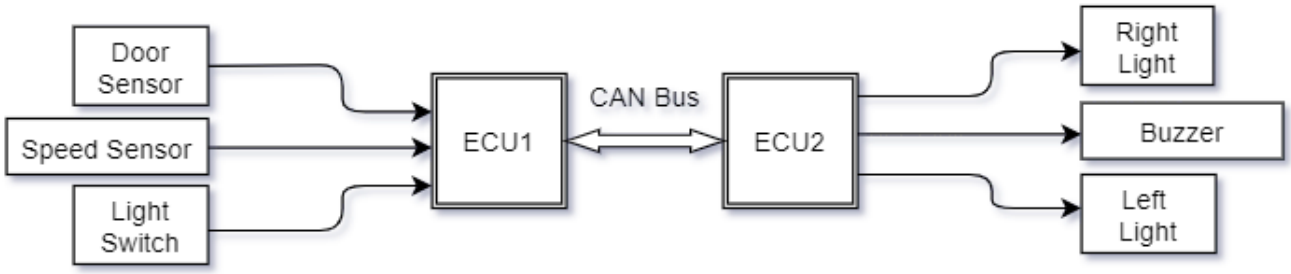
FWD – Advanced Embedded Systems Course

Project #3 – Embedded Software Design

1st : Fully Static Design.

❖ System Hardware Requirements



System Hardware Block Diagram

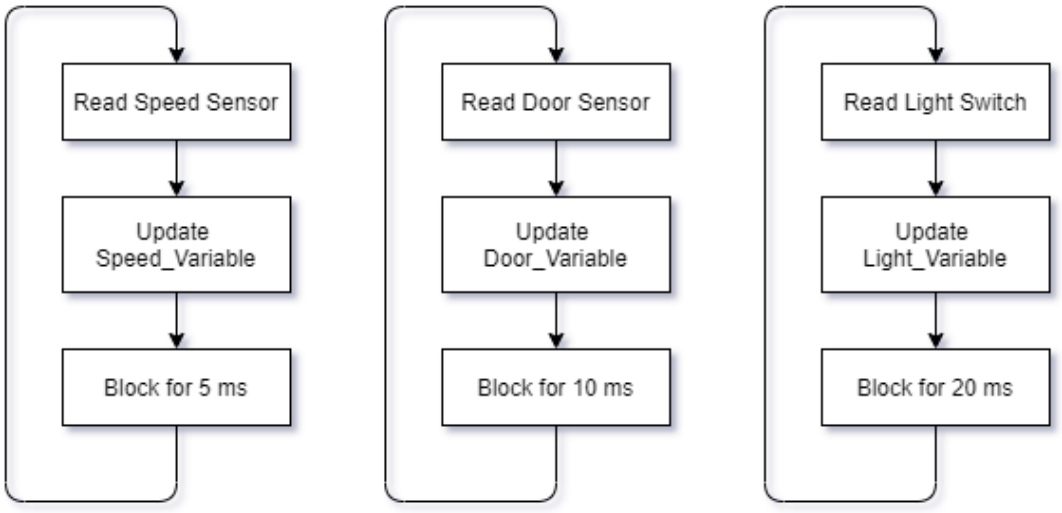
❖ System Software Requirements:

- **ECU1:**

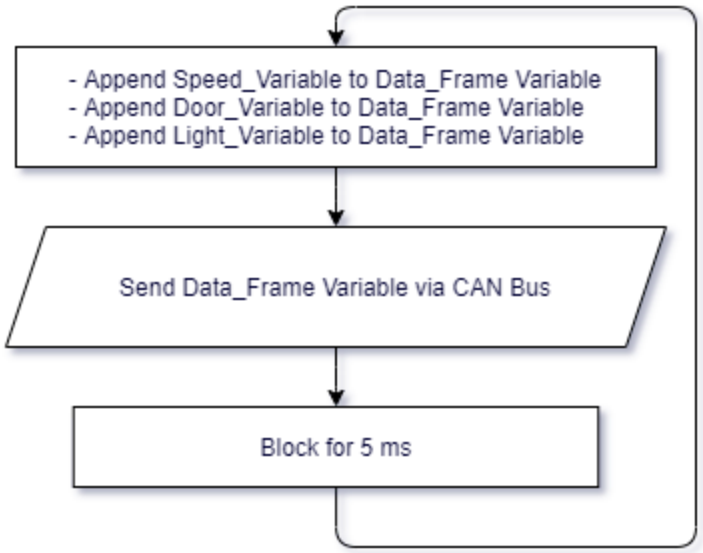
ECU1 will run a Real-Time Operating System (RTOS) to read the sensors/switch states and send them to ECU2 periodically via CAN Bus

- ECU1 Tasks properties:

Task	1) Speed Task	2) Door-Task	3) Light-Task
Properties	- Periodicity = 5ms - Priority = 1	- Periodicity = 10ms - Priority = 1	- Periodicity = 20ms - Priority = 1
Task job Flowchart	All tasks read the associate input and update the related variable state		

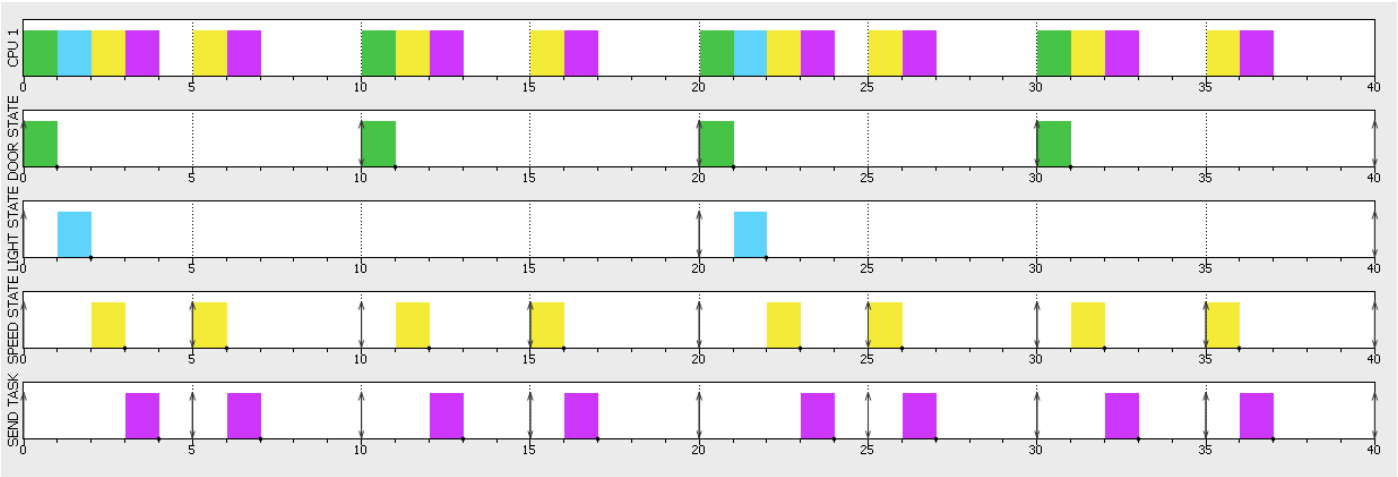


[Speed-Task] *[Door-Task]* *[Light-Task]*

Task	4) Send-Task
Properties	- Periodicity = 5ms - Priority = 2 (Send Task must run after all other tasks update the variables state)
Task job Flowchart	Task run every 5ms to create the frame to be sent then send it via CAN Bus  [Send-Task]

- ECU1 operating system offline simulation Gannet Chart: (Using SimSo Simulator)

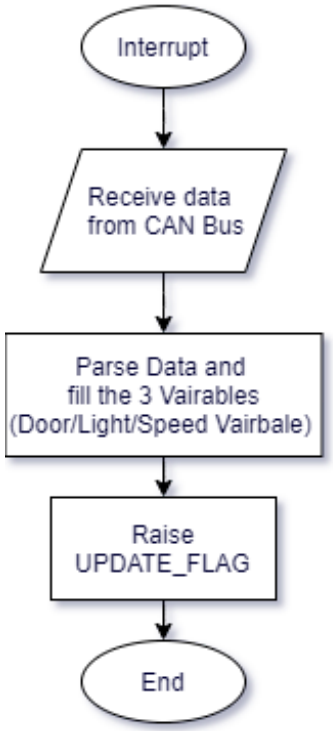
- Assuming all tasks have same execution time and total execution time is less than tick period (5ms)
- Send-Task always runs last one



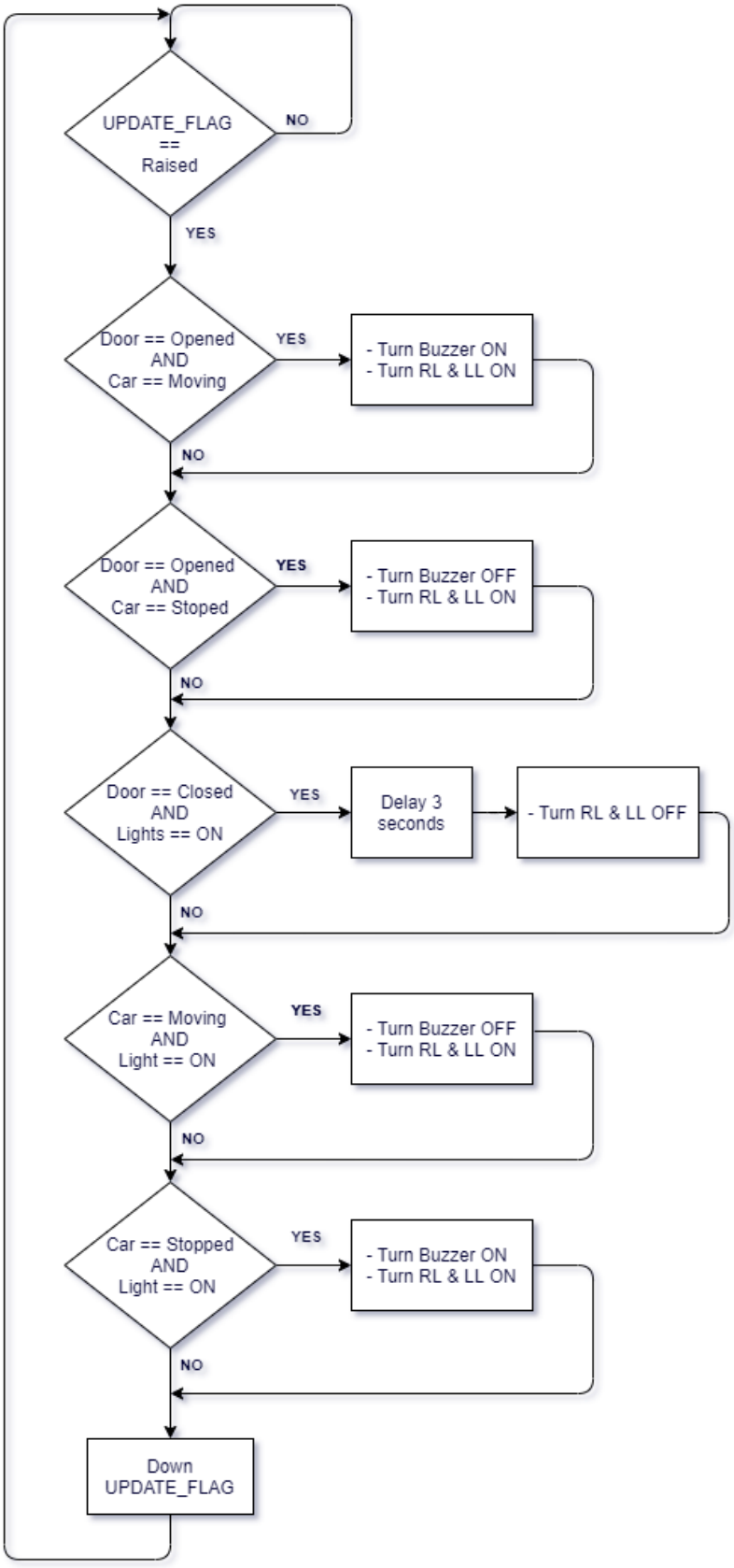
- **ECU2:**

ECU2 will runs an Event-Triggered Operating System that triggered when receives the sensors/switch states from ECU1 via CAN Bus then accordingly controls LeftLight(LL)/RightLight(RL)/Buzzer in a SuperLoop in the main

- ECU2 Tasks properties:

<table>
<tr><td>Task</td><td>CAN Bus Triggering ISR</td></tr>
<tr><td>Properties</td><td>- Triggered by CAN Bus Interrupt</td></tr>
<tr><td>Task job Flowchart</td><td>ISR receives the data frame via can bus then parse it and update variables and raise the update flag so the control task runs
 

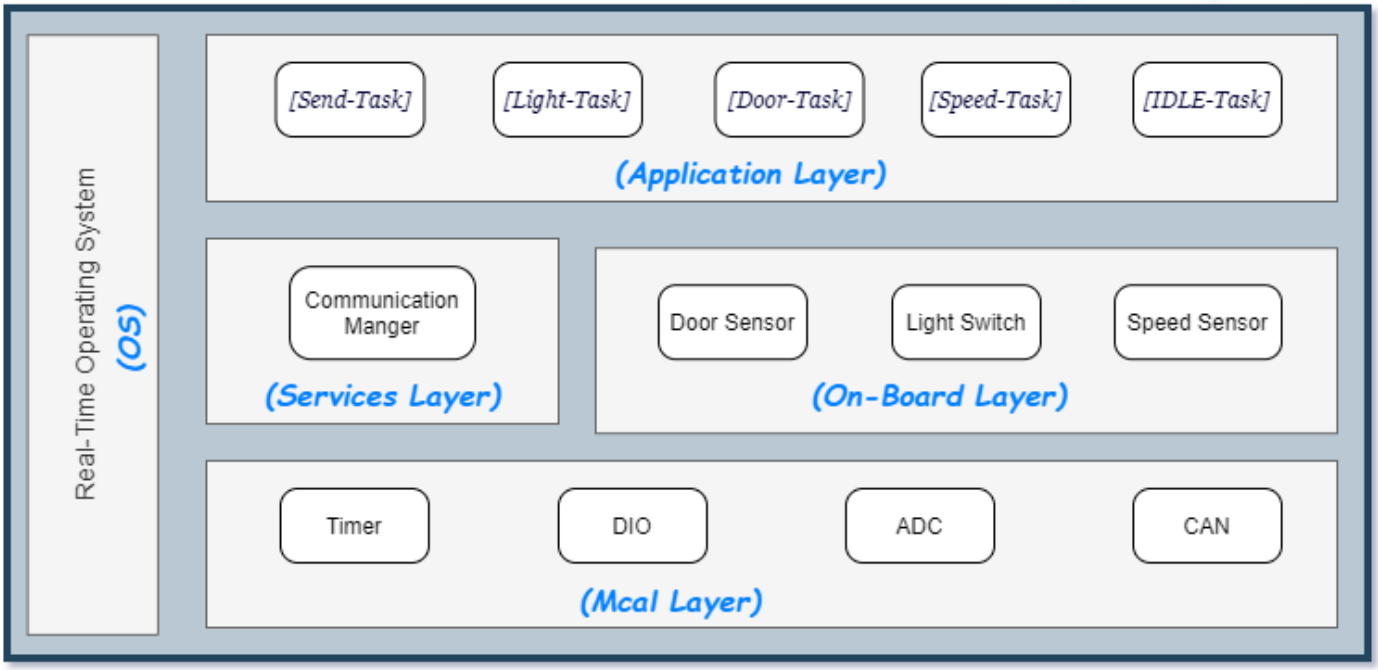
[CAN Bus ISR]</td></tr>
<tr><td>Task</td><td>SuperLoop in main</td></tr>
<tr><td>Properties</td><td>SuperLoop runs when theres an update, updates occur after CAN Bus Interrupt.</td></tr>
</table>

Task job Flowchart	When theres an update the task controls the lights and buzzer according to logic: If the door is opened while the car is moving → Buzzer ON, Lights OFF If the door is opened while the car is stopped → Buzzer OFF, Lights ON If the door is closed while the lights were ON → Lights are OFF after 3 seconds If the car is moving and the light switch is pressed → Buzzer OFF, Lights ON If the car is stopped and the light switch is pressed → Buzzer ON, Lights ON  *[Control_Task]*

❖ Static design analysis:

➢ ECU1:

○ Layered Architecture:

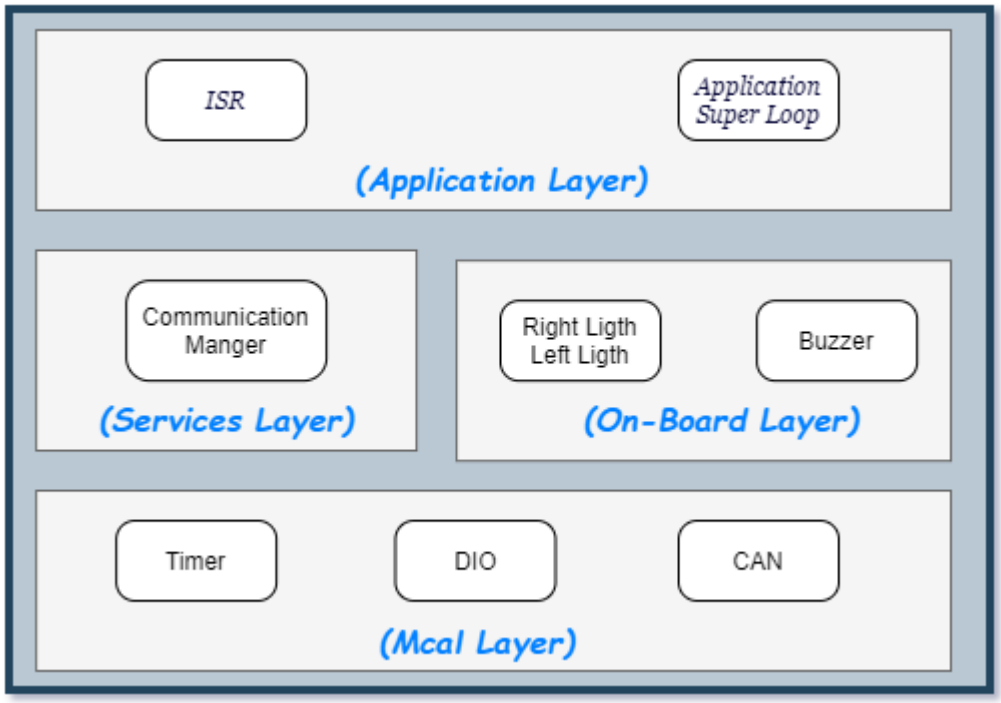


○ Modules APIs Discerption:

Layer	Module	Function Statement / Description	Arguments Description	Return Description
Mcal	DIO	void DIO_Init(DioConfigPtr_Type *ptr) ⇨ Function Initialize some GPIO port-pin as DIO	Struct holds the configurations for some GPIO port-pin	void
		void DIO_Write(Pin_Type pin, Port_Type port, Value_Type Value) ⇨ Function Writes HIGH/LOW on some GPIO port-pin	Required Port – Pin - value	void
		value_Type DIO_Read(Pin_Type pin, Port_Type port) ⇨ Function Read some GPIO port-pin state (HIGH/LOW)	Required Port - Pin	Value_Type enum states pin value (HIGH/LOW)
		void DIO_Toggle(Pin_Type pin, Port_Type port) ⇨ Function Toggles some GPIO port-pin state	Required Port - Pin	void
	ADC	void ADC_Init(DioConfigPtr_Type *ptr) ⇨ Function Initialize some GPIO port-pin as ADC	Struct holds the configurations for some GPIO port-pin	void
		float ADC_Read(Pin_Type pin, Port_Type port) ⇨ Function Reads the analog value from some GPIO port-pin	Required Port - Pin	Voltage on pin float value
	CAN	void CAN_Init(DioConfigPtr_Type *ptr) ⇨ Function Initialize some GPIO port-pin as CAN	Struct holds the configurations for some GPIO port-pin	void
		void CAN_Send(uint32_t *Data) ⇨ Function send data via CAN Bus	Pointer to the data to be sent	void
		void CAN_Receive(uint32 *Data) ⇨ Function receive data from CAN Bus	Pointer to store received data in it	void

	Timer	void Timer_Init(TimerConfigPtr_Type *ptr) ⇨ Function Initialize some Timer with some operation	Struct holds the configurations for some Timer operation	void
		void StartTimer(TimerType) ⇨ Function start some timer counting	Which timer	void
		void StopTimer(TimerType) ⇨ Function stops some timer counting	Which timer	void
		Void DelayMs(ms) ⇨ Delay function	Delay value in millisecond	void
On-Board	Door_Sensor	Void Init_DoorSensor (DoorConfigPtr *ptr) ⇨ Function initialize some GPIO pin to work with the sensor	Struct holds the configurations for initializing pin to work with the sensor	void
		DoorState_Type Get_DoorState((DoorConfigPtr *ptr) ⇨ Function returns some door sensor state (HIGH/LOW)	Pointer refers to the required door sensor	DoorState_Type enum with states HIGH/LOW
	Speed_Sensor	Void Init_SpeedSensor (SpeedConfigPtr *ptr) ⇨ Function initialize some GPIO pin to work with the sensor	Struct holds the configurations for initializing pin to work with the sensor	void
		float Get_SpeedState(SpeedConfigPtr *ptr) Function returns some speed sensor float value	Pointer refers to the required speed sensor	Speed float value
	Light_Switch	Void Init_Switch (SwitchConfigPtr *ptr) ⇨ Function initialize some GPIO pin to work with the switch	Struct holds the configurations for initializing pin to work with the switch	Void
		SwitchState_Type Get_SwitchState(SwitchConfigPtr *ptr) ⇨ Function returns some switch state (HIGH/LOW)	Pointer refers to the required switch	SwitchState_Type enum with states HIGH/LOW
Services	Comm Manger	Void CommMgr_Send (u8ID, u32 *Data) ⇨ Function sends some Data via some communication protocol	ID : represents the required comm protocol to send via Data : Pointer to data to be sent	void

- **ECU2:**
  - **Layered Architecture:**



  - **Modules APIs Discerption:**

Layer	Module	Function Statement / Description	Arguments Description	Return Description
MCAL	DIO	ECU1 Same Module		
	CAN	ECU1 Same Module		
	Timer	ECU1 Same Module		
Service	Comm Mang	ECU1 Same Module		
On-Board	Lights (RL/LL)	Void Init_Lights (LightsConfigPtr *ptr) ⇨ Function initialize some GPIO pins to work with the Lights	Struct holds the configurations for initializing pin to work with the lights	Void
		void Set_LightState(LightsConfigPtr *ptr) ⇨ Function sets lights state (HIGH/LOW)	Pointer refers to the required light sensor	void
	Buzzer	Void Init_Buzzer (BuzzerConfigPtr *ptr) ⇨ Function initialize some GPIO pins to work with the Buzzer	Struct holds the configurations for initializing pin to work with the Buzzer	Void
		void Set_BuzzerState(BuzzerConfigPtr *ptr) ⇨ Function sets Buzzer state (ACTIVE/INACTIVE)	Pointer refers to the required Buzzer	void

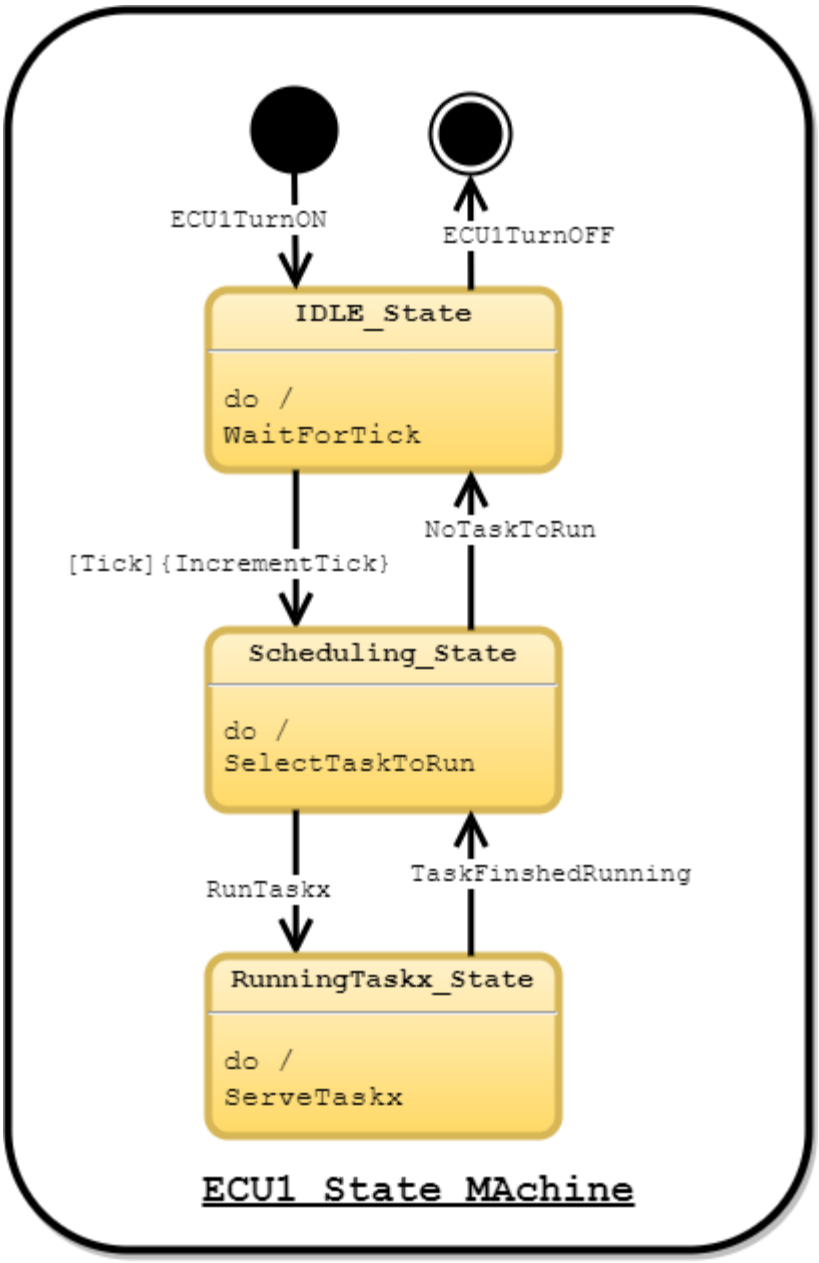
- **Folder Structure and Pseudocode Code:**

❖ Dynamic design analysis:

➢ ECU1 State Machine Diagram

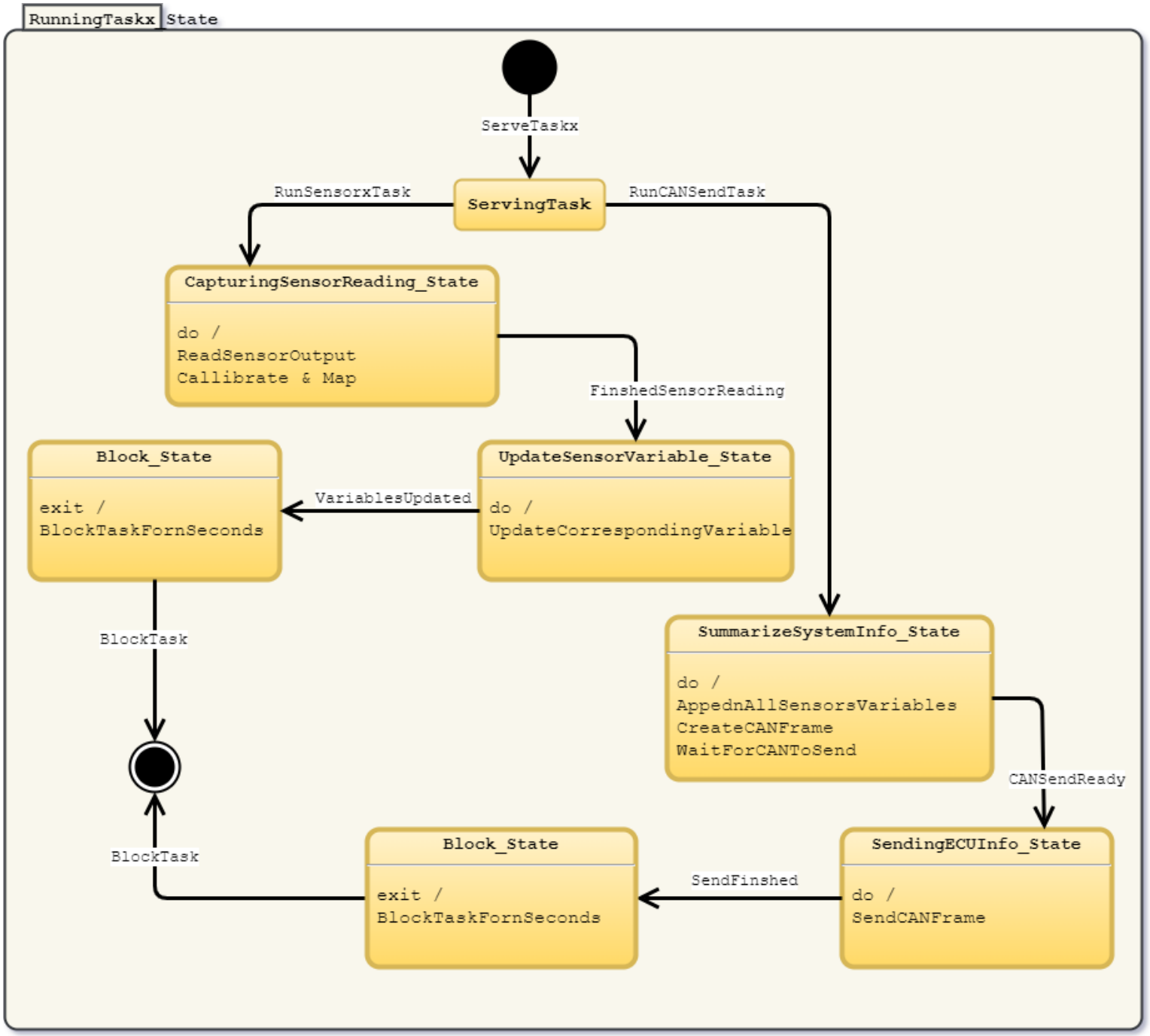
○ ECU1 Operation

ECU1 runs RTOS to handle all tasks:



- **ECU1 Components State Machine**

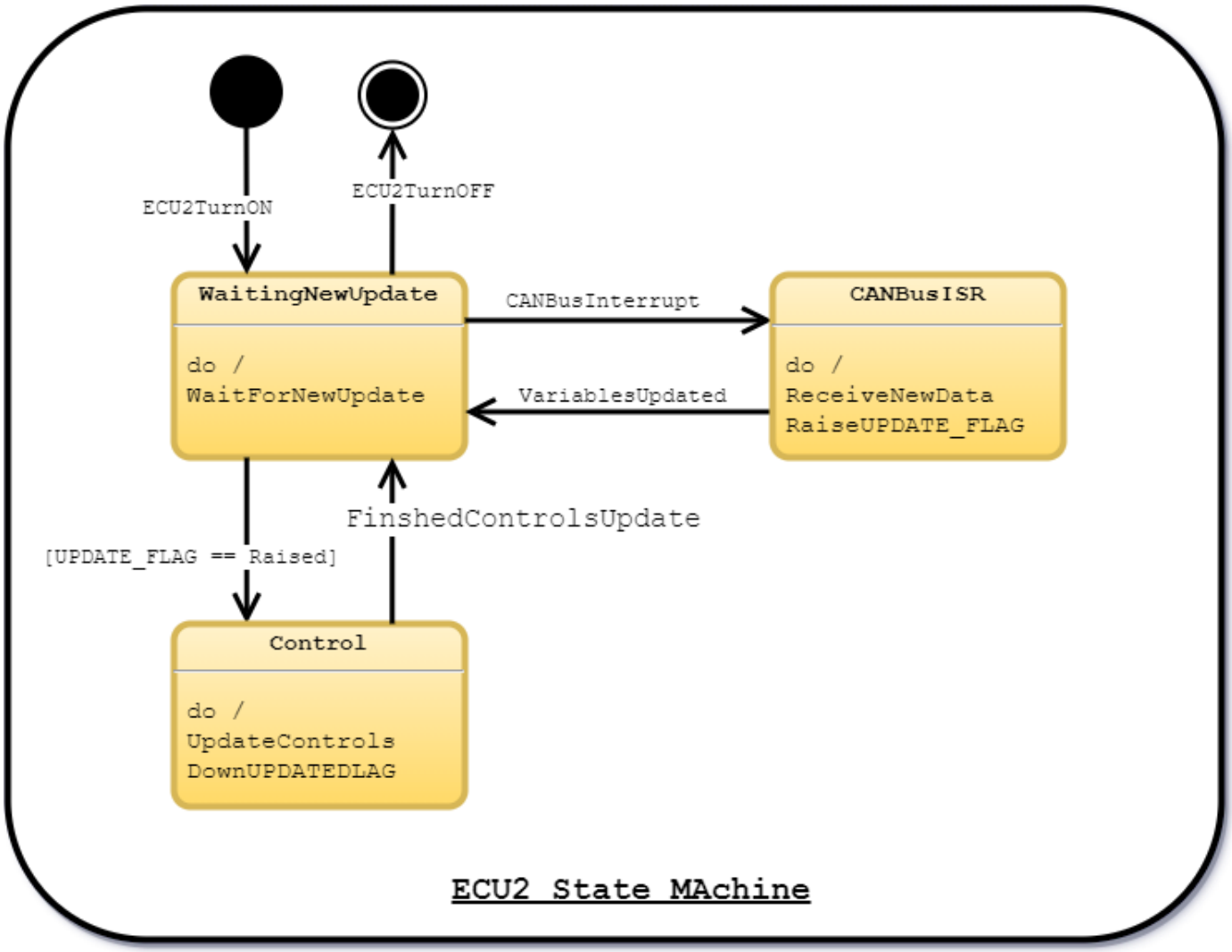
ECU1 Tasks states (RTOS Tasks) are composite states from the ECU Operation "`RunningTaskx_State`" State, ECU1 Tasks are two types, first the tasks serving the sensors/switch with same operation algorithm and summarized under "`RunSensorxTask`" Condition branch, second the Sending task under "`RunCANSendTask`" Condition branch:



- **ECU1 Sequence Diagram**

- **ECU2 State Machine Diagram**
  - **ECU2 Operation**

ECU1 Runs an Event-Triggered Architecture to immediately receive ECU1 updates vis CAN Bus and then update the control applied on ECU2 attached components (Lights/Buzzer):



- **ECU2 Components State Machine**

Control is a composite state to control Lights/Buzzer according the required logic:

If the door is opened while the car is moving → Buzzer ON, Lights OFF
If the door is opened while the car is stopped → Buzzer OFF, Lights ON
If the door is closed while the lights were ON → Lights are OFF after 3 seconds
If the car is moving and the light switch is pressed → Buzzer OFF, Lights ON
If the car is stopped and the light switch is pressed → Buzzer ON, Lights ON