MCT-411 Automotive Embedded Networking

SRS Document

SOFTWARE REQUIREMENTS SPECIFICATION

Submitted by:

Ahmed Essam Salem
ahmed-medo1999@hotmail.com

Table of Contents

List of Figures

1. Introduction

1.1 Purpose

The purpose of this document is to build a CAN bus networking system that connects number of Electronic Control Units (ECUs) and allow them to communicate with each other with the aid of AUTOSAR standardization to improve complexity management of integrated E/E architectures, facilitate portability, composability, integration of SW components.

1.2 Intended Use and Reading Suggestions

This project is a prototype for the development of a communication system between 3 microcontrollers on CAN bus and it is restricted within the college premises. This has been implemented under the guidance of college professors. This project is useful for communication between various electronic devices which is embedded in automobiles.

1.3 Project Scope

This project is used to implement a CAN bus system that allows centralized control over ECUs that are connected to the network so that controlling ECUs becomes easy using Arm Cortex M4 Tiva C board as microcontroller. The first ECU functions as main ECU to control the output state of other ECUs by sending a command on the CAN network based on the on-board switches' states. Other ECUs receive the commands sent on the bus and execute them using output LEDs. All ECUs are restricted with specific timing constraints during their functionality. More over, this was applied by the AUTOSAR to standardize the software architecture of ECUs.

1.4 Document Conventions

This document uses the following conventions.

CAN	Controller Area Network
ECU	Electronic Control Unit
AUTOSAR	Automotive Operating System Architecture
E/E	Electrical/Electronic
SW	Software
LED	Light Emitting Diode
API	Application Program Interface

1.5 Referencess

- https://www.autosar.org/fileadmin/user_upload/standards/classic/4-3/AUTOSAR_SWS_DIODriver.pdf
- https://www.autosar.org/fileadmin/user_upload/standards/classic/4-3/AUTOSAR_SWS_PortDriver.pdf
- https://www.autosar.org/fileadmin/user_upload/standards/classic/4-3/AUTOSAR_EXP_LayeredSoftwareArchitecture.pdf

2. Overall Description

2.1 Operating Environment

Specifically, an ECU prepares and broadcast information (switch state) via the CAN bus (consisting of two wires, CAN low and CAN high). The broadcasted data is accepted by all other ECUs on the CAN network - and each ECU then check the data and decide an action based on the data sent. Depends on the data order and timing, an action is taken (LEDs blinking) by each ECU.

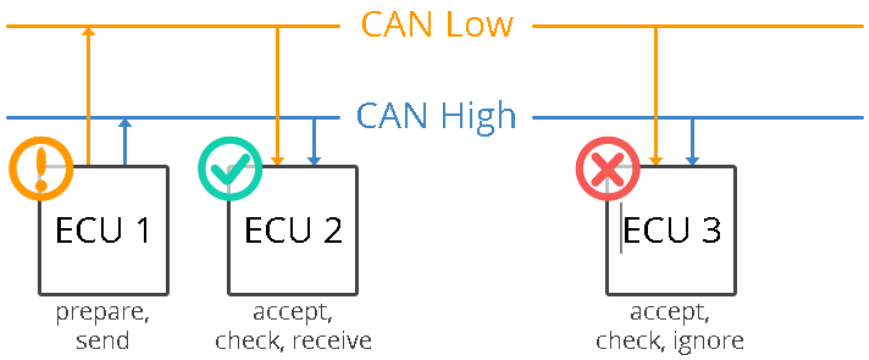

Figure 1 CAN Bus ECU Nodes Connection

2.2 Design and Implementation Constraints

1. The system consists of three ECUs connected together on CAN Bus.
2. ECU 1 acts as central node that controls the state machines of other ECUs.
3. Both ECUs 2 and 3 consist of periodic state machine tasks.
4. ECU 1 reads the input switches (the on-board switches) and sends a command on CAN network based on their state (ON/OFF).
5. Switch 1 of ECU 1 controls the state machine of ECU 2 and Switch 2 of ECU 1 controls the state machine of ECU 3, more over both switches control both ECUs' state machines.
6. Both ECUs initial state is considered to be Red LED blink.
7. Both ECUs read the commands on the CAN network pending on their state and some timing constraints.
8. Either switches pressed, the output of state machine is a LED blinking in a clockwise direction as RED ⟶ GREEN ⟶ BLUE, both switches pressed returns both ECUs to RED.

Constraints:

i. ECU 1 sends a message on the CAN bus of the switches' states every 500 ms.
ii. ECU 2 & 3's state machines are checked and executed each 10 ms periodicity.
iii. ECU 2 & 3's LEDs are turned on for at least 1 second window before it turns to another LED color although another switch press command was sent on the CAN.

2.3 Assumptions and Dependencies

Let us assume that this is a CAN bus system installed on a modern convertible car and the driver presses on a button to turn on the car windshield wiper system for a few time, and another button for the open and close of the car's rooftop for another time.

Assuming each subsystem is connected with an ECU and the main subsystem is the one interfacing with the buttons, we have designed a CAN network to enable communication between the systems with specific timing constraints on each ECU.

This project is implemented with dependency on projects made with college labs before.

System Features and Requirements

1. System Features

This section focuses on static views of a conceptual layered software architecture. The Layered Software Architecture describes the software architecture of AUTOSAR. It describes in an top-down approach the hierarchical structure of AUTOSAR software, maps the Basic Software Modules to software layers and shows their relationship.

The first ECU features and general function are discussed in more detail in this section as an architecture overview.

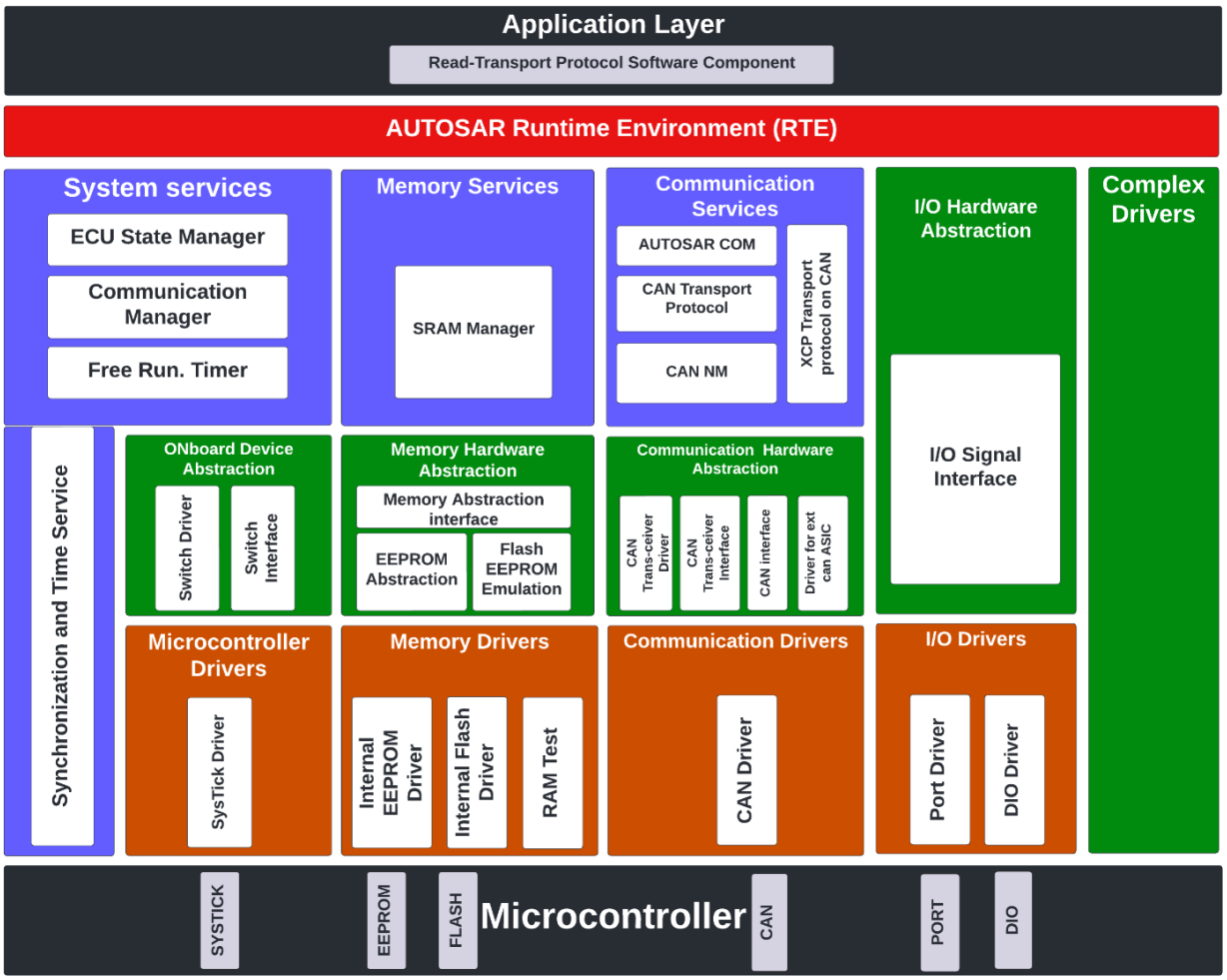

Figure 2 ECU 1 Layered Architecture

2. Functional Requirements

There are requirements specified to function the ECU as designed. Here we provide a flowchart to visually express how the software will be performing and how different functions depend on each other.

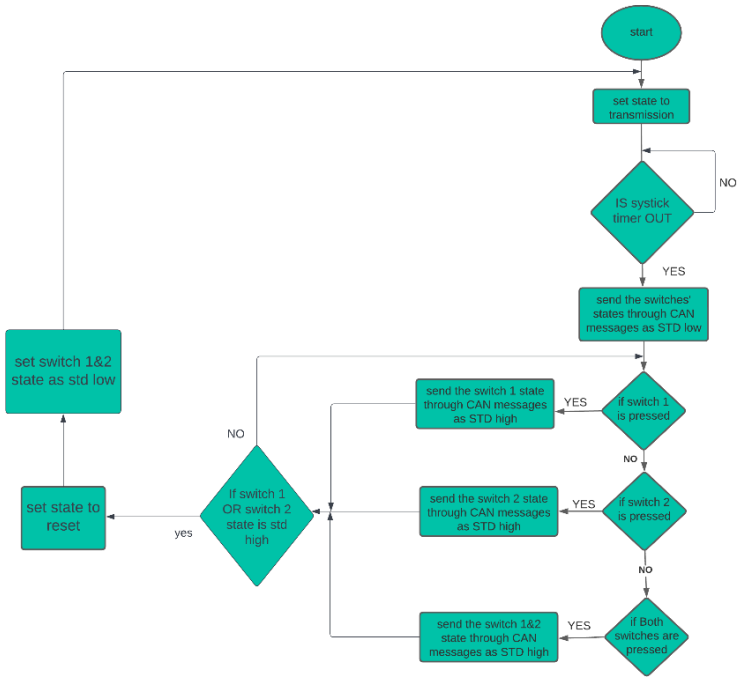

Figure 3 ECU 1 Operating Flowchart

Data flow diagram was also used to see the processes and what data is transferred between different functions. This way, each process or function is expressed as a block, while lines show what information passes between processes. As you can see in the figure simple data flow diagram is presented. Without program code, it is easy to read how the program operates:

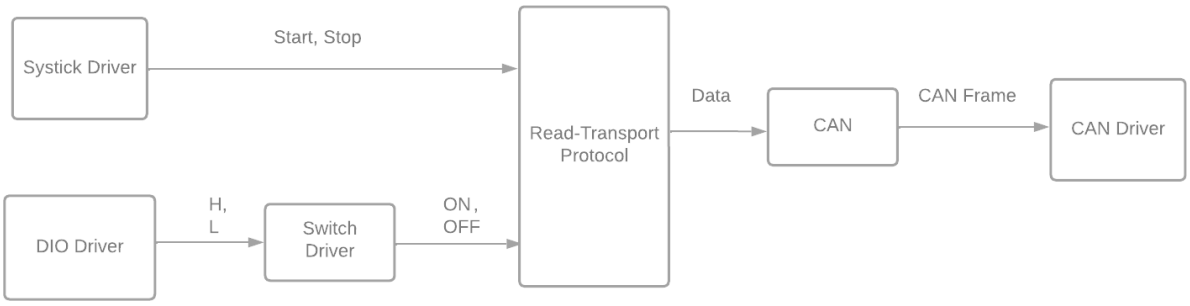

Figure 4 ECU 1 Dataflow Diagram

3 Nonfunctional Requirements

Each API contains and is implemented by function calls – language statements that request software to perform particular actions and services. An API specification is represented for each component used in the ECU:

PORT Driver:

Function Name	Port_Init
Input Parameters	ConfigPtr Description: Pointer to configuration set
Output Parameters	None
Input/Out Parameters	None
Fun. Description	Initializes the Port Driver module.

AUTOSAR API Function Requirements:

[SWS_Port_00041] ⌈The function Port_Init shall initialize ALL ports and port pins with the configuration set pointed to by the parameter ConfigPtr. ⌋

[SWS_Port_00078] ⌈The Port Driver module's environment shall call the function Port_Init first in order to initialize the port for use. ⌋

[SWS_Port_00213] ⌈If Port_Init function is not called first, then no operation can occur on the MCU ports and port pins. ⌋

[SWS_Port_00042] ⌈The function Port_Init shall initialize all configured resources. ⌋

[SWS_Port_00043] ⌈The function Port_Init shall avoid glitches and spikes on the affected port pins. ⌋

[SWS_Port_00071] ⌈The Port Driver module's environment shall call the function Port_Init after a reset in order to reconfigure the ports and port pins of the MCU. ⌋

[SWS_Port_00002] ⌈The function Port_Init shall initialize all variables used by the PORT driver module to an initial state. ⌋

[SWS_Port_00003] ⌈The Port Driver module's environment may also uses the function Port_Init to initialize the driver software and reinitialize the ports and port pins to another configured state depending on the configuration set passed to this function. ⌋

[SWS_Port_00055] ⌈The function Port_Init shall set the port pin output latch to a default level (defined during configuration) before setting the port pin direction to output. ⌋

Requirement SWS_Port_00055 ensures that the default level is immediately output on the port pin when it is set to an output port pin.

[SWS_Port_00121] ⌈The function Port_Init shall always have a pointer as a parameter, even though for the configuration variant VARIANT-PRE-COMPILE, no configuration set shall be given. In this case, the Port Driver module's environment shall pass a NULL pointer to the function Port_Init. ⌋

The Port Driver module's environment shall not call the function Port_Init during a running operation. This shall only apply if there is more than one caller of the PORT module. Configuration of Port_Init: All port pins and their functions, and alternate functions shall be configured by the configuration tool.

API Configuration Parameters:

Parameter	Description
GPIO_CR_NO	defined as 0x1F to allow changes for needed pins.
DIO	defined as 0 for Port_ConfigType pin mode member access.
CAN	defined as 1 for Port_ConfigType pin mode member access.
PortA	defined as GPIO_PORTA_BASE to specify the port base address.
PortB	defined as GPIO_PORTB_BASE to specify the port base address.
PortC	defined as GPIO_PORTC_BASE to specify the port base address.
PortD	defined as GPIO_PORTD_BASE to specify the port base address.
PortE	defined as GPIO_PORTE_BASE to specify the port base address.
PortF	defined as GPIO_PORTF_BASE to specify the port base address.
Pin_0	defined as GPIO_PIN_0 to specify the bit field value.
Pin_1	defined as GPIO_PIN_1 to specify the bit field value.
Pin_2	defined as GPIO_PIN_2 to specify the bit field value.
Pin_3	defined as GPIO_PIN_3 to specify the bit field value.
Pin_4	defined as GPIO_PIN_4 to specify the bit field value.
Pin_5	defined as GPIO_PIN_5 to specify the bit field value.
Pin_6	defined as GPIO_PIN_6 to specify the bit field value.
Pin_7	defined as GPIO_PIN_7 to specify the bit field value.
ClkFreq	defined as 25000000 for CAN clock frequency possibility.
BR	defined as 500000 for CAN driver baud rate.

DIO Driver:

Function Name	Dio_Init
Input Parameters	None
Output Parameters	None
Input/Out Parameters	None
Fun. Description	Initializes the DIO Driver module.

API Configuration Parameters:

Parameter	Description
GPIO_PORT	defined as PortF to specify GPIO port F base address.
Channel_0	defined as Pin_0 to specify bit field value of Switch 2 pin on port F.
Channel_1	defined as Pin_1 to specify bit field value of Switch 1 pin on port F.
GPIOPort_Interrupt	defined as INT_GPIOF to assign port F interrupt.
STD_HIGH	defined as 0x00 to determine that the switch is pressed.
STD_LOW	defined as 0x01 to determine that the switch is not pressed.

Function Name	Dio_ReadChannel
Input Parameters	ChannelId Description: ID of DIO Channel
Output Parameters	None
Input/Out Parameters	None
Return Value	Dio_LevelType (STD_HIGH: The physical level of the corresponding Pin is STD_HIGH, STD_LOW: The physical level of the corresponding Pin is STD_LOW)
Fun. Description	Returns the value of the specified DIO channel.

AUTOSAR Function Requirements:

[SWS_Dio_00027] ⌈The Dio_ReadChannel function shall return the value of the specified DIO channel.

[SWS_Dio_00083] ⌈If the microcontroller supports the direct read-back of a pin value, the Dio module's read functions shall provide the real pin level, when they are used on a channel which is configured as an output channel.⌋

[SWS_Dio_00084] ⌈If the microcontroller does not support the direct read-back of a pin value, the Dio module's read functions shall provide the value of the output register, when they are used on a channel which is configured as an output channel.⌋

[SWS_Dio_00005] ⌈The Dio module's read and write services shall ensure for all services, that the data is consistent (Interruptible read-modify-write sequences are not allowed).⌋

[SWS_Dio_00118] ⌈If development errors are enabled and an error ocurred the Dio module's read functions shall return with the value '0'. ⌋

[SWS_Dio_00026] ⌈The configuration process for Dio module shall provide symbolic names for each configured DIO channel, port and group.⌋

CAN Driver:

Function Name	CAN_Init
Input Parameters	None
Output Parameters	None
Input/Out Parameters	None
Fun. Description	Initializes the CAN Driver module.

API Configuration Parameters:

Parameter	Description
sCANMessage1	Global variable object for CAN message 1.
sCANMessage2	Global variable object for CAN message 2.
sCANMessage3	Global variable object for CAN message 3.
SW1Data	Global variable for CAN 1 message object data.
SW2Data	Global variable for CAN 2 message object data.
BothSWData	Global variable for CAN 3 message object data.
g_bErrFlag	Global volatile variable for indication of CAN message transmit error.
Msg1TX_Object	defined as 1 for CAN message 1 object ID.
Msg2TX_Object	defined as 2 for CAN message 1 object ID.
Msg3TX_Object	defined as 3 for CAN message 1 object ID.
MsgIDMask	defined as 0x00 for CAN message objects mask ID.
Msg1_ID	defined as 0x1001 for ID of CAN message 1 to send on.
Msg2_ID	defined as 0x2001 for ID of CAN message 2 to send on.
Msg3_ID	defined as 0x3001 for ID of CAN message 3 to send on.
CAN_GPIOPort	defined as PortB to specify GPIO port B base address.
GPIO_RXPin	defined as Pin_4 to specify bit field value of CAN RX pin on port.
GPIO_TXPin	defined as Pin_5 to specify bit field value of CAN TX pin on port.
CANChannel	defined as CAN0_BASE to specify the CAN base address.
CANChannel_Interrupt	defined as INT_CAN to assign CAN 0 interrupt.

Function Name	CAN_Write
Input Parameters	Switch1_Status Description: Value of DIO Channel 0 Switch2_Status Description: Value of DIO Channel 1
Output Parameters	None
Input/Out Parameters	None
Fun. Description	Passes a CAN message to CanDrv for transmission depending on input switches' state.

Function Name	CANIntHandler
Input Parameters	None
Output Parameters	None
Input/Out Parameters	None
Fun. Description	Reads the CAN interrupt controller status.

SysTick Driver:

Function Name	Systick_Init
Input Parameters	None
Output Parameters	None
Input/Out Parameters	None
Fun. Description	Initializes the SysTick Driver.

Switch Driver:

Function Name	Switch_Init
Input Parameters	None
Output Parameters	None
Input/Out Parameters	None
Fun. Description	Initializes the ECU Switch module.

Function Name	Switch_GetStatus
Input Parameters	ChannelId Description: ID of DIO Channel
Output Parameters	None
Input/Out Parameters	None
Return Value	uint8 (SW_HIGH: 0x00, SW_LOW: 0x01)
Fun. Description	Returns the value of the switch DIO channel.

Read-Transport Protocol Software Component:

Function Name	App_RP_Init
Input Parameters	None
Output Parameters	None
Input/Out Parameters	None
Fun. Description	Initializes the Read Protocol Component.

API Configuration Parameters:

Parameter	Description
PB1	Global variable for Switch 1 read value.
PB2	Global variable for Switch 2 read value.
Switch1	defined as 1 to specify numeric ID of DIO channel.
Switch2	defined as 0 to specify numeric ID of DIO channel.

Function Name	App_RP
Input Parameters	None
Output Parameters	None
Input/Out Parameters	None
Fun. Description	Assigns the returned switch read values into global variables.

Function Name	App_TP
Input Parameters	None
Output Parameters	None
Input/Out Parameters	None
Fun. Description	Writes the switch value over CAN Bus each 500ms.

-NO compilation warnings exist in code.

3 External Interface Requirements

3.1 User Interfaces

- C Compiler version

3.2 Hardware Interfaces

- Windows
- IDE that supports TI's microcontroller (MCU) and embedded processor portfolios.
- Arm Cortex M4 Tiva C board

3.3 Software Interfaces

Following are the software used for the project.

Software used	Description
Operating System	We have chosen Windows operating system for its best support and user-friendliness.
CCS (Code Composer Studio)	To develop and debug application, we have chosen CCS as an IDE.
C	To implement the project we have chosen C programming language.

System Features and Requirements

1. System Features

This section focuses on static views of a conceptual layered software architecture. The Layered Software Architecture describes the software architecture of AUTOSAR. It describes in an top-down approach the hierarchical structure of AUTOSAR software, maps the Basic Software Modules to software layers and shows their relationship.

The *second* ECU features and general function are discussed in more detail in this section as an architecture overview.

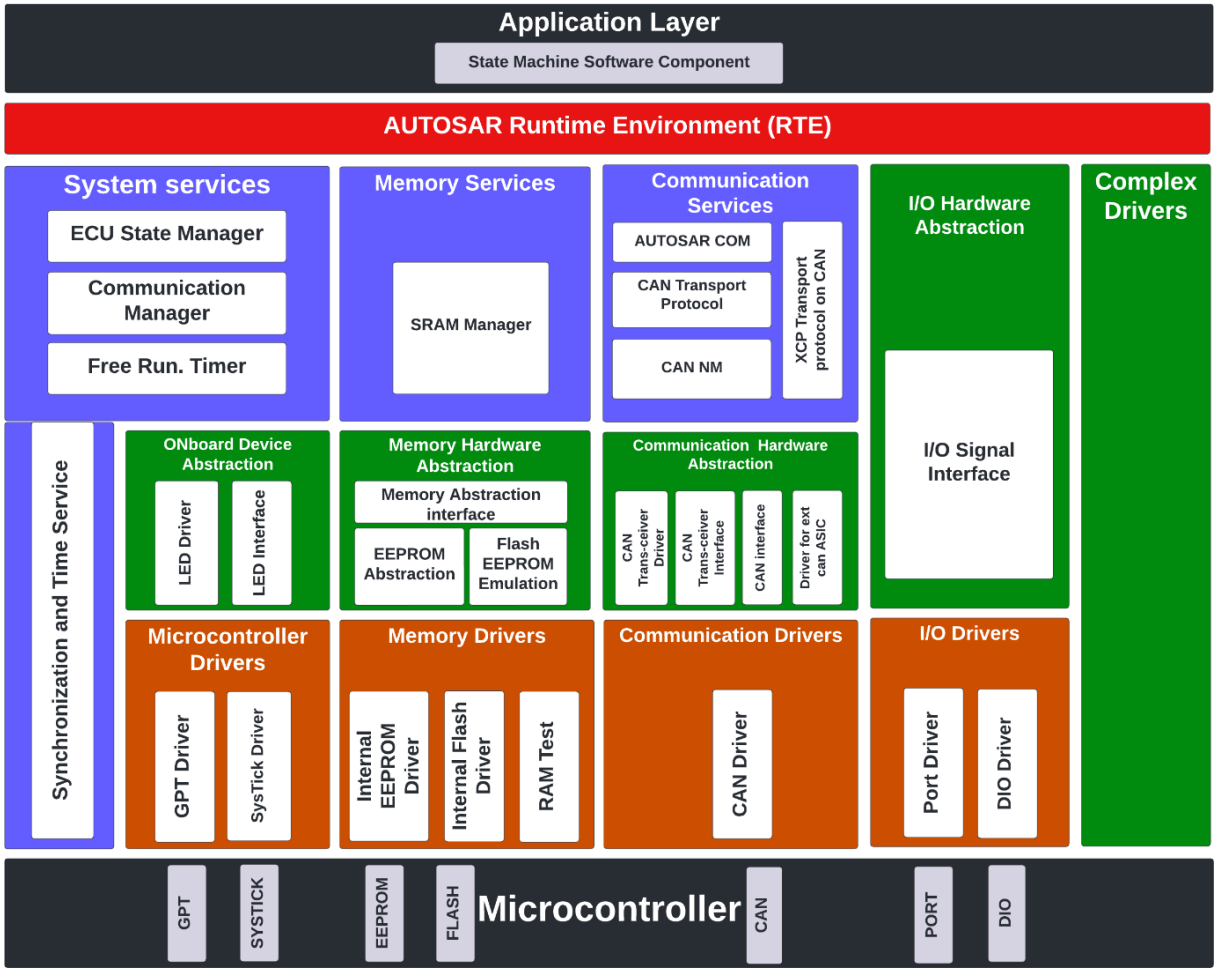

Figure 5 ECU 2 Layered Architecture

2. Functional Requirements

There are requirements specified to function the ECU as designed. Here we provide a flowchart to visually express how the software will be performing and how different functions depend on each other.

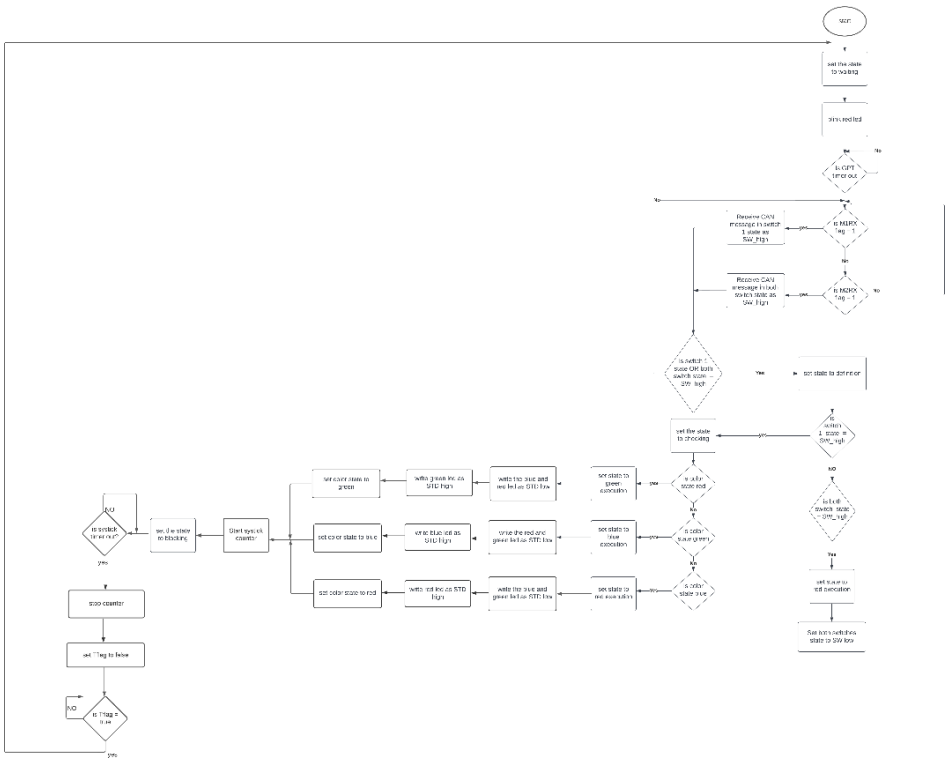

Figure 6 ECU 2 Operating Flowchart

Data flow diagram was also used to see the processes and what data is transferred between different functions. This way, each process or function is expressed as a block, while lines show what information passes between processes. As you can see in the figure simple data flow diagram is presented. Without program code, it is easy to read how the program operates:

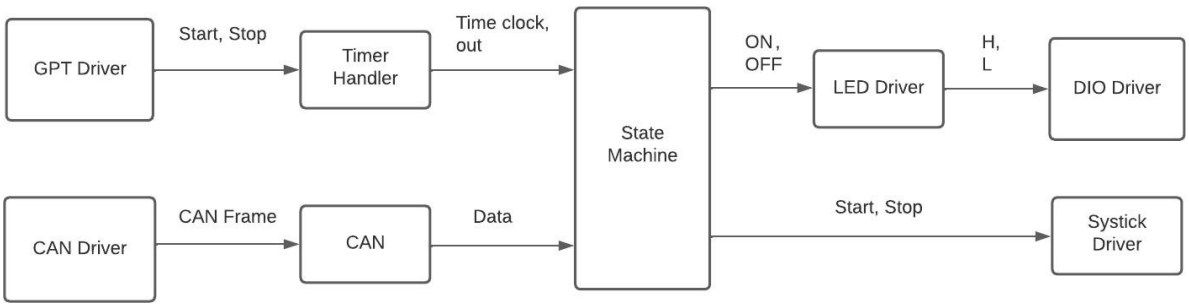

Figure 7 ECU 2 Dataflow Diagram

State diagram is needed and more useful for *second* ECU to represent each possible state of ECU and what inputs cause it to change to another state.

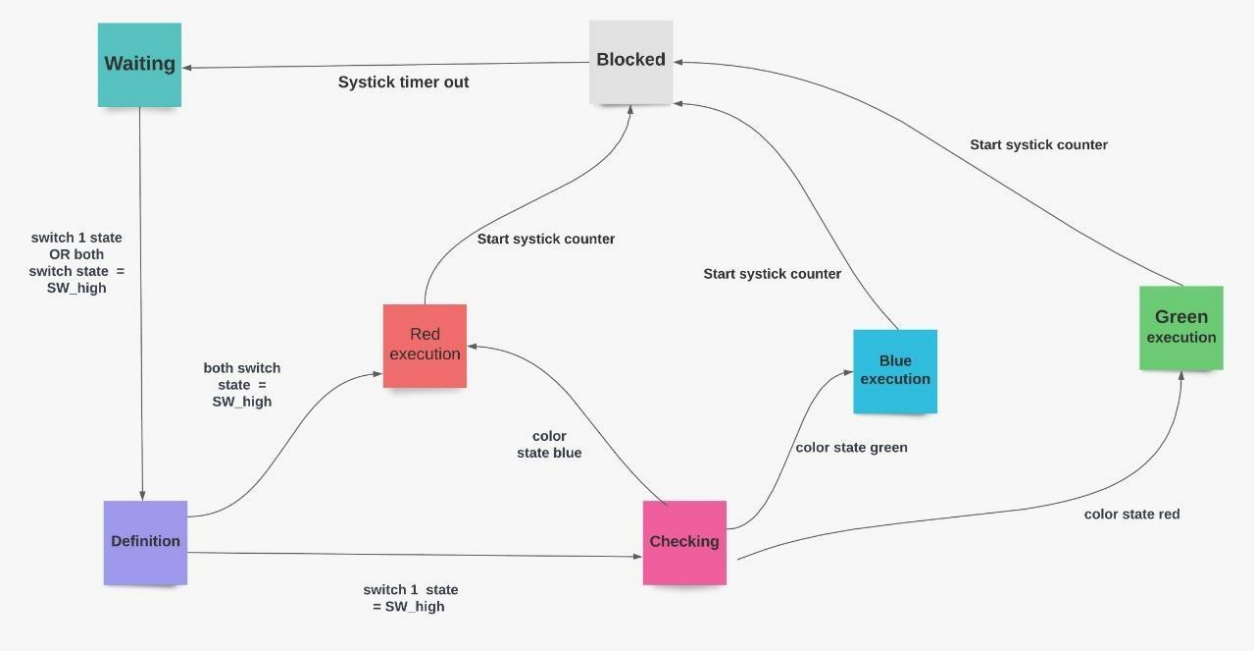

Figure 8 ECU 2 State Machine Diagram

3 Nonfunctional Requirements

Each API contains and is implemented by function calls – language statements that request software to perform particular actions and services. An API specification is represented for each component used in the ECU:

PORT Driver:

Function Name	Port_Init
Input Parameters	ConfigPtr Description: Pointer to configuration set
Output Parameters	None
Input/Out Parameters	None
Fun. Description	Initializes the Port Driver module.

AUTOSAR API Function Requirements:

[SWS_Port_00041] ⌈The function Port_Init shall initialize ALL ports and port pins with the configuration set pointed to by the parameter ConfigPtr. ⌋

[SWS_Port_00078] ⌈The Port Driver module's environment shall call the function Port_Init first in order to initialize the port for use. ⌋

[SWS_Port_00213] ⌈If Port_Init function is not called first, then no operation can occur on the MCU ports and port pins. ⌋

[SWS_Port_00042] ⌈The function Port_Init shall initialize all configured resources. ⌋

[SWS_Port_00043] ⌈The function Port_Init shall avoid glitches and spikes on the affected port pins. ⌋

[SWS_Port_00071] ⌈The Port Driver module's environment shall call the function Port_Init after a reset in order to reconfigure the ports and port pins of the MCU. ⌋

[SWS_Port_00002] ⌈The function Port_Init shall initialize all variables used by the PORT driver module to an initial state. ⌋

[SWS_Port_00003] ⌈The Port Driver module's environment may also uses the function Port_Init to initialize the driver software and reinitialize the ports and port pins to another configured state depending on the configuration set passed to this function. ⌋

[SWS_Port_00055] ⌈The function Port_Init shall set the port pin output latch to a default level (defined during configuration) before setting the port pin direction to output. ⌋

Requirement SWS_Port_00055 ensures that the default level is immediately output on the port pin when it is set to an output port pin.

[SWS_Port_00121] ⌈The function Port_Init shall always have a pointer as a parameter, even though for the configuration variant VARIANT-PRE-COMPILE, no configuration set shall be given. In this case, the Port Driver module's environment shall pass a NULL pointer to the function Port_Init. ⌋

The Port Driver module's environment shall not call the function Port_Init during a running operation. This shall only apply if there is more than one caller of the PORT module. Configuration of Port_Init: All port pins and their functions, and alternate functions shall be configured by the configuration tool.

API Configuration Parameters:

Parameter	Description
GPIO_CR_NO	defined as 0x1F to allow changes for needed pins.
DIO	defined as 0 for Port_ConfigType pin mode member access.
CAN	defined as 1 for Port_ConfigType pin mode member access.
PortA	defined as GPIO_PORTA_BASE to specify the port base address.
PortB	defined as GPIO_PORTB_BASE to specify the port base address.
PortC	defined as GPIO_PORTC_BASE to specify the port base address.
PortD	defined as GPIO_PORTD_BASE to specify the port base address.
PortE	defined as GPIO_PORTE_BASE to specify the port base address.
PortF	defined as GPIO_PORTF_BASE to specify the port base address.
Pin_0	defined as GPIO_PIN_0 to specify the bit field value.
Pin_1	defined as GPIO_PIN_1 to specify the bit field value.
Pin_2	defined as GPIO_PIN_2 to specify the bit field value.
Pin_3	defined as GPIO_PIN_3 to specify the bit field value.
Pin_4	defined as GPIO_PIN_4 to specify the bit field value.
Pin_5	defined as GPIO_PIN_5 to specify the bit field value.
Pin_6	defined as GPIO_PIN_6 to specify the bit field value.
Pin_7	defined as GPIO_PIN_7 to specify the bit field value.
ClkFreq	defined as 25000000 for CAN clock frequency possibility.
BR	defined as 500000 for CAN driver baud rate.

DIO Driver:

Function Name	DioChannel_Init
Input Parameters	None
Output Parameters	None
Input/Out Parameters	None
Fun. Description	Initializes the DIO Driver module.

API Configuration Parameters:

Parameter	Description
GPIO_PORT	defined as PortF to specify GPIO port F base address.
Channel_0	defined as Pin_1 to specify bit field value of Red LED pin on port F.
Channel_1	defined as Pin_2 to specify bit field value of Blue LED pin on port F.
Channel_2	defined as Pin_3 to specify bit field value of Green LED pin on port F.
STD_HIGH	defined as 0x01 to determine that the LED is on.
STD_LOW	defined as 0x00 to determine that the LED is off.

Function Name	Dio_WriteChannel
Input Parameters	ChannelId Description: ID of DIO Channel Level Description: Value to be written
Output Parameters	None
Input/Out Parameters	None
Fun. Description	Service to set a level of a channel.

AUTOSAR Function Requirements:

[SWS_Dio_00028] ⌈If the specified channel is configured as an output channel, the Dio_WriteChannel function shall set the specified Level for the specified channel. ⌋

[SWS_Dio_00029] ⌈If the specified channel is configured as an input channel, the Dio_WriteChannel function shall have no influence on the physical output. ⌋

[SWS_Dio_00079] ⌈If the specified channel is configured as an input channel, the Dio_WriteChannel function shall have no influence on the result of the next Read-Service.

[SWS_Dio_00119] ⌈If development errors are enabled and an error ocurred, the Dio module's write functions shall NOT process the write command. ⌋

[SWS_Dio_00026] ⌈The configuration process for Dio module shall provide symbolic names for each configured DIO channel, port and group.⌋

CAN Driver:

Function Name	CAN_Init
Input Parameters	None
Output Parameters	None
Input/Out Parameters	None
Fun. Description	Initializes the CAN Driver module.

API Configuration Parameters:

Parameter	Description
sCANMessageRX	Global variable object for CAN messages.
pui8MsgDataRX	Global variable for CAN messages object data.
Msg1_ID	defined as 0x1001 for ID of CAN message 1 to receive on.
Msg2_ID	defined as 0x3001 for ID of CAN message 2 to receive on.
Msg1RX_Object	defined as 1 for CAN message 1 object ID.
Msg2RX_Object	defined as 2 for CAN message 2 object ID.
MsgIDMask	defined as 0xffffff for CAN message objects mask ID.
g_M1RXFlag	Global volatile variable for CAN message 1 receive indication.
g_M2RXFlag	Global volatile variable for CAN message 2 receive indication.
g_bErrFlag	Global volatile variable for indication of CAN message transmit error.
CAN_GPIOPort	defined as PortB to specify GPIO port B base address.
Pinoxepin	defined as Pin_4 to specify bit field value of CAN RX pin on port.
GPIO_TXPin	defined as Pin_5 to specify bit field value of CAN TX pin on port.
CANChannel	defined as CAN0_BASE to specify the CAN base address.
CANChannel_Interrupt	defined as INT_CAN0 to assign CAN 0 interrupt.
SW_HIGH	defined as 0x00 to determine that the switch is pressed.
SW_LOW	defined as 0x01 to determine that the switch is not pressed.

Function Name	CAN_Read
Input Parameters	None
Output Parameters	None
Input/Out Parameters	None
Fun. Description	Reads if a CAN message is sent on CanDrv for receive.

API Configuration Parameters:

Parameter	Description
SW1State	Global variable for CAN message 1 data of Switch 1 state.
BothSWState	Global variable for CAN message 2 data of both Switches state.

Function Name	CANIntHandler
Input Parameters	None
Output Parameters	None
Input/Out Parameters	None
Fun. Description	Reads the CAN interrupt controller status.

SysTick Driver:

Function Name	Systick_Init
Input Parameters	None
Output Parameters	None
Input/Out Parameters	None
Fun. Description	Initializes the SysTick Driver.

Function Name	SystickIntHandler
Input Parameters	None
Output Parameters	None
Input/Out Parameters	None
Fun. Description	Sets the ECU state back to waiting and disables the systick timer.

Function Name	Systick_StartCount
Input Parameters	None
Output Parameters	None
Input/Out Parameters	None
Fun. Description	Enables the systick timer counter to start counting.

Function Name	Systick_EndCount
Input Parameters	None
Output Parameters	None
Input/Out Parameters	None
Fun. Description	Disables the systick timer from counting and reloads it.

GPT Driver:

Function Name	GPT_Init
Input Parameters	None
Output Parameters	None
Input/Out Parameters	None
Fun. Description	Initializes the GPT Driver.

API Configuration Parameters:

Parameter	Description
TIMERPeripheral	defined as SYSCTL_PERIPH_TIMER0 to specify Timer 0 periph value.
TIMERChannel	defined as TIMER0_BASE specify the Timer 0 base address.
TIMERConfig	defined as TIMER_CFG_SPLIT_PAIR | TIMER_CFG_B_PERIODIC to configure the timer to work independent and in periodic mode.
SubTimer	defined as TIMER_B to specify subtimer B value.
TIMERIntFlag	defined as TIMER_TIMB_TIMEOUT to specify bit mask of the interrupt source to be enabled.
TIMERChannel_Interrupt	defined as INT_TIMER0B to assign Timer 0B interrupt.

Function Name	TimerIntHandler
Input Parameters	None
Output Parameters	None
Input/Out Parameters	None
Fun. Description	Calls the ECU state machine periodically each 10ms.

LED Driver:

Function Name	LED_Init
Input Parameters	None
Output Parameters	None
Input/Out Parameters	None
Fun. Description	Initializes the ECU LED module.

Function Name	SetLED_ON
Input Parameters	LEDId Description: Numeric ID of LED Channel
Output Parameters	None
Input/Out Parameters	None
Fun. Description	Writes on LED Channel to set it on.

Function Name	SetLED_OFF
Input Parameters	LEDId Description: Numeric ID of LED Channel
Output Parameters	None
Input/Out Parameters	None
Fun. Description	Writes on LED Channel to set it off.

Function Name	LED_Toggle
Input Parameters	LEDId Description: Numeric ID of LED Channel
Output Parameters	None
Input/Out Parameters	None
Fun. Description	Writes on LED Channel to toggle it.

State Machine Software Component:

Function Name	App_SM_Init
Input Parameters	None
Output Parameters	None
Input/Out Parameters	None
Fun. Description	Initializes the ECU and color states.

API Configuration Parameters:

Parameter	Description
state	Global variable for ECU state declaration
colorstate	Global variable for ECU LED color state declaration
waiting	defined as 0 to specify ECU state name.
definition	defined as 1 to specify ECU state name.
checking	defined as 2 to specify ECU state name.
red_execution	defined as 3 to specify ECU state name.
green_execution	defined as 4 to specify ECU state name.
blue_execution	defined as 5 to specify ECU state name.
blocked	defined as 6 to specify ECU state name.
RED_LED	defined as 8 to specify ECU LED color state name.
GREEN_LED	defined as 9 to specify ECU LED color state name.
BLUE_LED	defined as 10 to specify ECU LED color state name.
Red	defined as 0 to specify red channel ID.
Green	defined as 1 to specify green channel ID.
Blue	defined as 2 to specify blue channel ID.

Function Name	App_SM
Input Parameters	None
Output Parameters	None
Input/Out Parameters	None
Fun. Description	Defines the states of ECU and how it operates.

-NO compilation warnings exist in code.

4 External Interface Requirements

4.1 User Interfaces

- C Compiler version

4.2 Hardware Interfaces

- Windows
- IDE that supports TI's microcontroller (MCU) and embedded processor portfolios.
- Arm Cortex M4 Tiva C board

4.3 Software Interfaces

Following are the software used for the project.

Software used	Description
Operating System	We have chosen Windows operating system for its best support and user-friendliness.
CCS (Code Composer Studio)	To develop and debug application, we have chosen CCS as an IDE.
C	To implement the project we have chosen C programming language.

System Features and Requirements

1. System Features

This section focuses on static views of a conceptual layered software architecture. The Layered Software Architecture describes the software architecture of AUTOSAR. It describes in an top-down approach the hierarchical structure of AUTOSAR software, maps the Basic Software Modules to software layers and shows their relationship.

The *third* ECU features and general function are discussed in more detail in this section as an architecture overview.

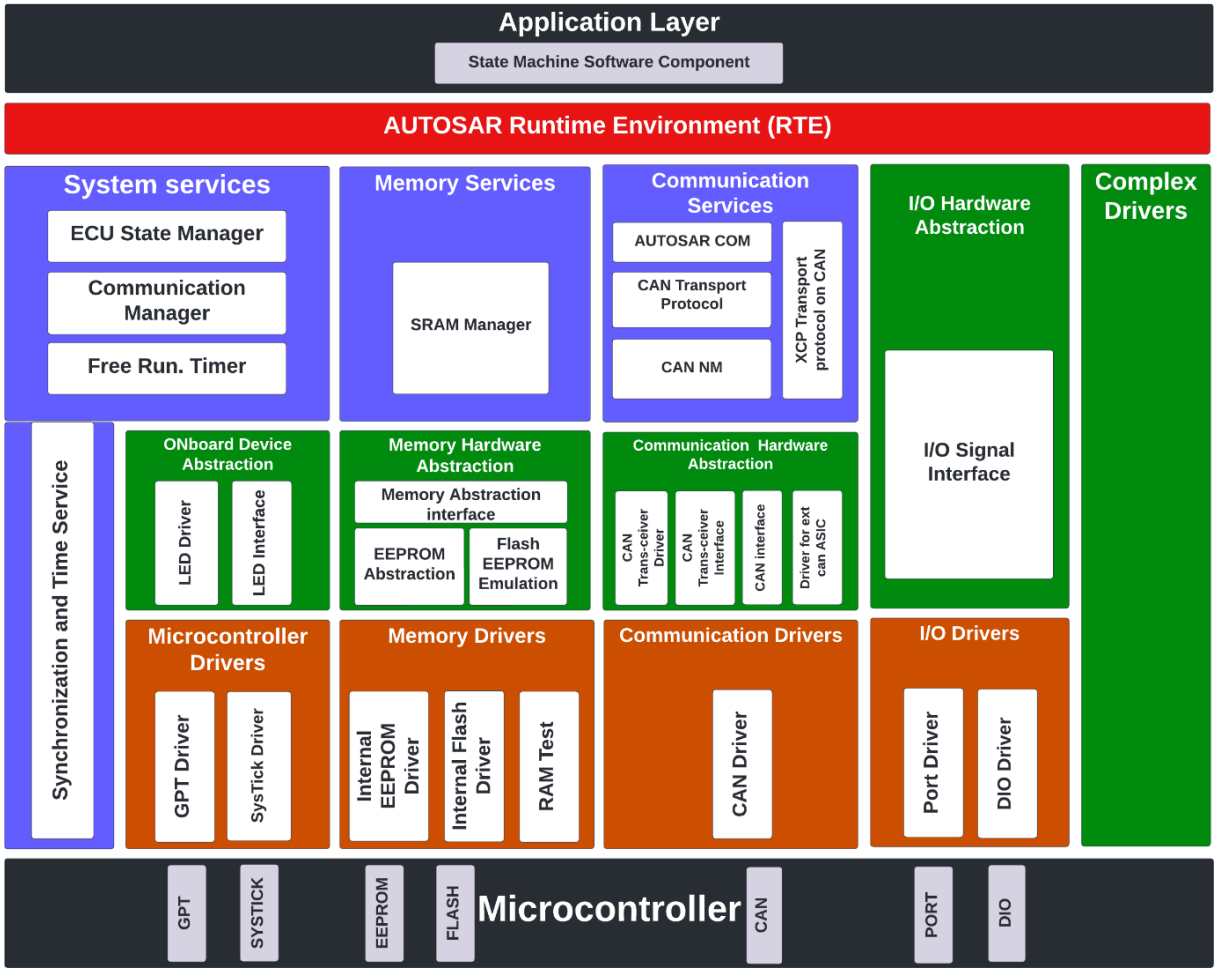

Figure 9 ECU 3 Layered Architecture

2. Functional Requirements

There are requirements specified to function the ECU as designed. Here we provide a flowchart to visually express how the software will be performing and how different functions depend on each other.

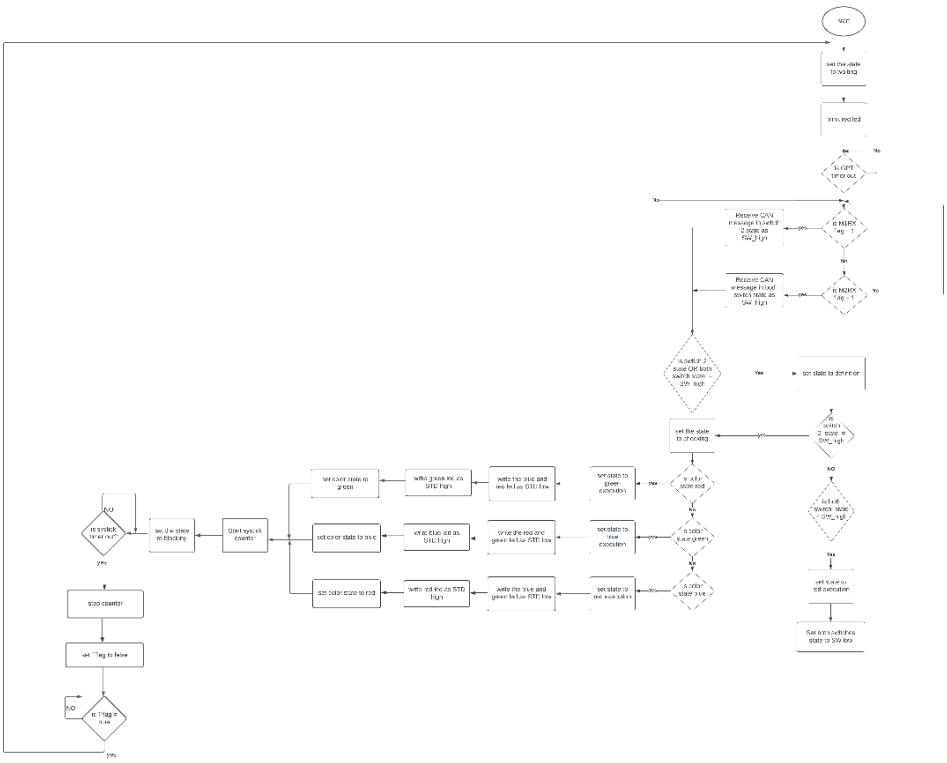

Figure 10 ECU 3 Operating Flowchart

Data flow diagram was also used to see the processes and what data is transferred between different functions. This way, each process or function is expressed as a block, while lines show what information passes between processes. As you can see in the figure simple data flow diagram is presented. Without program code, it is easy to read how the program operates:

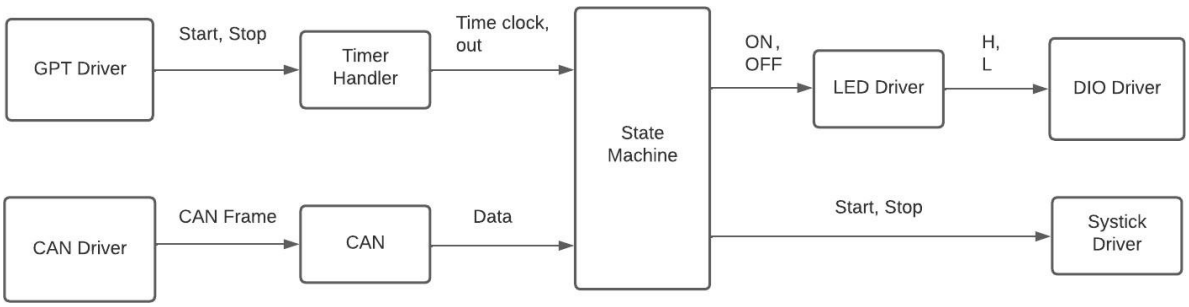

Figure 11 ECU 3 Dataflow Diagram

State diagram is needed and more useful for *third* ECU to represent each possible state of ECU and what inputs cause it to change to another state.

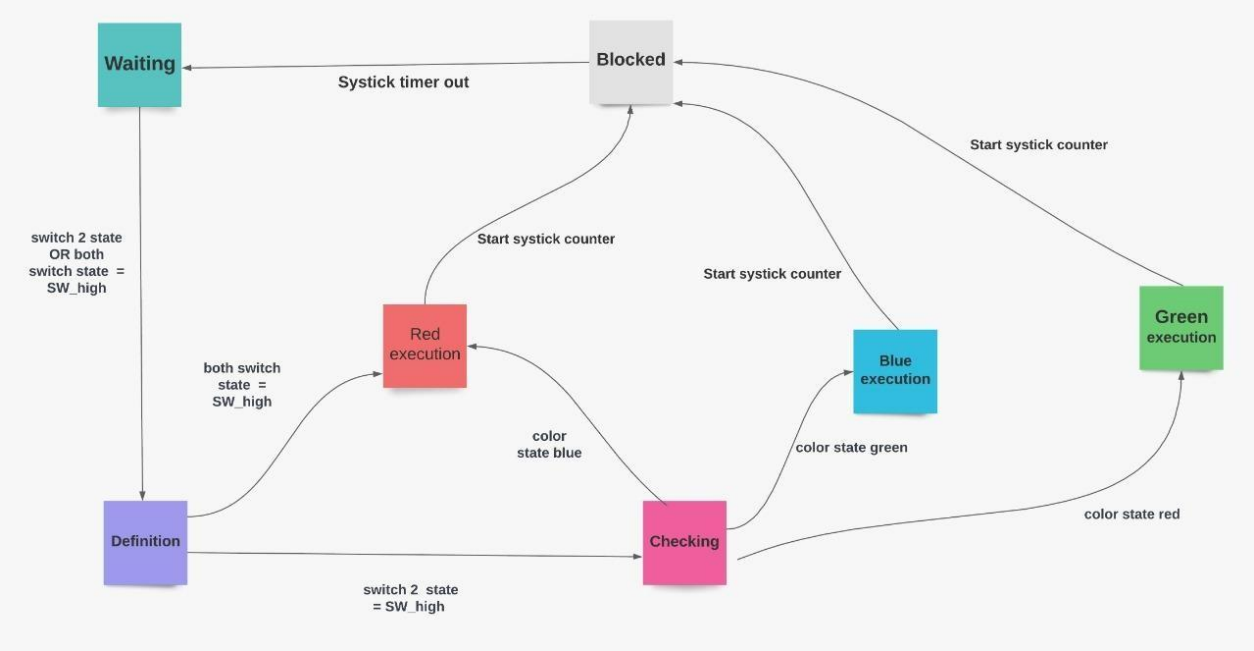

Figure 12 ECU 3 State Machine Diagram

3 Nonfunctional Requirements

Each API contains and is implemented by function calls – language statements that request software to perform particular actions and services. An API specification is represented for each component used in the ECU:

PORT Driver:

Function Name	Port_Init
Input Parameters	ConfigPtr Description: Pointer to configuration set
Output Parameters	None
Input/Out Parameters	None
Fun. Description	Initializes the Port Driver module.

AUTOSAR API Function Requirements:

[SWS_Port_00041] ⌈The function Port_Init shall initialize ALL ports and port pins with the configuration set pointed to by the parameter ConfigPtr. ⌋

[SWS_Port_00078] ⌈The Port Driver module's environment shall call the function Port_Init first in order to initialize the port for use. ⌋

[SWS_Port_00213] ⌈If Port_Init function is not called first, then no operation can occur on the MCU ports and port pins. ⌋

[SWS_Port_00042] ⌈The function Port_Init shall initialize all configured resources. ⌋

[SWS_Port_00043] ⌈The function Port_Init shall avoid glitches and spikes on the affected port pins. ⌋

[SWS_Port_00071] ⌈The Port Driver module's environment shall call the function Port_Init after a reset in order to reconfigure the ports and port pins of the MCU. ⌋

[SWS_Port_00002] ⌈The function Port_Init shall initialize all variables used by the PORT driver module to an initial state. ⌋

[SWS_Port_00003] ⌈The Port Driver module's environment may also uses the function Port_Init to initialize the driver software and reinitialize the ports and port pins to another configured state depending on the configuration set passed to this function. ⌋

[SWS_Port_00055] ⌈The function Port_Init shall set the port pin output latch to a default level (defined during configuration) before setting the port pin direction to output. ⌋

Requirement SWS_Port_00055 ensures that the default level is immediately output on the port pin when it is set to an output port pin.

[SWS_Port_00121] ⌈The function Port_Init shall always have a pointer as a parameter, even though for the configuration variant VARIANT-PRE-COMPILE, no configuration set shall be given. In this case, the Port Driver module's environment shall pass a NULL pointer to the function Port_Init. ⌋

The Port Driver module's environment shall not call the function Port_Init during a running operation. This shall only apply if there is more than one caller of the PORT module. Configuration of Port_Init: All port pins and their functions, and alternate functions shall be configured by the configuration tool.

API Configuration Parameters:

Parameter	Description
GPIO_CR_NO	defined as 0x1F to allow changes for needed pins.
DIO	defined as 0 for Port_ConfigType pin mode member access.
CAN	defined as 1 for Port_ConfigType pin mode member access.
PortA	defined as GPIO_PORTA_BASE to specify the port base address.
PortB	defined as GPIO_PORTB_BASE to specify the port base address.
PortC	defined as GPIO_PORTC_BASE to specify the port base address.
PortD	defined as GPIO_PORTD_BASE to specify the port base address.
PortE	defined as GPIO_PORTE_BASE to specify the port base address.
PortF	defined as GPIO_PORTF_BASE to specify the port base address.
Pin_0	defined as GPIO_PIN_0 to specify the bit field value.
Pin_1	defined as GPIO_PIN_1 to specify the bit field value.
Pin_2	defined as GPIO_PIN_2 to specify the bit field value.
Pin_3	defined as GPIO_PIN_3 to specify the bit field value.
Pin_4	defined as GPIO_PIN_4 to specify the bit field value.
Pin_5	defined as GPIO_PIN_5 to specify the bit field value.
Pin_6	defined as GPIO_PIN_6 to specify the bit field value.
Pin_7	defined as GPIO_PIN_7 to specify the bit field value.
ClkFreq	defined as 25000000 for CAN clock frequency possibility.
BR	defined as 500000 for CAN driver baud rate.

DIO Driver:

Function Name	DioChannel_Init
Input Parameters	None
Output Parameters	None
Input/Out Parameters	None
Fun. Description	Initializes the DIO Driver module.

API Configuration Parameters:

Parameter	Description
GPIO_PORT	defined as PortF to specify GPIO port F base address.
Channel_0	defined as Pin_1 to specify bit field value of Red LED pin on port F.
Channel_1	defined as Pin_2 to specify bit field value of Blue LED pin on port F.
Channel_2	defined as Pin_3 to specify bit field value of Green LED pin on port F.
STD_HIGH	defined as 0x01 to determine that the LED is on.
STD_LOW	defined as 0x00 to determine that the LED is off.

Function Name	Dio_WriteChannel
Input Parameters	ChannelId Description: ID of DIO Channel Level Description: Value to be written
Output Parameters	None
Input/Out Parameters	None
Fun. Description	Service to set a level of a channel.

AUTOSAR Function Requirements:

[SWS_Dio_00028] ⌈If the specified channel is configured as an output channel, the Dio_WriteChannel function shall set the specified Level for the specified channel. ⌋

[SWS_Dio_00029] ⌈If the specified channel is configured as an input channel, the Dio_WriteChannel function shall have no influence on the physical output. ⌋

[SWS_Dio_00079] ⌈If the specified channel is configured as an input channel, the Dio_WriteChannel function shall have no influence on the result of the next Read-Service.

[SWS_Dio_00119] ⌈If development errors are enabled and an error ocurred, the Dio module's write functions shall NOT process the write command. ⌋

[SWS_Dio_00026] ⌈The configuration process for Dio module shall provide symbolic names for each configured DIO channel, port and group.⌋

CAN Driver:

Function Name	CAN_Init
Input Parameters	None
Output Parameters	None
Input/Out Parameters	None
Fun. Description	Initializes the CAN Driver module.

API Configuration Parameters:

Parameter	Description
sCANMessageRX	Global variable object for CAN messages.
pui8MsgDataRX	Global variable for CAN messages object data.
Msg1_ID	defined as 0x2001 for ID of CAN message 1 to receive on.
Msg2_ID	defined as 0x3001 for ID of CAN message 2 to receive on.
Msg1RX_Object	defined as 1 for CAN message 1 object ID.
Msg2RX_Object	defined as 2 for CAN message 2 object ID.
MsgIDMask	defined as 0xffffff for CAN message objects mask ID.
g_M1RXFlag	Global volatile variable for CAN message 1 receive indication.
g_M2RXFlag	Global volatile variable for CAN message 2 receive indication.
g_bErrFlag	Global volatile variable for indication of CAN message transmit error.
CAN_GPIOPort	defined as PortB to specify GPIO port B base address.
Pinoxepin	defined as Pin_4 to specify bit field value of CAN RX pin on port.
GPIO_TXPin	defined as Pin_5 to specify bit field value of CAN TX pin on port.
CANChannel	defined as CAN0_BASE to specify the CAN base address.
CANChannel_Interrupt	defined as INT_CAN0 to assign CAN 0 interrupt.
SW_HIGH	defined as 0x00 to determine that the switch is pressed.
SW_LOW	defined as 0x01 to determine that the switch is not pressed.

Function Name	CAN_Read
Input Parameters	None
Output Parameters	None
Input/Out Parameters	None
Fun. Description	Reads if a CAN message is sent on CanDrv for receive.

API Configuration Parameters:

Parameter	Description
SW2State	Global variable for CAN message 1 data of Switch 1 state.
BothSWState	Global variable for CAN message 2 data of both Switches state.

Function Name	CANIntHandler
Input Parameters	None
Output Parameters	None
Input/Out Parameters	None
Fun. Description	Reads the CAN interrupt controller status.

SysTick Driver:

Function Name	Systick_Init
Input Parameters	None
Output Parameters	None
Input/Out Parameters	None
Fun. Description	Initializes the SysTick Driver.

Function Name	SystickIntHandler
Input Parameters	None
Output Parameters	None
Input/Out Parameters	None
Fun. Description	Sets the ECU state back to waiting and disables the systick timer.

Function Name	Systick_StartCount
Input Parameters	None
Output Parameters	None
Input/Out Parameters	None
Fun. Description	Enables the systick timer counter to start counting.

Function Name	Systick_EndCount
Input Parameters	None
Output Parameters	None
Input/Out Parameters	None
Fun. Description	Disables the systick timer from counting and reloads it.

GPT Driver:

Function Name	GPT_Init
Input Parameters	None
Output Parameters	None
Input/Out Parameters	None
Fun. Description	Initializes the GPT Driver.

API Configuration Parameters:

Parameter	Description
TIMERPeripheral	defined as SYSCTL_PERIPH_TIMER0 to specify Timer 0 periph value.
TIMERChannel	defined as TIMER0_BASE specify the Timer 0 base address.
TIMERConfig	defined as TIMER_CFG_SPLIT_PAIR | TIMER_CFG_B_PERIODIC to configure the timer to work independent and in periodic mode.
SubTimer	defined as TIMER_B to specify subtimer B value.
TIMERIntFlag	defined as TIMER_TIMB_TIMEOUT to specify bit mask of the interrupt source to be enabled.
TIMERChannel_Interrupt	defined as INT_TIMER0B to assign Timer 0B interrupt.

Function Name	TimerIntHandler
Input Parameters	None
Output Parameters	None
Input/Out Parameters	None
Fun. Description	Calls the ECU state machine periodically each 10ms.

LED Driver:

Function Name	LED_Init
Input Parameters	None
Output Parameters	None
Input/Out Parameters	None
Fun. Description	Initializes the ECU LED module.

Function Name	SetLED_ON
Input Parameters	LEDId Description: Numeric ID of LED Channel
Output Parameters	None
Input/Out Parameters	None
Fun. Description	Writes on LED Channel to set it on.

Function Name	SetLED_OFF
Input Parameters	LEDId Description: Numeric ID of LED Channel
Output Parameters	None
Input/Out Parameters	None
Fun. Description	Writes on LED Channel to set it off.

Function Name	LED_Toggle
Input Parameters	LEDId Description: Numeric ID of LED Channel
Output Parameters	None
Input/Out Parameters	None
Fun. Description	Writes on LED Channel to toggle it.

State Machine Software Component:

Function Name	App_SM_Init
Input Parameters	None
Output Parameters	None
Input/Out Parameters	None
Fun. Description	Initializes the ECU and color states.

API Configuration Parameters:

Parameter	Description
state	Global variable for ECU state declaration
colorstate	Global variable for ECU LED color state declaration
waiting	defined as 0 to specify ECU state name.
definition	defined as 1 to specify ECU state name.
checking	defined as 2 to specify ECU state name.
red_execution	defined as 3 to specify ECU state name.
green_execution	defined as 4 to specify ECU state name.
blue_execution	defined as 5 to specify ECU state name.
blocked	defined as 6 to specify ECU state name.
RED_LED	defined as 8 to specify ECU LED color state name.
GREEN_LED	defined as 9 to specify ECU LED color state name.
BLUE_LED	defined as 10 to specify ECU LED color state name.
Red	defined as 0 to specify red channel ID.
Green	defined as 1 to specify green channel ID.
Blue	defined as 2 to specify blue channel ID.

Function Name	App_SM
Input Parameters	None
Output Parameters	None
Input/Out Parameters	None
Fun. Description	Defines the states of ECU and how it operates.

-NO compilation warnings exist in code.

4 External Interface Requirements

4.1 User Interfaces

- C Compiler version

4.2 Hardware Interfaces

- Windows
- IDE that supports TI's microcontroller (MCU) and embedded processor portfolios.
- Arm Cortex M4 Tiva C board

4.3 Software Interfaces

Following are the software used for the project.

Software used	Description
Operating System	We have chosen Windows operating system for its best support and user-friendliness.
CCS (Code Composer Studio)	To develop and debug application, we have chosen CCS as an IDE.
C	To implement the project we have chosen C programming language.

Here is a link that includes all files and videos needed:

https://drive.google.com/drive/folders/1E6Y4hTL3iuqxWG_h9gC7pdlWVb_81kz3?usp=sharing