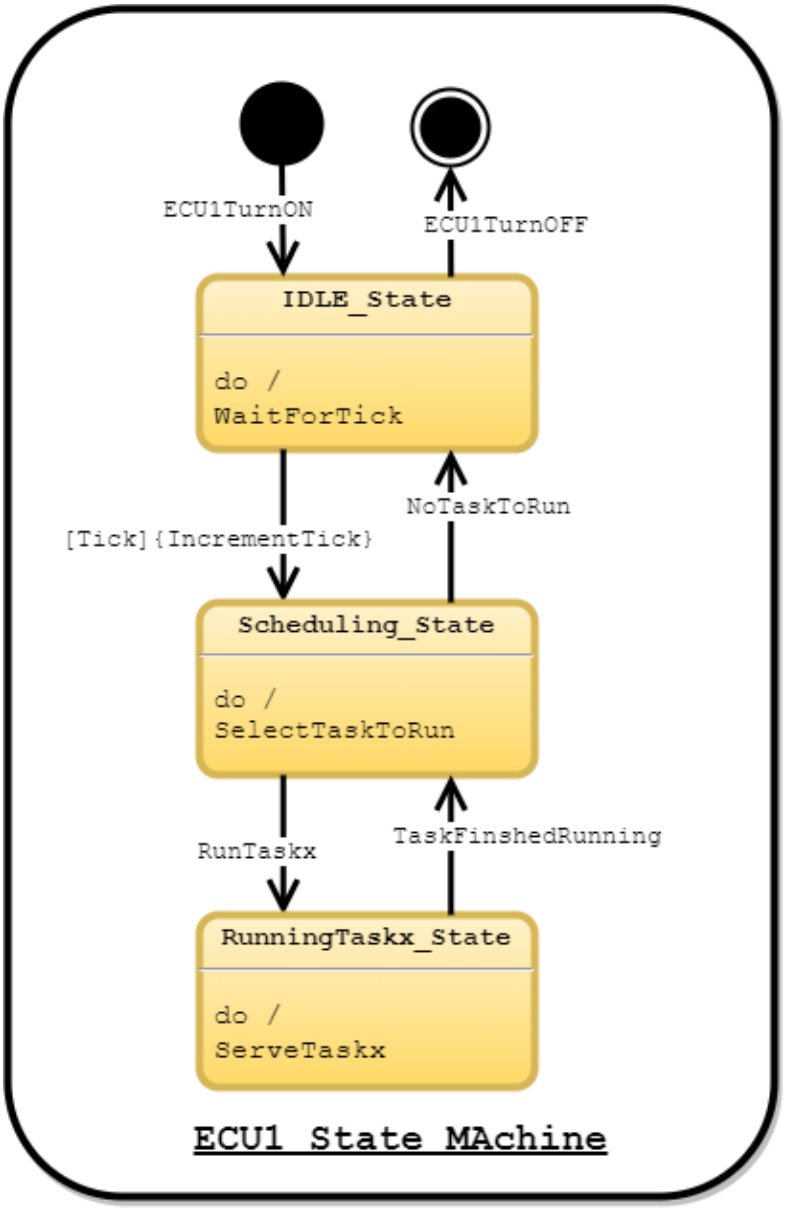
FWD – Advanced Embedded Systems Course

Project #3 – Embedded Software Design

1st : Fully Dynamic Design.

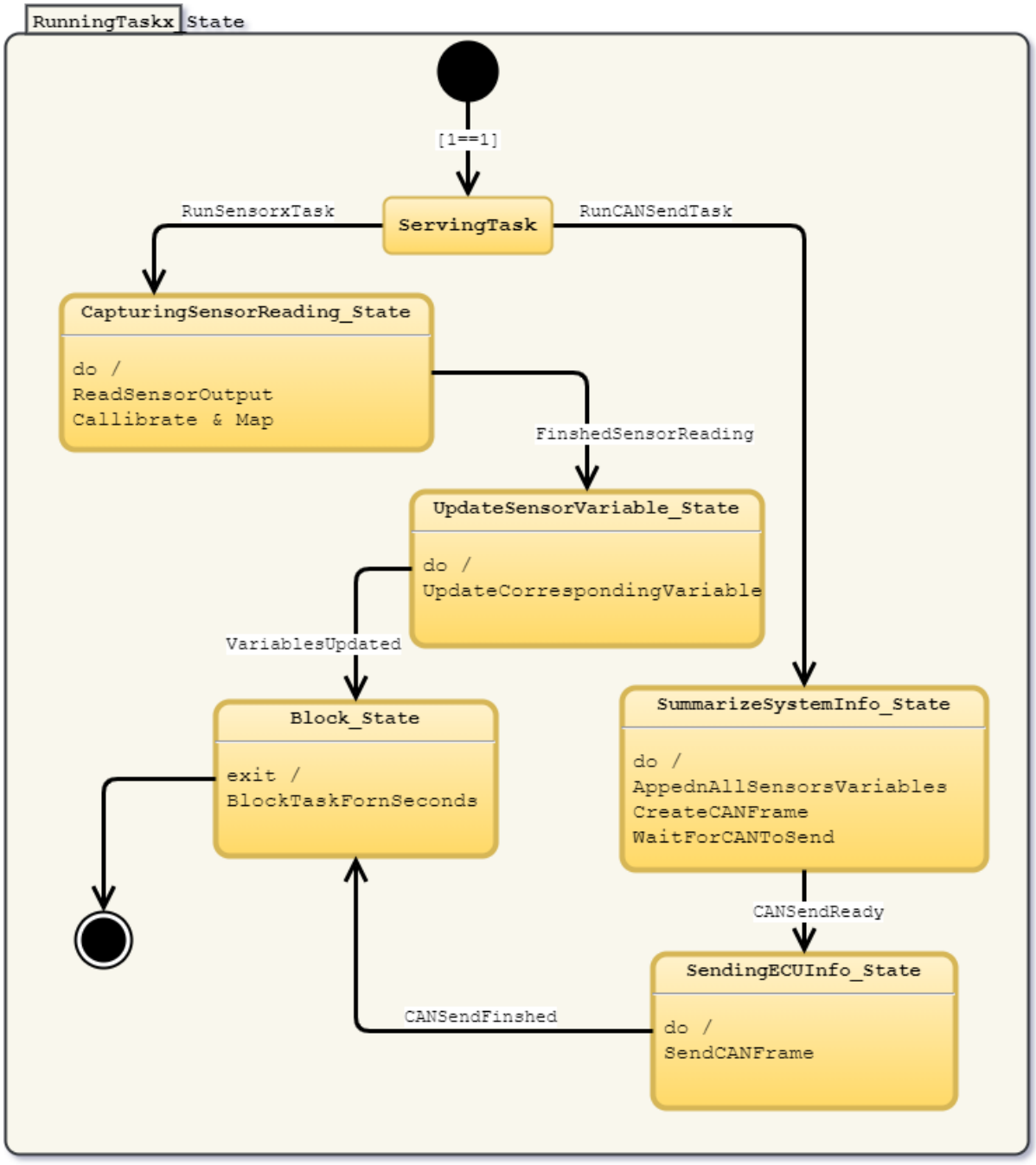
- **Dynamic design analysis:**
  - **ECU1 State Machine Diagram**
    - **ECU1 Operation**

ECU1 runs RTOS to handle all tasks:

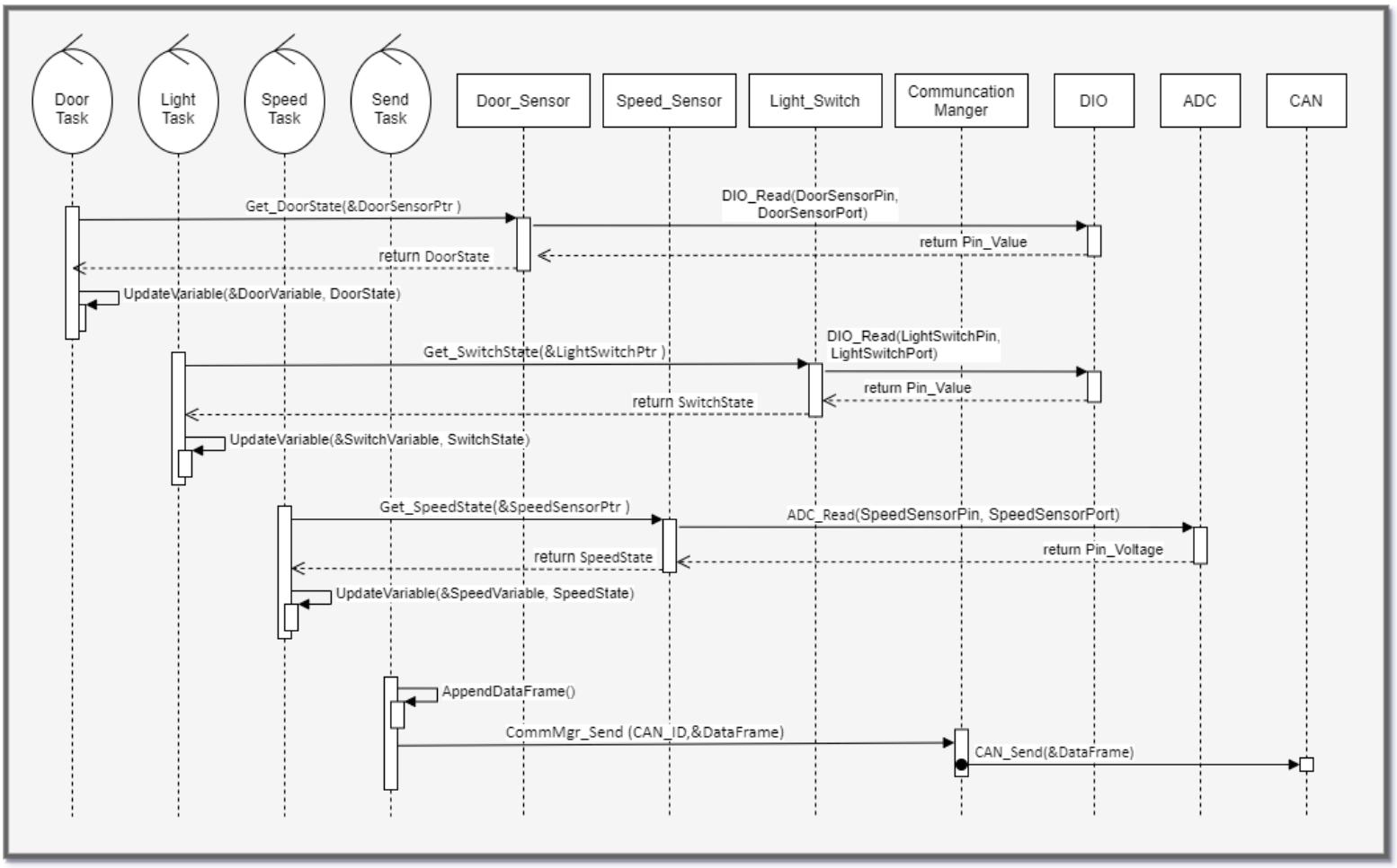


- **ECU1 Components State Machine**

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:

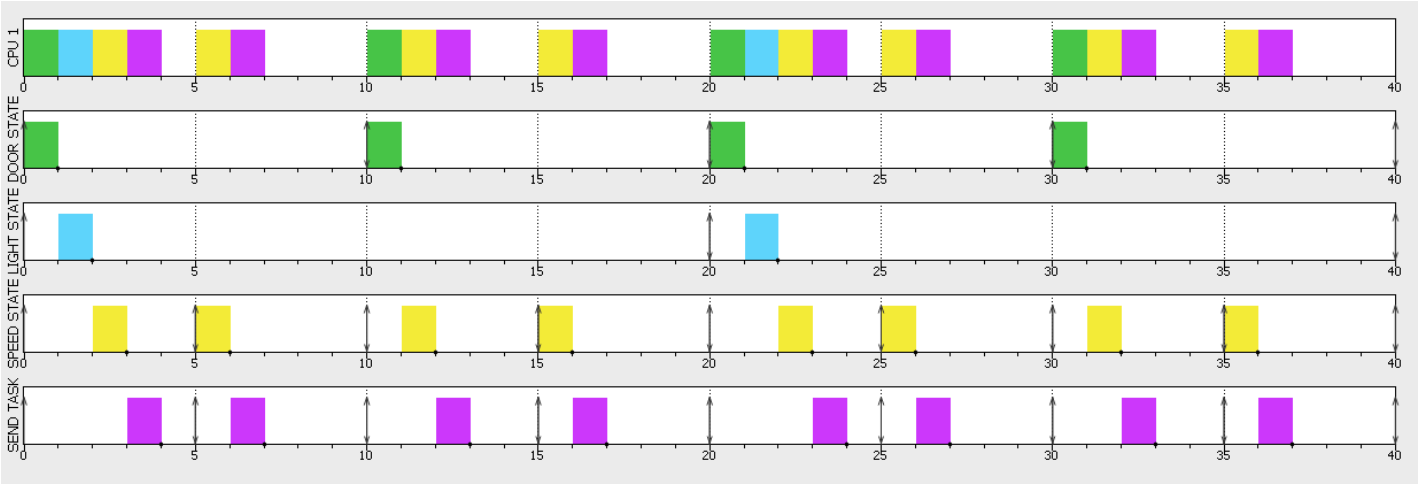


- **ECU1 Sequence Diagram**



➢ **ECU1 CPU Load**

ECU1 Runs a Real-Time Operating System,



⇨ Tasks Periodicities:
- ➔ Send_Task Periodicity = 5ms
- ➔ Speed_Task Periodicity = 5ms
- ➔ Light_Task Periodicity = 20ms
- ➔ Door_Task Periodicity = 10ms
- ➔ OS_Hyperperiod = 20ms

⇨ Tasks Execution time (Assumed for all tasks):
- ➔ Assume all tasks take 1ms for execution.

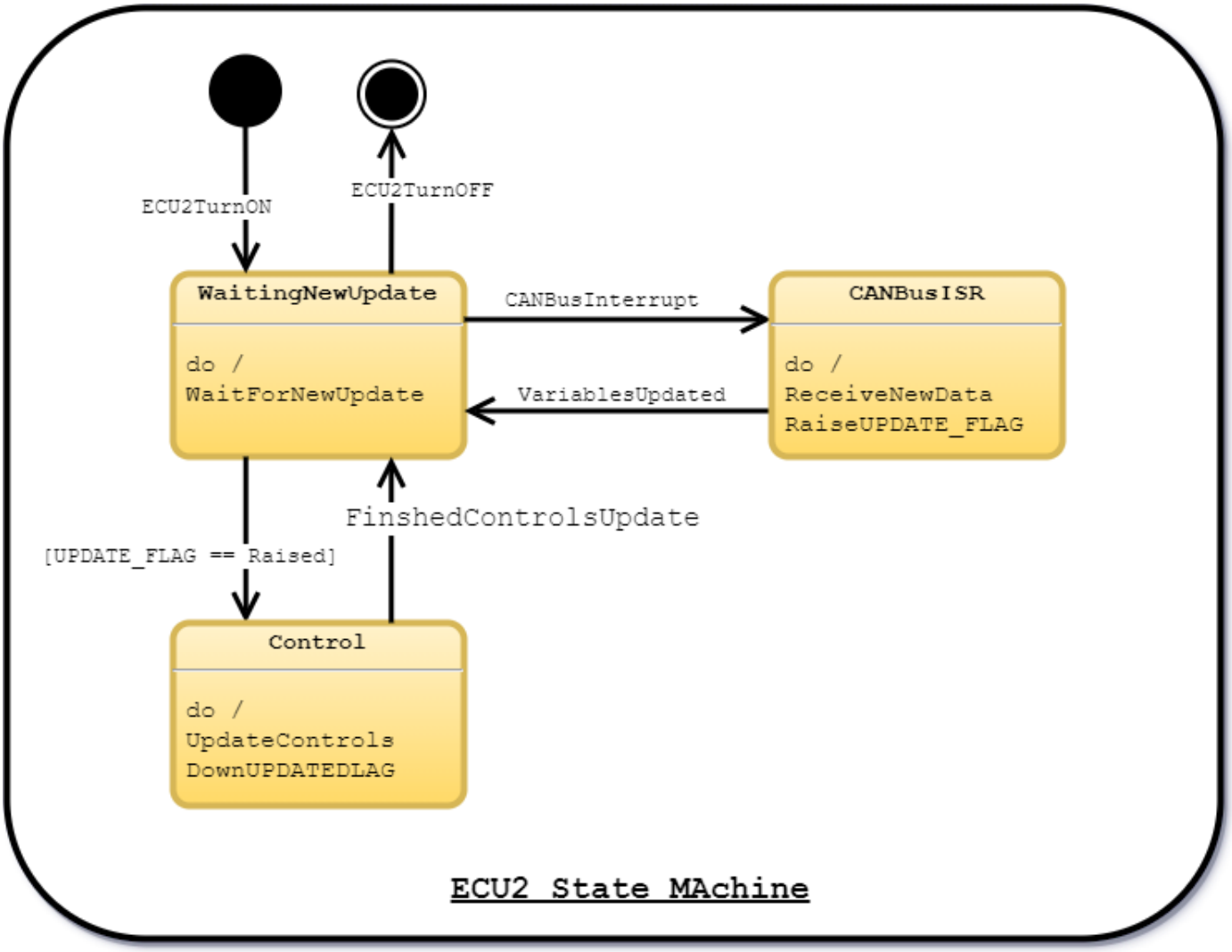
⇨ CPU Load = ((4*1) + (4*1) + (1*1) + (2*1)) / (20)
= 0.55
= 55%

ECU2 Pseudo code:

```
1
2   void Door_Task ()
3   {
4       Read Sendor Current Value
5       Update Related Variable
6       block for
7   }
8
9   void Speed_Task ()
10  {
11      Read Sendor Current Value
12      Update Related Variable
13      block for
14  }
15
16  void Light_Task ()
17  {
18      Read Sendor Current Value
19      Update Related Variable
20  }
21
22  void Send_Task ()
23  {
24      Create the frame to send
25      send the frame via CAN Bus
26  }
27
```

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

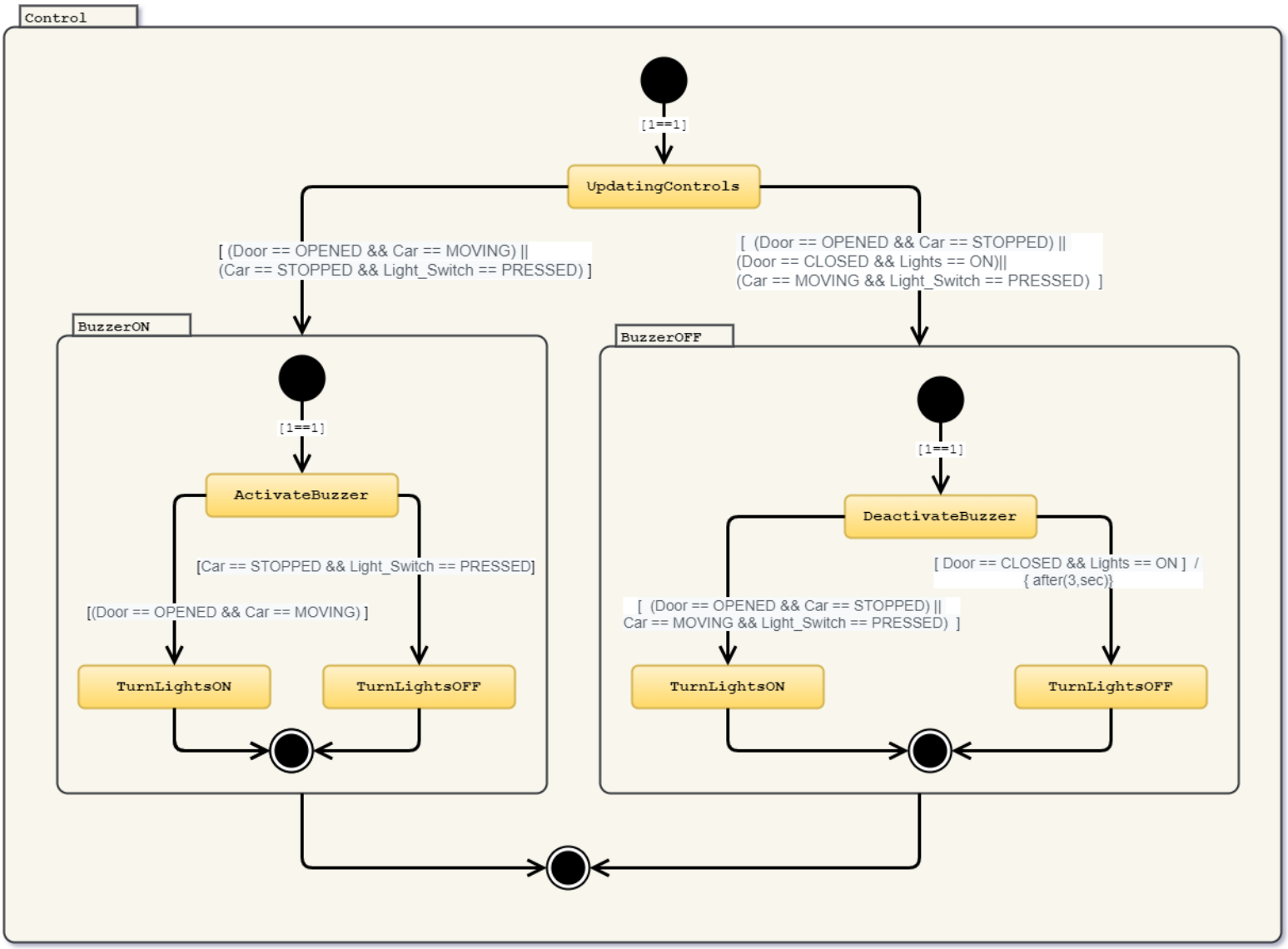
ECU1 Runs an Event-Triggered Architecture to immediately receive ECU1 updates vis CAN Bus and then update the control applied on ECU2 attached components (Lights/Buzzer):



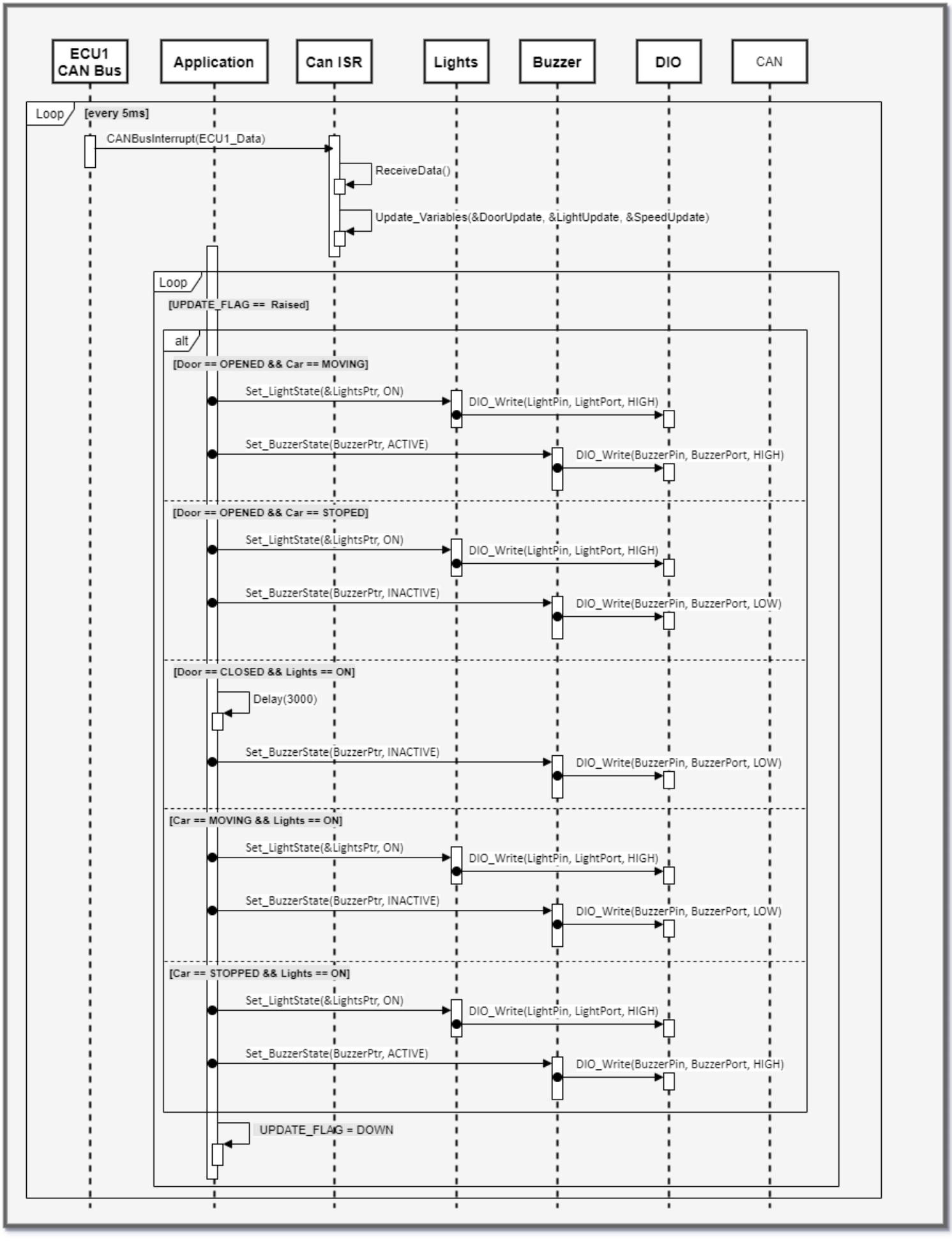
- **ECU2 Components State Machine**

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



- **ECU2 Sequence Diagram**



- **ECU2 CPU Load**

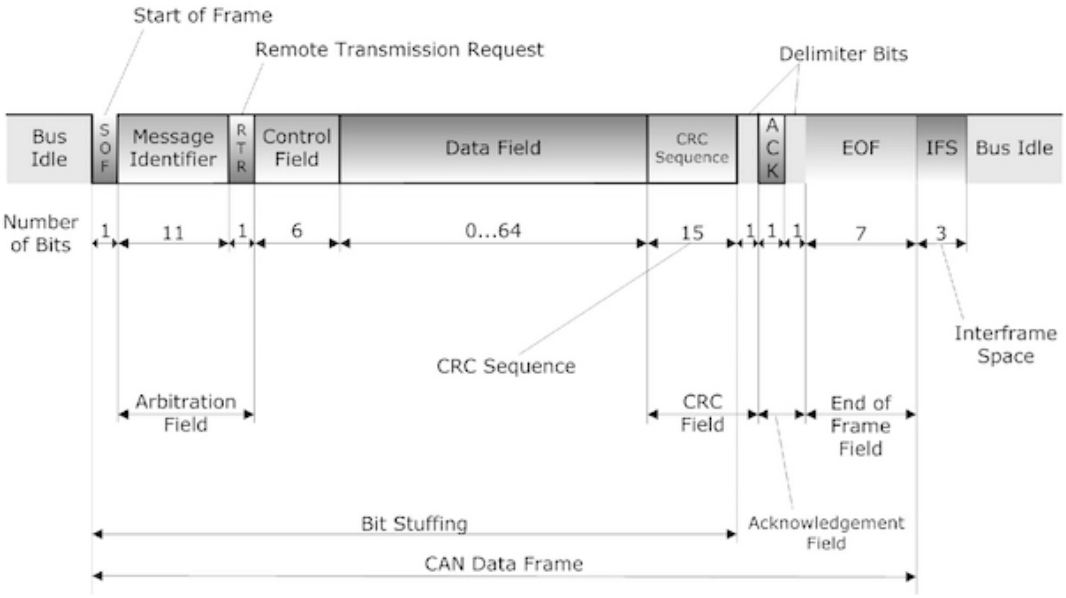
ECU2 runs an Event-Triggered Operating System, OS run at 2 instances, first at executing the CAN_Bus_ISR to receive the coming data from ECU1, second after ISR finished executing it raises the UPDATES_FLAG allowing an algorithm implemented in the main Superloop to execute to update the controls then wait for the next interrupt, this sequence is repeated every 5ms which is the interrupt rate from ECU1.

- ⇨ Assume the execution time of the CAN_Bus_ISR = 1ms
- ⇨ Assume the execution for the Superloop Algorithm = 2ms
- ⇨ OS Hyperperiod = 5m (interrupt rate)
- ⇨ CPU Load = ((1*1) + (1*2)) / 5 = 0.6 = 60 %

ECU2 Pseudo code:

```
1
2   void main (void) {
3     while (1)
4     {
5         if (UPDATE_FLAG is RAISED)
6         {
7             if (Case1_Condition is True)
8             {
9                 Do Case1_Action
10            }
11            if (Case2_Condition is True)
12            {
13                Do Case2_Action
14            }
15            if (Case3_Condition is True)
16            {
17                Do Case3_Action
18            }
19            if (Case4_Condition is True)
20            {
21                Do Case4_Action
22            }
23            if (Case5_Condition is True)
24            {
25                Do Case5_Action
26            }
27            down UPDATE_FLAG
28            Sleep MCU
29        }
30    }
31  }
32
33  void CAN_ISR ()
34  {
35      Receive The Coming Data
36      Update The Variables Value by calling UpdateFunction(newvalues)
37  }
38
39  void UpdateFunction()(newvalues){
40      Update Variables
41      Raise UPDATE_FLAG
42  }
```

- **Bus Load**
  - ➔ Message sent via CAN_Bus from ECU1 to ECU2 has periodicity = 5ms,
    Means sending 200 Message/sec
  - ➔ Assuming the message sent is 8-Bytes width,
    so total CAN frame width = 111bits



As shown in the previous picture, the entire frame has a length between 47 and 111 bits, depending on the length of the data field, which can be between 0 and 8 bytes (0 and 64 bits).

- ➔ Therefore, total number of bits send via the bus in 1 second =
  Message Rate * Message Size = 200 * 111 = 22,200 bits/s
- ➔ Using the CAN_Bus common baud rate of 125 Kbits/s,
  Bit sending time = (1/ (125*1024)) sec
- ➔ Frame sending time = 22200 * (1/ (125*1024)) = 0.1734375 sec
- ➔ Therefore, Bus Load in one second = 17.34 %