(CSE211s) INTRO. TO EMBEDDED SYSTEMS

PROJECT DOCUMENTATION

Team members:

- Shehab Mahmoud Salah (2100320)
- Abdelrahman Hany Mohamed (2100627)
- Youssef Ahmed Mohamed (2101006)
- Omar Mamon Hamed (2100767)
- Seif El Din Tamer Shawky (2100268)
- Seif Eldeen Ahmed Abdulaziz (2100339)
- Habiba El-sayed Mowafy (2100792)
- Moaz Ragab (2100938)
- Mohamed Waleed Sayed (2100623)
- Ahmed Ashraf Ali (2100255)

0. Project Outline:

Visit the **Repository**

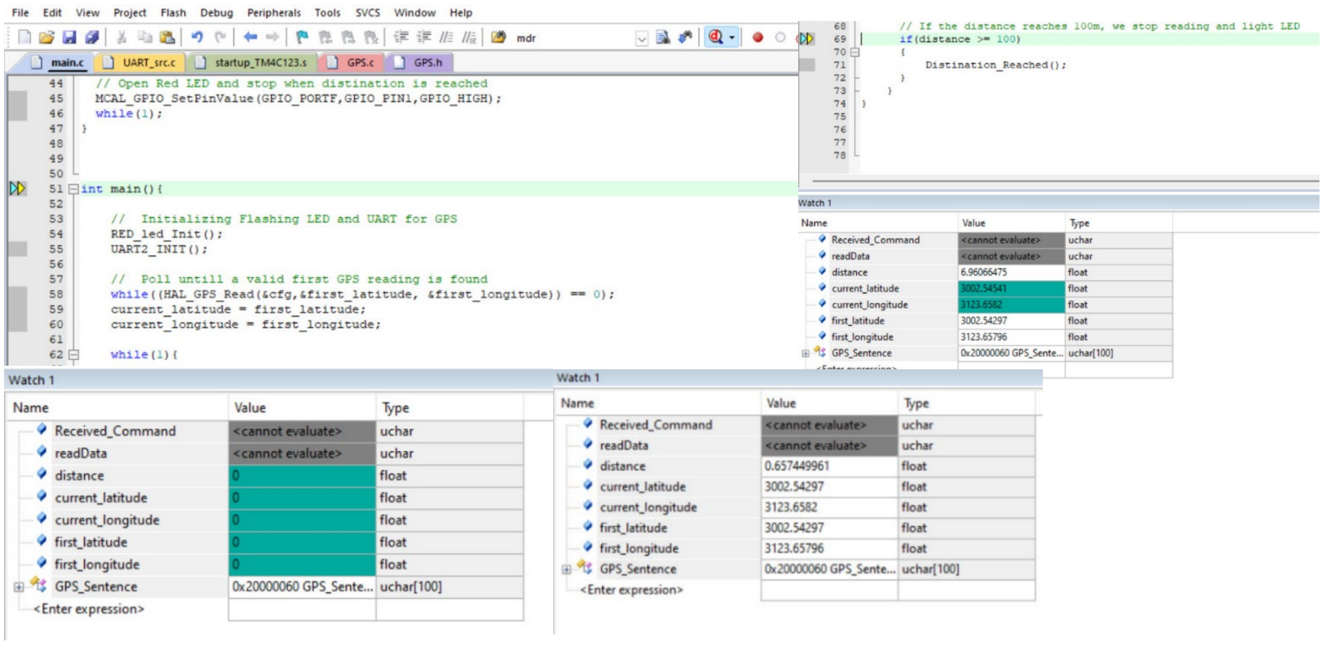
In the given project, we are assigned to a **GPS TRACKING SYSTEM** developed using embedded C programming, by gathering real-time positional coordinates while a microcontroller is in motion (using TM4C123G LaunchPad) after power-on until a destination point is reached. The collected data will be efficiently transferred to a personal computer and visualized on a map application. In the following pages, we'll briefly guide you through the whole process.

FIRST MILESTONE

1. "TIVA" in Action:

Video documenting the TM4C123G LaunchPad up and running:
https://drive.google.com/file/d/1StEVD5kAbZR1trFrzM2KsKkFHEyRKAmb/view

The following *screenshots* were also captured real-time, as we tested our **GPS** module:



File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
mdr
main.c
UART_src.c
startup_TM4C123.s
GPS.c
GPS.h

```c
    // Open Red LED and stop when distination is reached
    MCAL_GPIO_SetPinValue(GPIO_PORTF,GPIO_PIN1,GPIO_HIGH);
    while(1);
}



int main(){

    //  Initializing Flashing LED and UART for GPS
    RED_led_Init();
    UART2_INIT();

    //  Poll untill a valid first GPS reading is found
    while((HAL_GPS_Read(&cfg,&first_latitude, &first_longitude)) == 0);
    current_latitude = first_latitude;
    current_longitude = first_longitude;

    while(1){
```

```c
        // If the distance reaches 100m, we stop reading and light LED
        if(distance >= 100)
        {
            Distination_Reached();
        }
    }
}
```

Watch 1

Name	Value	Type
Received_Command	<cannot evaluate>	uchar
readData	<cannot evaluate>	uchar
distance	6.96066475	float
current_latitude	3002.54541	float
current_longitude	3123.6582	float
first_latitude	3002.54297	float
first_longitude	3123.65796	float
GPS_Sentence	0x20000060 GPS_Sente...	uchar[100]

Watch 1

Name	Value	Type
Received_Command	<cannot evaluate>	uchar
readData	<cannot evaluate>	uchar
distance	0	float
current_latitude	0	float
current_longitude	0	float
first_latitude	0	float
first_longitude	0	float
GPS_Sentence	0x20000060 GPS_Sente...	uchar[100]
<Enter expression>		

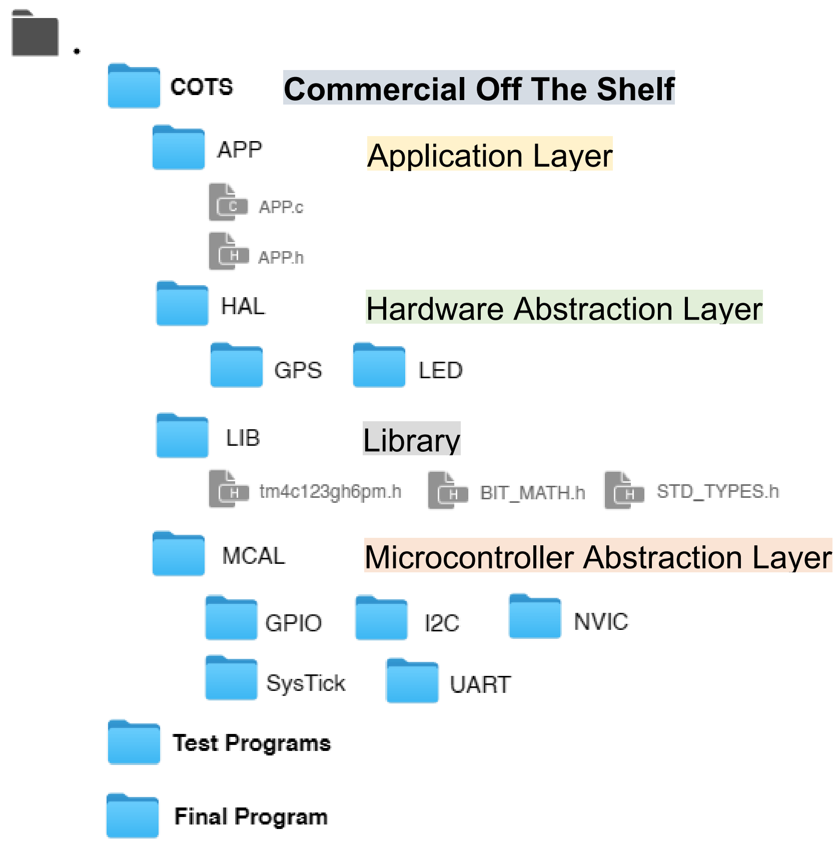
Watch 1

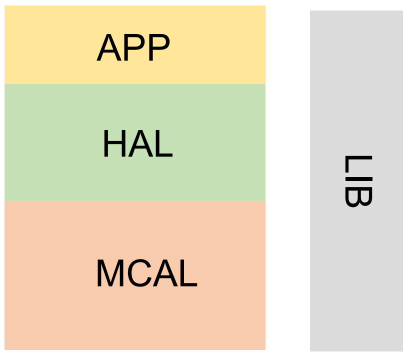
Name	Value	Type
Received_Command	<cannot evaluate>	uchar
readData	<cannot evaluate>	uchar
distance	0.657449961	float
current_latitude	3002.54297	float
current_longitude	3123.6582	float
first_latitude	3002.54297	float
first_longitude	3123.65796	float
GPS_Sentence	0x20000060 GPS_Sente...	uchar[100]
<Enter expression>		

2. Source Code:

Entire project source code can be found on our team's GitHub repo:
https://github.com/MashaWaleed/GPS-System-TIVAC-CSE211

The tree structure of our repo is explained briefly on this page:





The layers, as depicted in the figures above are demystified as follows:

(APP) Application Layer:
It's where the main flow of the program resides and is software specific.

(HAL) Hardware Abstraction Layer:
provides a high-level interface to the hardware. It makes the application code more portable as the same application code can work with different hardware just by using a different HAL implementation.

(MCAL) Microcontroller Abstraction Layer:
manages the microcontroller hardware. It includes our main drivers, i.e: GPIOs, communication interfaces (SPI, I2C, UART), ADCs, etc.

(LIB) Library:
Include third-party or proprietary libraries that the project might depend on. They provide various functions and utilities that are not specific to the hardware or the application but are used by them, such as data structures, math functions, or communication protocols.

SECOND MILESTONE

Project Test Run | https://bit.ly/4anEBYA

3. Functionality & Features:

The GPS system implements the following five core functionalities:

- **GPS Data Acquisition**
  - **Description:** The system acquires GPS data through the UART interface using a dedicated GPS module. The GPS module transmits NMEA sentences, which are industry-standard text-based messages containing various GPS data. The system specifically targets the "`$GPRMC`" sentence for acquiring latitude and longitude coordinates.
  - **Implementation:**
    - **Components Used:**
      - TIVA ARM Microcontroller.
      - GPS module to communicate with UART.
      - UART driver mainly for communication, and other initialized drivers.
    - **Connection Setup and Configuration:**
      - The GPS module is connected to the TIVA microcontroller through the UART interface using appropriate signal connections (TX/RX).
      - The UART driver is configured with the appropriate baud rate (9600 baud for this system) and data format (8 data bits, no parity, 1 stop bit) compatible with the GPS module.
    - **Code/Function Additions:**
      - A dedicated function **`HAL_GPS_Read`** is implemented to continuously receive data from the UART and identify the "`$GPRMC`" sentence using a pattern match. Once identified, the data is **parsed** into separate tokens using the `strtok_r` function, extracting the required information, including **latitude** and **longitude** coordinates.

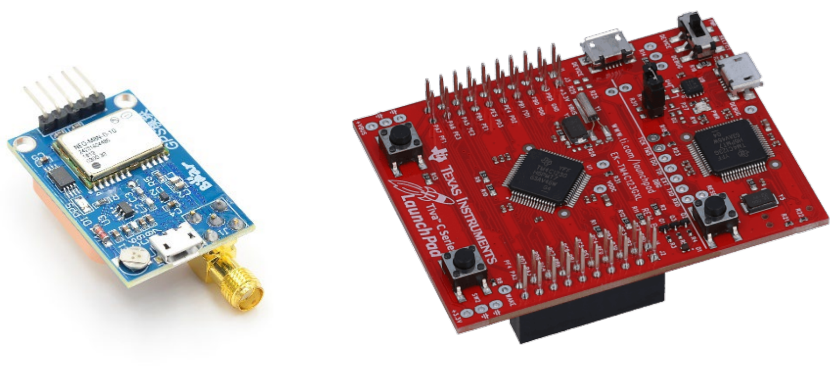


Figure 1 | (Left) NEO-M6N GPS Module (Right) TM4C123GXL TIVA

- **Distance Calculation**
  - **Description:** The system calculates the distance traveled between consecutive GPS coordinates using the **Haversine formula**. The Haversine formula is a sophisticated method for calculating distances on a sphere, like Earth, considering its curvature. It accurately calculates the shortest distance between two points on the Earth's surface, as opposed to using simple Euclidean distance calculations.
  - **Implementation:**
    - **Components Used:**
      - Mathematical libraries for trigonometric operations (sin, cos, atan2)
    - **Code/Function Additions:**
      - A dedicated function `APP_Compute_Distance` is implemented to calculate the distance between two sets of latitude and longitude coordinates using the Haversine formula. The function converts the latitude and longitude from degrees to radians, calculates the difference between the latitudes and longitudes, and applies the Haversine formula to calculate the great-circle distance between the two points.

- **Distance Tracking and Notification**
  - **Description:** The system continuously tracks the total distance traveled. It displays the accumulated distance on an LCD and triggers an LED notification when the distance exceeds a predefined threshold (**100 meters** in this case). This threshold can be easily adjusted according to the specific application.
  - **Implementation:**
    - **Components Used:**
      - TIVA ARM Microcontroller
      - LCD display with 16x2 character display capacity
      - Red-lit LEDs for visual notification
    - **Connection Setup and Configuration:**
      - The LCD is connected to the microcontroller using appropriate GPIO pins for data (D4-D7) and control signals (RS, RW, E).
      - The red LEDs are connected to the microcontroller using GPIO pins.
      - The LCD and LED GPIO pins are configured for output. The LCD display is initialized using a function `HAL_LCD_Init`, setting up the control pins and setting the LCD to 4-bit mode for data communication.
    - **Code/Function Additions:**

            - Functions are implemented to display the distance traveled on the LCD using `HAL_LCD_Send_Number` and display custom messages using `HAL_LCD_Send_String`.
            - Functions are implemented to trigger red LEDs when the destination is reached (distance threshold is exceeded) using `HAL_LED_SET`.

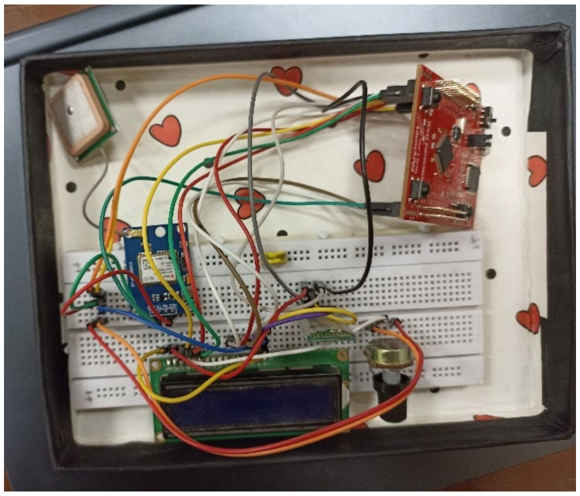


Figure 2 | The breadboard setup with the NEO-6M GPS module, TIVA, LCD display, and a GPS antenna. These components are wired as a GPS data collection and display system.

- **Data Dumping**
    - **Description:** The system allows the user to dump the logged GPS data to a PC via the UART interface. This enables the user to access the recorded trajectory for further analysis, visualization, or storage.
    - **Implementation:**
        - **Components Used:**
            - TIVA ARM Microcontroller.
            - UART driver.
        - **Connection Setup and Configuration:**
            - The microcontroller is connected to a PC via the UART interface using a dedicated serial port.
            - The UART driver is configured for communication with the PC (usually at 9600 baud).
        - **Code/Function Additions:**
            - Functions are implemented to read logged data from the EEPROM and send it to the PC via the UART using `MCAL_UART_Send_String`. The data is sent in a human-readable format with each GPS coordinate represented as a string separated by commas (e.g., "$lat,long\n\r").

- **Data Visualization and Trajectory Plotting**
    - **Description:** The system allows for visualization and plotting of the recorded GPS trajectory using a Python script (`communication.py`). This script retrieves logged data from the microcontroller via the UART interface and generates an interactive map using the Folium library, displaying the path traced by the user.
    - **Implementation:**
        - **Components Used:**
            - **Python:** The core programming language for the visualization script.
            - **Folium:** A Python library for creating interactive Leaflet maps, enabling the visualization of geographical data.
            - **Serial Communication Library:** A Python library for establishing serial communication (e.g., `pyserial`) to receive data from the microcontroller.
            - **Tkinter:** Python's built-in GUI toolkit is used for the interactive user interface of the Python script, allowing the user to select the COM port and initiate the data retrieval process.
        - **Connection Setup and Configuration:**
            - The microcontroller is connected to a PC via a dedicated serial port.
            - The Python script utilizes the chosen COM port to communicate with the microcontroller.
            - The Python script requires the user to select the appropriate COM port connected to the microcontroller.
            - Once the COM port is selected, the script initiates communication with the microcontroller and retrieves the logged data.
            - The script then processes the data, extracting **latitude** and **longitude** coordinates, and generates an interactive map using Folium, **plotting** the recorded trajectory.
            - The interactive map is saved as an HTML file (`map.html`) for easy access and viewing in a web browser.
        - **Code/Function Additions:**
            - **Data Retrieval:** A function `receive_data` in the Python script handles the process of receiving data from the microcontroller via the UART interface. It parses the incoming data and extracts latitude and longitude coordinates.
            - **Map Generation:** The script uses the Folium library to create an interactive map object. A `folium.Map` object is initialized with the starting coordinates from the received data and a zoom level for the initial view.

- **Trajectory Plotting:** The script uses `folium.PolyLine` to draw a line on the map, representing the traced trajectory. The coordinates received from the microcontroller are used to define the line's points.
- **Map Saving:** The `m.save("map.html")` command saves the generated map as an HTML file.

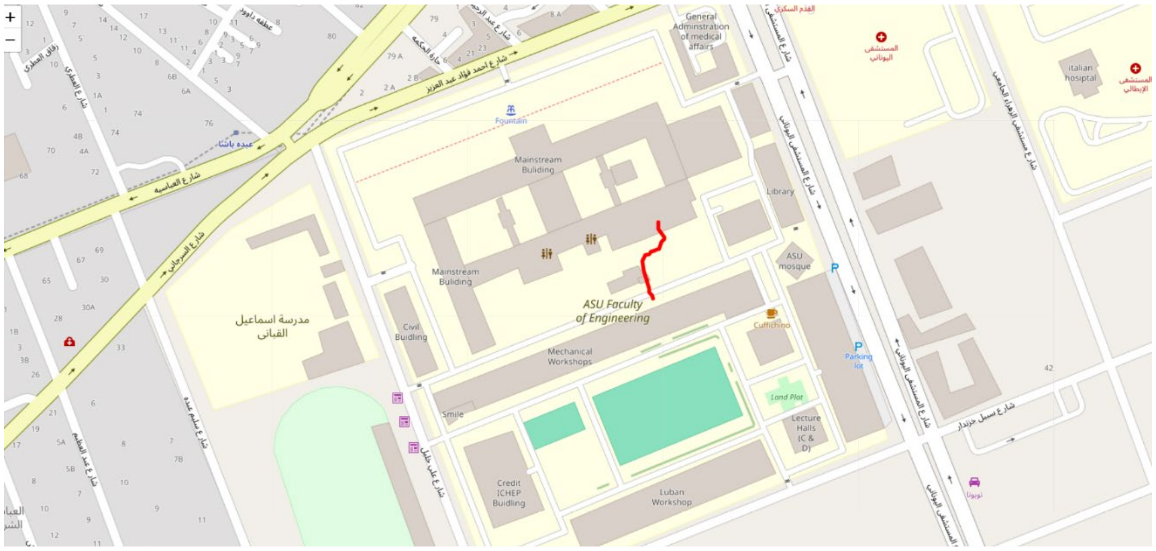


Figure 3 | the traced trajectory plotted onto the GPS-fetched map.

4. Testing & Validation:

The GPS system was carefully tested in a series of scenarios, covering both functional and performance aspects.

- **GPS Data Acquisition:** The system was tested in different locations with varying GPS signal strengths to ensure accurate acquisition of latitude and longitude coordinates. The system's response was monitored accordingly.
- **Distance Calculation:** The distance calculation function was tested with known distances between GPS coordinates obtained from online mapping services. The results were analyzed to validate the accuracy and reliability of the Haversine formula implementation.
- **Distance Tracking and Notification:** The system was tested by moving within a defined area (approximately 100 meters) while monitoring the LCD display and LED notification. The system accurately tracked the movement as shown in the video attached earlier.
- **Data Logging and Dumping:** The system was tested to ensure correct storage and retrieval of GPS coordinates to and from the EEPROM. After logging data, the system was commanded to dump the data to a PC via the UART interface.

5. Conclusion and Future Work

This project successfully demonstrated the integration of a GPS module with a microcontroller to track real-time positional coordinates. The system effectively gathered and stored data, communicated with a PC, and visualized the trajectory on a map. The project's goals were achieved through the collaborative efforts of the team, ensuring accurate GPS data acquisition, distance calculation, and real-time tracking.

For future work, the system could be enhanced by:

1. Improving Power Efficiency: Implementing low-power modes for the microcontroller to extend battery life during prolonged use.

2. Enhanced Data Visualization: Integrating more advanced mapping and data visualization tools to provide more detailed insights.

3. Real-time Data Transmission: Enabling live data transmission over the internet for remote tracking and monitoring applications.

These improvements would increase the system's robustness and applicability in various real-world scenarios. The knowledge gained from this project lays a strong foundation for future advancements in embedded GPS tracking systems.