# FWD\_Advanced Embedded Diploma

**Automotive Door Control System Design**

**By Mohamed Samy**

**Table Content**

[**1. Block Diagram (Req. 1) 1**](#_ui0bxbsu420s)

[**2. Static Design (Req. 2) 2**](#_c41b8bk7qvpt)

[2.1. Layered Architecture (ECU1) 2](#_bh8ace736ov5)

[2.1.1. On Board Components: 2](#_9romo9pxmnpt)

[2.1.2. Modules: 2](#_soo2jbljxm8l)

[2.1.3. Folder structure: 3](#_8y2rypo8beda)

[2.1.4. Typedefs and Enmus used: 4](#_knm5t4i76l7i)

[2.1.5. APIs and Private functions: 5](#_porjut52relh)

[2.2. Layered Architecture (ECU2) 11](#_abyzltmo26gw)

[2.2.1. On Board Components: 11](#_nuwcdr5qjzzh)

[2.2.2. Modules: 11](#_am3o6bneya69)

[2.2.3. Folder structure: 12](#_d4tfk1rotd2o)

[2.2.4. Typedefs and Enmus used: 13](#_jx3qmem2s9ro)

[2.2.5. APIs and Private functions: 14](#_xn12q75rch3c)

[**3. Dynamic Design (Req. 3) 19**](#_uv9n7d2xb34w)

[3.1. On Board Components State Machine (ECU1) 19](#_sf27ukyktukz)

[3.2. ECU State Machine (ECU1): 20](#_cm0l84hu2hbc)

[3.3. ECU Sequence Diagram (ECU1): 21](#_xcqxvyxaehju)

[3.4. CPU Load (ECU1): 21](#_z6w2xkam21q4)

[3.5. On Board Components State Machine (ECU2) 22](#_1arsohs5scwr)

[3.6. ECU State Machine (ECU2): 23](#_om1fgtexerso)

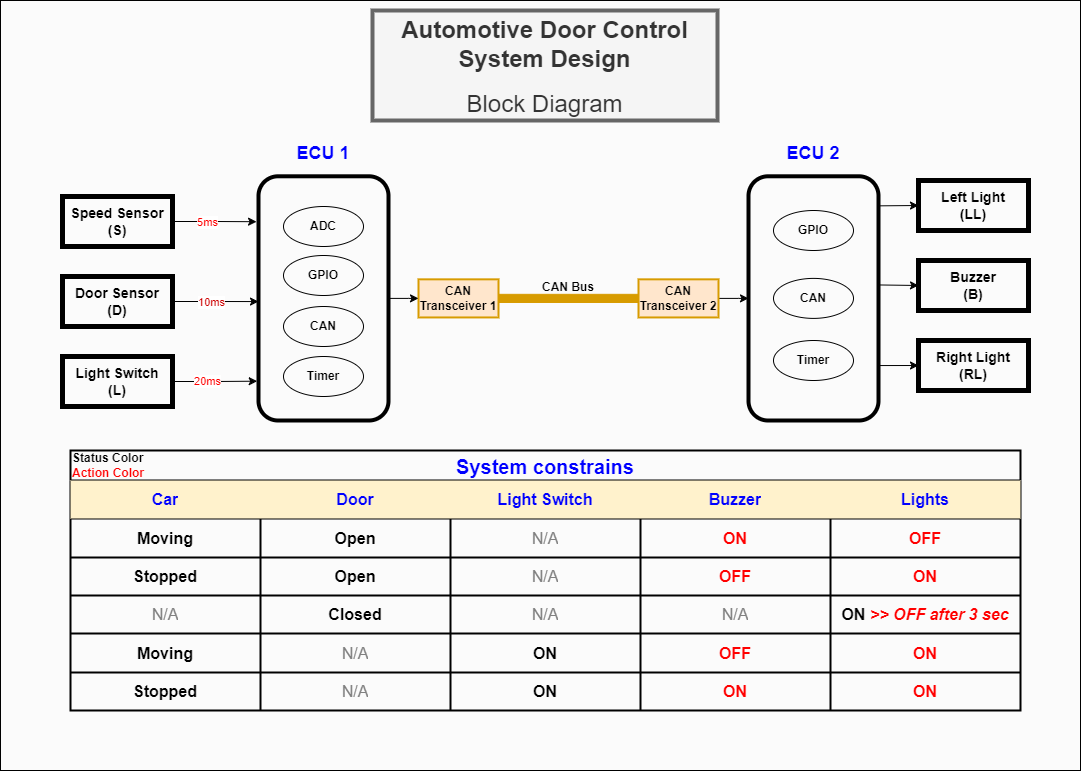
[3.7. ECU Sequence Diagram (ECU2): 24](#_dmfgw7ocejya)

[3.8. CPU Load (ECU2): 24](#_w9r6xmpb4kpj)

[**4. CAN Bus Load (Req. 3) 25**](#_nhdwc7p9e9j6)

# Block Diagram (Req. 1)

System block diagram that show all system blocks and design constraints



# Static Design (Req. 2)

## Layered Architecture (ECU1)ECU1 Layered Architecture

### On Board Components:

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

### Modules:

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

### 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** | |

### 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)*** |

### 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 |

## **Layered Architecture (ECU2)**

### On Board Components:ECU1 Layered Architecture

* Buzzer (Digital)
* Lights (Digital)

### Modules:

* Timer Module
* GPIO Module
* CAN Module
* Buzzer Module
* Lights Module
* Application Module

### 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** | |

### 

### 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)*** |

### 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 |

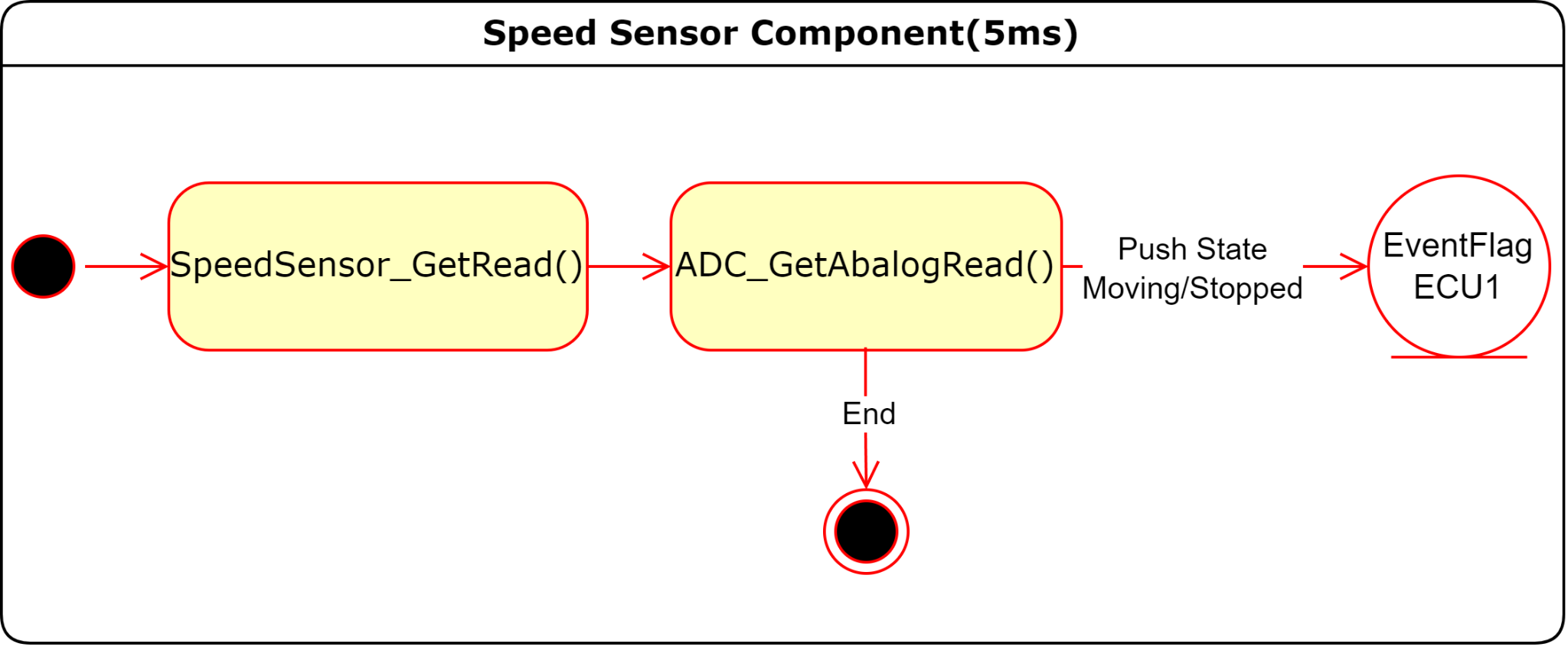
| **BCM Manager Module** | |
| --- | --- |
| APIs | |
| RetState\_t BCM\_Manager(void) | |
| Layer | OS Layer |
| Input parameters | (void) Utilise event flag as input |
| Return value | RetState\_t |
| Description | Read sensors states over communication bus and store them inside EventFlag\_ECU2 |

| **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. |

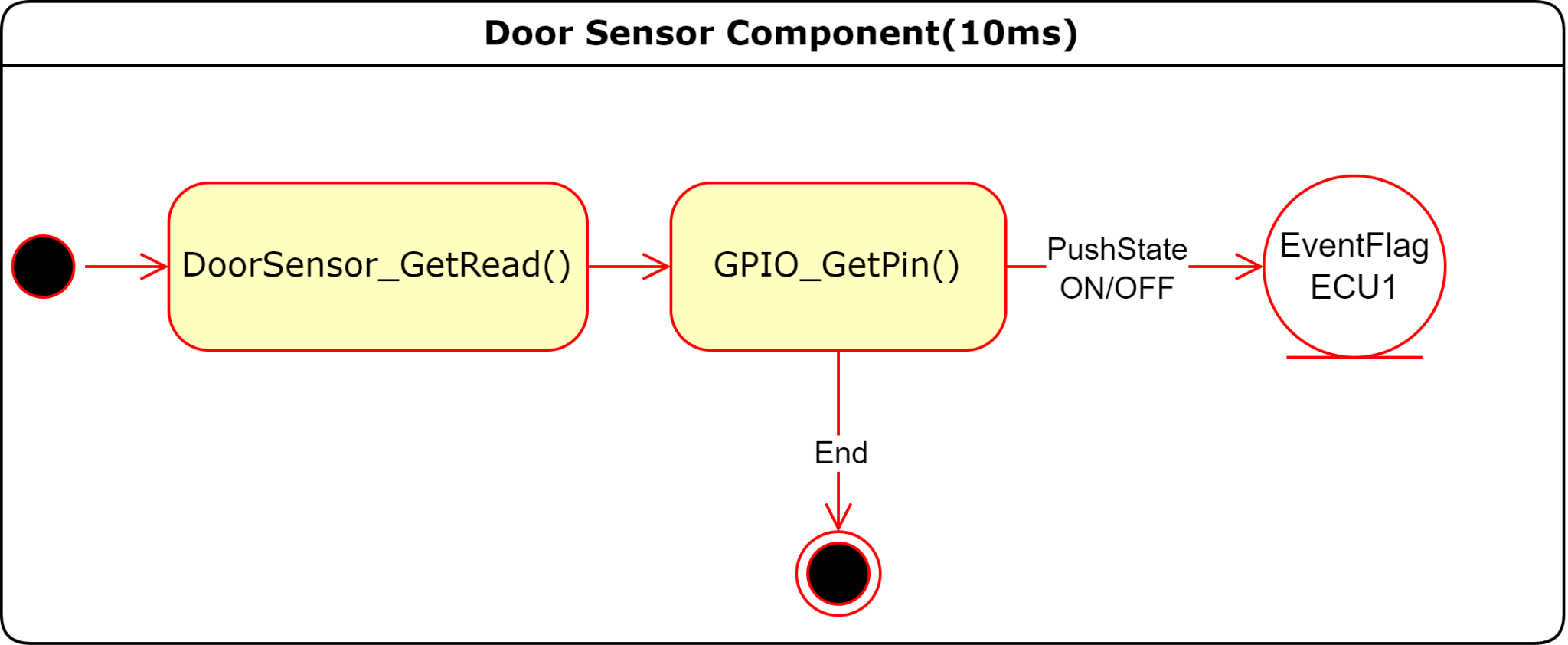
# Dynamic Design (Req. 3)

## On Board Components State Machine (ECU1)

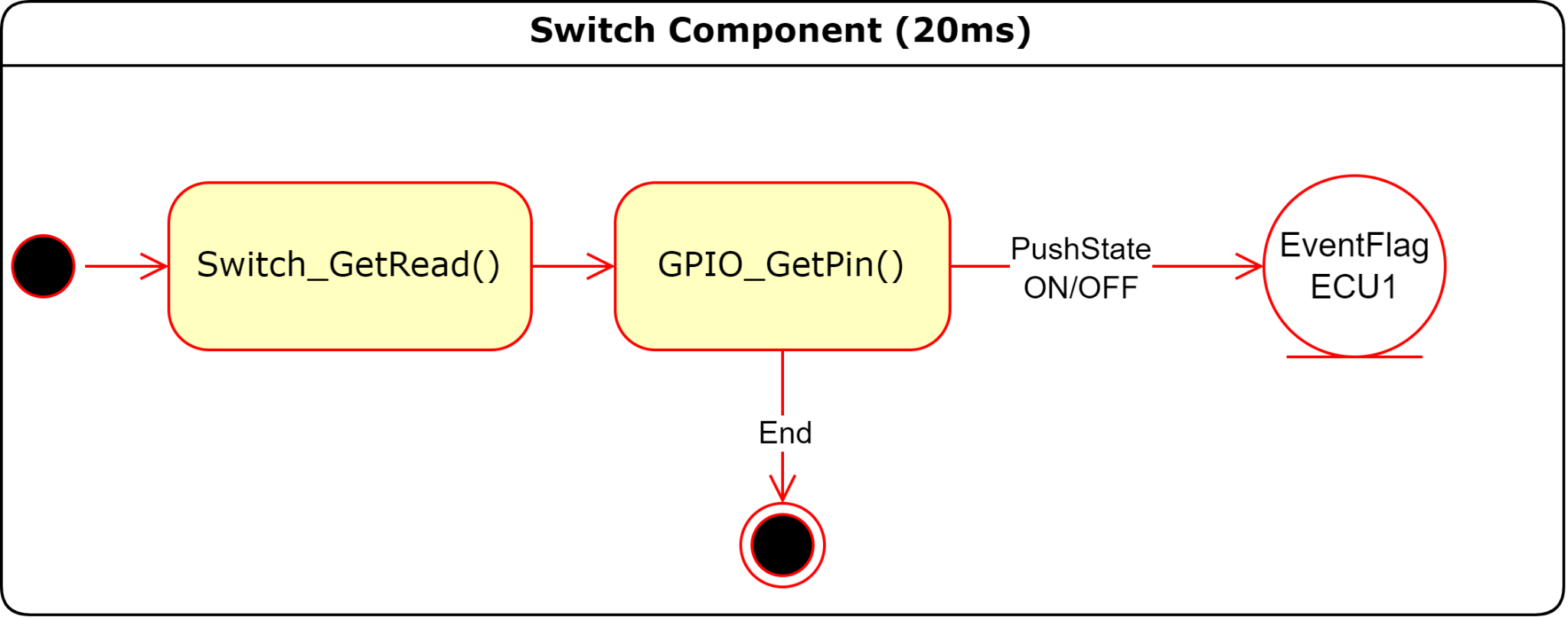
* + 1. Speed Sensor Component:



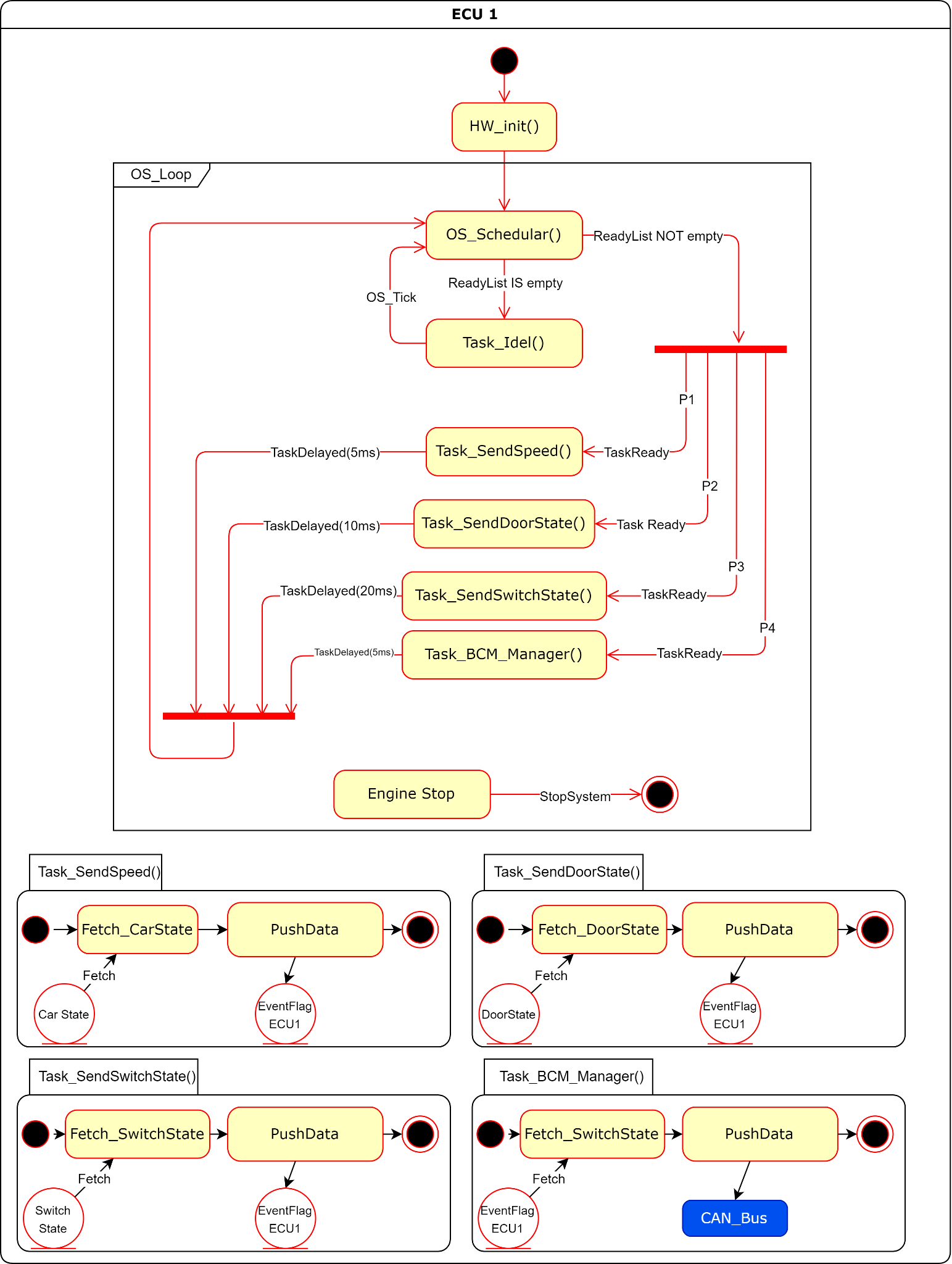
* + 1. Door Sensor Component:



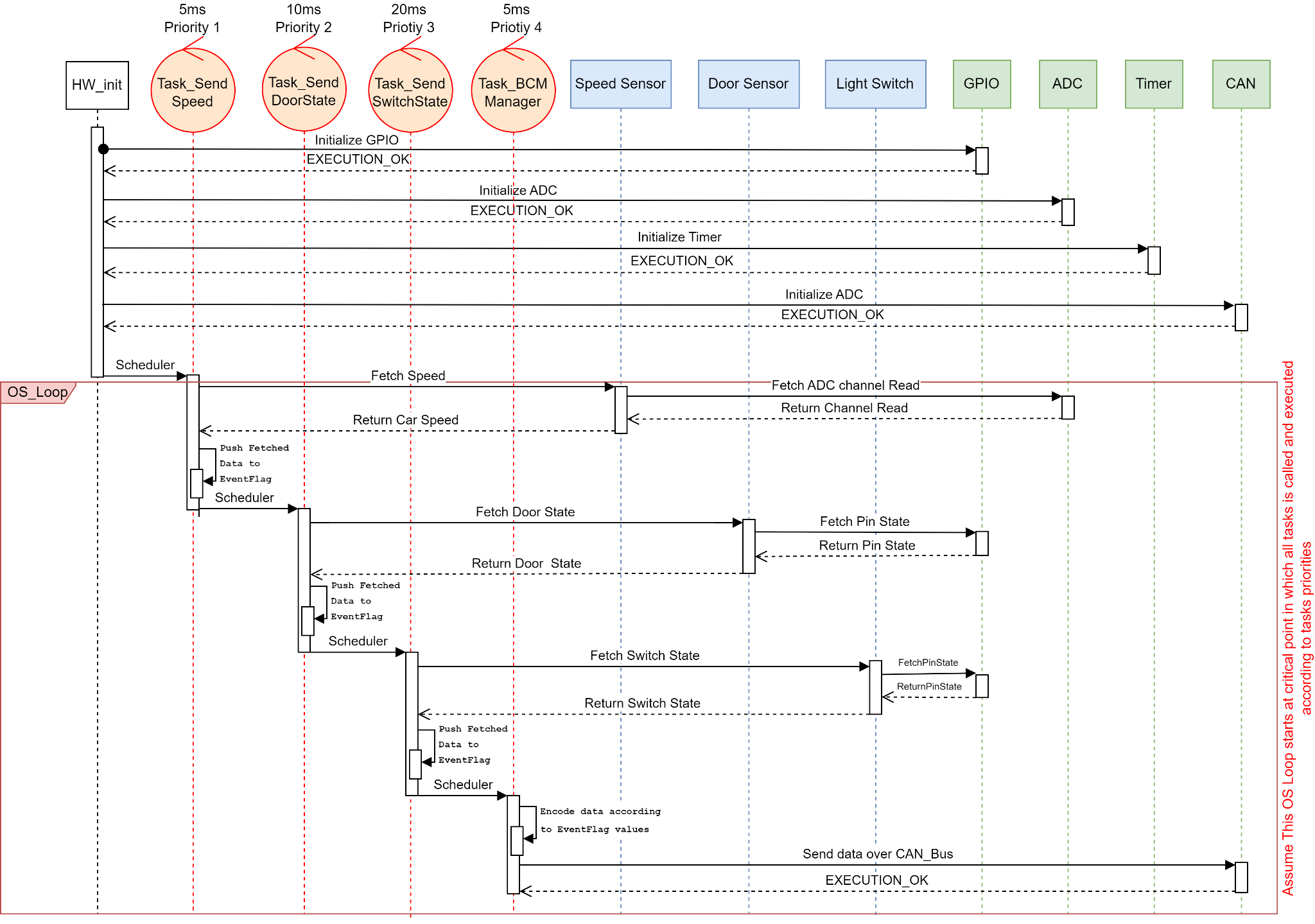
* + 1. Switch Component:



## ECU State Machine (ECU1):



## ECU Sequence Diagram (ECU1):

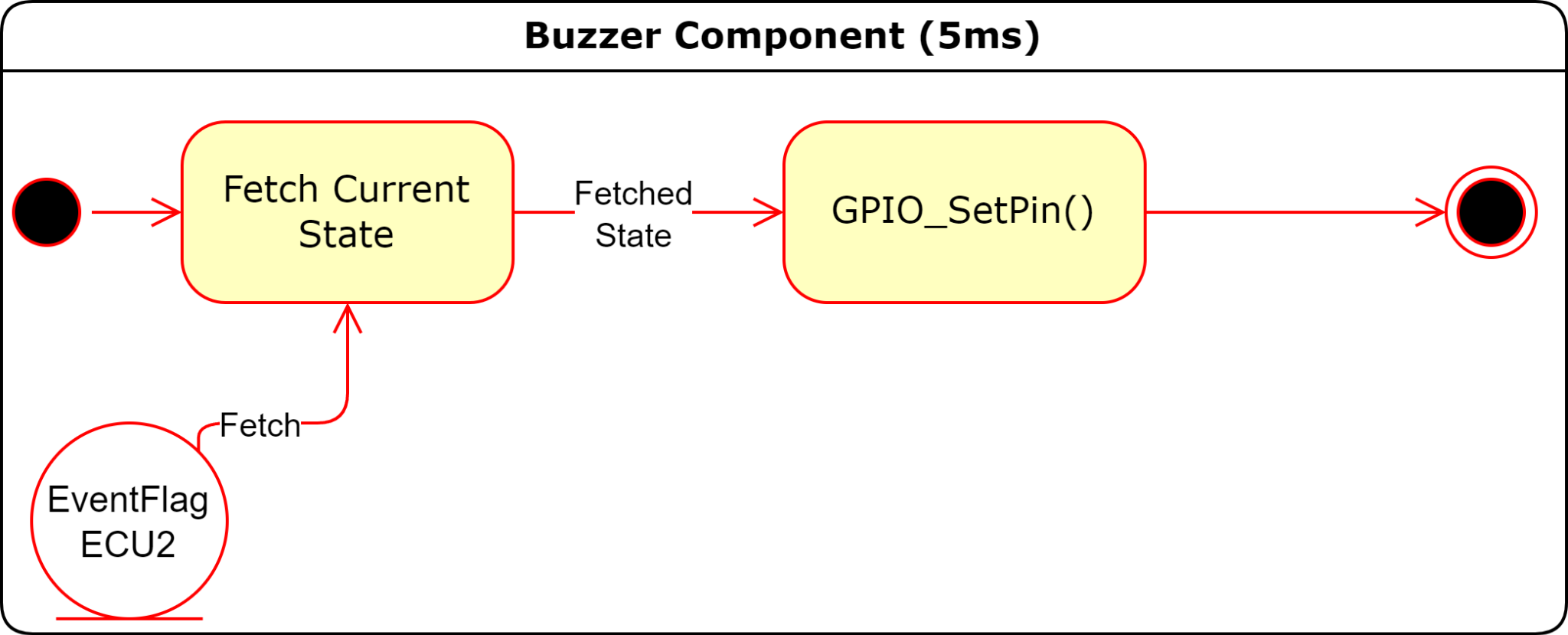


## CPU Load (ECU1):

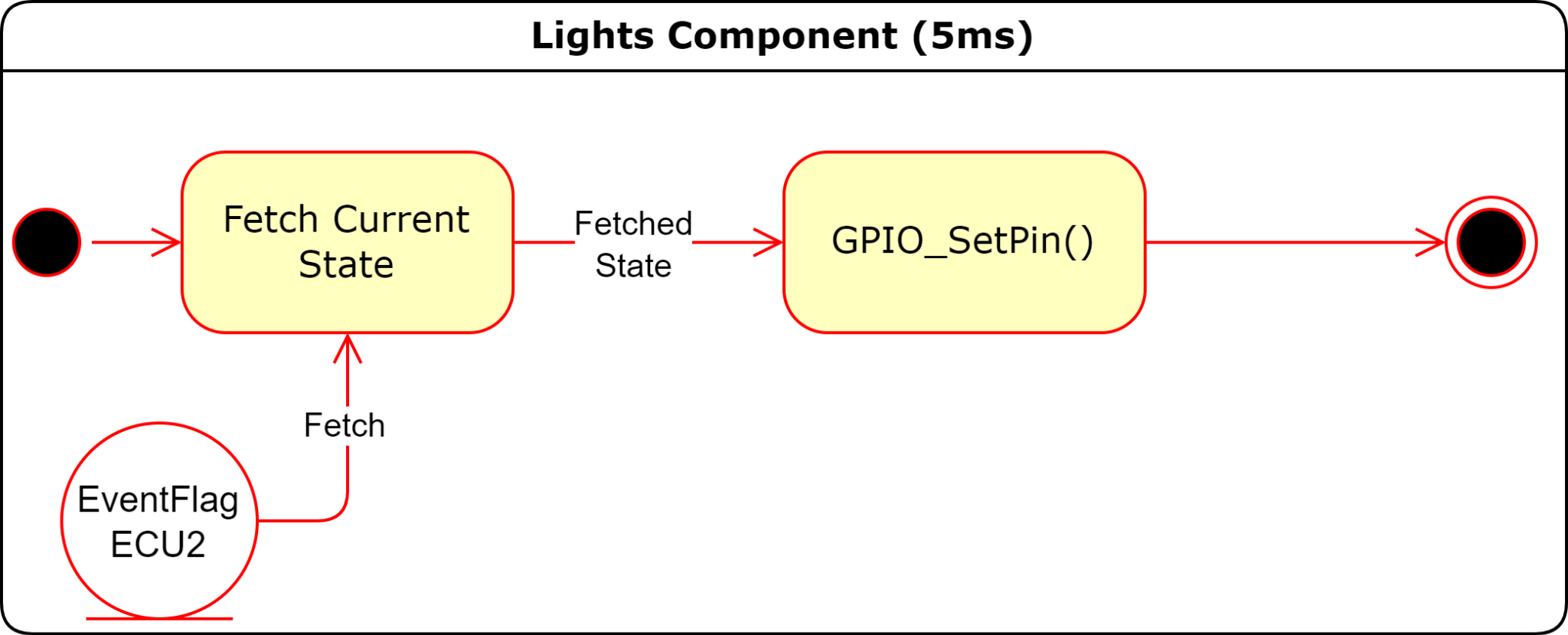
Assume:-

## On Board Components State Machine (ECU2)

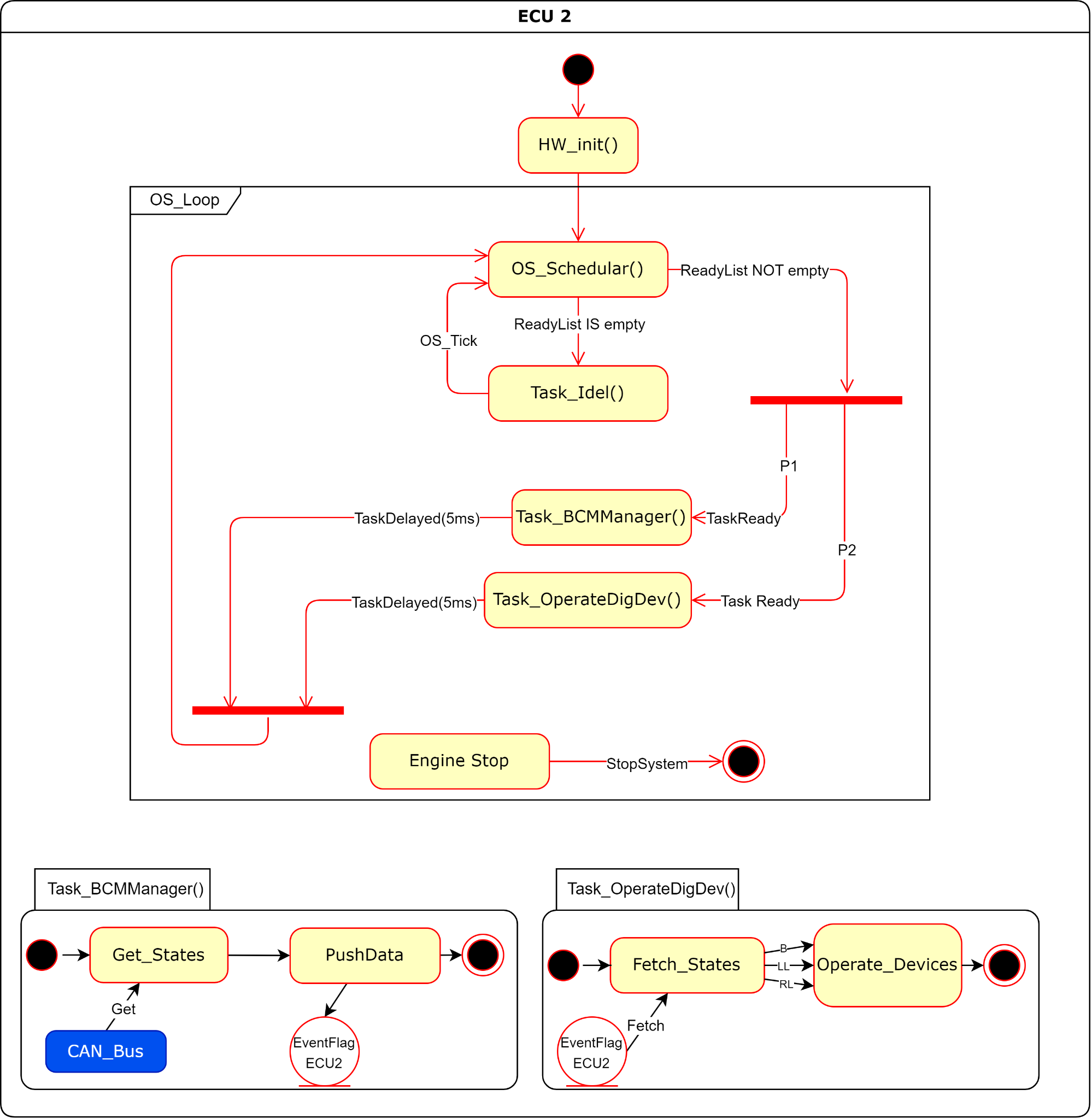
* + 1. Buzzer Component:



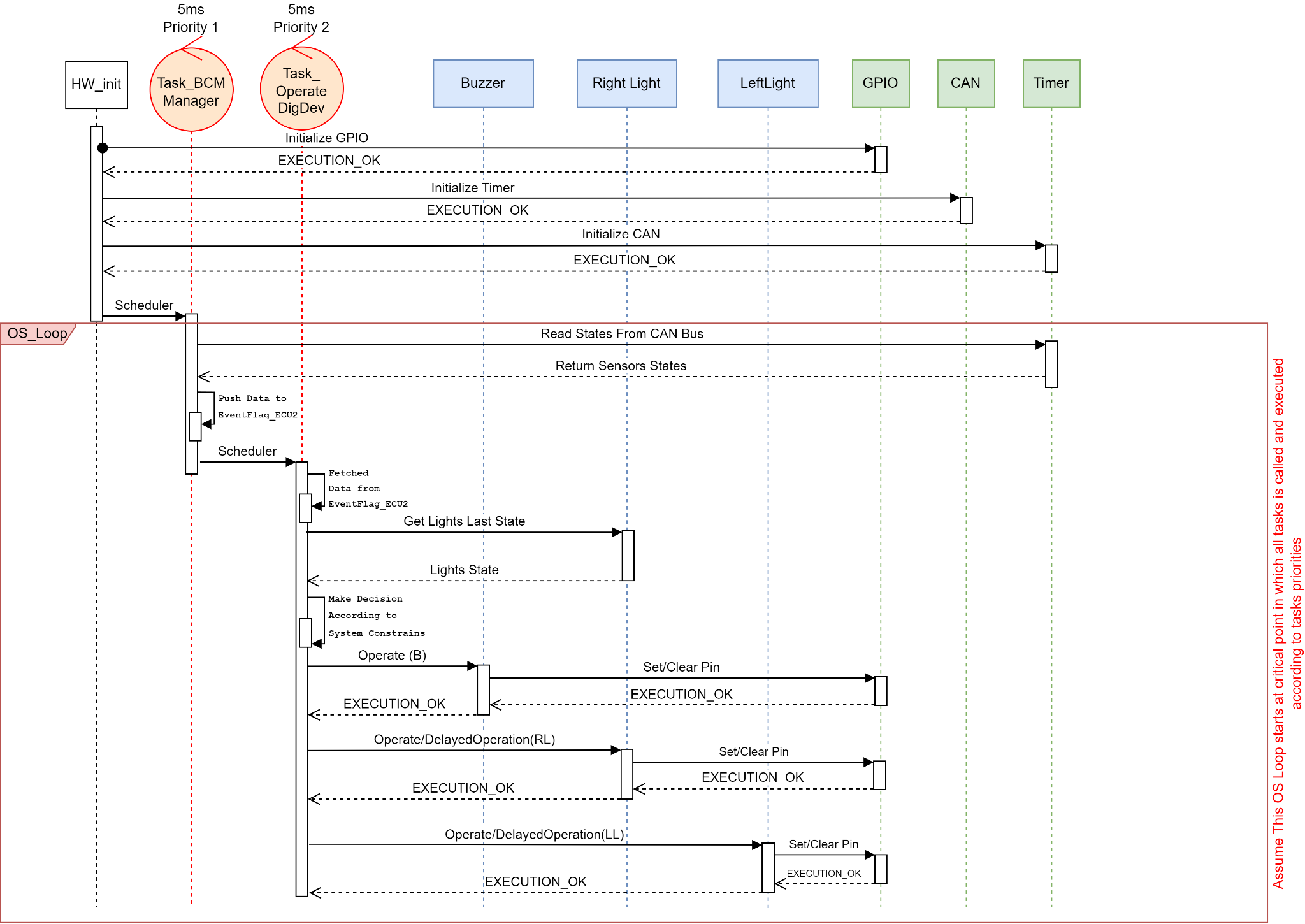
* + 1. Lights Component:



## ECU State Machine (ECU2):



## ECU Sequence Diagram (ECU2):



## CPU Load (ECU2):

Assume:-

# 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:***

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

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

**Answer = 8.16 %**