Table of Contents

Table Of Figures

ECU1- MASTER

Button	
Button_getState	
SW_index	This is of type Dio_ChannelType and represents the index or channel of the button (switch) that the function is checking.
Uint8	BUTTON_PRESSED (1): Indicates that the button is in a pressed state. BUTTON_RELEASED (0): Indicates that the button is in a released state.
Fun. Description	This function is a basic button reading mechanism with debouncing to ensure accurate detection of button states in a microcontroller-based system.
get_input	
NONE	
PRESSED_BTN	BTN1: Indicates that Button 1 is pressed. BTN2: Indicates that Button 2 is pressed. BOTH_BTNS: Indicates that both Button 1 and Button 2 are pressed simultaneously. NONE: Indicates that neither button is pressed.
Fun. Description	This function is a simple input detection mechanism designed for handling button presses and releases.

Systic	
Systick_INIT	
*config	Pointer to Systick_config structure, contains configuration parameters for initializing the SysTick timer. The structure likely includes fields such as reload_value and mode.
NONE	
Fun. Description	This function is responsible for initializing the SysTick timer by configuring the SysTick timer's control, reload, and current value registers based on the specified mode and reload value in the provided configuration.
Systick_delay	
uint16 delay_ms	An unsigned 16-bit integer representing the delay duration in milliseconds.
NONE	
Fun. Description	This function is a delay function that utilizes the SysTick timer to introduce a specified delay in milliseconds.
Systick_Stop	
NONE	
NONE	
Fun. Description	Disable the SysTick Timer by Clear the ENABLE Bit
Systick_SetCallBack	
void (*Ptr2Func)(void)	A function pointer to the user-defined callback function. The callback function should take no arguments (void) and return nothing (void).
NONE	
Fun. Description	This function is used to set a callback function that will be executed when a SysTick interrupt occurs.
Systick_Handler	
NONE	
NONE	
Fun. Description	This function is an interrupt service routine (ISR) for the SysTick timer interrupt. This ISR is automatically invoked when the SysTick timer reaches zero and triggers an interrupt.

Scheduler	
task_init	
1.void (*P)(void) 2.int period	1. A function pointer representing the task function. The function being pointed to takes no arguments (void) and returns void. 2. An integer specifying the period or execution interval of the task.
NONE	
Fun. Description	The function initializes a task using the provided function pointer (p) and period (period). It assigns the function pointer and period to a structure (t_identifier) that likely holds information about the task. The task information is stored in an array (array) at the current index (task_i), and task_i is incremented for the next task.
OS_start	
NONE	
NONE	
Fun. Description	The function runs in an infinite loop, periodically checking the systick counter and executing tasks if it's time to do so. It uses a flag (systick_flag) to ensure that each task is executed only once per systick timer period. Tasks are executed based on their specified periods, and the systick counter is incremented after executing a task.
Os_NewTimerTick	
NONE	
NONE	
Fun. Description	The function is typically called in response to a timer tick or interrupt. It increments the systick counter to keep track of time. If the counter exceeds the specified overflow value (OS_overflow), it is reset to 0 to prevent overflow. It resets the systick_flag to 0, indicating that a new timer tick has occurred.

Mcu	
Mcu_Init	
NONE	
NONE	
Fun. Description	This function initializes and configures various peripherals and system components based on the provided configurations.
BusFault_Init	
NONE	
NONE	
Fun. Description	The function sets the priority level for the Bus Fault exception using the NVIC_SYS_PRI1_R register and it enables the Bus Fault Exception in the NVIC by setting the appropriate bit (BUS_FAULT_ENABLE_BIT_POS) in the NVIC_SYS_HND_CTRL_R register.
BusFault_Handler	
NONE	
NONE	
Fun. Description	The purpose of an exception handler like Bus_Fault_Handler is to handle a Bus Fault.

Port	
Port_Init	
Port_ConfigType *ConfigPtr	A pointer to an array of Port_ConfigType structures, each containing configuration information for a specific GPIO pin.
NONE	
Fun. Description	This function is part of a port initialization module and is responsible for configuring the GPIO pins based on the provided configuration. The actual configuration details (pin direction, initial value, etc.) are specified in the Port_ConfigType structures passed as input.
Port_RefreshPortDirection	
NONE	
NONE	
Fun. Description	The specific bit manipulation operations (SET_BIT, CLEAR_BIT) are assumed to be defined elsewhere in the code. The Port_Configuration array is assumed to be a global array containing the configuration details for each configured pin. The GPIO_LOCK_KEY and other register offsets are assumed to be defined elsewhere in the code.
Port_SetPinMode	
1.Port_PinType Pin 2.Port_PinModeType Mode	1. The logical pin identifier (e.g., PA0, PB3, etc.). 2. The desired mode for the specified pin (e.g., DIO, ADC, ANALOG_COMPARTOR, CAN, I2C, PWM, QEI, SSI, UART).
NONE	
Fun. Description	The code includes optional error checking based on the PORT_DEV_ERROR_DETECT macro. The specific bit manipulation operations (SET_BIT, CLEAR_BIT) are assumed to be defined elsewhere in the code. The Port_Configuration array is assumed to be a global array containing the configuration details for each configured pin. The function supports different modes, and each mode may have specific configurations for the PMCx bits in the PORT_CTL_REG_OFFSET.

Port_SetPinDirection	
1.Port_PinType Pin 2.Port_PinDirectionType Direction	1. The logical pin identifier (e.g., PA0, PB3, etc.). 2. The desired direction for the specified pin (PORT_PIN_OUT for output, PORT_PIN_IN for input).
NONE	
Fun. Description	The code includes optional error checking based on the PORT_DEV_ERROR_DETECT macro. The specific bit manipulation operations (SET_BIT, CLEAR_BIT) are assumed to be defined elsewhere in the code. The Port_Configuration array is assumed to be a global array containing the configuration details for each configured pin. The function supports setting the direction of pins as either input or output based on the specified Direction.
Port_SetPortClock	
Port_ConfigType *ConfigPtr	A pointer to a configuration structure (Port_ConfigType) that contains information about the GPIO port, such as the port number.
NONE	
Fun. Description	This function assumes that the SYSCTL_RCGCGPIO_R and SYSCTL_PRGPIO_R registers are defined and accessible in the code. The specific bit manipulation operations (|=, &, <<) are assumed to be defined elsewhere in the code. The function relies on the wait loop to ensure that the clock for the specified GPIO port is stable before proceeding. The duration of the wait may depend on the microcontroller's clock configuration.
Port_ModeErrorCheck	
Port_ConfigType *ConfigPtr	A pointer to a configuration structure (Port_ConfigType) that contains information about the port configuration, including the mode, direction, analog mode, port number, and pin number.
Boolean TRUE	if configuration errors are detected based on the specified mode; FALSE otherwise.
Fun. Description	The function is incomplete and contains logical errors. Corrections are required to accurately validate the specified configurations for each mode. The conditions inside each case need to be carefully reviewed and corrected to ensure proper error checking. The function currently returns FALSE at the end, but this may need to be adjusted based on the corrected error-checking logic.

can	
CAN_Init	
1.ConfigPtr 2.array_size	1. This is a pointer to an array of structures of type CAN_Struct_Config. Each element in the array contains configuration parameters for a specific CAN interface. 2. This is an integer representing the size of the array pointed to by ConfigPtr. It indicates the number of CAN interfaces to be initialized.
NONE	
Fun. Description	1.Clock Setup: Configures the clock for the specified CAN interface (ConfigPtr[i].canID). 2.Control Register Setup: Configures the control register (CTL_R) for the specified CAN interface. It sets and clears specific bits based on the CAN interface identifier (ConfigPtr[i].canID). 3.Mode Setup: Configures the mode of the CAN interface (e.g., Normal, Loopback, etc.) based on the configuration provided. 4.Interrupt Setup: Configures interrupts for the CAN interface if the interrupt mode is not set to ALL_INTERRUPT_DISABLED. 5.Bitrate Setup: Configures the bitrate for the CAN interface based on the provided BitRateConfig. Control Register Clearing: Clears specific bits in the control register to finalize the configuration.
CAN_Clock_setup	
CAN_ID	This is an enum representing the identifier of the CAN interface for which the clock setup is to be performed. The enumeration CAN_NUM likely includes values like CAN0 and CAN1.
NONE	
Fun. Description	1.Switch Statement: The function uses a switch statement based on the value of CAN_ID to determine which CAN interface to configure. 2.Clock Enable: It sets the corresponding bit in the SYSCTL_RCGCCAN_R register to enable the clock for the selected CAN interface. This is typically used to power up the CAN module. 3.Delay Loop: After enabling the clock, there is a delay loop that waits until the Peripheral Ready (PRCAN) bit for the specified CAN interface is set in the SYSCTL_PRCAN_R register. This delay is introduced to allow time for the clock to stabilize before proceeding with further operations related to the CAN module.

CAN_Mode_setup	
1.CAN_NUM CAN_ID 2.CAN_MODE mode	1. Represents the identifier of the CAN interface for which the mode setup is to be performed. It's an enumeration or some other type that likely includes values like CAN0 and CAN1. 2. Represents the desired operating mode for the CAN interface. It's an enumeration or some other type that likely includes values like NORMAL, SILENT, LOOPBACK, and LOOPBACK_SILENT.
NONE	
Fun. Description	In summary, the CAN_Mode_setup function configures the operating mode of a specified CAN interface based on the provided inputs.
CAN_Interrupt_setup	
1.CAN_NUM CAN_ID 2.intr_mode	1. Represents the identifier of the CAN interface for which the mode setup is to be performed. It's an enumeration or some other type that likely includes values like CAN0 and CAN1. 2. Represents the desired interrupt configuration for the CAN interface. It's an enumeration or some other type that likely includes values like ALL_INTERRUPT_ENABLED, ERROR_INTERRUPT_ENABLED, and STATUS_INTERRUPT_ENABLED.
NONE	
Fun. Description	the CAN_Interrupt_setup function configures the interrupt settings for a specified CAN interface based on the provided inputs.
CAN_Bitrate_setup	
1.CAN_ID 2.config	1. Represents the identifier of the CAN interface for which the mode setup is to be performed. It's an enumeration or some other type that likely includes values like CAN0 and CAN1. 2. A pointer to a structure of type BitRate_Struct_Config. This structure presumably contains the configuration parameters for the CAN bitrate setup, including fields like BRP, SJW, TSEG1, and TSEG2.
NONE	
Fun. Description	The CAN_Bitrate_setup function is responsible for configuring the bitrate settings for a specified CAN interface based on the provided inputs.

CAN_SetObject_transmit	
1.message_index 2.can_index	1. Represents the index of the message object for which the CAN settings are being configured. This index likely corresponds to an entry in an array or a specific message configuration. 2. Represents the index of the CAN configuration to be used. It's assumed that there is an array or some other data structure (can_config) that holds configurations for multiple CAN interfaces.
NONE	
Fun. Description	The CAN_SetObject_transmit function is designed to configure the settings for a specific message object on a specified CAN interface in preparation for a transmission.
CAN_Transmit_Object	
1.can_index 2.message_index 3.data_ptr	1. Represents the index of the CAN configuration to be used. It's assumed that there is an array or some other data structure (can_config) that holds configurations for multiple CAN interfaces. 2. Represents the index of the message object for which the CAN transmission is being initiated. This index likely corresponds to an entry in an array or a specific message configuration. 3. A pointer to an array of data bytes that are to be transmitted in the CAN message. The size of the array is not explicitly mentioned in the provided code, but it is assumed to contain at least 8 bytes of data.
NONE	
Fun. Description	The CAN_Transmit_Object function is designed to initiate the transmission of a CAN message for a specified message object on a specified CAN interface.
CANRecieveMessageSet	
1.can_index 2.message_index	1.Index specifying the CAN module (CAN0 or CAN1) to configure for message reception. 2. Index specifying the message configuration to be used for reception.
NONE	
Fun. Description	The CANReceiveMessageSet function is responsible for configuring a CAN module (CAN0 or CAN1) to receive messages based on the specified message configuration.

CANMessageget	
1.can_index 2.message_index	1. Index specifying the CAN module (CAN0 or CAN1) from which to retrieve a received message. 2. Index specifying the message configuration for which to retrieve the received data.
Data_Receive	The function returns a pointer to an array (Data_Receive) containing the received data.
Fun. Description	The CANMessageget function is responsible for retrieving the received data from a specified CAN message object.
CAN_get_status	
Uint8 can_index	Represents the index of the CAN configuration to be used. It's assumed that there is an array or some other data structure (can_config) that holds configurations for multiple CAN interfaces.
NONE	
Fun. Description	The CAN_get_status function retrieves and interprets status information for a specified CAN interface (CAN0 or CAN1). It checks various bits in the status register and updates elements in a data structure (can_status_array). The interpreted status information includes the bus status, warning limit, error state, receive status, transmit status, and the type of the last error encountered.
getErrorCounter	
Unit8 can_index	Represents the index of the CAN configuration to be used. It's assumed that there is an array or some other data structure (can_config) that holds configurations for multiple CAN interfaces.
NONE	
Fun. Description	The getErrorCounter function retrieves and updates the transmit and receive error counters for a specified CAN interface (CAN0 or CAN1). It does this by extracting specific bits from the error register (CAN0_ERR_R or CAN1_ERR_R) and storing the results in the can_error_counter data structure.

Dio	
Dio_Init	
*ConfigPtr	A pointer to a configuration structure (Dio_ConfigType) containing the initialization parameters for the DIO module.
NONE	
Fun. Description	This function initializes the Digital I/O (DIO) module based on the provided configuration. The DIO module is responsible for controlling and configuring digital input and output pins.
Dio_WriteChannel	
1.ChannelId 2.Level	1. Identifier for the specific DIO channel to be written. 2. Desired logic level to be written to the channel (STD_HIGH for logic high, STD_LOW for logic low).
NONE	
Fun. Description	This function is responsible for writing a digital signal (logic level) to a specific channel. Ittakes the channel ID and the desired logic level as inputs and writes the corresponding signal to the associated GPIO (General Purpose Input/Output) port.
Dio_ReadChannel	
ChannelId	Identifier for the specific DIO channel to be read.
Dio_LevelType	The logic level currently present on the specified DIO channel (STD_HIGH for logic high, STD_LOW for logic low).
Fun. Description	This function is responsible for reading the logic level from a specific channel and it takes the channel ID as an input and returns the logic level (either STD_HIGH or STD_LOW) currently present on the specified DIO channel.
Dio_GetVersionInfo	
*versioninfo	Pointer to a Std_VersionInfoType structure where the version information will be stored.
NONE	
Fun. Description	This function is responsible for retrieving version information of the DIO module and is typically used for version and compatibility checks during the initialization phase.
Dio_FlipChannel	
1.ChannelId	Identifier of the digital channel to be flipped.
NONE	
Fun. Description	This function is responsible for toggling the output level of a specified channel. It is particularly useful for scenarios where you want to invert the logic level of a digital output channel.

Dio_ReadChannelGroup	
ChannelGroupIdPtr	Pointer to a structure containing information about the channel group, including the port index, bit offset, and mask.
Dio_PortLevelType	The function returns the combined level of the specified channel group.
Fun. Description	This function is responsible for reading the levels of a specified group of digital channels. This function reads a group of channels within a specific port and returns the combined level of those channels.
Dio_WriteChannelGroup	
1.ChannelGroupIdPtr 2.Level	1.Pointer to a structure containing information about the channel group, including the port index, bit offset, and mask. 2. The level to be written to the specified channel group.
NONE	
Fun. Description	This function is responsible for writing the specified levels to a group of digital channels within a specific port. This function allows the simultaneous control of multiple channels within a group.

Gpt	
Delay_timerA	
Int mtime	The desired delay time in arbitrary units. The actual delay time depends on the clock frequency and the configuration of the TimerA.
NONE	
Fun. Description	The actual delay achieved by this function depends on the clock frequency and the value loaded into TIMER0_TAILR_R. The function utilizes busy-waiting, and it may not be precise for very short delays or in the presence of interrupts. Consider using hardware timers with interrupts for more accurate timing.

Adc	
ADC_Init_Start	
ADC_config configPtr	A pointer to an array of ADC_config structures. Each structure holds configuration parameters for a specific ADC module defined as ADC_config structure.
NONE	
Fun. Description	The function uses register pointers (ADC_ACTSS_R_ptr, ADC_EMUX_R_ptr, etc.) to access and configure the ADC registers based on the specified ADC module number and it uses bit manipulation to enable/disable specific bits in the ADC registers for configuration. The interrupt-related configurations are done only if the specified mode is ADC_INTERRUPT.
ADC_Read	
Uint8 adc_index	An index representing the ADC channel or input for which the conversion is to be performed. It is used to identify the specific ADC configuration from the global array ADC_Configuartions.
Uint32 result	A 32-bit unsigned integer representing the digital result of the ADC conversion.
Fun. Description	This function provides a simple interface for initiating and obtaining ADC conversions for a specific ADC channel based on the provided ADC index.
ADC_Read_Volt	
1.Uint8 adc_index 2.Float32 Source	1. An index representing the ADC channel or input for which the conversion is to be performed. 2.A scaling factor or reference voltage used to convert the digital result to a voltage value.
Float32	A 32-bit floating-point variable that holds the digital result of the ADC conversion.
Fun. Description	This function is called to obtain the digital result of the ADC conversion on the specified ADC channel (adc_index). This digital result is then scaled to a voltage value using the provided scaling factor (Source). The scaling factor represents the reference voltage or a known value that allows conversion from digital to analog voltage. The division by 4095 is used to normalize the result to a voltage range between 0 and the provided reference voltage (Source).The final calculated voltage value is returned as the output of the function.

Eeprom	
eeprom_start	
None	
int	Returns an integer value indicating the success or failure of the EEPROM initialization process.
Fun. Description	This function utilizes the delay_n function to introduce delays. The specific implementation of delay_n is not provided in the code snippet. The function uses register writes and reads to interact with the EEPROM module and the system control registers. The return value serves as an indicator of the success or failure of the EEPROM initialization process.
eeprom_error_rcovery	
NONE	
NONE	
Fun. Description	This function seems to be a part of error-handling routines for the EEPROM module, designed to recover from certain error conditions by initiating an error recovery process and waiting for its completion.
eeprom_init	
NONE	
NONE	
Fun. Description	This function acts as a higher-level initialization routine for the EEPROM module, handling both the normal initialization process and potential error recovery. It provides a simple mechanism to ensure that the EEPROM is properly initialized and ready for use in subsequent operations.
eeprom_write	
1.Int data 2.Uint addr 3.Uint b1k	1. The data to be written to the EEPROM. 2. The offset within the specified block where the data should be written. 3. The block number indicating the EEPROM block where the data should be written.
NONE	
Fun. Description	The function directly manipulates the EEPROM registers to set the block, offset, and write data and waits for the completion of the EEPROM operation before returning. The waiting is done by polling the WORKING bit in the EEDONE register. The data type used for writing data to the EEPROM is int.

eeprom_read	
1.Uint8 addr 2.uint8 b1k	1. The offset within the specified block from where data should be read. 2. The block number indicating the EEPROM block from which data should be read.
Int data	Returns an int representing the data read from the specified EEPROM location.
Fun. Description	The function directly manipulates the EEPROM registers (EEPROM_EEBLOCK_R, EEPROM_EEOFFSET_R, EEPROM_EERDWR_R) to set the block, offset, and read data. It waits for the completion of the EEPROM operation before returning. The waiting is done by polling the WORKING bit in the EEDONE register. The data type used for storing the read data is int.

Temp_sens	
temperatureSensoRead	
NONE	
Uint32 average	The average value of the multiple ADC readings taken from the temperature sensor.
Fun. Description	The function takes multiple readings, calculates their average, and returns the result.

Led	
White_led	
NONE	
NONE	
Fun. Description	This function purpose is to turn on white LEDs by setting the appropriate digital output channels associated with the red, blue, and green components of the LEDs.
Blue_led	
NONE	
NONE	
Fun. Description	This function purpose is to turn on blue LEDs by setting the appropriate digital output channel associated with the blue LED.
Red_led	
NONE	
NONE	
Fun. Description	This function purpose is to turn on red LEDs by setting the appropriate digital output channel associated with the red LED.
Green_led	
NONE	
NONE	
Fun. Description	This function purpose is to turn on green LEDs by setting the appropriate digital output channel associated with the green LED.
Clear_leds	
NONE	
NONE	
Fun. Description	This function purpose is to turn off all LEDs by setting the associated digital output channels to a logic level of STD_LOW.

comm_pc	
reverse	
1.Char str[] 2.Int length	1. The character array (string) that needs to be reversed. 2. The length of the character array (str).
NONE	
Fun. Description	The function modifies the input array in place and does not create a new reversed array. The reversal is done by swapping characters symmetrically from both ends towards the center of the array.
intToAsciiAndSendPC	
Uint32 number	The integer to be converted to ASCII and sent to the PC.
NONE	
Fun. Description	The sendPC function is assumed to handle the process of sending the string to a PC, and its implementation is not provided in this code snippet. The size of the buffer array is chosen to accommodate integers with up to 10 digits plus a sign and null terminator. Adjust the size based on the expected maximum length of the integer representation.
sendPC	
Uint8 *pData	A pointer to an array of uint8 that is intended to be sent to the PC.
NONE	
Fun. Description	It takes the input string (pData) and passes it to the UART0_SendString function, presumably for transmission over a communication interface (UART in this case).

UART	
UART_SetClock	
UART_Num	Enumeration representing the UART module number. It is of type UART_ModuleNum.
NONE	
Fun. Description	This function is responsible for configuring the clock for a specified UART module in a microcontroller-based system. It follows a switch-case structure to handle different UART modules (UART0 to UART7).
UART_Init	
*ConfigPtr	Pointer to an array of UART_Config structures. Each structure contains configuration parameters for a specific UART module.
NONE	
Fun. Description	The UART_Init function initializes and configures multiple UART (Universal Asynchronous Receiver/Transmitter) modules based on the provided configuration parameters. It supports flexible setup for different UART modules, allowing customization of clock sources, baud rates, data frame length, and other control parameters.
UART0_SendByte	
Uint8 data	The 8-bit data byte to be transmitted through UART0.
NONE	
Fun. Description	The UART0_SendByte function is responsible for sending a single byte of data through the UART0 (Universal Asynchronous Receiver/Transmitter 0) module. This function utilizes busy-waiting to ensure that the transmit FIFO (First In, First Out) is empty before attempting to send the byte. Once the FIFO is empty, it writes the data byte to the UART0 Data Register (UART0_DR_R), initiating the transmission.
UART0_RecieveByte	
NONE	
Uint8	Returns an 8-bit data byte received from UART0.
Fun. Description	This function receives a single byte of data from UART0 (Universal Asynchronous Receiver/Transmitter 0). This function uses busy-waiting to ensure that the receive FIFO (First In, First Out) is not empty before attempting to read a byte. Once a byte is available in the FIFO, it is read from the UART0 Data Register (UART0_DR_R) and returned to the caller.

UART0_SendString	
Uint8 *pData	A pointer to the null-terminated string (array of characters) to be transmitted.
NONE	
Fun. Description	This function is responsible for transmitting a null-terminated string of characters via UART0 and it iterates through each character in the input string until it encounters the null terminator ('\0'). For each character in the string, it calls the UART0_SendByte function to send that character over UART0.
UART0_RecieveString	
uint8 *pData	A pointer to the character array where the received string will be stored. The array should have sufficient space to store the received characters.
NONE	
Fun. Description	This function is designed to receive a string of characters from UART0 until it encounters a specific termination character, in this case, the '#' symbol. It receives individual bytes using the UART0_ReceiveByte function and populates the provided character array (pData) until the termination character is received. The function then replaces the termination character with the null terminator '\0' to properly terminate the string.
UART0_SendData	
1.Uint8 data 2.Uint32 uSize	1.A pointer to the array containing the data to be transmitted. 2. The number of bytes to be transmitted.
NONE	
Fun. Description	This function is designed to send a specified number of bytes of data through UART0. It takes a pointer to a data array (pData) and the size of data (uSize) as inputs, and it transmits the specified number of bytes through UART0 using the UART0_SendByte function.
UART0_RecieveData	
1.uint8 *pData 2.uint32 uSize	1.A pointer to the array containing the data to be transmitted. 2. The number of bytes to be transmitted.
NONE	
Fun. Description	This function is designed to receive a specified number of bytes of data through. It takes a pointer to a data array (pData) and the size of data (uSize) as inputs, and it receives the specified number of bytes through UART0 using the UART0_ReceiveByte function.

app	
CANInitHandler	
NONE	
NONE	
Fun. Description	This function appears to be an interrupt handler for a Controller Area Network (CAN) module. The purpose of this handler is to manage incoming CAN messages and respond accordingly based on the interrupt status.
App_init	
NONE	
NONE	
Fun. Description	The App_init function configures various settings related to the Controller Area Network (CAN) module, initializes message objects for receiving and transmitting data, sets up pointers for data manipulation, and initializes tasks for multitasking. Additionally, it reads a system state from EEPROM and sets the initial state of the system.
MainTask	
NONE	
NONE	
Fun. Description	This function is a task that monitors the system's main state and conditions, and based on specific criteria, it assesses temperature, voltage, and communication errors. It updates the system state accordingly, transitioning between different error states, such as overheat and system fault.
KeepAlive	
NONE	
NONE	
Fun. Description	This function is responsible for sending a periodic "keep-alive" message over the Controller Area Network (CAN) bus, containing the current system state. It helps to ensure continuous communication and monitor the health of the system. Additionally, the function checks for communication errors and updates the system state accordingly.

ChangeState	
NONE	
NONE	
Fun. Description	This function is responsible for managing state transitions and error recovery. It evaluates specific conditions and updates the system state, clears error counters, and performs EEPROM read/write operations accordingly.
GetButtons	
NONE	
NONE	
Fun. Description	The function is responsible for determining the state of a button (or buttons) based on the current system state. It adapts button handling based on the specific error conditions or states, allowing for different responses to button inputs depending on the system's health.
PCcommunication	
NONE	
NONE	
Fun. Description	This function is responsible for communicating information about the system state to an external device, likely a PC (Personal Computer). It formats and sends messages based on the current state of the embedded system, providing status updates or error notifications.
LEDTASK	
NONE	
NONE	
Fun. Description	This function manages the state of LEDs (Light Emitting Diodes) based on the current state of the embedded system. It switches between different LED patterns and states to visually indicate the system's health or error conditions.

ECU2- SLAVE

app	
CANInitHandler	
NONE	
NONE	
Fun. Description	This function appears to be an interrupt handler for a Controller Area Network (CAN) module in an embedded system, likely written in C. The purpose of this handler is to manage incoming CAN messages and respond accordingly based on the interrupt status.
App_init	
NONE	
NONE	
Fun. Description	The App_init function configures various settings related to the Controller Area Network (CAN) module, initializes message objects for receiving and transmitting data, sets up pointers for data manipulation, and initializes tasks for multitasking. Additionally, it reads a system state from EEPROM and sets the initial state of the system.
voltageRead	
NONE	
Uint32	Returns a uint32 value representing the average voltage calculated from multiple ADC readings.
Fun. Description	The voltageRead function is designed to measure the voltage using an Analog-to-Digital Converter (ADC) and calculate the average of multiple readings to provide a more stable and accurate result. The function takes a specified number of readings from the ADC, calculates their sum, and then computes the average. The final average voltage value is returned by the function.
Test_task_1	
NONE	
NONE	
Fun. Description	This function is a task that performs several operations related to the monitoring and communication aspects of an embedded system. It checks the system state, reads temperature and voltage values, sends them over a Controller Area Network (CAN) bus, monitors communication status, and updates the system state based on communication errors.

GetButtons	
NONE	
NONE	
Fun. Description	The GetButtons function is responsible for determining the state of a button (or buttons) based on the current system state and sending the corresponding button data over the Controller Area Network (CAN) bus. It utilizes a switch-case structure to handle different cases (system states) and takes appropriate actions based on the button input.
LEDTASK	
NONE	
NONE	
Fun. Description	This function is a task responsible for controlling the LEDs (Light Emitting Diodes) based on the current state of the system. It responds to different error conditions and takes appropriate actions to display corresponding LED patterns. Additionally, it manages timers and resets error states under specific conditions.

Architecture layer Hardware

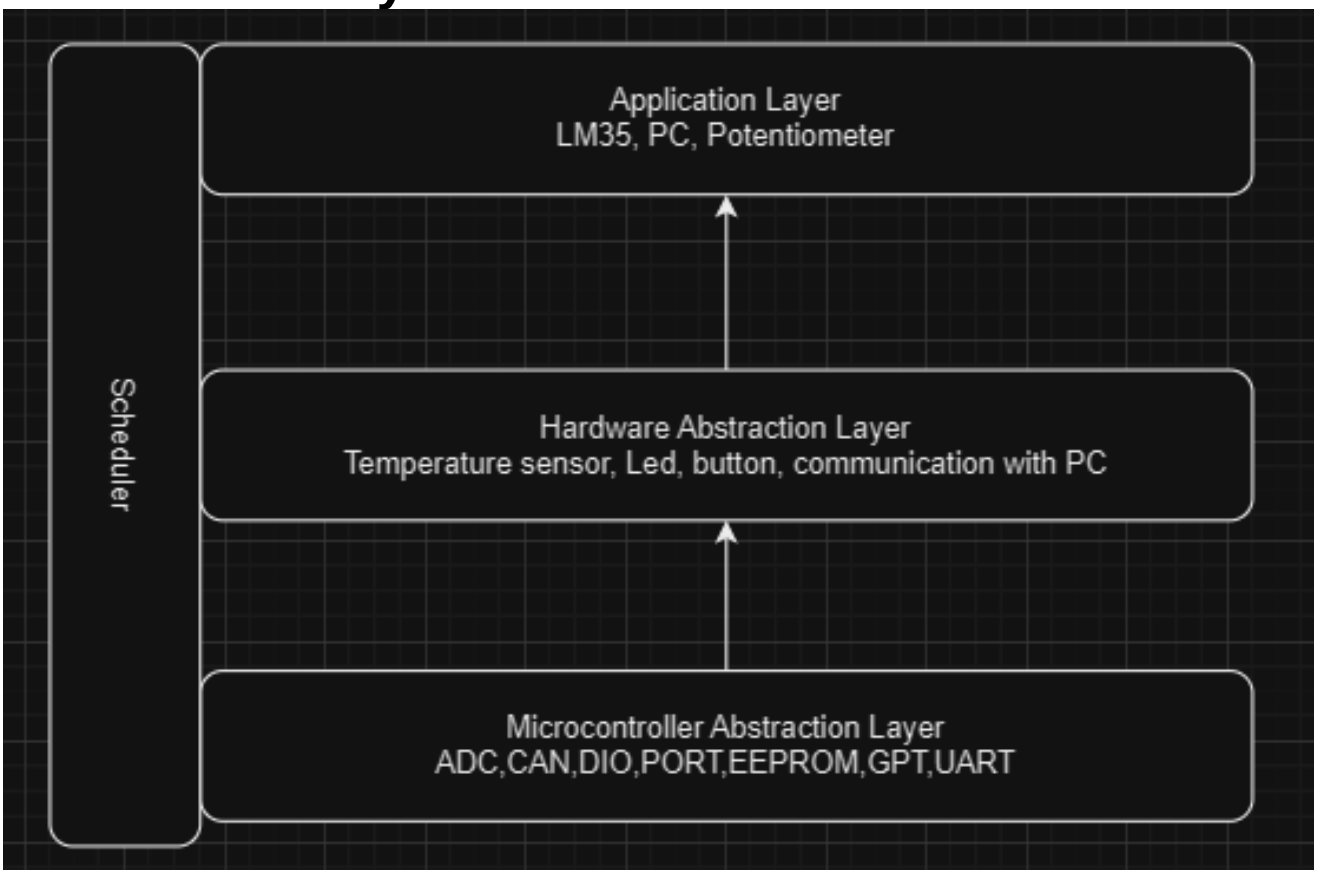

Figure 1: Architecture layer

Flowcharts

Overheated, normal states Flowchart

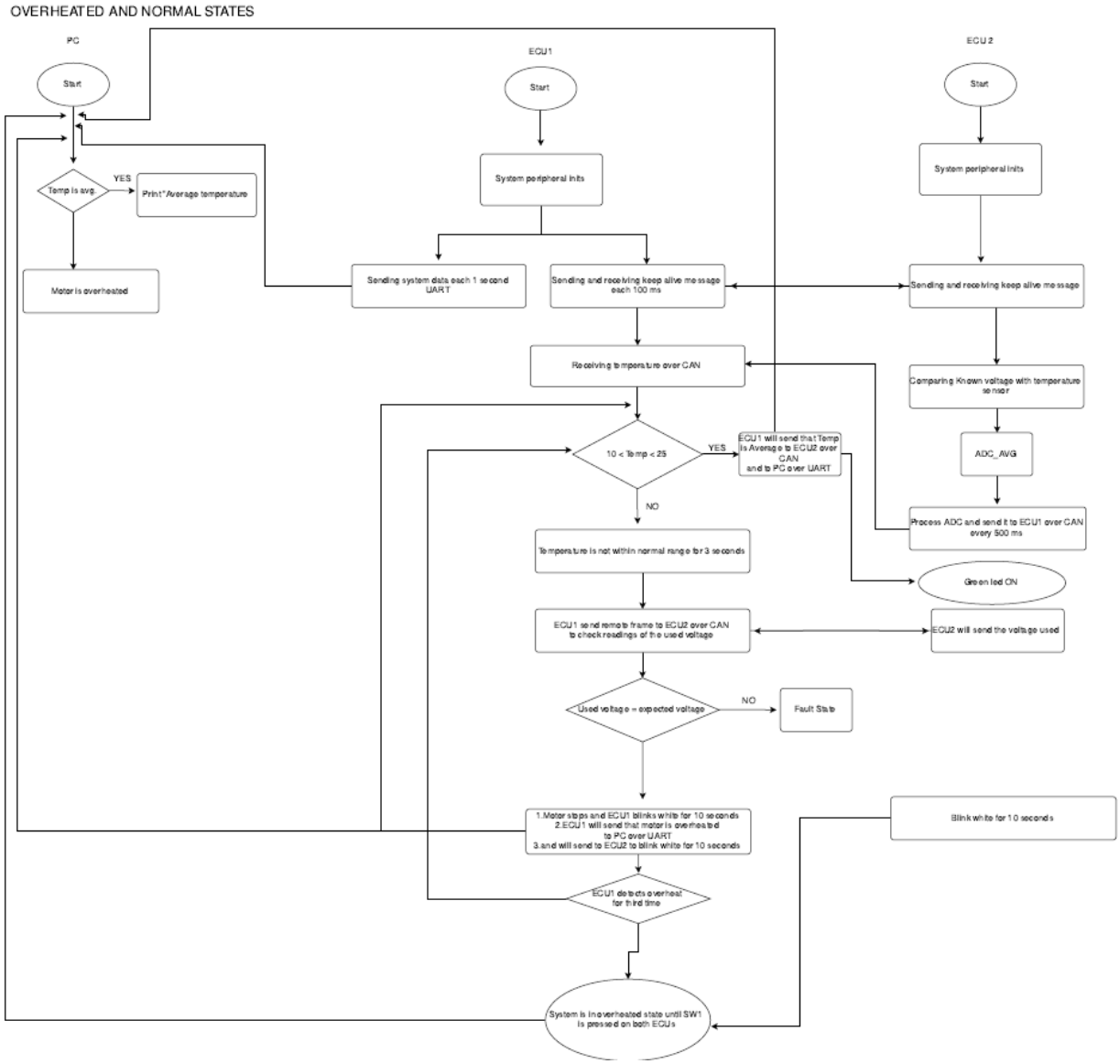

Figure 2:overheated, normal states flowchart

Fault, working systems Flowchart

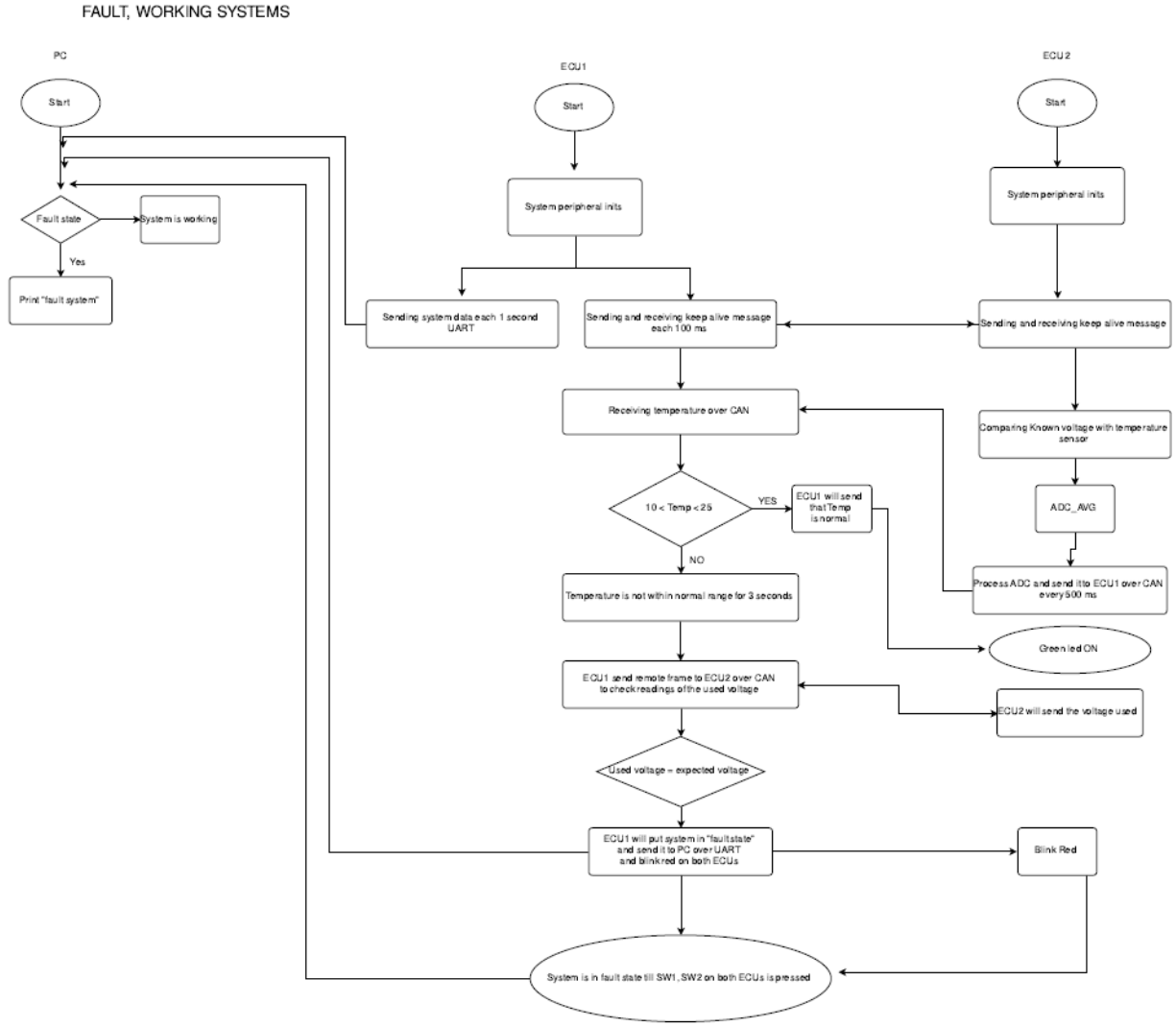

Figure 3: Fault, working systems flowchart

Communication Flowchart

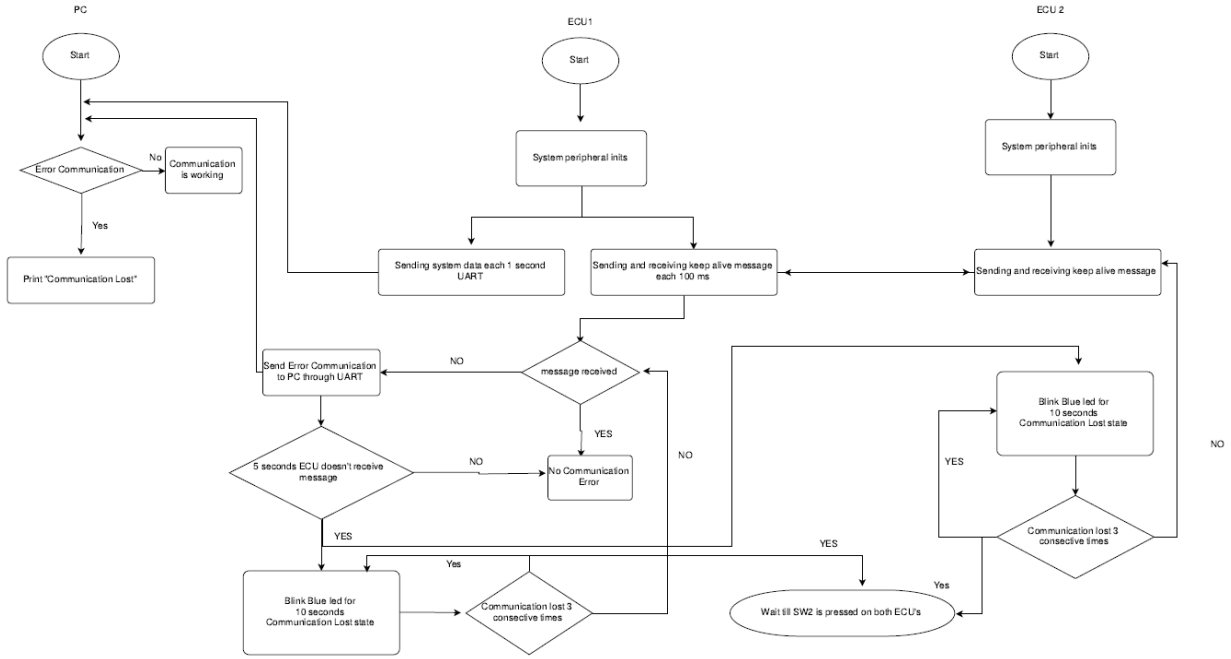

Figure 4: communication flowchart