.press = []
        for article in press_list:
            self.press.append(Covid19_Article(article))

    def __str__(self):
        s = ""
        for i, article in enumerate(self.press):
            s += "{}. {}".format(i, str(article))
        return s

class Covid19_Article():
    # Not used currently, but available if wanted.

    def __init__(self, article_json):
        self.title = article_json[
            "title"] if article_json["title"] != None else "No Title"
        self.url = article_json["url"]
        self.published_date = article_json["publishDate"]
        self.publisher = article_json["publication"]
        self.author = article_json[
            "author"] if "author" in article_json else "Unknown Author"

    def __str__(self):
        return "{}\n\tPublished on {} at {} by {}\n\tLink to article:
        {}\n".format(self.title, self.published_date, self.publisher,
            self.author, self.url)

    def __repr__(self):
        return self.title

class Covid19_Dataset():
    """
    Class that maintains the whole dataset, it tries to load from storage,

```

```
but creates and saves
if not available.
```

Methods:

```
save() - saves the whole dataset to a priorly specified file
load() - attempts to load the dataset
get_all_current() - gets all current data for all nodes in the dataset
get_current(name) - gets the current data for the specified node
get_by_date(name, month, day) - gets the data for a specific node for
                                a specific data
get_all_historical(name) - returns all historical information
                            currently
                                stored
get_historical(name, save=False) - gets historical information for
    name,
                                optionally saves the dataset if
                                changes are made
def retrieve_historical(name) - static method used to actually obtain
    the historical
                                data by creating a list of nodes
```

```
'''
```

```
def __init__(self):
    try:
        self.load()
    except:
        print("Couldn't load, generating new")
        self.names = states
        self.current = {state_data["state"]: Covid19_State_Node(
            state_data) for state_data in get_covid19_data("states",
                "current") if state_data["state"] in states}
        self.current["US"] = Covid19_USA_Node(
            get_covid19_data("us", "current")[0])
        self.current["WORLD"] = Covid19_World_Node(
            get_covid19_data("world", "current")[0]["data"])
        self.press = Covid19_Press()
        self.history = {state: {} for state in self.names}
        self.history["US"] = {}
        for state in self.history:
            self.get_historical(state)
        self.save()

def save(self):
    # save file
    data_file = open(dataset_save_file, 'wb')
    data_file.write(pickle.dumps(self.__dict__))
```

```

data_file.close()

# save time saved, so data doesn't get too stale
time_file = open(last_saved_file, "w")
time_file.write(str(datetime.datetime.now()).split(".")[0])

print("Dataset saved at {}".format(datetime.datetime.now()))

def load(self):
    # load time and check if stale
    time_file = open(last_saved_file, "r")
    last_saved = time_file.read()
    time_file.close()
    modify_time = datetime.datetime.strptime(
        last_saved, "%Y-%m-%d %H:%M:%S")
    time_since_modify = datetime.datetime.now() - modify_time
    if time_since_modify > dataset_timeout:
        raise RuntimeError # will now be reloaded instead

    # if recent enough, read the file
    file = open(dataset_save_file, "rb")
    dataPickle = file.read()
    file.close()

    self.__dict__ = pickle.loads(dataPickle)
    print("Successfully loaded Covid-19 Dataset")

def get_all_current(self):
    return self.current

def get_current(self, name):
    return self.current[name]

def get_by_date(self, name, month, day):
    history = self.get_historical(name)
    # make the data string
    date_string = "2020-"
    if len(month) < 2:
        date_string += "0"
    date_string += month
    date_string += "-"
    if len(day) < 2:
        date_string += "0"
    date_string += day

    # search for that day
    for day in history:

```

```

        if day.date == date_string:
            return day
        # otherwise day does not exist, raise error
        raise KeyError

def get_all_historical(self, name):
    """
    returns all available historical data
    """
    return self.history

def get_historical(self, name, save=False):
    if len(self.history[name]) == 0:
        self.history[name] = Covid19_Dataset.retrieve_historical(name)
    if save:
        self.save()
    return self.history[name]

@staticmethod
def retrieve_historical(name):
    historical = []
    name = name.lower()
    # assign the correct level
    level = "state"
    if name == "world":
        level = "world"
    elif name == "us":
        level = "us"
    name = None

    # retrieve the history
    history = get_covid19_data(level, "daily", name)

    # pick which class to make
    if level == "world":
        Make_Node = Covid19_World_Node
    elif level == "us":
        Make_Node = Covid19_USA_Node
    else:
        Make_Node = Covid19_State_Node

    # add nodes to the history
    for time in history:
        try:
            historical.append(Make_Node(time))
        except KeyError:

```

```

        break
    return historical

if __name__ == "__main__":
    data = Covid19_Dataset()

```

7.2 Serial/MINMON Python Code

```

'''
start_bunker_knowledge.py

Main file used, this is the access point into the whole system. It runs
a MINMON, terminal style interface where commands can be input, sent,
and displayed on the TFT screen. This is where the baud rate and com
port can be changed.

'''

import sys
sys.path.insert(1, "../RetrievalSystem")
from processing import *
from retrieval import *
import serial
import time
import threading
from plotly.subplots import make_subplots
import plotly.express as px
import plotly.graph_objects as go
import pandas as pd

us_state_abbrev = {
    'Alabama': 'AL',
    'Alaska': 'AK',
    'American Samoa': 'AS',
    'Arizona': 'AZ',
    'Arkansas': 'AR',
    'California': 'CA',
    'Colorado': 'CO',
    'Connecticut': 'CT',
    'Delaware': 'DE',
    'District of Columbia': 'DC',
    'Florida': 'FL',
    'Georgia': 'GA',
    'Guam': 'GU',
    'Hawaii': 'HI',

```

```

    'Idaho': 'ID',
    'Illinois': 'IL',
    'Indiana': 'IN',
    'Iowa': 'IA',
    'Kansas': 'KS',
    'Kentucky': 'KY',
    'Louisiana': 'LA',
    'Maine': 'ME',
    'Maryland': 'MD',
    'Massachusetts': 'MA',
    'Michigan': 'MI',
    'Minnesota': 'MN',
    'Mississippi': 'MS',
    'Missouri': 'MO',
    'Montana': 'MT',
    'Nebraska': 'NE',
    'Nevada': 'NV',
    'New Hampshire': 'NH',
    'New Jersey': 'NJ',
    'New Mexico': 'NM',
    'New York': 'NY',
    'North Carolina': 'NC',
    'North Dakota': 'ND',
    'Northern Mariana Islands': 'MP',
    'Ohio': 'OH',
    'Oklahoma': 'OK',
    'Oregon': 'OR',
    'Pennsylvania': 'PA',
    'Puerto Rico': 'PR',
    'Rhode Island': 'RI',
    'South Carolina': 'SC',
    'South Dakota': 'SD',
    'Tennessee': 'TN',
    'Texas': 'TX',
    'Utah': 'UT',
    'Vermont': 'VT',
    'Virgin Islands': 'VI',
    'Virginia': 'VA',
    'Washington': 'WA',
    'West Virginia': 'WV',
    'Wisconsin': 'WI',
    'Wyoming': 'WY',
    'USA': 'US',
    'World': 'WORLD'
}

abbrev_to_name = {v: k for k, v in us_state_abbrev.items()}

```

```

class Serial_Control():

    def __init__(self, port, baud_rate):
        # Loading the dataset can take a while
        print("-Please wait for startup (could take up to 10 seconds)")
        self.dataset = Covid19_Dataset()
        self.port = port
        self.baud_rate = baud_rate
        self.big_board_serial = None

    def start(self):
        # starts the serial connection and puts us in terminal mode
        self.big_board_serial = serial.Serial(self.port, self.baud_rate)
        self.startup()
        self.terminal()

    def startup(self):
        world_data = str(self.dataset.get_current("WORLD"))
        # send world data to display on startup
        self.send_data(world_data, receipt=False)
        print("\n
            *****")
        print(" * Welcome to Bunker Knowledge, the Covid-19 Database Terminal! *")
        print("
            *****\n")

        print("-To get help, please type help\n")

    def terminal(self):
        # terminal MINMON style command checker
        while True:
            command = self.get_command().strip()
            try:
                if command == "exit":
                    # ends the program
                    break
                elif command == "help":
                    # help_file.txt has the what is printed here
                    self.help()
                elif 'graph' in command:
                    # graphs the data for the specified name
                    split_command = command.split(" ")
                    name = " ".join((split_command)[: -1])
                    name = self.clean_name(name)
                    data = str(self.dataset.get_current(name))

```



```

        self.send_data(data)
        self.graph(name)

    elif len(command.split(" ")) >= 3:
        # particular date, send that data
        # [name] [month] [day] is command format
        split_command = command.split(" ")
        day = split_command[-1]
        month = split_command[-2]
        name = " ".join((split_command)[: -2])
        name = self.clean_name(name)
        data = str(self.dataset.get_by_date(name, month, day))
        self.send_data(data)
    else:
        # singular name, send data for today's date
        name = self.clean_name(command)
        data = str(self.dataset.get_current(name))
        self.send_data(data)

except KeyError:
    error = "No data for {}".format(command)
    print(error)
    error += "\nEND\n"
    self.big_board_serial.write(error.encode())

self.big_board_serial.close()

def clean_name(self, name):
    # function to take the name of a state and convert it correctly into the
    # stored
    # two letter postal code
    if len(name) != 2 and name != "world":
        # not using state code
        name = name.split(" ")
        for i, x in enumerate(name):
            name[i] = x.capitalize() # capitalize multi letter states
        name = " ".join(name)
        name = us_state_abbrev[name]
    name = name.upper()
    return name

def send_data(self, data, receipt=True):
    # sends the data line by line, and sends the delimiter specified
    # by the psoc
    data = data.split("\n")
    for dat in data:
        self.big_board_serial.write(dat.encode())

```

```

        self.big_board_serial.write("\n".encode()) # delimited
    if receipt:
        print(r'Sent: {}'.format(data))

def get_command(self):
    # waits for a new command
    return input(" > ").lower()

def help(self):
    # Prints from my help file command instructions
    help_file = "help_file.txt"
    with open(help_file, "r") as f:
        text = f.read()
        print(text)

def graph(self, name):
    # graphs the data, was originally intended to be used to send over port,
    # but shifted because of
    # graphics library limitations
    data = get_timeseries(self.dataset, name)
    df = pd.DataFrame(
        data, columns=["Date", "Confirmed", "Active", "Deaths", "Recovered",
            "New Cases", "New Deaths"])

    # Add data
    fig = make_subplots(rows=1, cols=2)
    name = abbrev_to_name[name]

    # Create and style trace 1, line graphs of info
    fig.add_trace(go.Scatter(x=df["Date"], y=df["Confirmed"],
        name='Confirmed',
        line=dict(color='red', width=4)), row=1, col=1)
    fig.add_trace(go.Scatter(x=df["Date"], y=df["Active"], name='Active',
        line=dict(color='orange', width=4)), row=1, col=1)
    fig.add_trace(go.Scatter(x=df["Date"], y=df["Deaths"], name='Deaths',
        line=dict(color='black', width=4)), row=1, col=1)
    fig.add_trace(go.Scatter(x=df["Date"], y=df["Recovered"],
        name='Recovered',
        line=dict(color='green', width=4)), row=1, col=1)

    # Create and style trace 2, bar graph of new cases
    fig.add_trace(go.Bar(x=df["Date"], y=df[
        "New Cases"], name="Daily New Cases"), row=1, col=2)
    fig.add_trace(go.Bar(x=df["Date"], y=df[
        "New Deaths"], name="Daily New Deaths"), row=1, col=2)

    fig.update_layout(title="{} Covid-19 Statistics".format(name),

```

```

        xaxis_title='Date',
        yaxis_title='Cases')

fig.show()

def get_timeseries(dataset, name):
    # retrieves the dates, confirmed, active, deaths, and recovered time series
    # from
    # a dataset
    history = dataset.get_historical(name)
    history.reverse() # already sorted, reverse
    data = {"Date": [], "Confirmed": [],
            "Active": [], "Deaths": [], "Recovered": [],
            "New Cases": [], "New Deaths": []}
    oldconfirmed = 0
    olddeaths = 0
    for date in history:
        data["Date"].append(pd.to_datetime(
            date.date, infer_datetime_format=True))
        data["Confirmed"].append(date.confirmed)
        data["Active"].append(date.active)
        data["Deaths"].append(date.deaths)
        data["Recovered"].append(date.recovered)
        data["New Cases"].append(date.confirmed - oldconfirmed)
        data["New Deaths"].append(date.deaths - olddeaths)
        oldconfirmed = date.confirmed
        olddeaths = date.deaths
    return data

if __name__ == "__main__":
    port = "COM3"
    baud = 57600
    connection = Serial_Control(port, baud)
    connection.start()

```

7.3 Serial and Drawing PSOC C Code

```

/* =====
 *
 * Copyright MIT 6.115, 2013
 * All Rights Reserved
 * UNPUBLISHED, LICENSED SOFTWARE.
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION
 * WHICH IS THE PROPERTY OF MIT 6.115.

```

```

*
* This file is necessary for your project to build.
* Please do not delete it.
*
* =====
*/

#include <device.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#include "GUI.h"
#include "tft.h"

// Function defintions
void MainTask(void); // Used to initialize the GUI
void create_rects(); // Creates and stores the rectangles for stat display on GUI
void display_stats(int word_count); // Displays sent stats on the GUI
void clear_data(int word_count); // Clears the word array
void color_change(char* word, char* check_word, GUI_COLOR color); // Changes the
    color for particular words

// Serial Port Data Processing Constants

#define MARGIN 10 // 10 from the age to look nice
#define Y_INC 20 // go down by 20 pixels when y is incremented
#define WORD_SIZE 50
#define NUM_RECTS 30 // US has 28 data points, can expand if needed

// Screen size for TFT 35628MP from MPJA, 240x320
#define SCREEN_X 240
#define SCREEN_Y 320

// New GUI Color Constants
#define WHITE 0xFFFFFF
#define RED 0xFF0000
#define GREEN 0x00FF00
#define BLUE 0x0000FF
#define CYAN 0x00FFFF
#define ORANGE 0xFF9933
#define MAGENTA 0xFF00FF
#define YELLOW 0xFFFF00

// Serial Port Data Processing variables

```

```

char end_command[4] = "END";
uint8 printindex = 0; // used to keep the index of which word we're on
char delimiter = '\n'; // this delimiter expected in the sent data
int wordindex = 0; // used to track which character of a word we're modifying
GUI_RECT rects[NUM_RECTS]; // predefined rectangles to store in
char data[NUM_RECTS][WORD_SIZE]; // stores the sent data to display

CY_ISR(RX_INT)
{
    char sent = UART_ReadRxData();
    if (sent == delimiter) {
        // end of word, store in array, print if end of sending
        // first sent is name, then date, then alternating key, value pairs
        if (strcmp(data[printindex], end_command) == 0) {
            // END, so go to display, printindex is number of words +1
            display_stats(printindex);
            clear_data(printindex);
            printindex = 0;
        } else {
            // This word is done, move to next word
            printindex++;
        }
        wordindex = 0; // start at the beginning of next word
    } else {
        // add sent to the current word
        data[printindex][wordindex] = sent; // add to word
        data[printindex][wordindex+1] = '\0'; // make it a properly terminated
        string
        wordindex++; // prepare for next character
    }
    UART_WriteTxData(sent);
}

void main()
{
    CyGlobalIntEnable;
    rx_int_StartEx(RX_INT); // start RX interrupt (look for CY_ISR with RX_INT
    address)
    // for code that writes received bytes to LCD_1.

    UART_Start(); // initialize UART
    UART_ClearRxBuffer();
    SPIM_1_Start(); // initialize SPIM component

    LED_1_Write(1);
}

```

```

MainTask();                // all of the emWin exmples use MainTask() as the
    entry point

for(;;) {
    // Unused for now, only interrupts
}

}

void MainTask()
{
    GUI_Init();                // initilize graphics library
    GUI_Clear();
    GUI_SetFont(&GUI_Font8x16);
    GUI_SetTextMode(GUI_TM_NORMAL);
    create_rects();
}

void create_rects() {
    // Creates the rectangles needed to display our data, creates two full width
    // rectangles then pairs the rest of the way down.
    int x = 0;
    int y = 0;
    uint i;
    for (i = 0; i < NUM_RECTS; i++) {
        if (i < 2) {
            // full width for name and date, aka first two rectangles
            GUI_RECT new_rect = {MARGIN, MARGIN+(y*Y_INC), SCREEN_X-MARGIN,
                MARGIN+((y+1)*Y_INC)};
            rects[i] = new_rect;
            y++; // go to next line
        } else {
            if (x == 0) {
                // left side
                GUI_RECT new_rect = {MARGIN, MARGIN+(y*Y_INC), (SCREEN_X-MARGIN)
                    /2, MARGIN+((y+1)*Y_INC)};
                rects[i] = new_rect;
            } else {
                // right side
                GUI_RECT new_rect = {(SCREEN_X-MARGIN) /2, MARGIN+(y*Y_INC),
                    SCREEN_X-MARGIN, MARGIN+((y+1)*Y_INC)};
                rects[i] = new_rect;
            }
            if (x == 1) {
                y++; // go to next line if end of this line
            }
            x ^= 1; // toggle the side for the next one
        }
    }
}

```

```

    }

}

void display_stats(int word_count) {
    // displays the first display_count things in data
    // expects rects and data to be filled
    GUI_Clear();
    int i;
    for (i = 0; i < word_count; i++) {
        if (i > 1 && i % 2 == 1) {
            // check if last rectangle was a key word so the number changes color
            color_change(data[i-1], "Confirmed", RED);
            color_change(data[i-1], "Infected", RED);
            color_change(data[i-1], "ested", CYAN);
            color_change(data[i-1], "Recovered", GREEN);
            color_change(data[i-1], "Active", YELLOW);

        } else {
            GUI_SetColor(WHITE);
        }
        GUI_DispStringInRectWrap(data[i], &rects[i], GUI_TA_CENTER,
            GUI_WRAPMODE_WORD);
    }
}

void color_change(char* word, char* check_word, GUI_COLOR color) {
    // changes color of the text if check_word is a substring of word
    if (strstr(word, check_word) != NULL) {
        GUI_SetColor(color);
    }
}

void clear_data(int word_count) {
    // clears all the words we've used to prepare for next data send
    int i;
    for (i = 0; i < word_count; i++) {
        memset(data[i], 0, WORD_SIZE);
    }
}

/* [] END OF FILE */

```
