CSE211s: Embedded Project Documentation

Submitted To:

Dr. Ashraf Salem

Dr. Bassem Amin

Contents

1. Introduction

This project aims to develop a GPS-based tracking system that detects the user's real-time position and provides additional features such as a buzzer alert upon reaching a destination, an LCD display for location information, and a GUI (Graphical User Interface) built with Qt to visualize the travelled path.

2. Objectives

- Implement **real-time GPS positioning** to track user movement.
- Use a **buzzer** to alert the user when they reach a predefined destination.
- Display distinct locations names in our college on an **LCD screen**.
- Develop a **Qt-based GUI** to map and visualize the travelled path.

3. System Components

Hardware Components

- GPS Module (e.g., NEO-6M) – Provides real-time latitude and longitude.
- Microcontroller (e.g., Arduino, ESP32, or Raspberry Pi) – Processes GPS data and controls peripherals.
- Buzzer – Activates when the user reaches the target location.
- LCD Display (16x2) – Shows current position and status.
- Power Supply (Battery or USB) – Powers the system.

Software Components

- Keil Embedded Firmware– Reads GPS data and controls the buzzer & LCD.
- Qt Framework– Creates a GUI to display the traveled path.
- Serial Communication – Links the microcontroller with the GUI application.

4. Working Principle

1. **GPS Data Acquisition:**
   - The GPS module continuously sends location data (latitude, longitude, speed).
2. **Position Detection & Buzzer Activation:**
   - The microcontroller checks if the current coordinates match the destination.
   - If matched, the **buzzer beeps** to notify the user.
3. **LCD Display:**
   - Shows real-time location, distance to destination, and speed.
4. **Qt GUI Visualization:**
   - The GUI receives location updates via serial/USB and plots the path on a map.

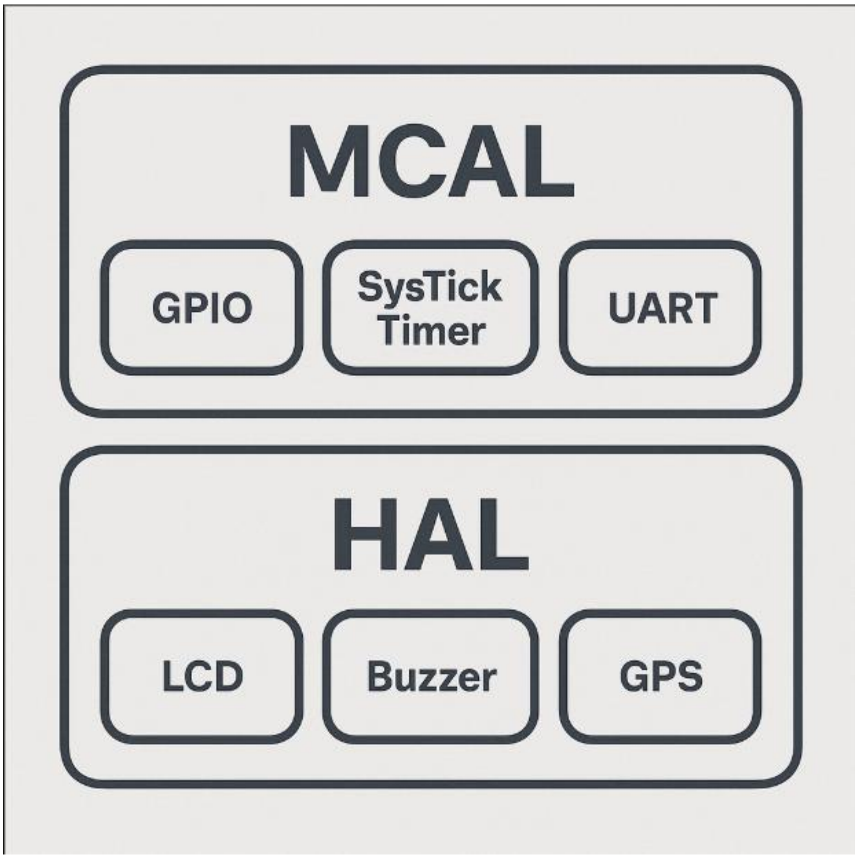
5. Features (That we assume to be bonus)

Destination Alert (Buzzer) – Audible indication when the target is reached.
Interactive Qt GUI – Displays the travelled route with markers.
Variable Lcd Brightness – Being controlled by potentiometer
Structured Code – layered architecture with Hardware Abstraction Layer (HAL) and Microcontroller Abstraction Layer (MCAL) for better modularity, reusability, and maintainability.
Path Calculation – Calculation of full path covered by our Gui
Lcd Driver –

6. Team Members and their Contribution:

Name	ID	Contribution
Mohammed Khaled Ahmed Elsaid	2201057	**GUI** and contributed in **UART** driver implementation
Ahmed Mustafa Abdulbadea Dawood	2200667	**GPS** driver implementation
Mosa Abdelaziz Morgan Abdelaziz	2200257	**LCD** Driver Implementation
Houssam Magdy Mohammed	2200796	**Systick** and Contributed in **UART** and **Main logic**
Bassam Hussam Mashaly	2200084	**Port-F** Driver Implementation
Hazem Youssef Mahmoud	2200183	**Main logic** and **Buzzer** configiration
Youssef Yacoub Radi Fareed	2200649	**GPIO** Driver Implementation

7.Drivers Explanation



GPIO Driver Documentation

Overview

The **General-Purpose Input/Output (GPIO) Driver** provides an interface to control the GPIO pins of the **Tiva C TM4C123GH6PM microcontroller**. It follows a **modular and layered architecture** (MCAL) for portability across different microcontrollers.

Key Features

✓ Configures pins as **input/output**
✓ Supports **pull-up/pull-down resistors**
✓ Enables **digital I/O operations**
✓ Provides **register-level abstraction**

2. Hardware Abstraction

Base Addresses & Registers

The driver accesses GPIO registers via **memory-mapped addresses** (APB bus). Key registers include:

Register	Function
GPIOx_DIR_R	Sets pin direction (input/output)
GPIOx_AFSEL_R	Alternate function selection
GPIOx_DEN_R	Digital enable
GPIOx_DATA_R	Reads/writes pin data
GPIOx_PUR_R	Pull-up resistor control
GPIOx_ODR_R	Pull-down resistor control

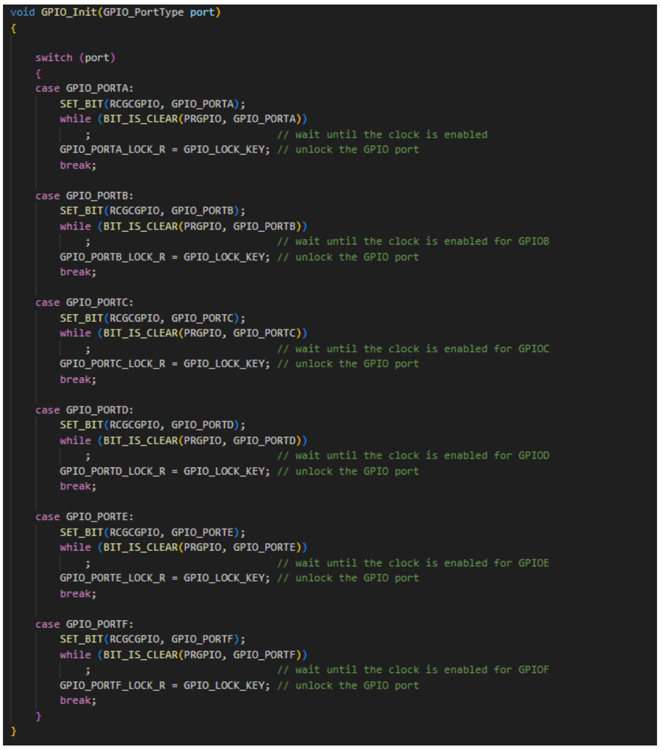
Supported Ports

- **PORTA, PORTB, PORTC, PORTD, PORTE, PORTF**
- Each port has **8 pins (0–7)**.

Functions Documentation:

GPIO_Init ()

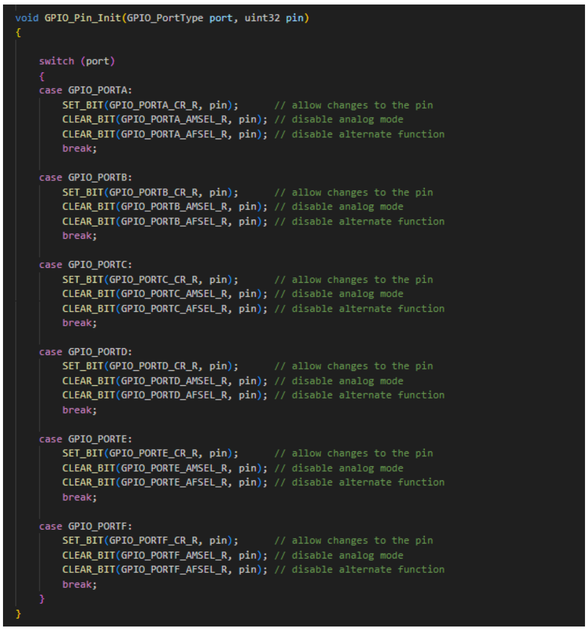
- **Description**: Enables clock gating for the specified GPIO port.
- **Code snippet**:



```c
void GPIO_Init(GPIO_PortType port)
{

    switch (port)
    {
    case GPIO_PORTA:
        SET_BIT(RCGCGPIO, GPIO_PORTA);
        while (BIT_IS_CLEAR(PRGPIO, GPIO_PORTA))
            ;                              // wait until the clock is enabled
        GPIO_PORTA_LOCK_R = GPIO_LOCK_KEY; // unlock the GPIO port
        break;

    case GPIO_PORTB:
        SET_BIT(RCGCGPIO, GPIO_PORTB);
        while (BIT_IS_CLEAR(PRGPIO, GPIO_PORTB))
            ;                              // wait until the clock is enabled for GPIOB
        GPIO_PORTB_LOCK_R = GPIO_LOCK_KEY; // unlock the GPIO port
        break;

    case GPIO_PORTC:
        SET_BIT(RCGCGPIO, GPIO_PORTC);
        while (BIT_IS_CLEAR(PRGPIO, GPIO_PORTC))
            ;                              // wait until the clock is enabled for GPIOC
        GPIO_PORTC_LOCK_R = GPIO_LOCK_KEY; // unlock the GPIO port
        break;

    case GPIO_PORTD:
        SET_BIT(RCGCGPIO, GPIO_PORTD);
        while (BIT_IS_CLEAR(PRGPIO, GPIO_PORTD))
            ;                              // wait until the clock is enabled for GPIOD
        GPIO_PORTD_LOCK_R = GPIO_LOCK_KEY; // unlock the GPIO port
        break;

    case GPIO_PORTE:
        SET_BIT(RCGCGPIO, GPIO_PORTE);
        while (BIT_IS_CLEAR(PRGPIO, GPIO_PORTE))
            ;                              // wait until the clock is enabled for GPIOE
        GPIO_PORTE_LOCK_R = GPIO_LOCK_KEY; // unlock the GPIO port
        break;

    case GPIO_PORTF:
        SET_BIT(RCGCGPIO, GPIO_PORTF);
        while (BIT_IS_CLEAR(PRGPIO, GPIO_PORTF))
            ;                              // wait until the clock is enabled for GPIOF
        GPIO_PORTF_LOCK_R = GPIO_LOCK_KEY; // unlock the GPIO port
        break;
    }
}
```

GPIO_Pin_Init ():

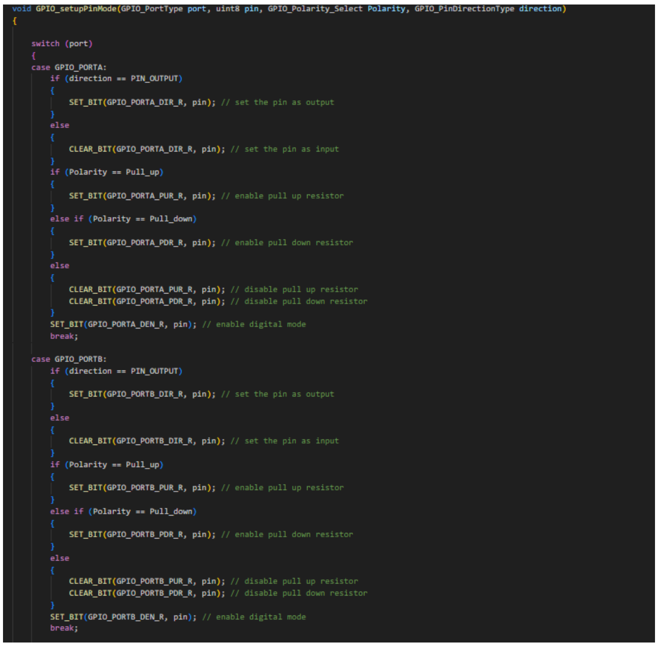
- **Description: Initializes a specific pin (sets default state).**
- **Code snippet**:



```c
void GPIO_Pin_Init(GPIO_PortType port, uint32 pin)
{

    switch (port)
    {
    case GPIO_PORTA:
        SET_BIT(GPIO_PORTA_CR_R, pin);      // allow changes to the pin
        CLEAR_BIT(GPIO_PORTA_AMSEL_R, pin); // disable analog mode
        CLEAR_BIT(GPIO_PORTA_AFSEL_R, pin); // disable alternate function
        break;

    case GPIO_PORTB:
        SET_BIT(GPIO_PORTB_CR_R, pin);      // allow changes to the pin
        CLEAR_BIT(GPIO_PORTB_AMSEL_R, pin); // disable analog mode
        CLEAR_BIT(GPIO_PORTB_AFSEL_R, pin); // disable alternate function
        break;

    case GPIO_PORTC:
        SET_BIT(GPIO_PORTC_CR_R, pin);      // allow changes to the pin
        CLEAR_BIT(GPIO_PORTC_AMSEL_R, pin); // disable analog mode
        CLEAR_BIT(GPIO_PORTC_AFSEL_R, pin); // disable alternate function
        break;

    case GPIO_PORTD:
        SET_BIT(GPIO_PORTD_CR_R, pin);      // allow changes to the pin
        CLEAR_BIT(GPIO_PORTD_AMSEL_R, pin); // disable analog mode
        CLEAR_BIT(GPIO_PORTD_AFSEL_R, pin); // disable alternate function
        break;

    case GPIO_PORTE:
        SET_BIT(GPIO_PORTE_CR_R, pin);      // allow changes to the pin
        CLEAR_BIT(GPIO_PORTE_AMSEL_R, pin); // disable analog mode
        CLEAR_BIT(GPIO_PORTE_AFSEL_R, pin); // disable alternate function
        break;

    case GPIO_PORTF:
        SET_BIT(GPIO_PORTF_CR_R, pin);      // allow changes to the pin
        CLEAR_BIT(GPIO_PORTF_AMSEL_R, pin); // disable analog mode
        CLEAR_BIT(GPIO_PORTF_AFSEL_R, pin); // disable alternate function
        break;
    }
}
```

GPIO_setupPinMode ():

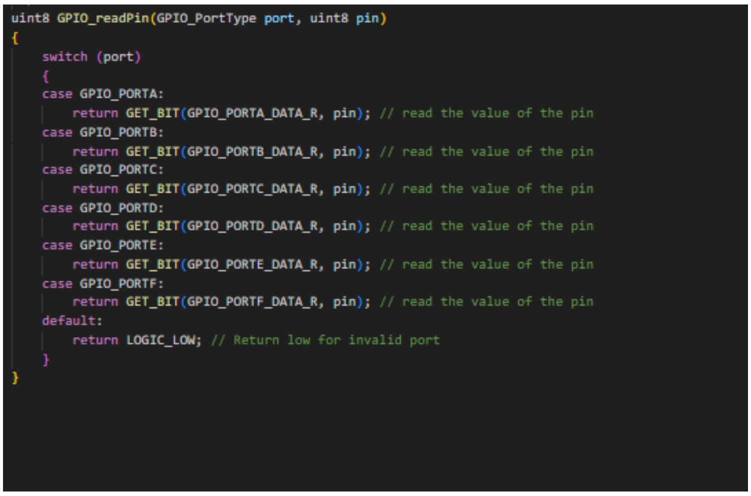
- **Description**: Configures a pin as **input/output** with pull-up/down or floating.
- **Code snippet**:



```c
void GPIO_setupPinMode(GPIO_PortType port, uint8 pin, GPIO_Polarity_Select Polarity, GPIO_PinDirectionType direction)
{

    switch (port)
    {
    case GPIO_PORTA:
        if (direction == PIN_OUTPUT)
        {
            SET_BIT(GPIO_PORTA_DIR_R, pin); // set the pin as output
        }
        else
        {
            CLEAR_BIT(GPIO_PORTA_DIR_R, pin); // set the pin as input
        }
        if (Polarity == Pull_up)
        {
            SET_BIT(GPIO_PORTA_PUR_R, pin); // enable pull up resistor
        }
        else if (Polarity == Pull_down)
        {
            SET_BIT(GPIO_PORTA_PDR_R, pin); // enable pull down resistor
        }
        else
        {
            CLEAR_BIT(GPIO_PORTA_PUR_R, pin); // disable pull up resistor
            CLEAR_BIT(GPIO_PORTA_PDR_R, pin); // disable pull down resistor
        }
        SET_BIT(GPIO_PORTA_DEN_R, pin); // enable digital mode
        break;

    case GPIO_PORTB:
        if (direction == PIN_OUTPUT)
        {
            SET_BIT(GPIO_PORTB_DIR_R, pin); // set the pin as output
        }
        else
        {
            CLEAR_BIT(GPIO_PORTB_DIR_R, pin); // set the pin as input
        }
        if (Polarity == Pull_up)
        {
            SET_BIT(GPIO_PORTB_PUR_R, pin); // enable pull up resistor
        }
        else if (Polarity == Pull_down)
        {
            SET_BIT(GPIO_PORTB_PDR_R, pin); // enable pull down resistor
        }
        else
        {
            CLEAR_BIT(GPIO_PORTB_PUR_R, pin); // disable pull up resistor
            CLEAR_BIT(GPIO_PORTB_PDR_R, pin); // disable pull down resistor
        }
        SET_BIT(GPIO_PORTB_DEN_R, pin); // enable digital mode
        break;
```

GPIO_readPin ():

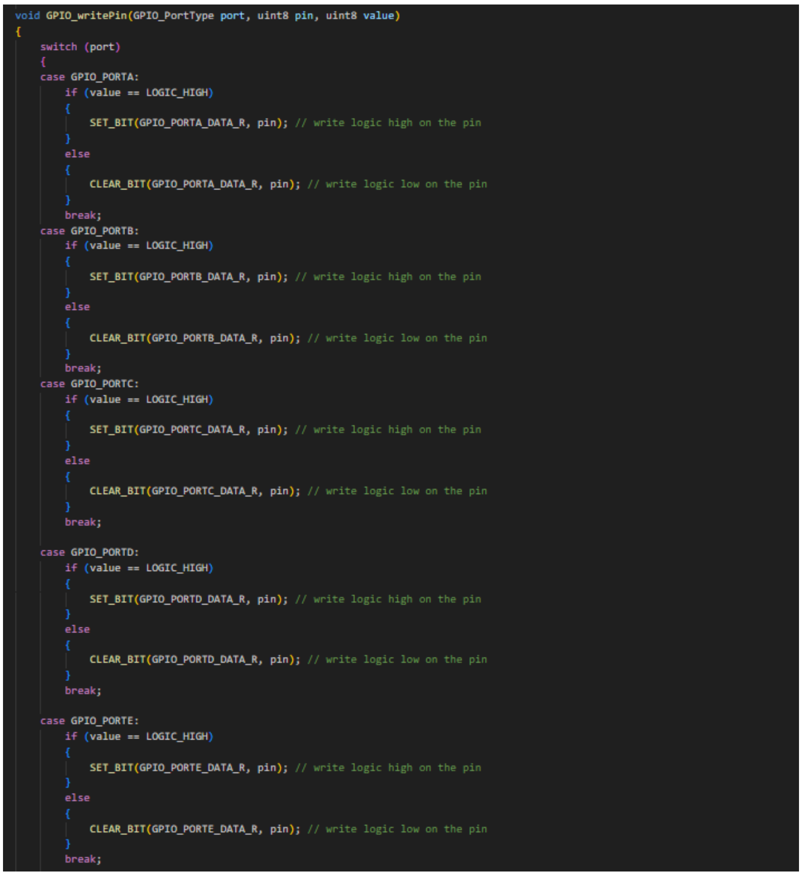
- **Description: Reads the digital value (HIGH/LOW) of a pin.**
- **Code snippet**:



```c
uint8 GPIO_readPin(GPIO_PortType port, uint8 pin)
{
    switch (port)
    {
    case GPIO_PORTA:
        return GET_BIT(GPIO_PORTA_DATA_R, pin); // read the value of the pin
    case GPIO_PORTB:
        return GET_BIT(GPIO_PORTB_DATA_R, pin); // read the value of the pin
    case GPIO_PORTC:
        return GET_BIT(GPIO_PORTC_DATA_R, pin); // read the value of the pin
    case GPIO_PORTD:
        return GET_BIT(GPIO_PORTD_DATA_R, pin); // read the value of the pin
    case GPIO_PORTE:
        return GET_BIT(GPIO_PORTE_DATA_R, pin); // read the value of the pin
    case GPIO_PORTF:
        return GET_BIT(GPIO_PORTF_DATA_R, pin); // read the value of the pin
    default:
        return LOGIC_LOW; // Return low for invalid port
    }
}
```

GPIO_writePin ():

- **Description: Writes a digital value (HIGH/LOW) to a pin.**
- **Code snippet**:



```c
void GPIO_writePin(GPIO_PortType port, uint8 pin, uint8 value)
{
    switch (port)
    {
    case GPIO_PORTA:
        if (value == LOGIC_HIGH)
        {
            SET_BIT(GPIO_PORTA_DATA_R, pin); // write logic high on the pin
        }
        else
        {
            CLEAR_BIT(GPIO_PORTA_DATA_R, pin); // write logic low on the pin
        }
        break;
    case GPIO_PORTB:
        if (value == LOGIC_HIGH)
        {
            SET_BIT(GPIO_PORTB_DATA_R, pin); // write logic high on the pin
        }
        else
        {
            CLEAR_BIT(GPIO_PORTB_DATA_R, pin); // write logic low on the pin
        }
        break;
    case GPIO_PORTC:
        if (value == LOGIC_HIGH)
        {
            SET_BIT(GPIO_PORTC_DATA_R, pin); // write logic high on the pin
        }
        else
        {
            CLEAR_BIT(GPIO_PORTC_DATA_R, pin); // write logic low on the pin
        }
        break;

    case GPIO_PORTD:
        if (value == LOGIC_HIGH)
        {
            SET_BIT(GPIO_PORTD_DATA_R, pin); // write logic high on the pin
        }
        else
        {
            CLEAR_BIT(GPIO_PORTD_DATA_R, pin); // write logic low on the pin
        }
        break;

    case GPIO_PORTE:
        if (value == LOGIC_HIGH)
        {
            SET_BIT(GPIO_PORTE_DATA_R, pin); // write logic high on the pin
        }
        else
        {
            CLEAR_BIT(GPIO_PORTE_DATA_R, pin); // write logic low on the pin
        }
        break;
```

UART Driver Documentation

Overview

The Universal Asynchronous Receiver/Transmitter (UART) Driver provides serial communication capabilities for the Tiva C TM4C123GH6PM microcontroller. It enables full-duplex communication between the microcontroller and peripheral devices (e.g., GPS module, PC). The driver follows a modular design (MCAL layer) for portability.

Key Feature

- Supports **8 UART modules** (UART0–UART7)
- Configurable **baud rate, parity, stop bits**
- **Polling-based** and **FIFO-buffered** operation
- **Memory-mapped register access** for low-level control

Hardware abstraction

Registers	Function
UARTx_DR_R	Data transmit/receive
UARTx_FR_R	Flag register (TX/RX status)
UARTx_IBRD_R	Integer baud rate divisor
UARTx_FBRD_R	Fractional baud rate divisor
UARTx_LCRH_R	Line control (data bits, parity)
UARTx_CTL_R	UART enable/disable

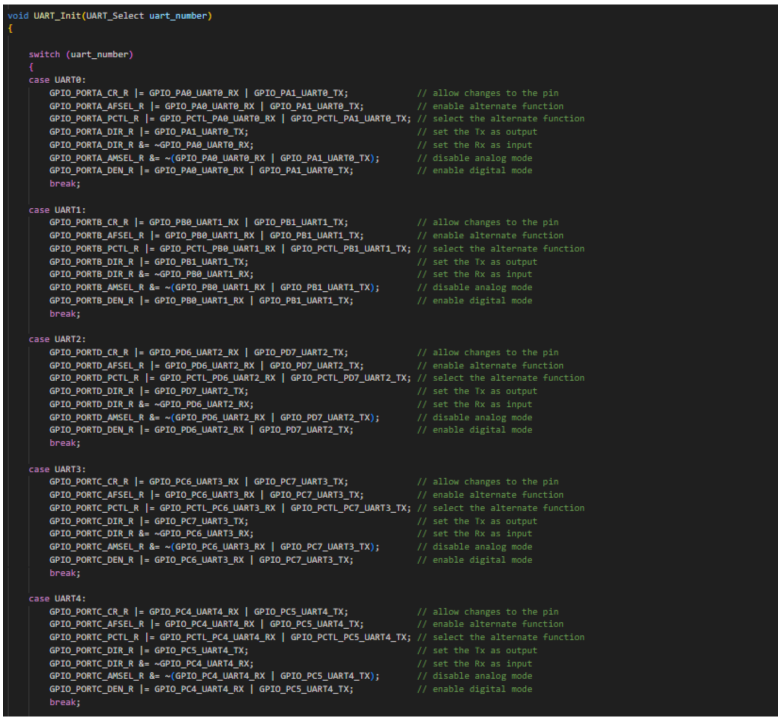
Supported UART Modules

- **UART0–UART7** (each with dedicated GPIO pins)
- **Default pins**:
  - **UART0**: PA0 (RX), PA1 (TX)
  - **UART1**: PB0 (RX), PB1 (TX)
  - **UART2**: PD6 (RX), PD7 (TX)
  - **UART3**: PC6 (RX), PC7 (TX)
  - **UART4**: PC4 (RX), PC5 (TX)
  - **UART5**: PE4 (RX), PE5 (TX)
  - **UART6**: PD4 (RX), PD5 (TX)
  - **UART7**: PE0 (RX), PE1 (TX)

Functions Documentation

UART_Init ()

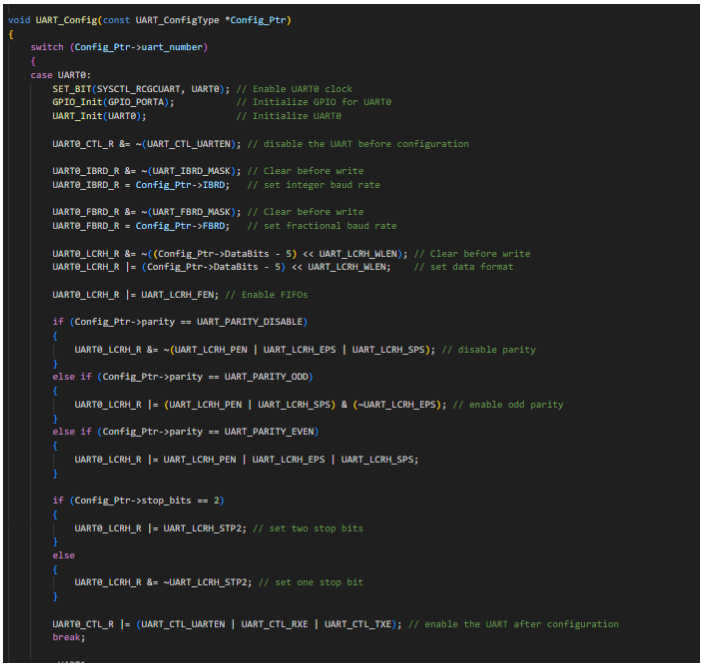
- **Description**: Enables clock gating for the specified UART module.
- **Code snippet**:



```c
void UART_Init(UART_Select uart_number)
{

    switch (uart_number)
    {
    case UART0:
        GPIO_PORTA_CR_R |= GPIO_PA0_UART0_RX | GPIO_PA1_UART0_TX;            // allow changes to the pin
        GPIO_PORTA_AFSEL_R |= GPIO_PA0_UART0_RX | GPIO_PA1_UART0_TX;         // enable alternate function
        GPIO_PORTA_PCTL_R |= GPIO_PCTL_PA0_UART0_RX | GPIO_PCTL_PA1_UART0_TX; // select the alternate function
        GPIO_PORTA_DIR_R |= GPIO_PA1_UART0_TX;                               // set the Tx as output
        GPIO_PORTA_DIR_R &= ~GPIO_PA0_UART0_RX;                              // set the Rx as input
        GPIO_PORTA_AMSEL_R &= ~(GPIO_PA0_UART0_RX | GPIO_PA1_UART0_TX);      // disable analog mode
        GPIO_PORTA_DEN_R |= GPIO_PA0_UART0_RX | GPIO_PA1_UART0_TX;           // enable digital mode
        break;

    case UART1:
        GPIO_PORTB_CR_R |= GPIO_PB0_UART1_RX | GPIO_PB1_UART1_TX;            // allow changes to the pin
        GPIO_PORTB_AFSEL_R |= GPIO_PB0_UART1_RX | GPIO_PB1_UART1_TX;         // enable alternate function
        GPIO_PORTB_PCTL_R |= GPIO_PCTL_PB0_UART1_RX | GPIO_PCTL_PB1_UART1_TX; // select the alternate function
        GPIO_PORTB_DIR_R |= GPIO_PB1_UART1_TX;                               // set the Tx as output
        GPIO_PORTB_DIR_R &= ~GPIO_PB0_UART1_RX;                              // set the Rx as input
        GPIO_PORTB_AMSEL_R &= ~(GPIO_PB0_UART1_RX | GPIO_PB1_UART1_TX);      // disable analog mode
        GPIO_PORTB_DEN_R |= GPIO_PB0_UART1_RX | GPIO_PB1_UART1_TX;           // enable digital mode
        break;

    case UART2:
        GPIO_PORTD_CR_R |= GPIO_PD6_UART2_RX | GPIO_PD7_UART2_TX;            // allow changes to the pin
        GPIO_PORTD_AFSEL_R |= GPIO_PD6_UART2_RX | GPIO_PD7_UART2_TX;         // enable alternate function
        GPIO_PORTD_PCTL_R |= GPIO_PCTL_PD6_UART2_RX | GPIO_PCTL_PD7_UART2_TX; // select the alternate function
        GPIO_PORTD_DIR_R |= GPIO_PD7_UART2_TX;                               // set the Tx as output
        GPIO_PORTD_DIR_R &= ~GPIO_PD6_UART2_RX;                              // set the Rx as input
        GPIO_PORTD_AMSEL_R &= ~(GPIO_PD6_UART2_RX | GPIO_PD7_UART2_TX);      // disable analog mode
        GPIO_PORTD_DEN_R |= GPIO_PD6_UART2_RX | GPIO_PD7_UART2_TX;           // enable digital mode
        break;

    case UART3:
        GPIO_PORTC_CR_R |= GPIO_PC6_UART3_RX | GPIO_PC7_UART3_TX;            // allow changes to the pin
        GPIO_PORTC_AFSEL_R |= GPIO_PC6_UART3_RX | GPIO_PC7_UART3_TX;         // enable alternate function
        GPIO_PORTC_PCTL_R |= GPIO_PCTL_PC6_UART3_RX | GPIO_PCTL_PC7_UART3_TX; // select the alternate function
        GPIO_PORTC_DIR_R |= GPIO_PC7_UART3_TX;                               // set the Tx as output
        GPIO_PORTC_DIR_R &= ~GPIO_PC6_UART3_RX;                              // set the Rx as input
        GPIO_PORTC_AMSEL_R &= ~(GPIO_PC6_UART3_RX | GPIO_PC7_UART3_TX);      // disable analog mode
        GPIO_PORTC_DEN_R |= GPIO_PC6_UART3_RX | GPIO_PC7_UART3_TX;           // enable digital mode
        break;

    case UART4:
        GPIO_PORTC_CR_R |= GPIO_PC4_UART4_RX | GPIO_PC5_UART4_TX;            // allow changes to the pin
        GPIO_PORTC_AFSEL_R |= GPIO_PC4_UART4_RX | GPIO_PC5_UART4_TX;         // enable alternate function
        GPIO_PORTC_PCTL_R |= GPIO_PCTL_PC4_UART4_RX | GPIO_PCTL_PC5_UART4_TX; // select the alternate function
        GPIO_PORTC_DIR_R |= GPIO_PC5_UART4_TX;                               // set the Tx as output
        GPIO_PORTC_DIR_R &= ~GPIO_PC4_UART4_RX;                              // set the Rx as input
        GPIO_PORTC_AMSEL_R &= ~(GPIO_PC4_UART4_RX | GPIO_PC5_UART4_TX);      // disable analog mode
        GPIO_PORTC_DEN_R |= GPIO_PC4_UART4_RX | GPIO_PC5_UART4_TX;           // enable digital mode
        break;
```

UART_Config ()

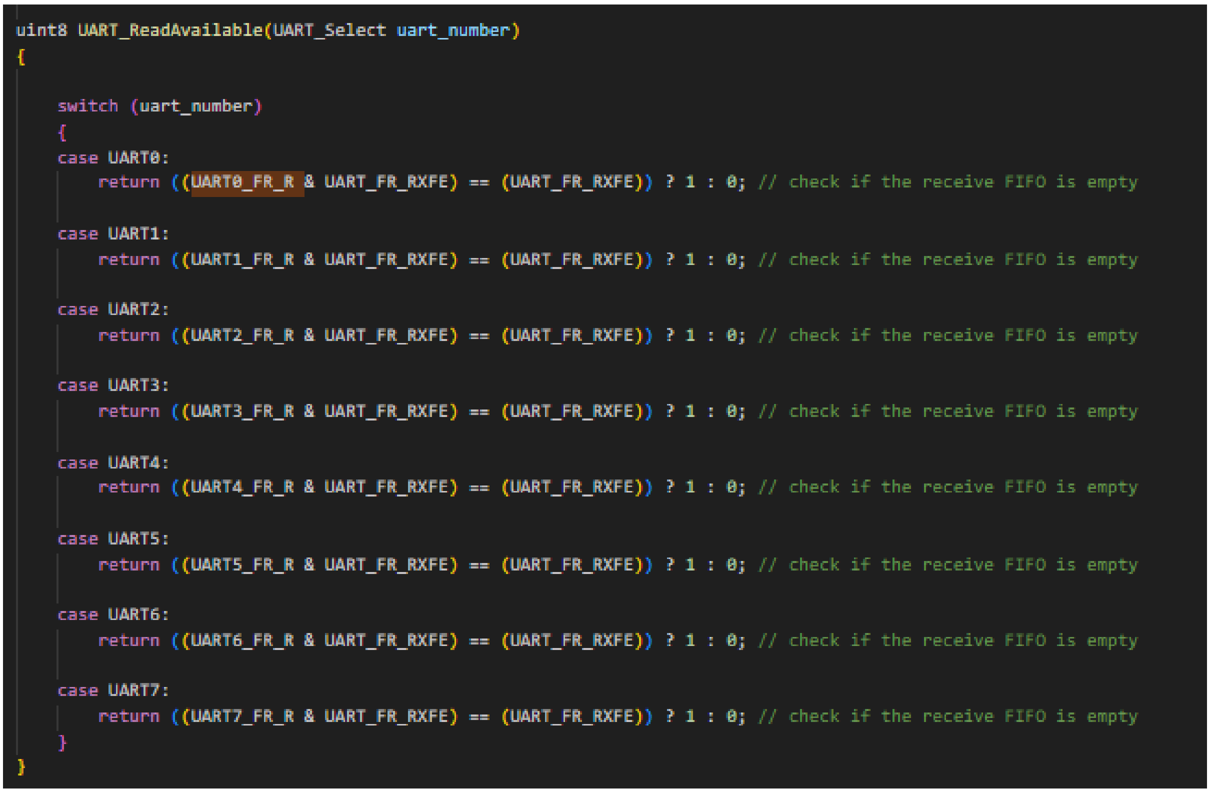
- **Description**: Configures UART parameters (baud rate, data bits, parity, stop bits).
- **Code snippet**:



```c
void UART_Config(const UART_ConfigType *Config_Ptr)
{
    switch (Config_Ptr->uart_number)
    {
    case UART0:
        SET_BIT(SYSCTL_RCGCUART, UART0); // Enable UART0 clock
        GPIO_Init(GPIO_PORTA);           // Initialize GPIO for UART0
        UART_Init(UART0);                // Initialize UART0

        UART0_CTL_R &= ~(UART_CTL_UARTEN); // disable the UART before configuration

        UART0_IBRD_R &= ~(UART_IBRD_MASK); // Clear before write
        UART0_IBRD_R = Config_Ptr->IBRD;   // set integer baud rate

        UART0_FBRD_R &= ~(UART_FBRD_MASK); // Clear before write
        UART0_FBRD_R = Config_Ptr->FBRD;   // set fractional baud rate

        UART0_LCRH_R &= ~((Config_Ptr->DataBits - 5) << UART_LCRH_WLEN); // Clear before write
        UART0_LCRH_R |= (Config_Ptr->DataBits - 5) << UART_LCRH_WLEN;    // set data format

        UART0_LCRH_R |= UART_LCRH_FEN; // Enable FIFOs

        if (Config_Ptr->parity == UART_PARITY_DISABLE)
        {
            UART0_LCRH_R &= ~(UART_LCRH_PEN | UART_LCRH_EPS | UART_LCRH_SPS); // disable parity
        }
        else if (Config_Ptr->parity == UART_PARITY_ODD)
        {
            UART0_LCRH_R |= (UART_LCRH_PEN | UART_LCRH_SPS) & (~UART_LCRH_EPS); // enable odd parity
        }
        else if (Config_Ptr->parity == UART_PARITY_EVEN)
        {
            UART0_LCRH_R |= UART_LCRH_PEN | UART_LCRH_EPS | UART_LCRH_SPS;
        }

        if (Config_Ptr->stop_bits == 2)
        {
            UART0_LCRH_R |= UART_LCRH_STP2; // set two stop bits
        }
        else
        {
            UART0_LCRH_R &= ~UART_LCRH_STP2; // set one stop bit
        }

        UART0_CTL_R |= (UART_CTL_UARTEN | UART_CTL_RXE | UART_CTL_TXE); // enable the UART after configuration
        break;
```

UART_ReadAvailable ():

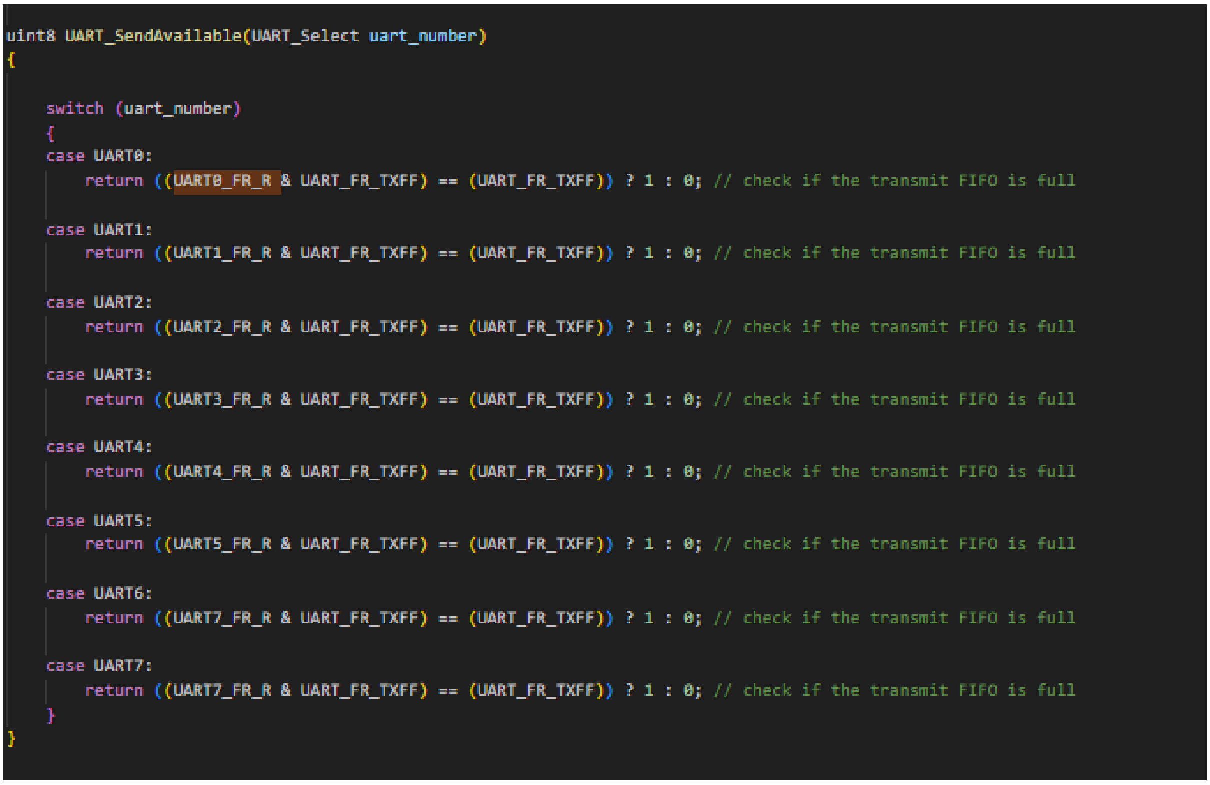
- **Description**: Checks if data is available in the UART receive buffer.
- **Code snippet**:



```
uint8 UART_ReadAvailable(UART_Select uart_number)
{

    switch (uart_number)
    {
    case UART0:
        return ((UART0_FR_R & UART_FR_RXFE) == (UART_FR_RXFE)) ? 1 : 0; // check if the receive FIFO is empty

    case UART1:
        return ((UART1_FR_R & UART_FR_RXFE) == (UART_FR_RXFE)) ? 1 : 0; // check if the receive FIFO is empty

    case UART2:
        return ((UART2_FR_R & UART_FR_RXFE) == (UART_FR_RXFE)) ? 1 : 0; // check if the receive FIFO is empty

    case UART3:
        return ((UART3_FR_R & UART_FR_RXFE) == (UART_FR_RXFE)) ? 1 : 0; // check if the receive FIFO is empty

    case UART4:
        return ((UART4_FR_R & UART_FR_RXFE) == (UART_FR_RXFE)) ? 1 : 0; // check if the receive FIFO is empty

    case UART5:
        return ((UART5_FR_R & UART_FR_RXFE) == (UART_FR_RXFE)) ? 1 : 0; // check if the receive FIFO is empty

    case UART6:
        return ((UART6_FR_R & UART_FR_RXFE) == (UART_FR_RXFE)) ? 1 : 0; // check if the receive FIFO is empty

    case UART7:
        return ((UART7_FR_R & UART_FR_RXFE) == (UART_FR_RXFE)) ? 1 : 0; // check if the receive FIFO is empty
    }
}
```

UART_SendAvaliable ():

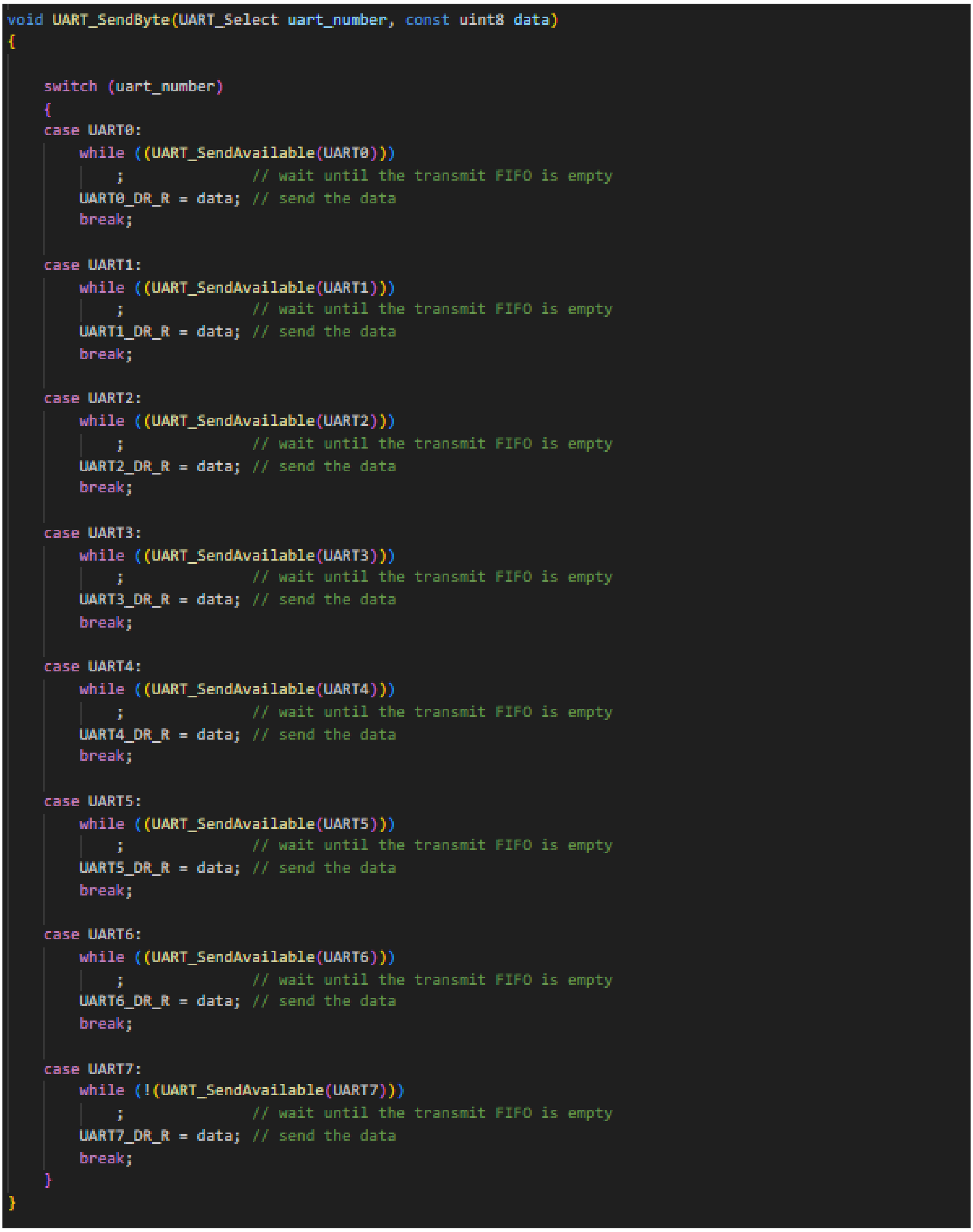
- **Description**: Checks if the UART transmit buffer is ready to send data.
- **Code snippet**:



```
uint8 UART_SendAvailable(UART_Select uart_number)
{

    switch (uart_number)
    {
    case UART0:
        return ((UART0_FR_R & UART_FR_TXFF) == (UART_FR_TXFF)) ? 1 : 0; // check if the transmit FIFO is full

    case UART1:
        return ((UART1_FR_R & UART_FR_TXFF) == (UART_FR_TXFF)) ? 1 : 0; // check if the transmit FIFO is full

    case UART2:
        return ((UART2_FR_R & UART_FR_TXFF) == (UART_FR_TXFF)) ? 1 : 0; // check if the transmit FIFO is full

    case UART3:
        return ((UART3_FR_R & UART_FR_TXFF) == (UART_FR_TXFF)) ? 1 : 0; // check if the transmit FIFO is full

    case UART4:
        return ((UART4_FR_R & UART_FR_TXFF) == (UART_FR_TXFF)) ? 1 : 0; // check if the transmit FIFO is full

    case UART5:
        return ((UART5_FR_R & UART_FR_TXFF) == (UART_FR_TXFF)) ? 1 : 0; // check if the transmit FIFO is full

    case UART6:
        return ((UART7_FR_R & UART_FR_TXFF) == (UART_FR_TXFF)) ? 1 : 0; // check if the transmit FIFO is full

    case UART7:
        return ((UART7_FR_R & UART_FR_TXFF) == (UART_FR_TXFF)) ? 1 : 0; // check if the transmit FIFO is full
    }
}
```

UART_SendByte ():

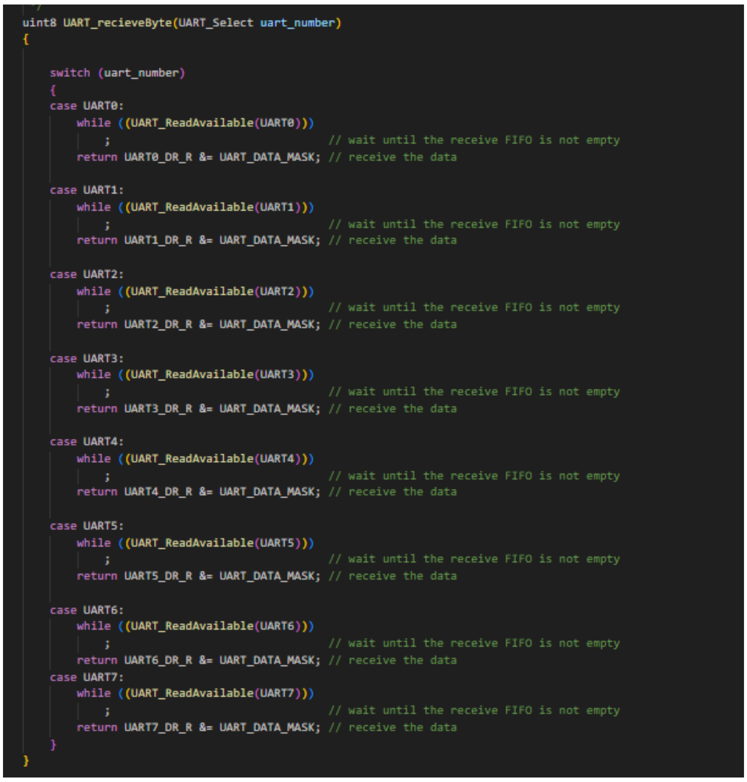
- **Description**: Sends a single byte over UART.
- **Code snippet**:



```
void UART_SendByte(UART_Select uart_number, const uint8 data)
{

    switch (uart_number)
    {
    case UART0:
        while ((UART_SendAvailable(UART0)))
            ;               // wait until the transmit FIFO is empty
        UART0_DR_R = data; // send the data
        break;

    case UART1:
        while ((UART_SendAvailable(UART1)))
            ;               // wait until the transmit FIFO is empty
        UART1_DR_R = data; // send the data
        break;

    case UART2:
        while ((UART_SendAvailable(UART2)))
            ;               // wait until the transmit FIFO is empty
        UART2_DR_R = data; // send the data
        break;

    case UART3:
        while ((UART_SendAvailable(UART3)))
            ;               // wait until the transmit FIFO is empty
        UART3_DR_R = data; // send the data
        break;

    case UART4:
        while ((UART_SendAvailable(UART4)))
            ;               // wait until the transmit FIFO is empty
        UART4_DR_R = data; // send the data
        break;

    case UART5:
        while ((UART_SendAvailable(UART5)))
            ;               // wait until the transmit FIFO is empty
        UART5_DR_R = data; // send the data
        break;

    case UART6:
        while ((UART_SendAvailable(UART6)))
            ;               // wait until the transmit FIFO is empty
        UART6_DR_R = data; // send the data
        break;

    case UART7:
        while (!(UART_SendAvailable(UART7)))
            ;               // wait until the transmit FIFO is empty
        UART7_DR_R = data; // send the data
        break;
    }
}
```

UART_recieveByte ()

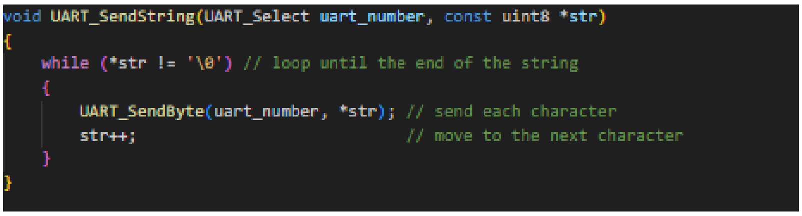
- **Description:** Receives a single byte from UART.
- **Code snippet**:



```c
uint8 UART_recieveByte(UART_Select uart_number)
{

    switch (uart_number)
    {
    case UART0:
        while ((UART_ReadAvailable(UART0)))
            ;                                 // wait until the receive FIFO is not empty
        return UART0_DR_R &= UART_DATA_MASK; // receive the data

    case UART1:
        while ((UART_ReadAvailable(UART1)))
            ;                                 // wait until the receive FIFO is not empty
        return UART1_DR_R &= UART_DATA_MASK; // receive the data

    case UART2:
        while ((UART_ReadAvailable(UART2)))
            ;                                 // wait until the receive FIFO is not empty
        return UART2_DR_R &= UART_DATA_MASK; // receive the data

    case UART3:
        while ((UART_ReadAvailable(UART3)))
            ;                                 // wait until the receive FIFO is not empty
        return UART3_DR_R &= UART_DATA_MASK; // receive the data

    case UART4:
        while ((UART_ReadAvailable(UART4)))
            ;                                 // wait until the receive FIFO is not empty
        return UART4_DR_R &= UART_DATA_MASK; // receive the data

    case UART5:
        while ((UART_ReadAvailable(UART5)))
            ;                                 // wait until the receive FIFO is not empty
        return UART5_DR_R &= UART_DATA_MASK; // receive the data

    case UART6:
        while ((UART_ReadAvailable(UART6)))
            ;                                 // wait until the receive FIFO is not empty
        return UART6_DR_R &= UART_DATA_MASK; // receive the data
    case UART7:
        while ((UART_ReadAvailable(UART7)))
            ;                                 // wait until the receive FIFO is not empty
        return UART7_DR_R &= UART_DATA_MASK; // receive the data
    }
}
```

UART_SendString ()

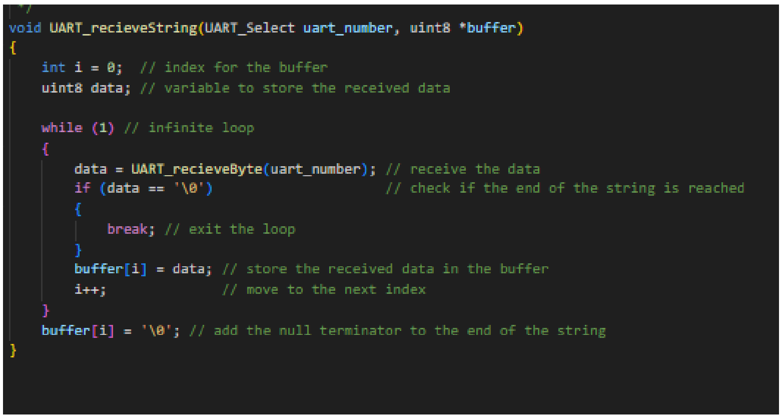
- **Description:** Sends a null-terminated string over UART.
- **Code snippet**:



```c
void UART_SendString(UART_Select uart_number, const uint8 *str)
{
    while (*str != '\0') // loop until the end of the string
    {
        UART_SendByte(uart_number, *str); // send each character
        str++;                            // move to the next character
    }
}
```

UART_recieveString ()

- **Description: Receives a string until a newline (\n) or buffer limit.**
- **Code snippet:**



```c
void UART_recieveString(UART_Select uart_number, uint8 *buffer)
{
    int i = 0;  // index for the buffer
    uint8 data; // variable to store the received data

    while (1) // infinite loop
    {
        data = UART_recieveByte(uart_number); // receive the data
        if (data == '\0')                     // check if the end of the string is reached
        {
            break; // exit the loop
        }
        buffer[i] = data; // store the received data in the buffer
        i++;              // move to the next index
    }
    buffer[i] = '\0'; // add the null terminator to the end of the string
}
```

SysTick Timer Driver Documentation

1. Overview

The SysTick Timer Driver provides precise timing functionality for the Tiva C TM4C123GH6PM microcontroller using the ARM Cortex-M core's built-in 24-bit system timer.

This driver is essential for:

- **Creating accurate delays**
- **Implementing time-based operations**
- **Building real-time scheduling systems**

Key Features

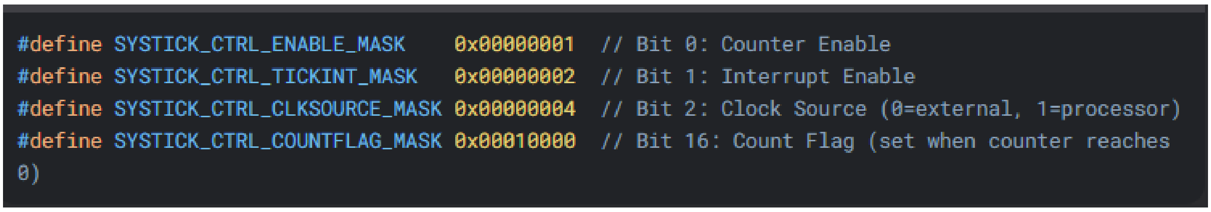
- **24-bit down counter with automatic reload.**
- **Configurable clock source (system clock or divided clock)**
- **Interrupt generation capability**
- **Polling-based delay functions**

Hardware Abstraction

Register	Function
SYSTICK_CTRL	Control and status register
SYSTICK_RELOAD	Reload value register
SYSTICK_CURRENT	Current value register

Register Bit Definitions

Code Snippet:

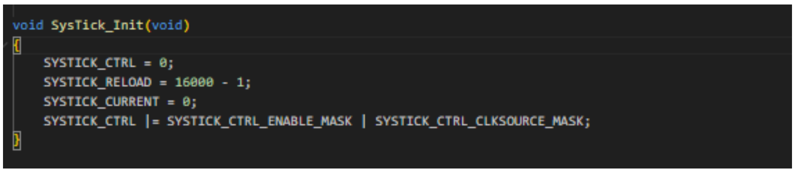


```
#define SYSTICK_CTRL_ENABLE_MASK    0x00000001  // Bit 0: Counter Enable
#define SYSTICK_CTRL_TICKINT_MASK   0x00000002  // Bit 1: Interrupt Enable
#define SYSTICK_CTRL_CLKSOURCE_MASK 0x00000004  // Bit 2: Clock Source (0=external, 1=processor)
#define SYSTICK_CTRL_COUNTFLAG_MASK 0x00010000  // Bit 16: Count Flag (set when counter reaches
0)
```

Functions Documentation

void SysTick_Init(void)

- **Description:**
  Initializes the SysTick timer with default configuration:
  - **Uses processor clock (system clock)**
  - **Disables interrupts**
  - **Clears current value**
- **Code Snippet:**



```
void SysTick_Init(void)
{
    SYSTICK_CTRL = 0;
    SYSTICK_RELOAD = 16000 - 1;
    SYSTICK_CURRENT = 0;
    SYSTICK_CTRL |= SYSTICK_CTRL_ENABLE_MASK | SYSTICK_CTRL_CLKSOURCE_MASK;
}
```

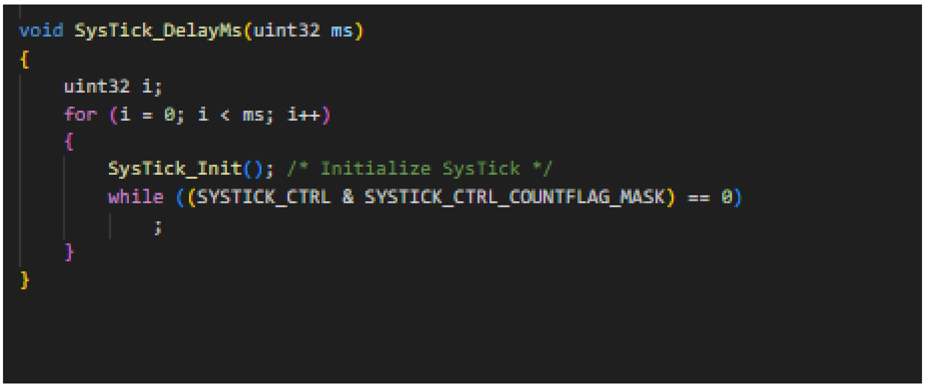
SysTick_DelayMs()

- **Description:**
  Creates a blocking delay for the specified number of milliseconds using polling.

- **Operation:**

Calculates reload value based on system clock (80MHz default)
Configures SysTick to trigger after specified time
Polls count flag to wait for delay completion

- **Code Snippet:**



```
void SysTick_DelayMs(uint32 ms)
{
    uint32 i;
    for (i = 0; i < ms; i++)
    {
        SysTick_Init(); /* Initialize SysTick */
        while ((SYSTICK_CTRL & SYSTICK_CTRL_COUNTFLAG_MASK) == 0)
            ;
    }
}
```

GPS Module Documentation

Overview

This module provides an interface for communicating with a GPS receiver and processing location data. It handles:

- **UART initialization for GPS communication**
- **NMEA sentence parsing (GPRMC format)**
- **Coordinate conversion utilities**
- **Distance calculations between geographic points**

Function Documentation

GPS_UART_Init ()

Description: Initializes UART communication with the GPS module

Typical Configuration:

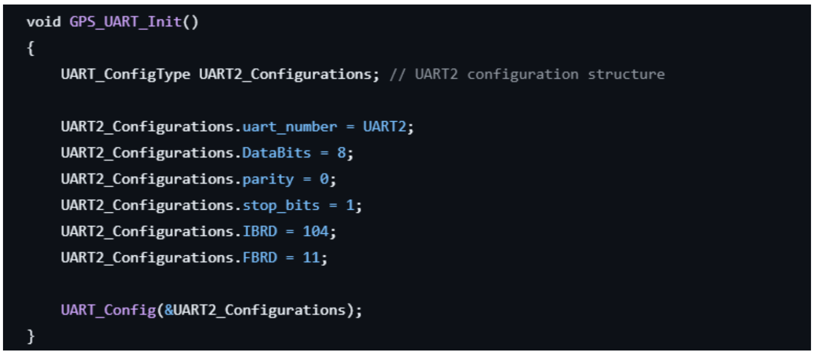
Baud rate: 9600 (standard for NMEA GPS modules)

Data bits: 8

Parity: None

Stop bits: 1

Code Snippet:



```c
void GPS_UART_Init()
{
    UART_ConfigType UART2_Configurations; // UART2 configuration structure

    UART2_Configurations.uart_number = UART2;
    UART2_Configurations.DataBits = 8;
    UART2_Configurations.parity = 0;
    UART2_Configurations.stop_bits = 1;
    UART2_Configurations.IBRD = 104;
    UART2_Configurations.FBRD = 11;

    UART_Config(&UART2_Configurations);
}
```

GetGPRMC ();

Description: Retrieves a GPRMC NMEA sentence from the GPS module

Operation:

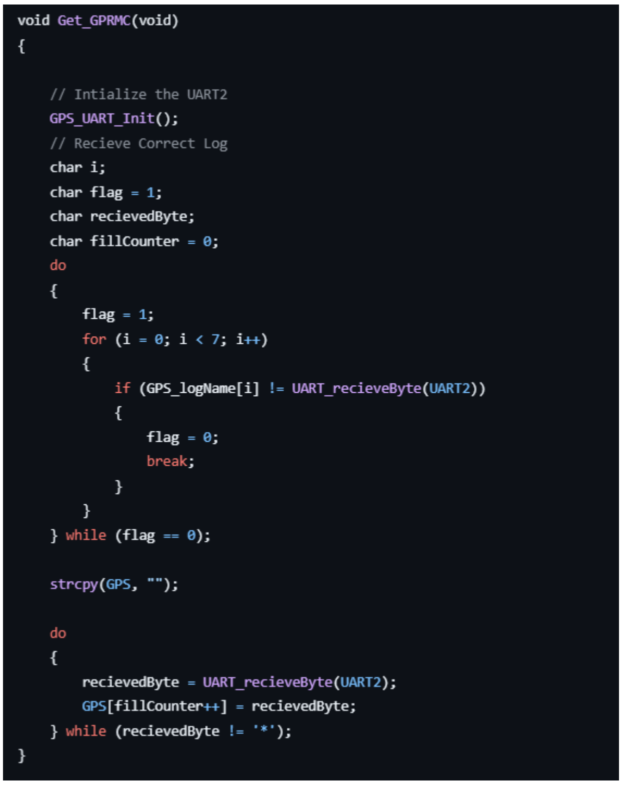
Reads raw data from UART

Validates sentence format ("$GPRMC")

Verifies checksum

Stores valid sentence for parsing

Code Snippet:

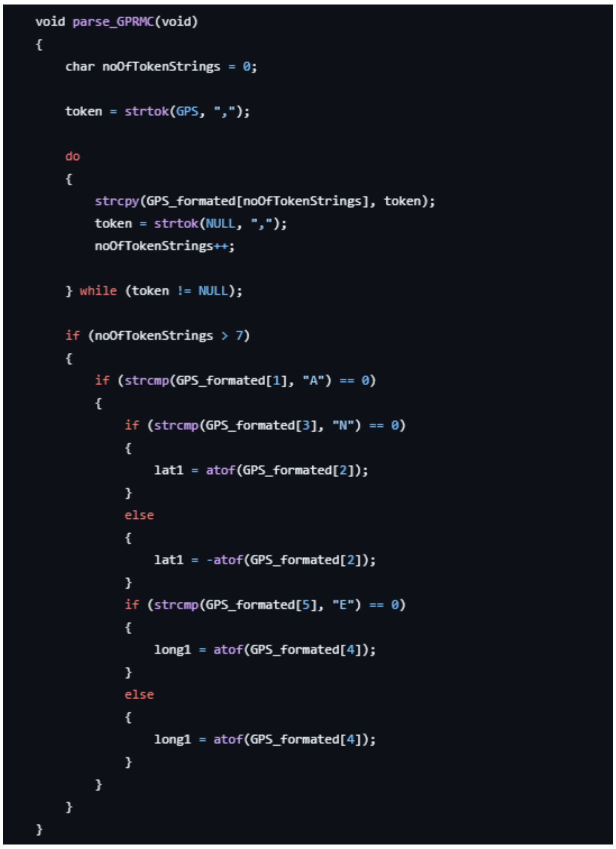


```c
void Get_GPRMC(void)
{

    // Intialize the UART2
    GPS_UART_Init();
    // Recieve Correct Log
    char i;
    char flag = 1;
    char recievedByte;
    char fillCounter = 0;
    do
    {
        flag = 1;
        for (i = 0; i < 7; i++)
        {
            if (GPS_logName[i] != UART_recieveByte(UART2))
            {
                flag = 0;
                break;
            }
        }
    } while (flag == 0);

    strcpy(GPS, "");

    do
    {
        recievedByte = UART_recieveByte(UART2);
        GPS[fillCounter++] = recievedByte;
    } while (recievedByte != '*');
}
```

Parse_GPRMC ();

- **Description**: Parses a GPRMC sentence into usable data
- **Extracted Data**:
  - UTC time
  - Latitude/longitude (converted to decimal degrees)
  - Ground speed (knots)
  - Course over ground (degrees)
  - Date
- **Output**: Updates global variables lat1 and long1

Code Snippet:



```
void parse_GPRMC(void)
{
    char noOfTokenStrings = 0;

    token = strtok(GPS, ",");

    do
    {
        strcpy(GPS_formated[noOfTokenStrings], token);
        token = strtok(NULL, ",");
        noOfTokenStrings++;

    } while (token != NULL);

    if (noOfTokenStrings > 7)
    {
        if (strcmp(GPS_formated[1], "A") == 0)
        {
            if (strcmp(GPS_formated[3], "N") == 0)
            {
                lat1 = atof(GPS_formated[2]);
            }
            else
            {
                lat1 = -atof(GPS_formated[2]);
            }
            if (strcmp(GPS_formated[5], "E") == 0)
            {
                long1 = atof(GPS_formated[4]);
            }
            else
            {
                long1 = atof(GPS_formated[4]);
            }
        }
    }
}
```

ConvertToRad():

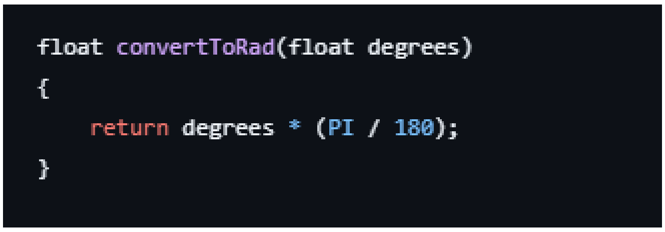
Description: Converts degrees to radians

Parameters:

Angle: Angle in degrees

Returns: Angle in radians

Code Snippet:



```
float convertToRad(float degrees)
{
    return degrees * (PI / 180);
}
```

ConvertToDegree():

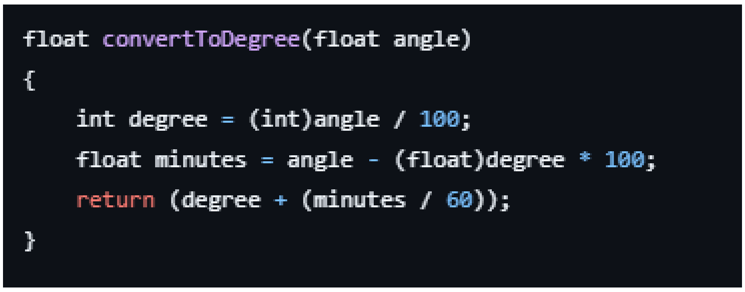
Description: Converts radians to degree

Parameters:

Angle: Angle in radians

Returns: Angle in degrees

Code Snippet:



```
float convertToDegree(float angle)
{
    int degree = (int)angle / 100;
    float minutes = angle - (float)degree * 100;
    return (degree + (minutes / 60));
}
```

Calculate Distance()

Description: Calculates great-circle distance between current position (lat1,long1) and target (lat2, long2)

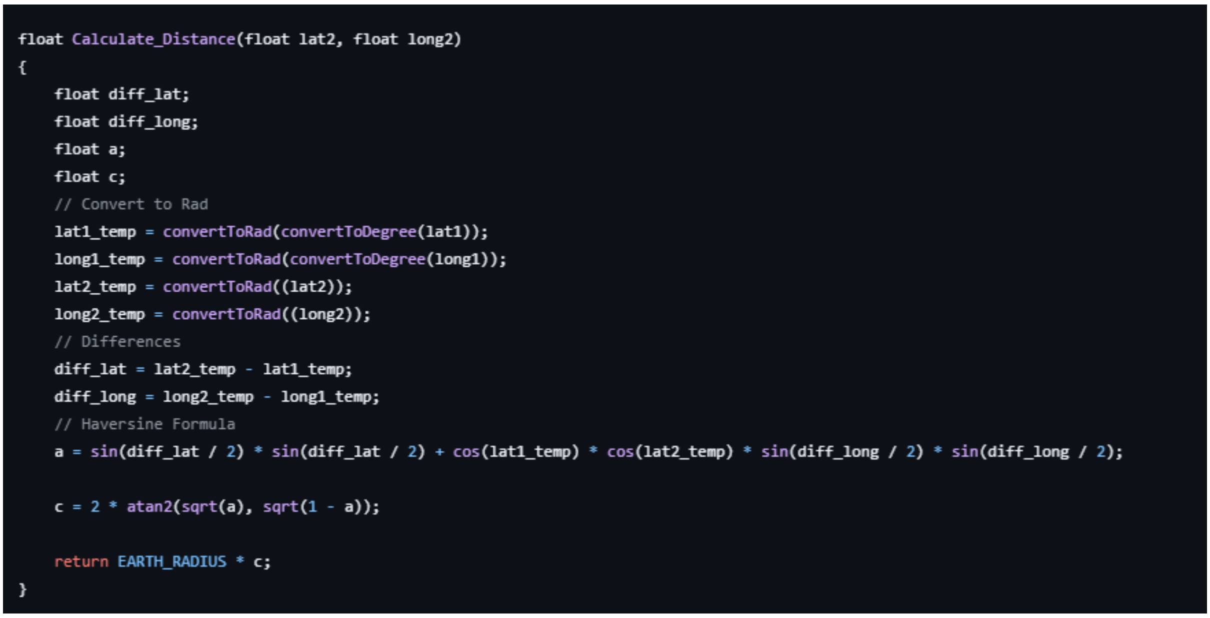
Parameters:

lat2: Target latitude (decimal degrees)

long2: Target longitude (decimal degrees)

Returns: Distance in meters

Algorithm: Haversine formula



```
float Calculate_Distance(float lat2, float long2)
{
    float diff_lat;
    float diff_long;
    float a;
    float c;
    // Convert to Rad
    lat1_temp = convertToRad(convertToDegree(lat1));
    long1_temp = convertToRad(convertToDegree(long1));
    lat2_temp = convertToRad((lat2));
    long2_temp = convertToRad((long2));
    // Differences
    diff_lat = lat2_temp - lat1_temp;
    diff_long = long2_temp - long1_temp;
    // Haversine Formula
    a = sin(diff_lat / 2) * sin(diff_lat / 2) + cos(lat1_temp) * cos(lat2_temp) * sin(diff_long / 2) * sin(diff_long / 2);

    c = 2 * atan2(sqrt(a), sqrt(1 - a));

    return EARTH_RADIUS * c;
}
```

LCD Driver Documentation

Overview

This driver provides an interface for controlling 16x2 character LCD displays in either 4-bit or 8-bit mode. It supports all standard LCD operations including:

- Text display
- Cursor positioning
- Screen clearing
- Special character display

Configuration Options

- **4-bit mode**: Uses 4 data lines (DB4-DB7)
- **8-bit mode**: Uses 8 data lines (DB0-DB7)

Hardware Configuration

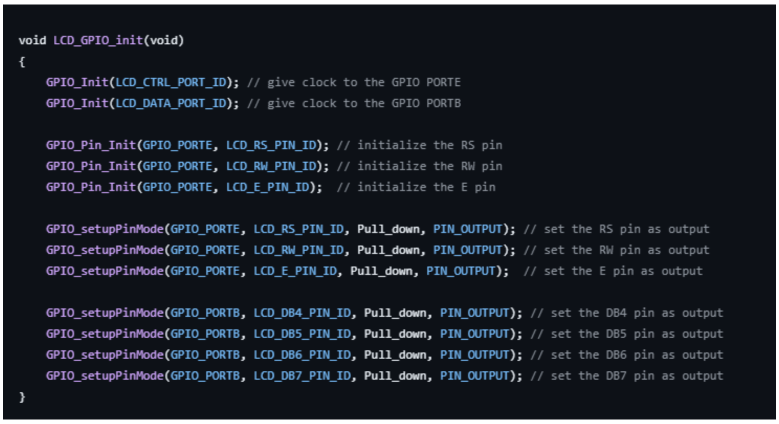
Signal	Default pin	Description
RS	PE1	Register Select
RW	PE2	Read/write
E	PE3	Enable
DB4-DB7	PB4-PB7	Data bus (4-bit mode)

Function Documentation

LCD_GPIO_init()

- Initializes all GPIO pins for LCD control
- Configures pins as digital outputs
- Called automatically by LCD_init()

Code Snippet:



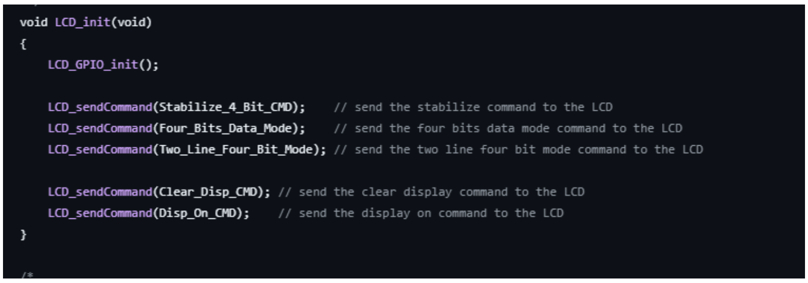
```c
void LCD_GPIO_init(void)
{
    GPIO_Init(LCD_CTRL_PORT_ID); // give clock to the GPIO PORTE
    GPIO_Init(LCD_DATA_PORT_ID); // give clock to the GPIO PORTB

    GPIO_Pin_Init(GPIO_PORTE, LCD_RS_PIN_ID); // initialize the RS pin
    GPIO_Pin_Init(GPIO_PORTE, LCD_RW_PIN_ID); // initialize the RW pin
    GPIO_Pin_Init(GPIO_PORTE, LCD_E_PIN_ID);  // initialize the E pin

    GPIO_setupPinMode(GPIO_PORTE, LCD_RS_PIN_ID, Pull_down, PIN_OUTPUT); // set the RS pin as output
    GPIO_setupPinMode(GPIO_PORTE, LCD_RW_PIN_ID, Pull_down, PIN_OUTPUT); // set the RW pin as output
    GPIO_setupPinMode(GPIO_PORTE, LCD_E_PIN_ID, Pull_down, PIN_OUTPUT);  // set the E pin as output

    GPIO_setupPinMode(GPIO_PORTB, LCD_DB4_PIN_ID, Pull_down, PIN_OUTPUT); // set the DB4 pin as output
    GPIO_setupPinMode(GPIO_PORTB, LCD_DB5_PIN_ID, Pull_down, PIN_OUTPUT); // set the DB5 pin as output
    GPIO_setupPinMode(GPIO_PORTB, LCD_DB6_PIN_ID, Pull_down, PIN_OUTPUT); // set the DB6 pin as output
    GPIO_setupPinMode(GPIO_PORTB, LCD_DB7_PIN_ID, Pull_down, PIN_OUTPUT); // set the DB7 pin as output
}
```

LCD_init()

Performs complete LCD initialization sequence:

- Power-on delay (15ms)
- Function set command
- Display on/off control
- Clear display
- Entry mode set

Code Snippet:

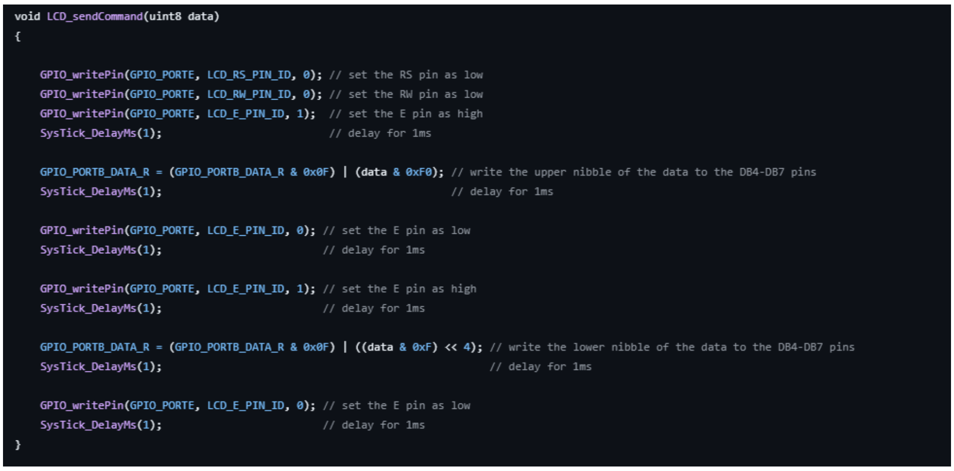


```c
void LCD_init(void)
{
    LCD_GPIO_init();

    LCD_sendCommand(Stabilize_4_Bit_CMD);    // send the stabilize command to the LCD
    LCD_sendCommand(Four_Bits_Data_Mode);    // send the four bits data mode command to the LCD
    LCD_sendCommand(Two_Line_Four_Bit_Mode); // send the two line four bit mode command to the LCD

    LCD_sendCommand(Clear_Disp_CMD); // send the clear display command to the LCD
    LCD_sendCommand(Disp_On_CMD);    // send the display on command to the LCD
}

/*
```

LCD_sendCommand()

- **Sends a command byte to the LCD**
- **Parameters: command**: One of the predefined LCD commands

Code Snippet:

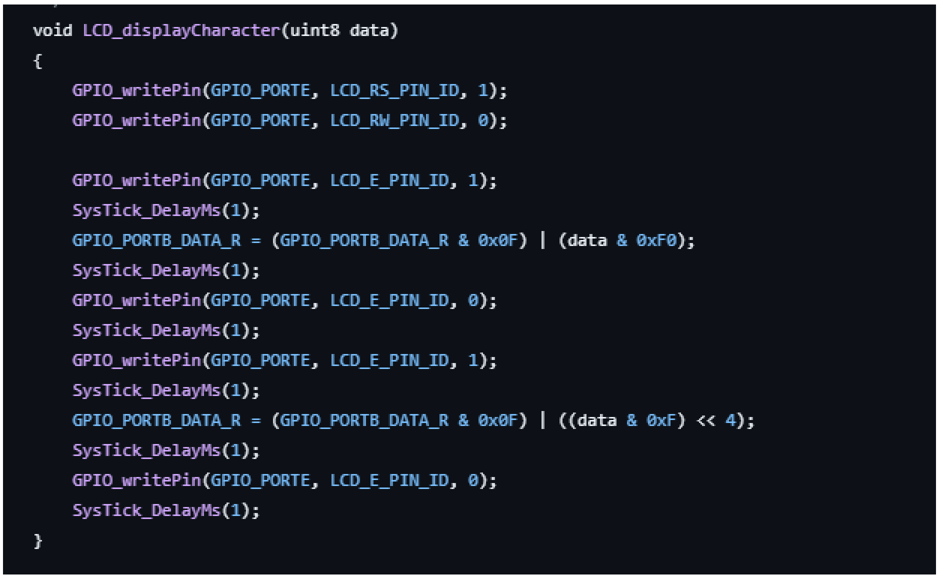


```
void LCD_sendCommand(uint8 data)
{

    GPIO_writePin(GPIO_PORTE, LCD_RS_PIN_ID, 0); // set the RS pin as low
    GPIO_writePin(GPIO_PORTE, LCD_RW_PIN_ID, 0); // set the RW pin as low
    GPIO_writePin(GPIO_PORTE, LCD_E_PIN_ID, 1);  // set the E pin as high
    SysTick_DelayMs(1);                          // delay for 1ms

    GPIO_PORTB_DATA_R = (GPIO_PORTB_DATA_R & 0x0F) | (data & 0xF0); // write the upper nibble of the data to the DB4-DB7 pins
    SysTick_DelayMs(1);                                             // delay for 1ms

    GPIO_writePin(GPIO_PORTE, LCD_E_PIN_ID, 0); // set the E pin as low
    SysTick_DelayMs(1);                         // delay for 1ms

    GPIO_writePin(GPIO_PORTE, LCD_E_PIN_ID, 1); // set the E pin as high
    SysTick_DelayMs(1);                         // delay for 1ms

    GPIO_PORTB_DATA_R = (GPIO_PORTB_DATA_R & 0x0F) | ((data & 0xF) << 4); // write the lower nibble of the data to the DB4-DB7 pins
    SysTick_DelayMs(1);                                                   // delay for 1ms

    GPIO_writePin(GPIO_PORTE, LCD_E_PIN_ID, 0); // set the E pin as low
    SysTick_DelayMs(1);                         // delay for 1ms
}
```

LCD_displayCharacter()

- **Displays a single character at current cursor position**
- **Parameters:**
  data: ASCII character to display

Code Snippet:

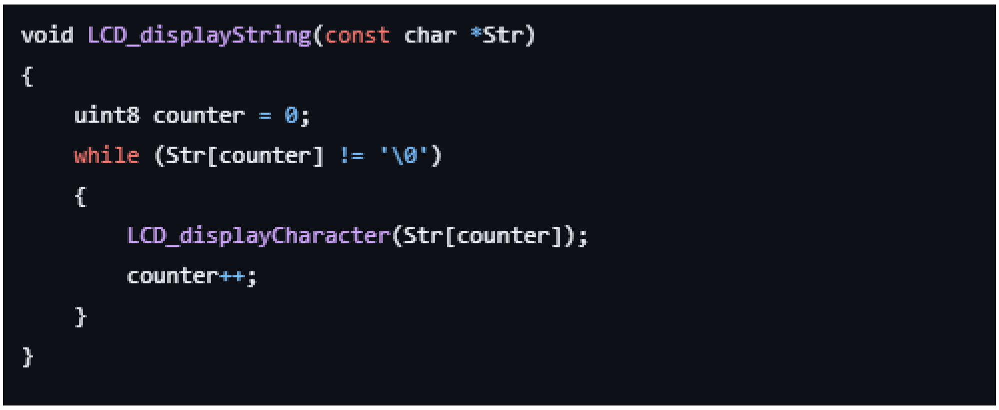


```
void LCD_displayCharacter(uint8 data)
{
    GPIO_writePin(GPIO_PORTE, LCD_RS_PIN_ID, 1);
    GPIO_writePin(GPIO_PORTE, LCD_RW_PIN_ID, 0);

    GPIO_writePin(GPIO_PORTE, LCD_E_PIN_ID, 1);
    SysTick_DelayMs(1);
    GPIO_PORTB_DATA_R = (GPIO_PORTB_DATA_R & 0x0F) | (data & 0xF0);
    SysTick_DelayMs(1);
    GPIO_writePin(GPIO_PORTE, LCD_E_PIN_ID, 0);
    SysTick_DelayMs(1);
    GPIO_writePin(GPIO_PORTE, LCD_E_PIN_ID, 1);
    SysTick_DelayMs(1);
    GPIO_PORTB_DATA_R = (GPIO_PORTB_DATA_R & 0x0F) | ((data & 0xF) << 4);
    SysTick_DelayMs(1);
    GPIO_writePin(GPIO_PORTE, LCD_E_PIN_ID, 0);
    SysTick_DelayMs(1);
}
```

LCD_displayString()

- **Displays a null-terminated string**
- **Parameters:**
    - **Str: Pointer to string buffer**

Code Snippet:

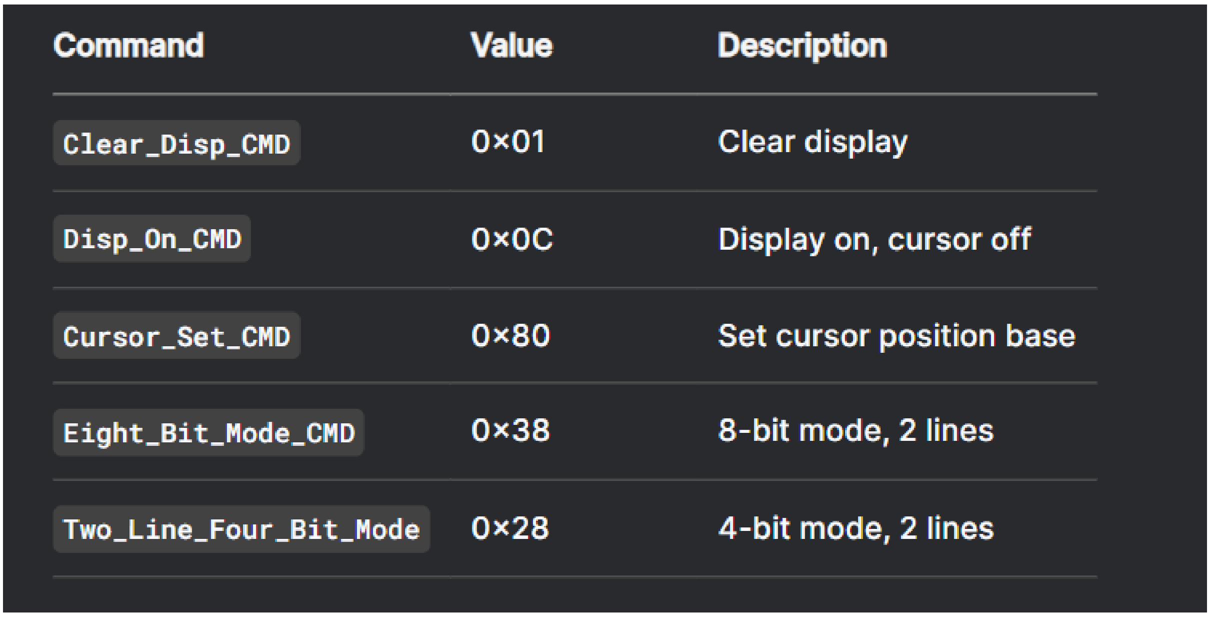


```
void LCD_displayString(const char *Str)
{
    uint8 counter = 0;
    while (Str[counter] != '\0')
    {
        LCD_displayCharacter(Str[counter]);
        counter++;
    }
}
```

Advanced Operations

Function Name	Short Description
LCD_moveCursor()	Positions the cursor at specified location
LCD_displayStringRowColumn()	Combines cursor positioning and string display
LCD_intgerToString()	-Displays numeric values as strings -Supports both integers and floating point
LCD_clearScreen()	Clears entire display and returns cursor to home position

Predefined Commands



Command	Value	Description
`Clear_Disp_CMD`	0×01	Clear display
`Disp_On_CMD`	0×0C	Display on, cursor off
`Cursor_Set_CMD`	0×80	Set cursor position base
`Eight_Bit_Mode_CMD`	0×38	8-bit mode, 2 lines
`Two_Line_Four_Bit_Mode`	0×28	4-bit mode, 2 lines

PortF Driver Documentation

Overview

This driver provides an interface for controlling the Tiva C TM4C123GH6PM microcontroller's Port F, which includes:

- **Three onboard LEDs (Red, Blue, Green)**
- **Two user switches (SW1, SW2)**
- **Implements proper initialization and locking/unlocking procedures**

Hardware Configuration

PIN	FUNCTION	MASK DEFINITION
PF0	SW2	SW_MASK (0X11)
PF1	RED LED	RED_LED (0X02)
PF2	BLUE LED	BLUE_LED (0X04)
PF3	GREEN LED	GREEN_LED (0X08)
PF4	SW1	SW_MASK (0X11)

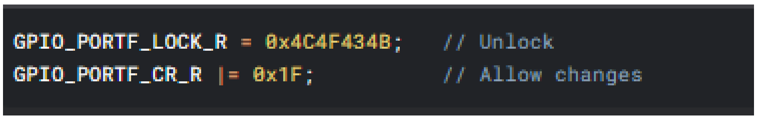
Function Documentation

Function Name	Short Description
PORTF_LEDS_Init()	Initializes PortF LEDs (PF1-PF3
PORTF_SW1_SW2_Init()	Initializes PortF switches (PF0 & PF4)
PORTF_SetLedValue()	Sets specified LED to ON/OFF state
PORTF_led_Toggle()	Toggles specified LED state
PORTF_leds_Off()	Turns off all LEDs
PORTF_GetSwitchValue()	Reads current state of specified switch

Technical Considerations

Port Unlocking

- **Required for PF0 (SW2) due to NMI functionality**
- **Sequence:**



```
GPIO_PORTF_LOCK_R = 0x4C4F434B;   // Unlock
GPIO_PORTF_CR_R |= 0x1F;          // Allow changes
```

Pull-Up Resistors

- **Enabled on switch inputs (PF0 & PF4) for proper button detection**
- **Configured through GPIO_PUR register**

Debouncing

- **Hardware debouncing recommended for switches**
- **Typical RC values: 0.1µF capacitor, 10kΩ resistor**

Buzzer Driver Documentation

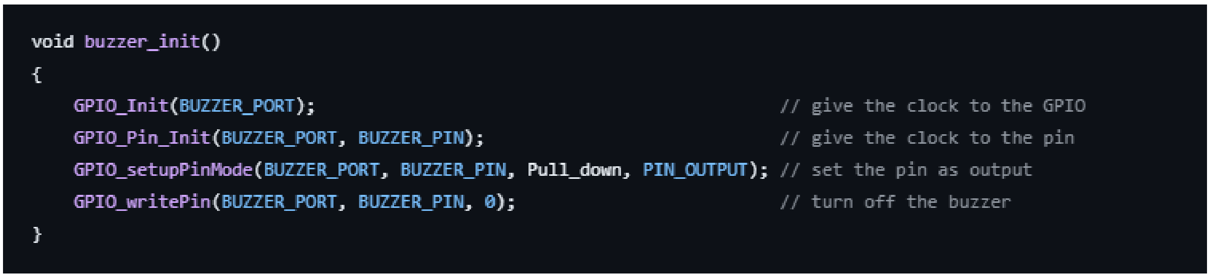
Overview

This driver provides control for a buzzer/piezo element connected to the microcontroller. It implements basic on/off functionality with proper GPIO initialization.

Function Documentation

buzzer_init()

- **Description:** Initializes the buzzer GPIO pin
- **Operations:**
- Enables clock for the buzzer port
- Configures the buzzer pin as digital output
- Initializes buzzer to OFF state



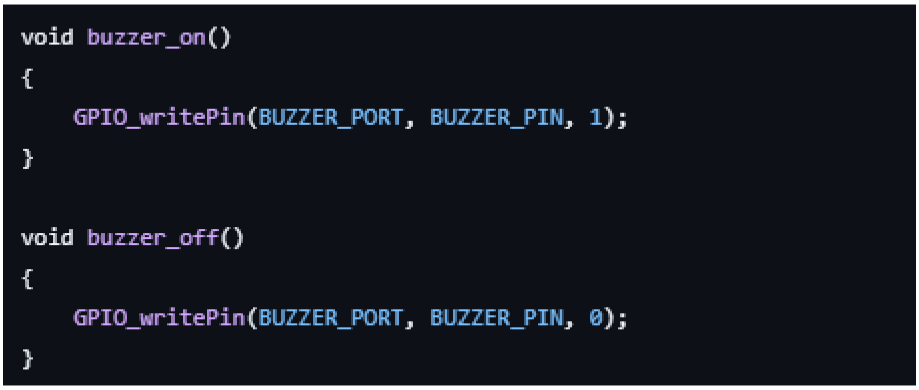
```
void buzzer_init()
{
    GPIO_Init(BUZZER_PORT);                                            // give the clock to the GPIO
    GPIO_Pin_Init(BUZZER_PORT, BUZZER_PIN);                            // give the clock to the pin
    GPIO_setupPinMode(BUZZER_PORT, BUZZER_PIN, Pull_down, PIN_OUTPUT); // set the pin as output
    GPIO_writePin(BUZZER_PORT, BUZZER_PIN, 0);                         // turn off the buzzer
}
```

buzzer_on()

- **Description:** Activates the buzzer
- **Operation:** Sets buzzer pin HIGH
- **Current Draw:** Typically, 20-30mA (check buzzer specs)
- **Note:** For PWM buzzers, this would start the tone

buzzer_off()

- **Description**: Deactivates the buzzer
- **Operation:** Sets buzzer pin LOW
- **Usage:** Should be called when alarm/notification completes



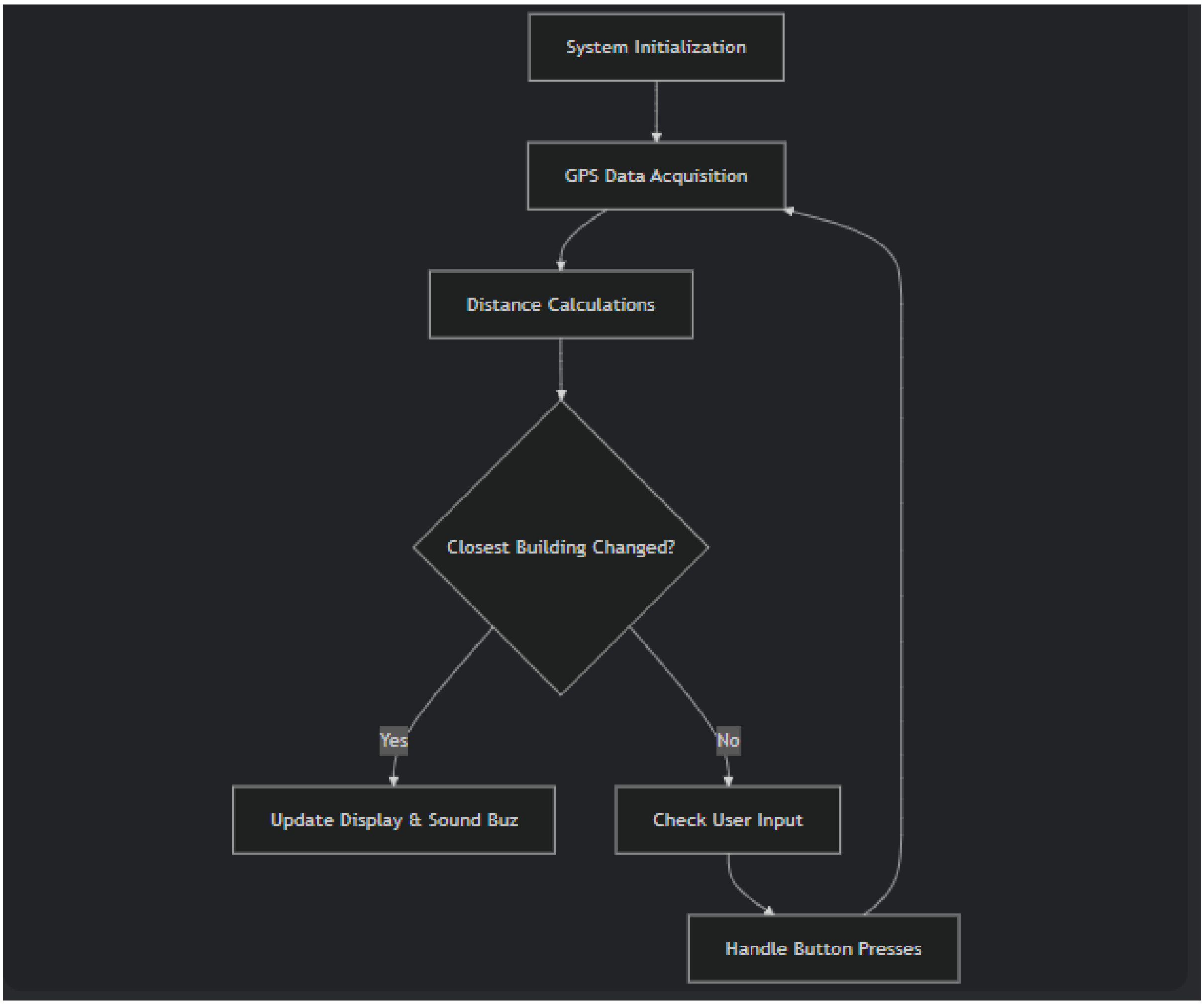
```
void buzzer_on()
{
    GPIO_writePin(BUZZER_PORT, BUZZER_PIN, 1);
}

void buzzer_off()
{
    GPIO_writePin(BUZZER_PORT, BUZZER_PIN, 0);
}
```

GPS Navigation System Application Documentation

Overview

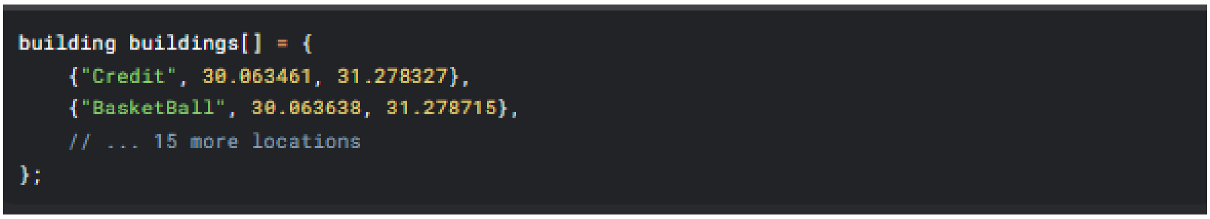
- This application implements a campus navigation system that:
- Tracks user's current position using GPS
- Identifies the closest building from predefined locations
- Calculates total distance travelled
- Provides visual and audible alerts when approaching buildings
- Offers user interaction through buttons and LCD display

Main Workflow



Key Functionality

Building Database



```
building buildings[] = {
    {"Credit", 30.063461, 31.278327},
    {"BasketBall", 30.063638, 31.278715},
    // ... 15 more locations
};
```

- Stores campus buildings with names and coordinates
- Easily expandable by adding new entries

Core Features

Closest Building Detection

- Continuously calculates distance to all buildings
- Updates display when closest building changes
- Triggers buzzer alert on building change

Distance Tracking

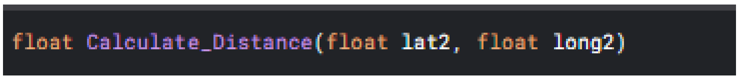
- Accumulates total distance traveled
- Accessible via SW2 button press
- Displays with 2 decimal precision

User Interaction

- **SW1**: Sends current coordinates via UART
- **SW2**: Shows total distance traveled
- **LEDs**: Visual feedback (Red LED on coordinate send)

Critical Blocks

Distance Calculation

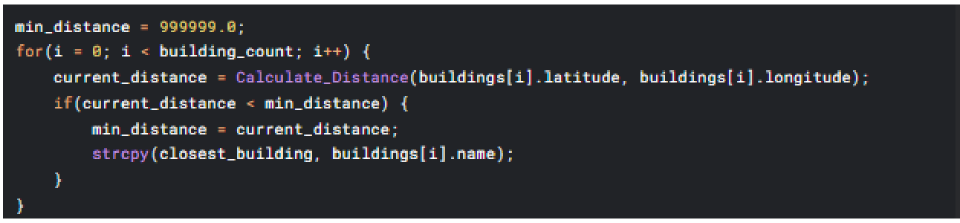


```
float Calculate_Distance(float lat2, float long2)
```

- Uses Haversine formula for great-circle distance
- Returns distance in meters between current position and target
- Earth radius: 6,371,000 meters (WGS-84)

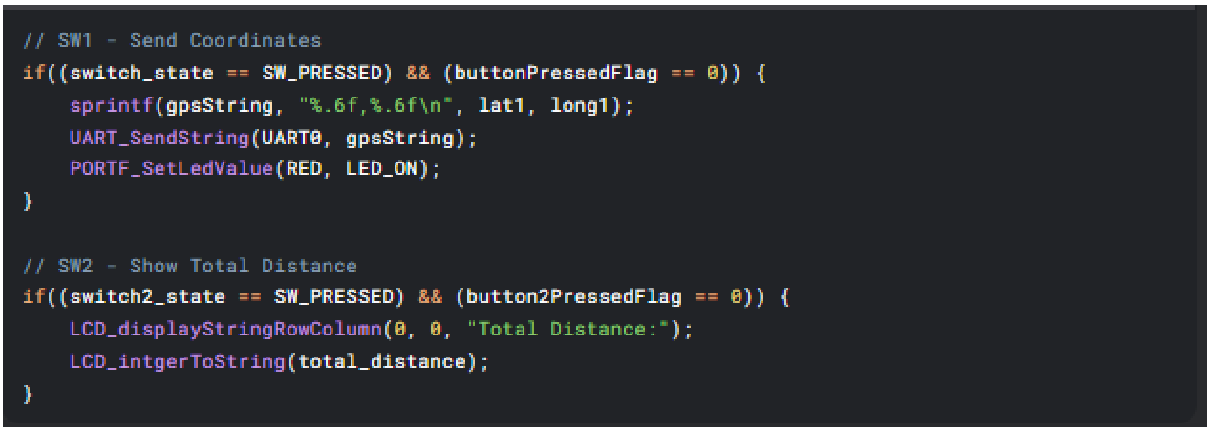
Building Proximity Detection

- Iterates through all buildings
- Tracks minimum distance
- Updates closest building name



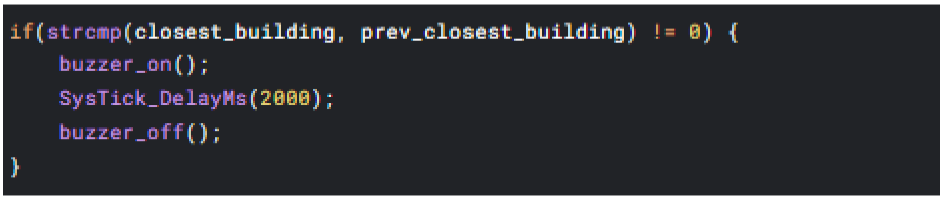
```
min_distance = 999999.0;
for(i = 0; i < building_count; i++) {
    current_distance = Calculate_Distance(buildings[i].latitude, buildings[i].longitude);
    if(current_distance < min_distance) {
        min_distance = current_distance;
        strcpy(closest_building, buildings[i].name);
    }
}
```

User Interface



```
// SW1 - Send Coordinates
if((switch_state == SW_PRESSED) && (buttonPressedFlag == 0)) {
    sprintf(gpsString, "%.6f,%.6f\n", lat1, long1);
    UART_SendString(UART0, gpsString);
    PORTF_SetLedValue(RED, LED_ON);
}

// SW2 - Show Total Distance
if((switch2_state == SW_PRESSED) && (button2PressedFlag == 0)) {
    LCD_displayStringRowColumn(0, 0, "Total Distance:");
    LCD_intgerToString(total_distance);
}
```

Buzzer Feedback



```
if(strcmp(closest_building, prev_closest_building) != 0) {
    buzzer_on();
    SysTick_DelayMs(2000);
    buzzer_off();
}
```

Performance Considerations

Optimizations

Building Database: Static array for fast access

Distance Calculation: Only recalculates when GPS updates

Edge Detection: Button press debouncing

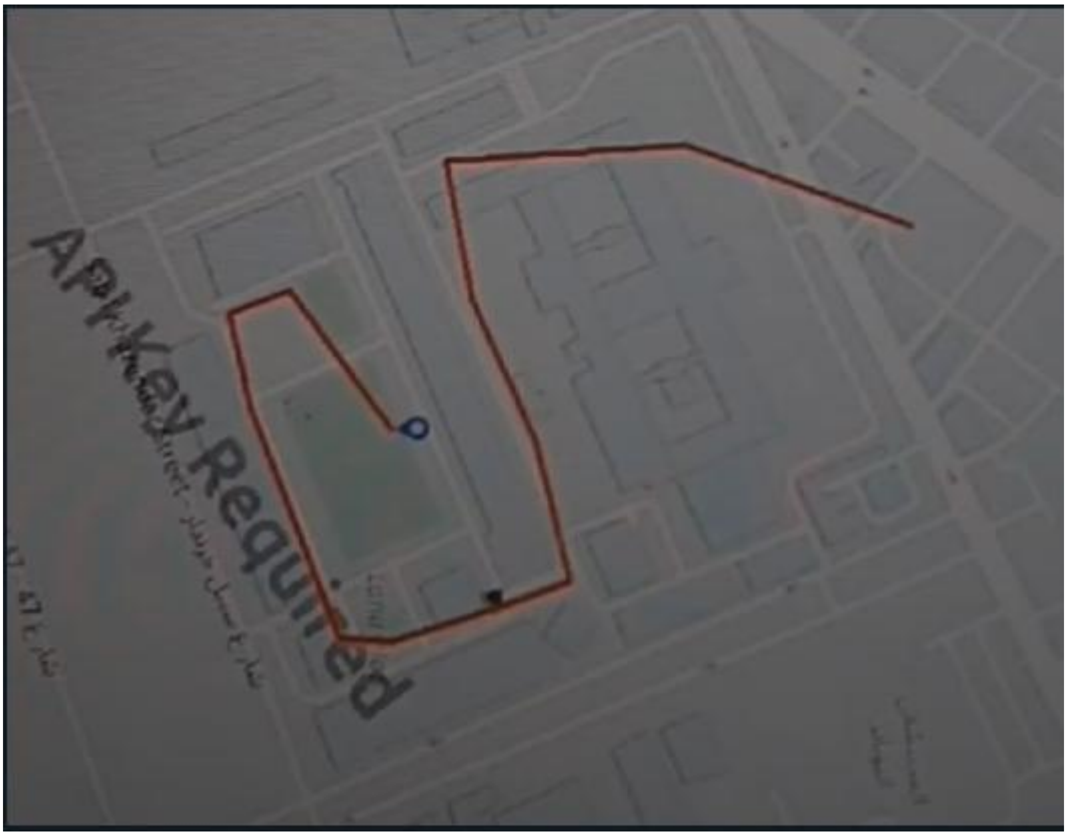
Project video

https://drive.google.com/drive/folders/1DfONqVmcC8QmbkDC5HBqe2poU8R_Lajd

Project Repo

https://github.com/Mohamedkhaled687/GPS_Tracking_System

Total Path snippet:


API Key Required