FWD_Advanced Embedded Diploma

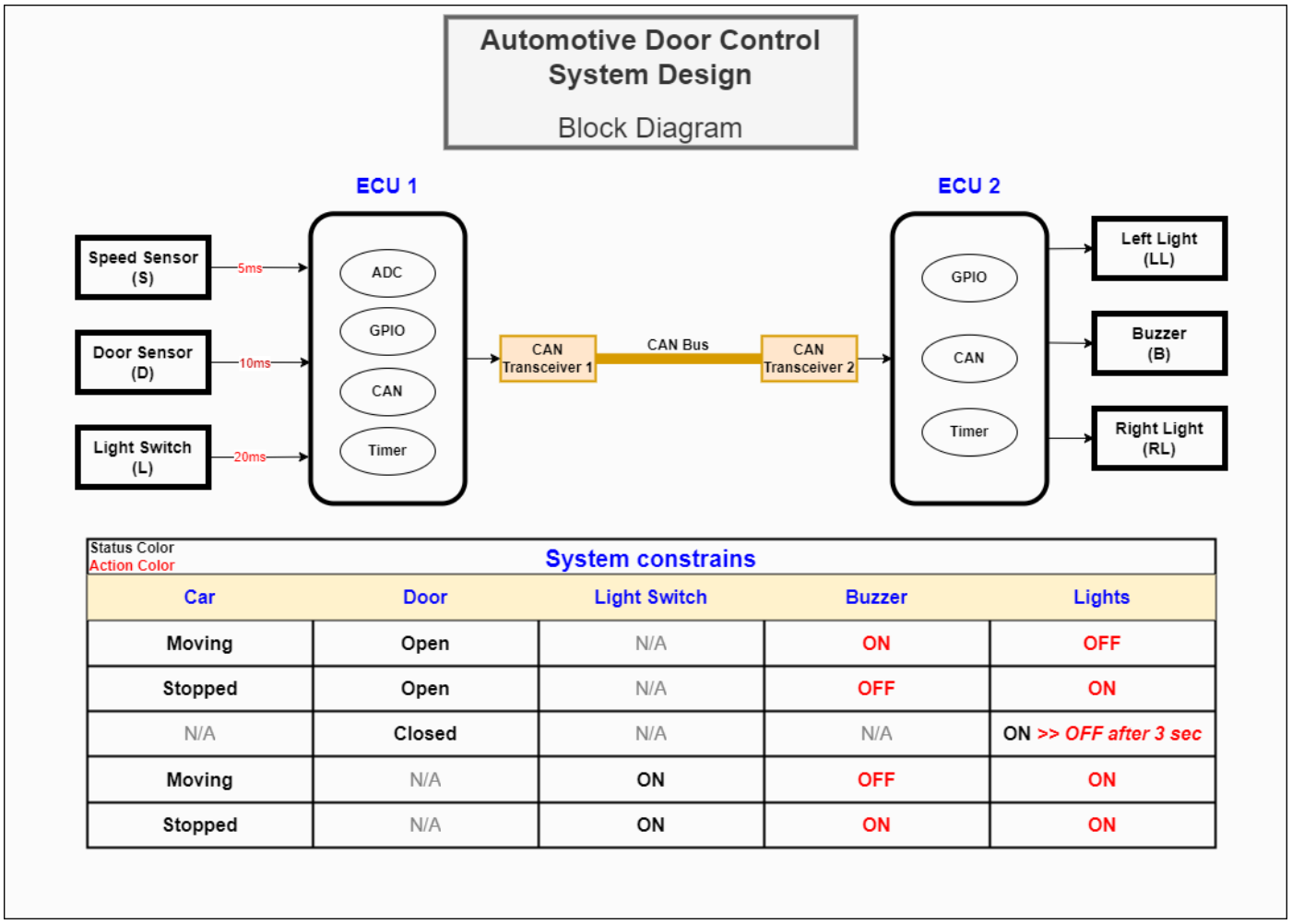
Automotive Door Control System Design

By Mohamed Samy

Table Content

1. Block Diagram (Req. 1)

System block diagram that show all system blocks and design constraints



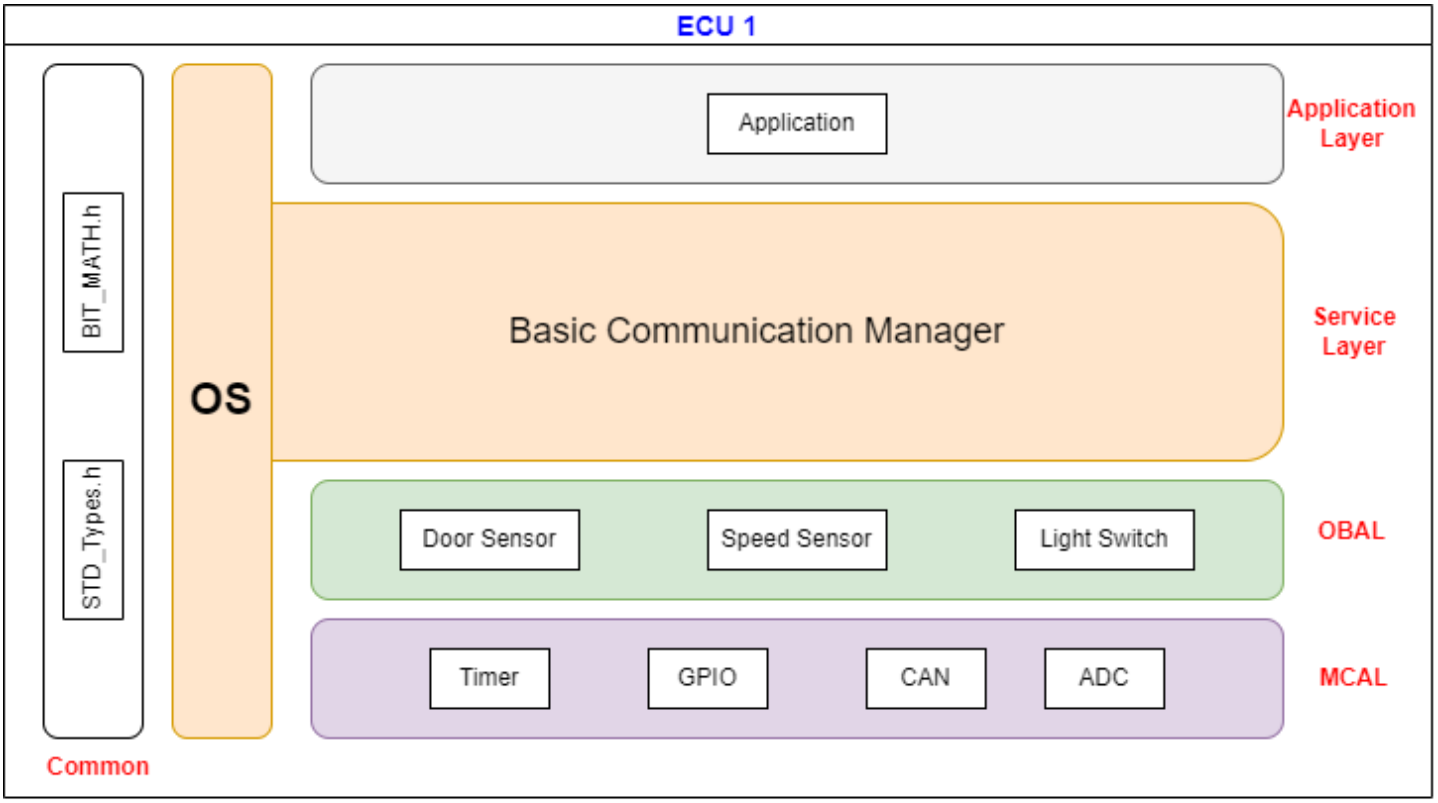
Status Color
Action Color

System constrains

Car	Door	Light Switch	Buzzer	Lights
Moving	Open	N/A	ON	OFF
Stopped	Open	N/A	OFF	ON
N/A	Closed	N/A	N/A	ON *>> OFF after 3 sec*
Moving	N/A	ON	OFF	ON
Stopped	N/A	ON	ON	ON

2. Static Design (Req. 2)

2.1. Layered Architecture (ECU1)



2.1.1. On Board Components:

- Door Sensor (Digital)
- Speed Sensor (Analog)
- Light Switch (Digital)

2.1.2. Modules:

- Timer Module
- GPIO Module
- CAN Module
- ADC Module
- Door Module
- Speed Module
- Switch Module
- Application Module

2.1.3. Folder structure:

Application Folder	`main.c`	
	`app.h`	
	`app.c`	
Service Folder	`OS_config.h`	
	`OS_interface.h`	
	`OS.c`	
OnBoard Folder	DoorSensor Folder	`DoorSensor_confg.h`
		`DoorSensor_interface.h`
		`DoorSensor.c`
	SpeedSensor Folder	`SpeedSensor_confg.h`
		`SpeedSensor_interface.h`
		`SpeedSensor.c`
	LightSwitch Folder	`LightSwitch _confg.h`
		`LightSwitch _interface.h`
		`LightSwitch.c`
MCAL Folder	Timer	`Timer_confg.h`
		`Timer_interface.h`
		`Timer.c`
	GPIO	`GPIO_confg.h`
		`GPIO_interface.h`
		`GPIO.c`
	CAN	`CAN_confg.h`
		`CAN_interface.h`
		`CAN.c`
	ADC	`ADC_confg.h`
		`ADC_interface.h`
		`ADC.c`
Common	`STD_Types.h`	
	`Bit_Math.h`	

2.1.4. Typedefs and Enmus used:

DataType	***Description***
Enum RetState_t	Used to return all function execution status
struct TimerConfig_st	Used to hold timer configurations
struct GPIOConfig_st	Used to hold GPIO configurations
Enum GPIO_Pin_t	GPIO pins names
Enum GPIO_Port_t	GPIO ports names
Enum GPIO_PinState_t	GPIO digital read states
struct CANConfig_st	Used to hold CAN configurations
Enum CAN_NodeID_t	CAN bus nodes available to be selected
Enum CAN_MsgSize_t	Data size available to be selected for CAN message
struct ADCConfig_st	Used to hold ADC configurations
Enum ADC_Channel_t	Available channels of ADC
Enum Doors_t	Car doors to be selected later (4 doors, 2 doors)
struct Door_st {Doors_t, GPIOConfig_st}	Used as object of door that holds all its data
Enum SpeedSensors_t	Car speed sensors to be selected
struct SpeedSensor_st{SpeedSensors_t, ADCConfig_st}	Used as object of speed sensors that holds all its data
Enum Switchs_t	Car switches to be selected
struct Switch_st{Switchs_t, GPIOConfig_st}	Used as object of switches that holds all its data
EventFlag_ECU1	Event flag group that stores sensors values ***(OS feature)***

2.1.5. APIs and Private functions:

Timer Module	
APIs	
`RetState_t Timer_init(void)`	
Layer	MCAL
Input parameters	(void) Utilise configuration array of the module
Return value	RetState_t
Description	Used to initiate timer peripherals with the required setting stated by user in the configuration array
`RetState_t Timer_start(void)`	
Layer	MCAL
Input parameters	(void)
Return value	RetState_t
Description	Used to start timer counting and trigger an interrupt for each tick in the timer.
`RetState_t callBack_register(<parameter>)`	
Layer	MCAL
Input parameters	(*void ptrToCBFunction)
Return value	RetState_t
Description	Used by upper layers to set the ISR action of timer interrupt.

GPIO Module	
APIs	
`RetState_t GPIO_init(void)`	
Layer	MCAL
Input parameters	(void) Utilise configuration array of the module
Return value	RetState_t
Description	Used to initiate GPIO pins according to GPIO

	configuration array
RetState_t GPIO_ConfigPin(<parameters>)	
Layer	MCAL
Input parameters	(GPIOConfig_st Config)
Return value	RetState_t
Description	Used to configure specific pin not mentioned in the configuration array
RetState_t GPIO_GetPin(<parameters>)	
Layer	MCAL
Input parameters	(GPIO_Pin_t Pinx, GPIO_Port_t Portx GPIO_PinState_t* PinReading)
Return value	RetState_t
Description	Used to get specific GPIO pin
RetState_t GPIO_SetPin(<parameters>)	
Layer	MCAL
Input parameters	(GPIO_Pin_t Pinx, GPIO_Port_t Portx)
Return value	RetState_t
Description	Used to set specific GPIO pin

CAN Module	
APIs	
RetState_t CAN_init(void)	
Layer	MCAL
Input parameters	(void) Utilise configuration array of the module
Return value	RetState_t
Description	Used to initiate CAN peripheral according to CAN configuration array
RetState_t CAN_SendMessage(<parameters>)	

Layer	MCAL
Input parameters	(CAN_NodeID_t NodeID, uint8_t* MsgBuffer, CAN_MsgSize_t MsgSize)
Return value	RetState_t
Description	Used to send specific message to specific node on the CAN bus

ADC Module	
APIs	
RetState_t ADC_init(void)	
Layer	MCAL
Input parameters	(void) Utilise configuration array of the module
Return value	RetState_t
Description	Used to initiate ADC peripheral according to ADC configuration array
RetState_t ADC_ConfigChannel(<parameters>)	
Layer	MCAL
Input parameters	(ADCConfig_st)
Return value	RetState_t
Description	Used to initiate ADC peripheral according to ADC configuration array
RetState_t ADC_GetAnalogRead(<parameters>)	
Layer	MCAL
Input parameters	(ADC_Channel_t Channel, float* SensorAnalogRead)
Return value	RetState_t
Description	Used to get the analog read of a sensor connected to a specific channel pre-defined by the user. Value is evaluated inside this API using ADC_GetDigtalRead private function to get current digital reading then map it according to the connected sensor configurations in the module.

Private Functions	
RetState_t ADC_GetDigtalRead(<parameters>)	
Layer	MCAL
Input parameters	(ADC_Channel_t Channel, uint16_t* DigReading)
Return value	RetState_t
Description	Used internally in the ADC_GetAnalogRead() to evaluate the current digital value.

Door Module	
APIs	
RetState_t DoorSensor_init(<parameters>)	
Layer	OBAL
Input parameters	(Door_st* SelectedDoorPtr)
Return value	RetState_t
Description	Used to initiate door sensor pin according to input structure stores pin used with the sensor and its configurations
RetState_t DoorSensor_GetRead(<parameters>)	
Layer	OBAL
Input parameters	(Door_st* SelectedDoorPtr, uint_t* DoorSensorReading)
Return value	RetState_t
Description	Used to get specific GPIO pin

Speed Module	
APIs	
RetState_t SpeedSensor_init(<parameters>)	
Layer	OBAL
Input parameters	(SpeedSensor_st* SelectedSensorPtr)
Return value	RetState_t

Description	Used to initiate a speed sensor according to the sensor object (structure) configurations.
RetState_t SpeedSensor_GetRead(<parameters>)	
Layer	OBAL
Input parameters	(SpeedSensor_st* SelectedSensorPtr, float* DoorSensorReading)
Return value	RetState_t
Description	Used to get specific sensor current speed

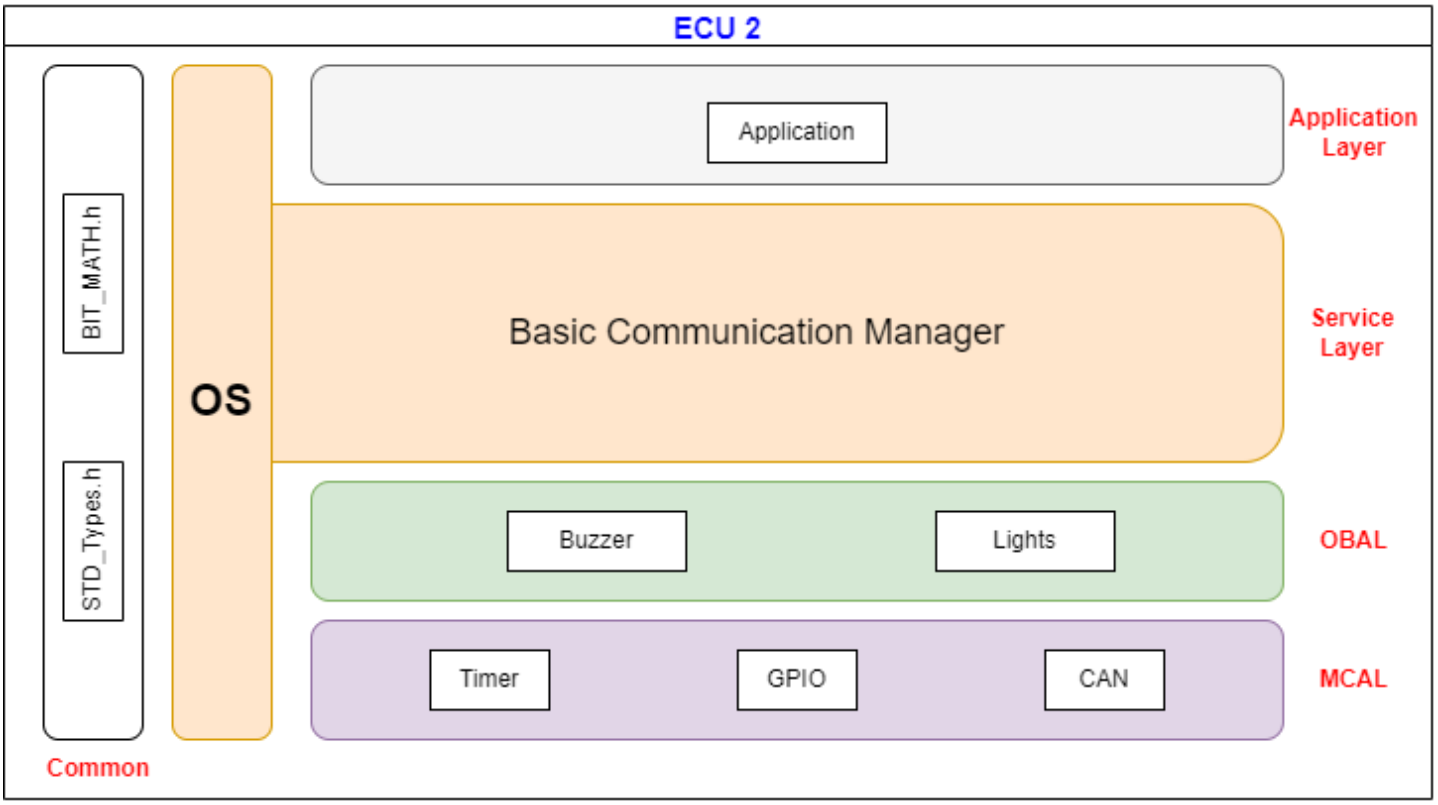
Switch Module	
APIs	
RetState_t Switch_init(<parameters>)	
Layer	OBAL
Input parameters	(Switch_st* Switch)
Return value	RetState_t
Description	Used to initiate a switch.
RetState_t Switch_GetRead(<parameters>)	
Layer	OBAL
Input parameters	(Switch_st* Switch, uint8_t* DoorSensorReading)
Return value	RetState_t
Description	Used to get current reading of the switch button.

BCM Manager Module	
APIs	
RetState_t BCM_Manager(void)	
Layer	OS Layer
Input parameters	(void) Utilise event flag as input
Return value	RetState_t
Description	Send Fetched sensors reading from EventFlag_ECU1

	and send them over communication bus

Application Module	
APIs	
RetState_t HW_init(void)	
Layer	Application Layer
Input parameters	(void)
Return value	RetState_t
Description	Initiate system setting according to lower layers configurations.
RetState_t SendSpeed(void)	
Layer	Application Layer
Input parameters	(void)
Return value	RetState_t
Description	Send car speed every 5ms utilising timer tick
RetState_t SendDoorState(void)	
Layer	Application Layer
Input parameters	(void)
Return value	RetState_t
Description	Send door state every 10ms utilising timer tick
RetState_t SendSwitchState(void)	
Layer	Application Layer
Input parameters	(void)
Return value	RetState_t
Description	Send Switch state every 20ms utilising timer tick

2.2. Layered Architecture (ECU2)



2.2.1. On Board Components:

- Buzzer (Digital)
- Lights (Digital)

2.2.2. Modules:

- Timer Module
- GPIO Module
- CAN Module
- Buzzer Module
- Lights Module
- Application Module

2.2.3. Folder structure:

Application Folder	`main.c`	
	`app.h`	
	`app.c`	
Service Folder	`OS_config.h`	
	`OS_interface.h`	
	`OS.c`	
OnBoard Folder	Buzzer Folder	`Buzzer_confg.h`
		`Buzzer_interface.h`
		`Buzzer.c`
	Light Folder	`Light_confg.h`
		`Light_interface.h`
		`Light.c`
MCAL Folder	Timer	`Timer_confg.h`
		`Timer_interface.h`
		`Timer.c`
	GPIO	`GPIO_confg.h`
		`GPIO_interface.h`
		`GPIO.c`
	ADC	`ADC_confg.h`
		`ADC_interface.h`
		`ADC.c`
Common	`STD_Types.h`	
	`Bit_Math.h`	

2.2.4. Typedefs and Enmus used:

DataType	***Description***
Enum RetState_t	Used to return all function execution status
struct TimerConfig_st	Used to hold timer configurations
struct GPIOConfig_st	Used to hold GPIO configurations
Enum GPIO_Pin_t	GPIO pins names
Enum GPIO_Port_t	GPIO ports names
Enum GPIO_PinState_t	GPIO digital read states
struct CANConfig_st	Used to hold CAN configurations
Enum CAN_MsgSize_t	Data size available to be selected for CAN message
Enum Buzzer_t	Enum used to select which buzzer to operate
struct Buzzer_st {Buzzer_t, GPIOConfig_st}	Used as object of Buzzer that holds all its data
Enum Light_t	Enum used to select which Light to operate
struct Light_st{Light_t, GPIOConfig_st}	Used as object of Light that holds all its data
Enum App_Device_t	Enum used to define application used devices (Buzzer , LeftLight, Right Light)
EventFlag_ECU2	Event flag group that stores sensors values ***(OS feature)***

2.2.5. APIs and Private functions:

Timer Module	
APIs	
`RetState_t Timer_init(void)`	
Layer	MCAL
Input parameters	(void) Utilise configuration array of the module
Return value	RetState_t
Description	Used to initiate timer peripherals with the required setting stated by user in the configuration array
`RetState_t Timer_start(void)`	
Layer	MCAL
Input parameters	(void)
Return value	RetState_t
Description	Used to start timer counting and trigger an interrupt for each tick in the timer.
`RetState_t callBack_register(<parameter>)`	
Layer	MCAL
Input parameters	(*void ptrToCBFunction)
Return value	RetState_t
Description	Used by upper layers to set the ISR action of timer interrupt.

GPIO Module	
APIs	
`RetState_t GPIO_init(void)`	
Layer	MCAL
Input parameters	(void) Utilise configuration array of the module
Return value	RetState_t
Description	Used to initiate GPIO pins according to GPIO

	configuration array
RetState_t GPIO_ConfigPin(<parameters>)	
Layer	MCAL
Input parameters	(GPIOConfig_st Config)
Return value	RetState_t
Description	Used to configure specific pin not mentioned in the configuration array
RetState_t GPIO_GetPin(<parameters>)	
Layer	MCAL
Input parameters	(GPIO_Pin_t Pinx, GPIO_Port_t Portx GPIO_PinState_t* PinReading)
Return value	RetState_t
Description	Used to get specific GPIO pin
RetState_t GPIO_SetPin(<parameters>)	
Layer	MCAL
Input parameters	(GPIO_Pin_t Pinx, GPIO_Port_t Portx)
Return value	RetState_t
Description	Used to set specific GPIO pin

CAN Module	
APIs	
RetState_t CAN_init(void)	
Layer	MCAL
Input parameters	(void) Utilise configuration array of the module
Return value	RetState_t
Description	Used to initiate CAN peripheral according to CAN configuration array
RetState_t CAN_ReadBus(<parameters>)	

Layer	MCAL
Input parameters	(uint8_t* MsgBuffer, CAN_MsgSize_t MsgSize)
Return value	RetState_t
Description	Used to read CAN bus (Transceiver buffer).

Buzzer Module	
APIs	
RetState_t Buzzer_init(<parameters>)	
Layer	OBAL
Input parameters	(Buzzer_st* SelectedBuzzerPtr)
Return value	RetState_t
Description	Used to initiate Buzzer pin according to input structure.
RetState_t Buzzer_Set(<parameters>)	
Layer	OBAL
Input parameters	(Buzzer_st* SelectedBuzzerPtr, GPIO_PinState_t BuzzerState)
Return value	RetState_t
Description	Used to turn buzzer on/off

Light Module	
APIs	
RetState_t Light_init(<parameters>)	
Layer	OBAL
Input parameters	(Light_st* SelectedSensorPtr)
Return value	RetState_t
Description	Used to initiate light pins according to input structure.
RetState_t Light_Set(<parameters>)	

Layer	OBAL
Input parameters	(Light_st* SelectedLightPtr, GPIO_PinState_t LightState)
Return value	RetState_t
Description	Used to turn Lights on/off

BCM Manager Module	
APIs	
RetState_t BCM_Manager(void)	
Layer	OS Layer
Input parameters	(void) Utilise event flag as input
Return value	RetState_t
Description	Read sensors states over communication bus and store them inside EventFlag_ECU2

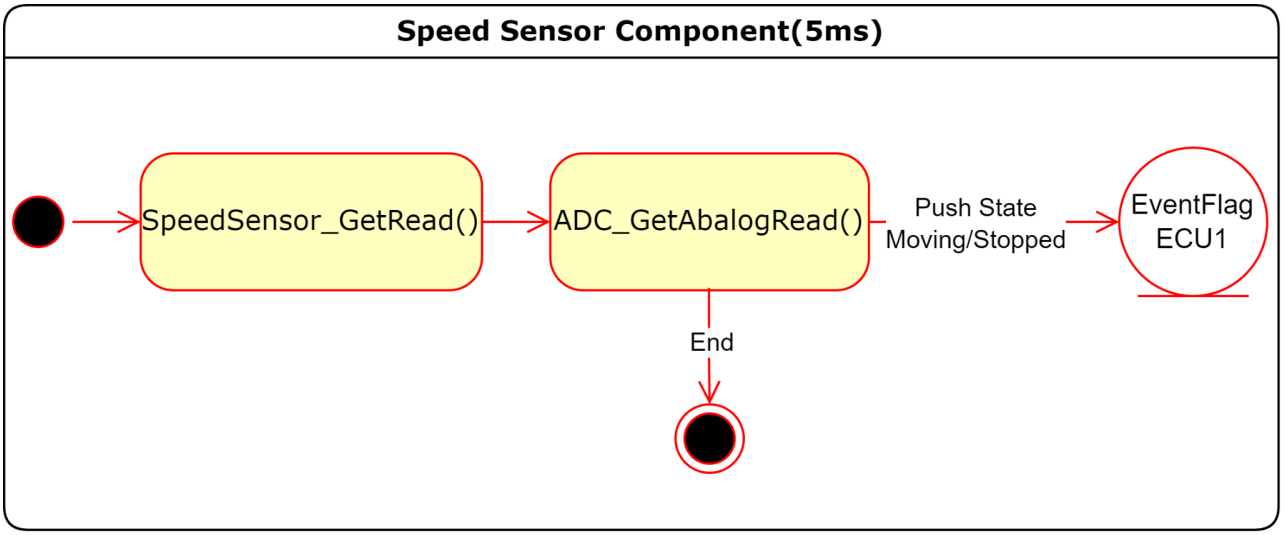
Application Module	
APIs	
RetState_t HW_init(void)	
Layer	Application Layer
Input parameters	(void)
Return value	RetState_t
Description	Initiate system setting according to lower layers configurations.
RetState_t OperateDigDev(<parameters>)	
Layer	Application Layer
Input parameters	(App_Device_t SelectedDevice, GPIO_PinState_t State_t)
Return value	RetState_t
Description	Implement logic to operate devices according to system constraints. Used to operate a specific device (Light, Buzzer. Operation based on data pushed from

	GetSensorsValues.

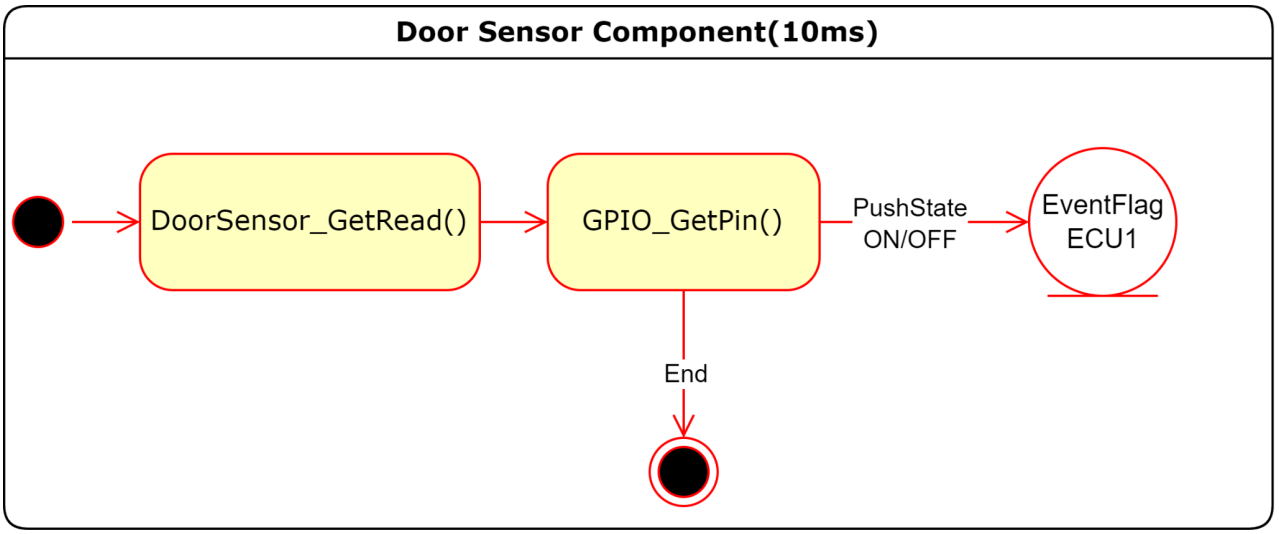
3. Dynamic Design (Req. 3)

3.1. On Board Components State Machine (ECU1)

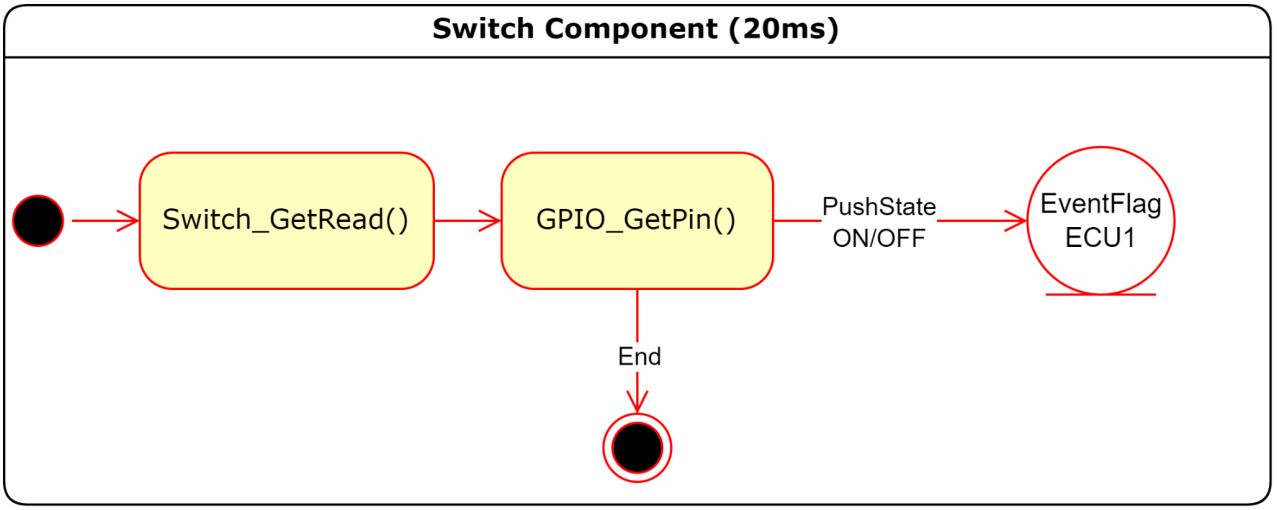
3.1.1. Speed Sensor Component:



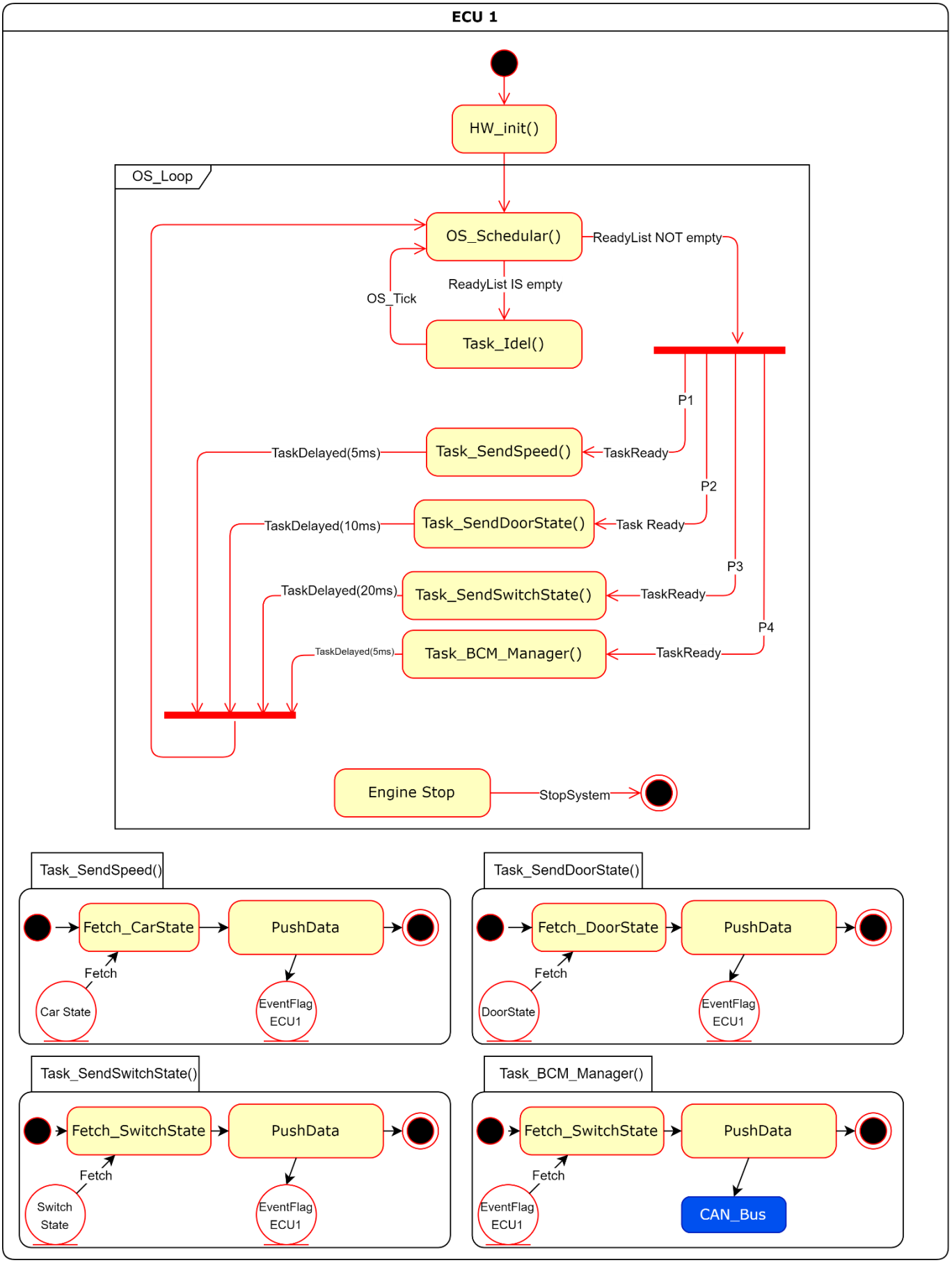
3.1.2. Door Sensor Component:



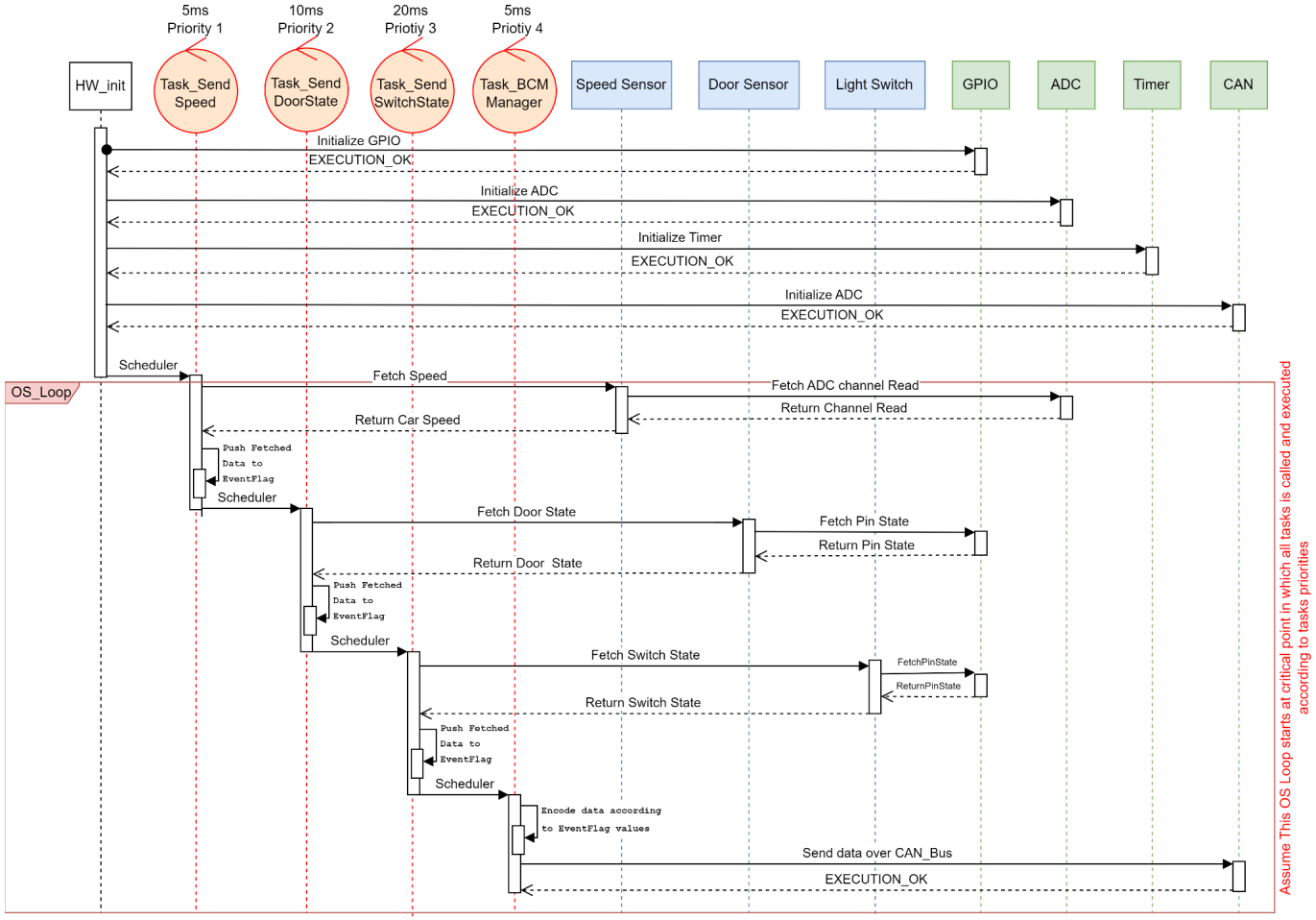
3.1.3. Switch Component:



3.2. ECU State Machine (ECU1):



3.3. ECU Sequence Diagram (ECU1):



3.4. CPU Load (ECU1):

Assume:-

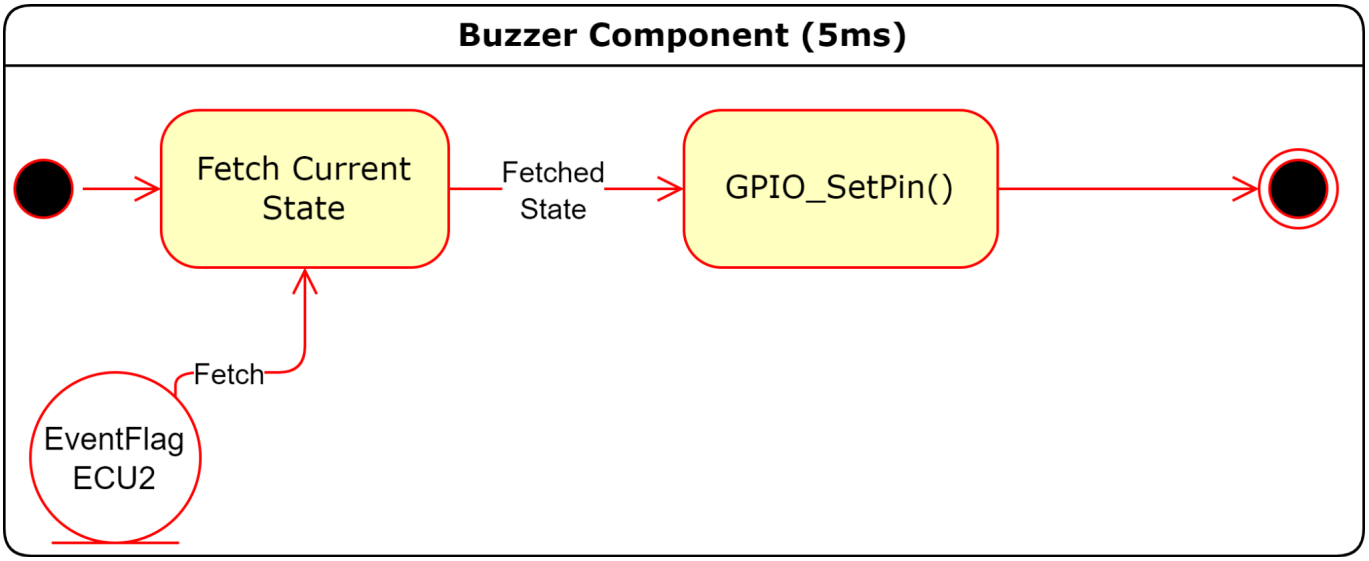
- $TaskSendSpeed_{exe\ time} = 1\ ms$
- $TaskSendDoorState_{exe\ time} = 0.5\ ms$
- $TaskSwitchState_{exe\ time} = 0.5\ ms$
- $TaskBCMManger_{exe\ time} = 1\ ms$

$$HyperPeriod = 20\ ms$$

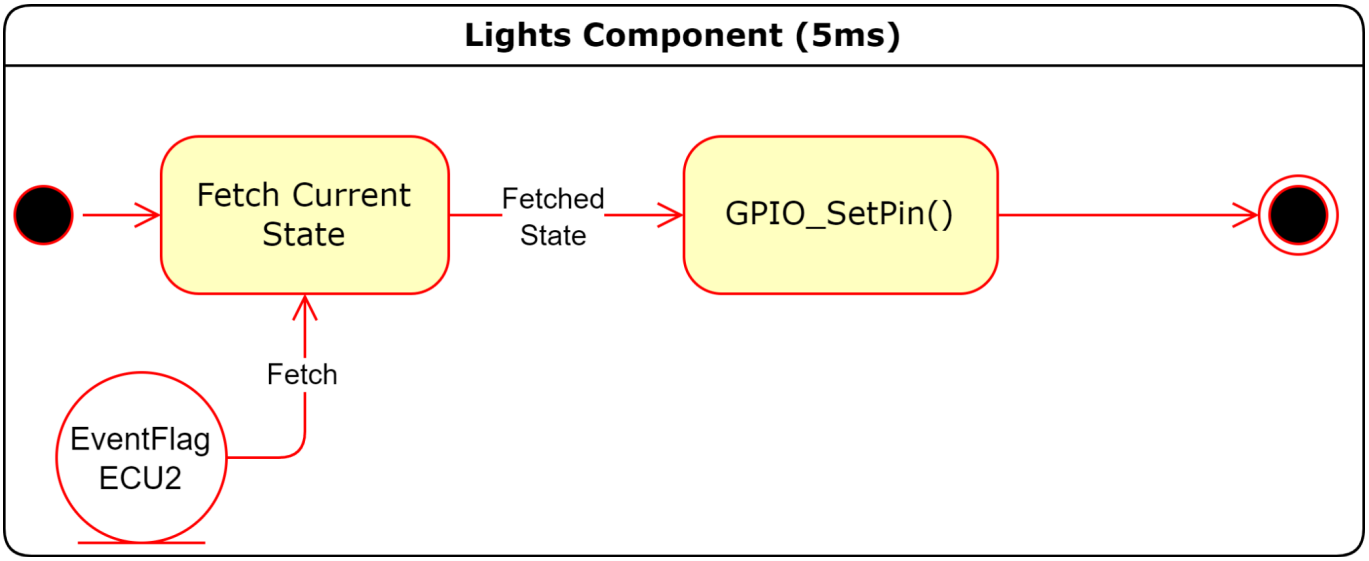
$$CPU_{Load} = \frac{Tasks\ execution\ time}{HyperPeriod} = \frac{(\frac{20}{5}\times 1)+(\frac{20}{10}\times 0.5)+(\frac{20}{20}\times 0.5)+(\frac{20}{5}\times 1)}{20} = 47.5\%$$

3.5. On Board Components State Machine (ECU2)

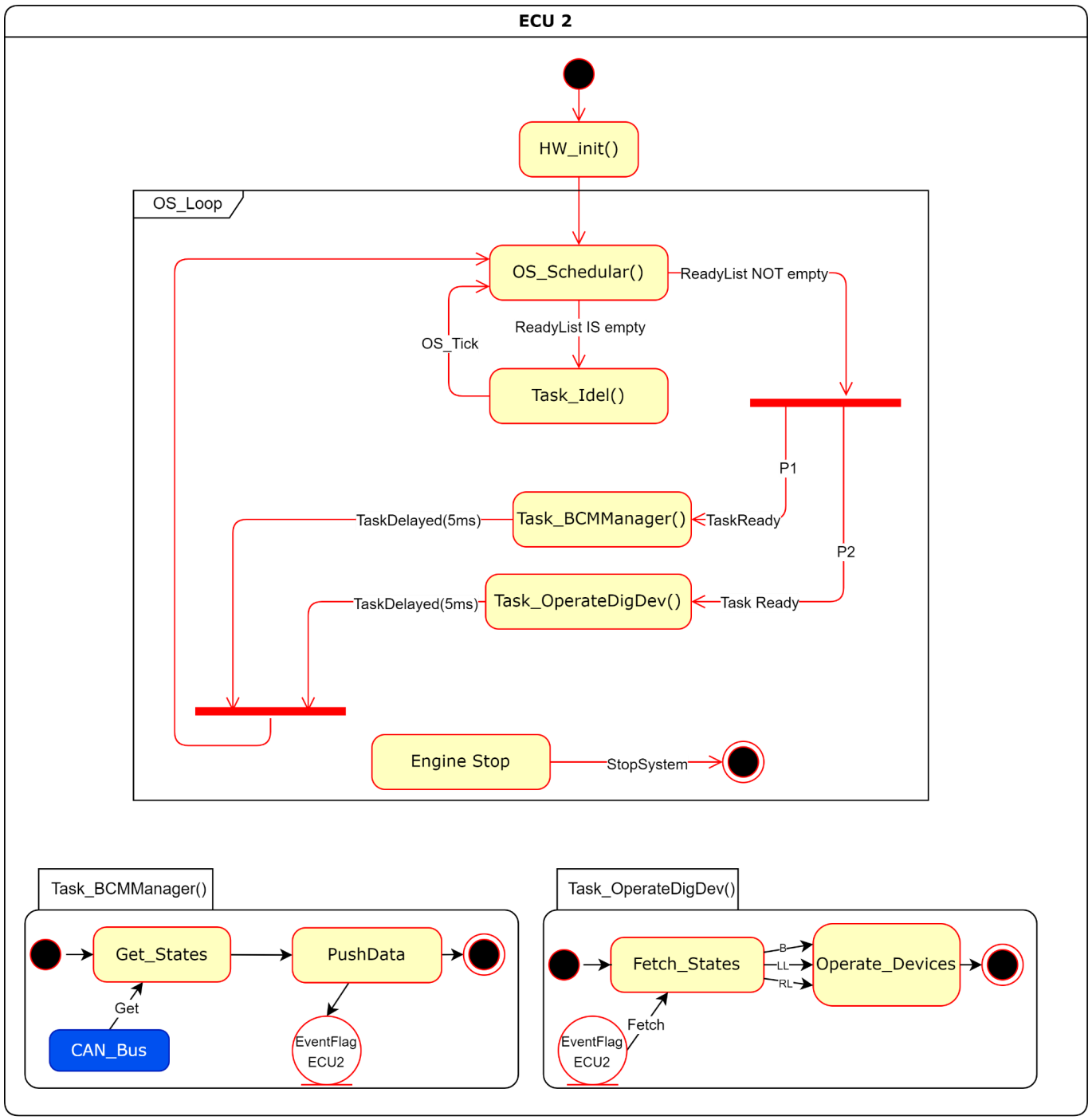
3.5.1. Buzzer Component:



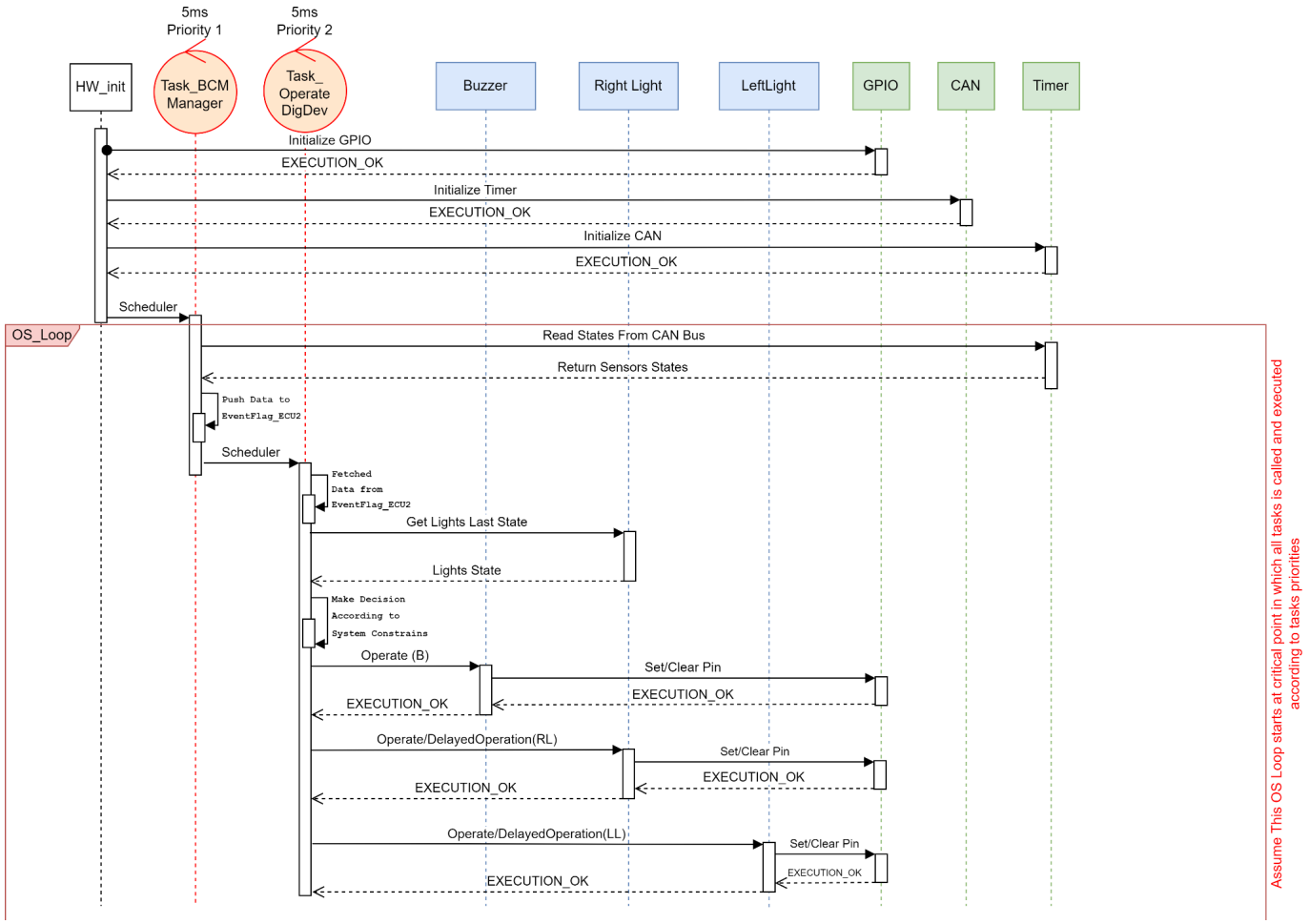
3.5.2. Lights Component:



3.6. ECU State Machine (ECU2):



3.7. ECU Sequence Diagram (ECU2):



3.8. CPU Load (ECU2):

Assume:-

- $TaskBCMManager_{exe\ time} = 1\ ms$
- $TaskOperateDigDev_{exe\ time} = 0.9\ ms$

$$HyperPeriod = 5\ ms$$

$$CPU_{Load} = \frac{Tasks\ execution\ time}{HyperPeriod} = \frac{(\frac{5}{5}\times 1)+(\frac{5}{5}\times 0.9)}{5} = 38\%$$

4. CAN Bus Load (Req. 3)

- ***CAN Bus Configurations:***

Assume using CAN_LowSpeed at rate of (125 kbit/second)
Assume each sensor state is encoded in 1-bit

Car Moving State	`0 -> Stopped`	`1 -> Moving`
Door Sensor State	`0 -> Closed`	`1 -> Opened`
Light Switch Sensor State	`0 -> OFF`	`1 -> ON`

So we have 3-bits of data ***rounded up to 1-Byte***

- ***CAN Frame size calculations:***

$$Calculation\ of\ bits\ in\ standard\ CAN\ frame\ =$$
$$1(SOF)\ +\ 11(Identifier)\ +\ 7(Control)\ +\ 8(Data)\ +\ 15(CRC)\ +\ 2\,(ACK)\ +\ 7(EOF)$$
$$=\ 51\ bit/frame$$

- ***CAN Frame Speed calculations:***

$$125000\ bit\ \rightarrow\ 1\ second$$
$$51\quad bit\ \rightarrow\ x\ second$$
$$x\ =\ \frac{51}{125000}\ =\ 0.408\ ms$$

So it takes 0.408 ms to send one frame

- ***CAN Bus Load calculations:***

Knowing that (BCM_Manager) in ECU1 is sending one frame periodically each 5ms

$$frames_{perSecond}\ =\ 1000ms\ /\ 5ms\ =\ 200\ frame/sec$$
$$CAN.Bus_{BusyTime}\ =\ 200\ *\ 0.408\ ms\ =\ 81.6ms$$
$$CAN.Bus_{Load}\ =\ 81.6/1000\ =\ 8.16\ \%$$

Answer = 8.16 %