# Final Project

STATIC AND DYNAMIC DESIGN

Nourhan Mansour
SPRINTS | ESW DESIGN MC

# Contents


1) Project requirements
2) Static Design
    a. Layered Architecture
    b. API Specification
        i. DIO module
        ii. ADC
        iii. UART
        iv. Light Sensor
        v. Door Sensor
        vi. Speed Sensor
        vii. Light Actuator
        viii. Buzzer Actuator
        ix. BCM
        x. SOS
        xi. SWC Message Sender
        xii. SWC Observer
        xiii. SWC Message Receiver
        xiv. SWC Actuator
3) Dynamic Design
    a. Sequence diagrams
    b. Component Diagrams
    c. FSM
    d. Folder Structure
    e. CPU Load and Bus Load

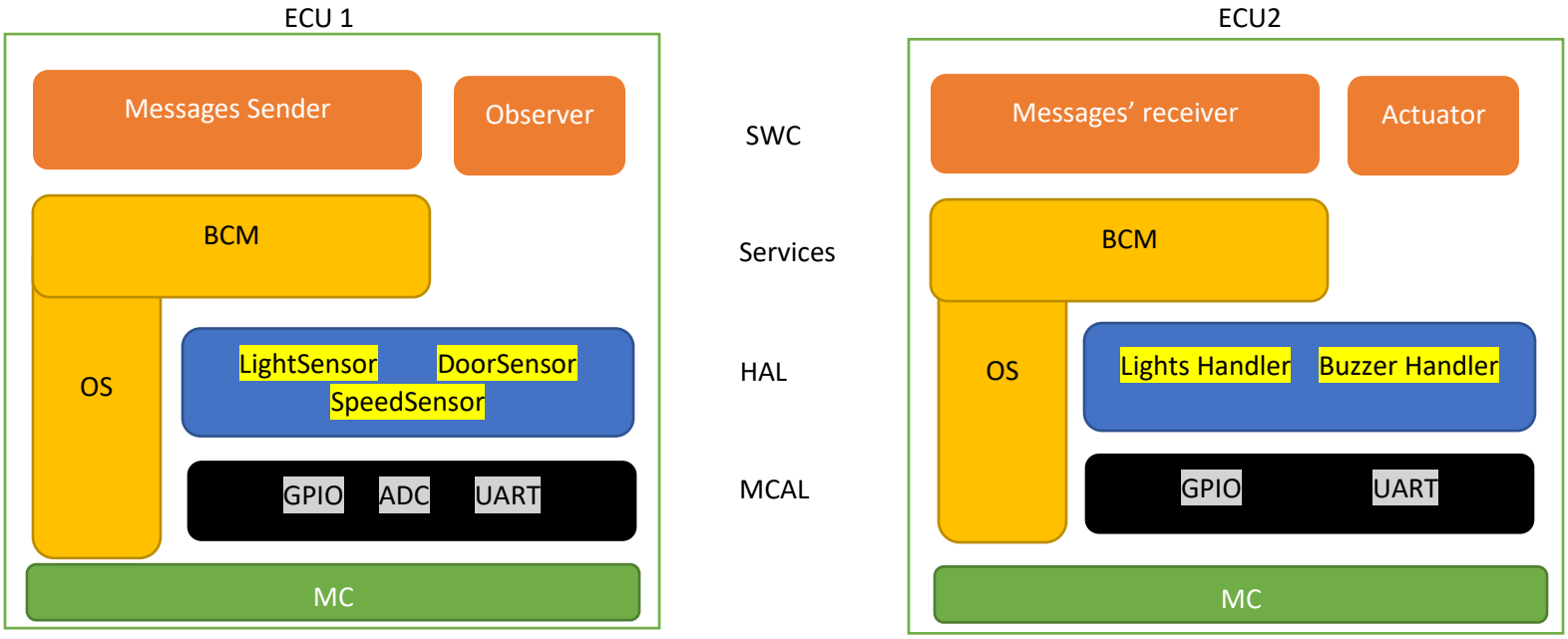
## 1. Project Requirements

2 ECUs communicate using BCM protocol over UART network
ECU1 handles inputs and ECU2 handles outputs.
Inputs: Door (10 ms periodicity) / Light (20 ms periodicity) / Speed sensors (5 ms periodicity).
Outputs: Buzzer and Lights.
Sensors' data are sent in 3 CAN messages periodically.
Define SOS tasks, CPU load, and Bus load.
Application actuator requirements:

- Door open && car moving -> Buzzer on / Lights off
- Door open && car stopped -> Buzzer off / Lights on
- Door Close && lights on -> Set timer then Lights off
- Car stopped && lights on -> Buzzer on / Lights on

## 2. Static design

### a. Layered Architecture




### b. API Specification

- *DIO Module*

Data Type Table

| Name | DIO_port | | |
|---|---|---|---|
| Type | Enumeration | | |
| Range | PORTA | 0 | Symbolic name for PortA |
| | PORTB | 1 | Symbolic name for PortB |

| | | | |
|---|---|---|---|
| | PORTC | 2 | Symbolic name for PortC |
| | PORTD | 3 | Symbolic name for PortD |
| Description | Define symbolic names for ports | | |

| | | |
|---|---|---|
| Name | DIO_pin | |
| Type | Enumeration | |
| Range | Pin0 ~ Pin7 | Available pins on each port |
| Description | Define symbolic names for available pins | |

| | | | |
|---|---|---|---|
| Name | DIO_Level | | |
| Type | Enumeration | | |
| Range | STD_LOW | 0 | Physical pin level = 0 |
| | STD_HIGH | 1 | Physical pin level = 1 |
| Description | Digital pin value | | |

| | | |
|---|---|---|
| Name | DIO_Direction | |
| Type | Enumeration | |
| Range | Input | 0 |
| | Output | 1 |
| Description | Define DIO port pin direction | |

## API Design Table

| | | | |
|---|---|---|---|
| Function name | Dio_InitPortPin | | |
| Arguments | Input | DIO_port | Enumeration |
| | | Port number / symbolic name | |
| | | DIO_Pin | Enumeration |
| | | Pin number / symbolic name | |
| | | DIO_Direction | Enumeration |

| | | |
|---|---|---|
| | | Define port pin direction. |
| | Output | |
| | Input/ Output | |
| Return | E_OK | 1 |
| | E_NOT_OK | 0 |
| Description | Responsible for initializing a port pin direction. Must be specified before read/write access on a pin. | |

| | | | |
|---|---|---|---|
| Function name | Dio_Read | | |
| Arguments | Input | DIO_Port | Enumeration |
| | | Port number / symbolic name | |
| | | DIO_Pin | Enumeration |
| | | Pin number / symbolic name | |
| | Output | DIO_LEVEL | Enumeration * |
| | | Pointer to Physical level of the specified pin. | |
| | Input/ Output | | |
| Return | E_OK | 1 | |
| | E_NOT_OK | 0 | |
| Description | Responsible for reading the physical current value of a hardware port pin. It should be able to read the value of the pin whether it's input or output without affecting its current state.<br>If the pin is uninitialized, the function should return an error and not do anything.<br>For the output parameters it must check for a null pointer exception before proceeding. | | |

| | | | |
|---|---|---|---|
| Function name | Dio_Write | | |
| Arguments | Input | DIO_Port | Enumeration |
| | | Port number / symbolic name | |

| | | | |
|---|---|---|---|
| | | DIO_Pin | Enumeration |
| | | Pin number / symbolic name | |
| | | DIO_LEVEL | Enumeration |
| | | Physical level to write on the specified pin. | |
| | Output | | |
| | Input / Output | | |
| Return | E_OK | 1 | |
| | E_NOT_OK | 0 | |
| Description | Responsible for writing the physical value of a hardware port pin. It should be able to write the value of the pin if it's output without.<br>If the pin is uninitialized, the function should return an error and not do anything. | | |

- *ADC*

Datatype Table

| Name | ADC_ChannelType | |
|---|---|---|
| Type | Uint8 | |
| Range | CHANNEL1 ~ CHANNEL8 | enumeration |
| Description | Definition of ADC channels | |

| Name | ADC_TriggerSrcType | |
|---|---|---|
| Type | Uint8 | |
| Range | FREE_RUNNING<br>ANALOG_COMPATRATOR<br>EXTERNAL_INTERRUPT<br>TMR_COMPARE_MATCH<br>TMR_OVF | enumeration |
| Description | Definition of ADC channels | |

| Name | ADC_ConfigType |
|---|---|
| Type | Structure |

| | | | |
|---|---|---|---|
| Elements | ADC_IE | Boolean | 0: Interrupts disabled<br>1: Interrupts enabled |
| | ADC_PreScalar | Enumeration | |
| | ADC_TriggerSrcType | Enumeration | |
| Description | Configuration parameters for timer module | | |

### API Design Table

| | | | |
|---|---|---|---|
| Function name | ADC_Init | | |
| Arguments | Input | ADC_ConfigType | Structure |
| | Output | | |
| | Input / Output | | |
| Return | E_OK | 1 | |
| | E_NOT_OK | 0 | |
| Description | Initialize and Enable ADC module, define interrupt usage, specify the prescalar, and trigger source | | |

| | | | |
|---|---|---|---|
| Function name | ADC_Start | | |
| Arguments | Input | ADC_ChannelType | Enumeration |
| | | Channel number for the ADC to start conversion. | |
| | Output | | |
| | Input / Output | | |
| Return | E_OK | 1 | |
| | E_NOT_OK | 0 | |
| Description | Function responsible for starting the work of the ADC channel specified.<br>If the Channel passed isn't valid for ADC operation, the function should return an error and do nothing. | | |

| | | | |
|---|---|---|---|
| Function name | ADC_Stop | | |
| Arguments | Input | ADC_ChannelType | Enumeration |

| | | |
|---|---|---|
| | | Channel number for the ADC. |
| | Output | |
| | Input / Output | |
| Return | E_OK | 1 |
| | E_NOT_OK | 0 |
| Description | Function responsible for stopping the work of the specified ADC channel.<br>If the Channel passed isn't valid for ADC operation or isn't started, the function should return an error and do nothing. | |

| | | | |
|---|---|---|---|
| Function name | ADC_GetConversionResult | | |
| Arguments | Input | ADC_ChannelType | Enumeration |
| | | Channel number for the ADC to get conversion. | |
| | Output | ADC_ChannelResult | Uint16 * |
| | | Place holder to store the latest result of the specified channel | |
| | Input / Output | | |
| Return | E_OK | 1 | |
| | E_NOT_OK | 0 | |
| Description | Function responsible for providing the latest completed conversion value for the required channel using an internal buffer storing all running channels' data. | | |

- *UART*

### Datatype Table

| | | |
|---|---|---|
| Name | UART_ChannelType | |
| Type | Enumeration | |
| Range | 1 - 3 | |
| Description | Define available UART channel values: | |

| | |
|---|---|
| | UART1_CHANNEL<br>UART2_CHANNEL<br>UART3_CHANNEL |

| | |
|---|---|
| Name | UART_BaudRateType |
| Type | Enumeration |
| Range | BPS_4800<br>BPS_9600<br>BPS_19200<br>BPS_57600<br>BPS_115200 |
| Description | Define available UART baud rate values: |

| | | | |
|---|---|---|---|
| Name | UART_ConfigType | | |
| Type | Structure | | |
| Elements | Parity | Boolean | 0: No Parity<br>1: Even Parity |
| | BaudRate | UART_BaudRateType | |
| | UART_Channel | UART_ChannelType | |
| Description | Configuration parameters for UART module | | |

## API Design Table

| | | | |
|---|---|---|---|
| Function name | UART_init | | |
| Arguments | Input | ConfigPtr | UART_ConfigType * |
| | | Pointer to channel configuration struct | |
| | Output | | |
| | Input / Output | | |
| Return | E_OK | 1 | |
| | E_NOT_OK | 0 | |

| | |
|---|---|
| Description | Initialize the specified UART channel for Receive and Transmit according to configuration parameters.<br>The function should return an error on invalid parameters. |

| | | | |
|---|---|---|---|
| Function name | UART_Transmit | | |
| Arguments | Input | Channel | UART_ChannelType |
| | | a channel to send data | |
| | | DataPtr | uint8 * |
| | | Pointer to data array to send | |
| | Output | | |
| | Input / Output | | |
| Return | E_OK | 1 | |
| | E_NOT_OK | 0 | |
| Description | Function to perform asynchronous transmit of an array of data. | | |
| Function name | UART_Recieve | | |
| Arguments | Input | Channel | UART_ChannelType |
| | | a channel to send data | |
| | | DataPtr | uint8 * |
| | | Pointer to store received data | |
| | Output | | |
| | Input / Output | | |
| Return | E_OK | 1 | |
| | E_NOT_OK | 0 | |
| Description | Function to perform asynchronous receive of an array of data using external buffer. | | |

- *Light Sensor*

### Datatype Table

| | |
|---|---|
| Name | LightSensorClassType |

| | | |
|---|---|---|
| Type | Structure | |
| Elements | DataValue | Uint8 |
| | SensorPort | DIO_PORT |
| | SensorPin | DIO_PIN |
| Description | Pointer to a new sensor instance to be used with the Sensor interface | |

### API Design Table

| | | | |
|---|---|---|---|
| Function name | LightSensor_Create | | |
| Arguments | Input | me | LightSensorClassType * |
| | | Pointer to configuration structure holding DIO information of one instance of the sensor | |
| | Output | | |
| | Input / Output | | |
| Return | E_OK | 1 | |
| | E_NOT_OK | 0 | |
| Description | Function responsible for initializing the Sensor DIO pins and assigning sensor readings to the dedicated data variable. | | |

| | | | |
|---|---|---|---|
| Function name | LightSensor_Main | | |
| Arguments | Input | me | LightSensorClassType * |
| | | Pointer to the sensor object (instance) | |
| | output | | |
| | Input / Output | | |
| Return | E_OK | 1 | |
| | E_NOT_OK | 0 | |
| Description | Periodic function to get the physical meaning of the average of the latest 10 readings of the sensor and store in the data variable | | |

| | |
|---|---|
| Function name | LightSensor_SetUpdateFrequency |

| | | | |
|---|---|---|---|
| Arguments | Input | me | LightSensorClassType * |
| | | Pointer to the sensor object (instance) | |
| | | Frequency | uint |
| | | Define the periodic update interval in ms | |
| | output | | |
| | Input / Output | | |
| Return | E_OK | 1 | |
| | E_NOT_OK | 0 | |
| Description | Function to set the update criteria for the sensor | | |

| | | | |
|---|---|---|---|
| Function name | LightSensor_ReadValue | | |
| Arguments | Input | me | LightSensorClassType * |
| | | Pointer to the sensor object (instance) | |
| | output | | |
| | Input / Output | | |
| Return | E_OK | 1 | |
| | E_NOT_OK | 0 | |
| Description | Return the value stored in the data variable of the associated instance / object | | |

- *Door Sensor*

Datatype Table

| | | |
|---|---|---|
| Name | DoorSensorClassType | |
| Type | Structure | |
| Elements | DataValue | Uint8 |
| | SensorPort | DIO_PORT |
| | SensorPin | DIO_PIN |
| Description | Pointer to a new sensor instance to be used with the Sensor interface | |

## API Design Table

| Function name | DoorSensor_Create | | |
|---|---|---|---|
| Arguments | Input | me | DoorSensorClassType * |
| | | Pointer to the sensor object (instance) | |
| | output | | |
| | Input / Output | | |
| Return | E_OK | 1 | |
| | E_NOT_OK | 0 | |
| Description | Create a new instance of the Sensor DIO pins and assigning sensor readings to the dedicated data variable. | | |

| Function name | DoorSensor_Main | | |
|---|---|---|---|
| Arguments | Input | me | DoorSensorClassType * |
| | | Pointer to the sensor object (instance) | |
| | output | | |
| | Input / Output | | |
| Return | E_OK | 1 | |
| | E_NOT_OK | 0 | |
| Description | Periodic function to get the physical meaning of the average of the latest 10 readings of the sensor and store in the data variable | | |

| Function name | DoorSensor_SetUpdateFrequency | | |
|---|---|---|---|
| Arguments | Input | me | DoorSensorClassType * |
| | | Pointer to the sensor object (instance) | |
| | | Frequency | uint |
| | | Define the periodic update interval in ms | |
| | output | | |
| | Input / Output | | |

| | | |
|---|---|---|
| Return | E_OK | 1 |
| | E_NOT_OK | 0 |
| Description | Function to set the update criteria for the sensor | |

| | | | |
|---|---|---|---|
| Function name | DoorSensor_ReadValue | | |
| Arguments | Input | me | DoorSensorClassType * |
| | | Pointer to the sensor object (instance) | |
| | output | | |
| | Input / Output | | |
| Return | E_OK | 1 | |
| | E_NOT_OK | 0 | |
| Description | Return the value stored in the data variable of the associated instance / object | | |

- *Speed Sensor*

### Datatype Table

### API Design Table

| | | | |
|---|---|---|---|
| Function name | SpeedSensor_Init | | |
| Arguments | Input | ADC_ChannelType | Enumeration |
| | | ADC channel to which the Sensor is assigned | |
| | Output | | |
| | Input / Output | | |
| Return | E_OK | 1 | |
| | E_NOT_OK | 0 | |
| Description | Function responsible for initializing the Sensor and starting the associated ADC channel conversion. | | |

| | | | |
|---|---|---|---|
| Function name | SpeedSensor_main | | |
| Arguments | Input | | |

| | | | |
|---|---|---|---|
| | Output | Speed_KMH | Uint16 * |
| | | Return the physical meaning of ADC conversion result in actual speed | |
| | Input / Output | | |
| Return | E_OK | 1 | |
| | E_NOT_OK | 0 | |
| Description | Main function to get the avg and physical meaning of the last 10 ADC readings. | | |

- *Light Actuator*

### Datatype Table

| Name | LightActuatorClassType | |
|---|---|---|
| Type | Structure | |
| Elements | DataValue | Uint8 |
| | SensorPort | DIO_PORT |
| | SensorPin | DIO_PIN |
| Description | Pointer to a new instance to be used with the actuator interface | |

### API Design Table

| Function name | LightActuator_Create | | |
|---|---|---|---|
| Arguments | Input | me | LightActuatorClassType * |
| | | Pointer to configuration structure holding DIO information of one instance of the light actuator | |
| | Output | | |
| | Input / Output | | |
| Return | E_OK | 1 | |
| | E_NOT_OK | 0 | |
| Description | Function responsible for initializing the DIO pins. | | |

| Function name | LightActuator_SetValue |
|---|---|

| | | | |
|---|---|---|---|
| Arguments | Input | me | LightActuatorClassType * |
| | | Pointer to one instance of the light actuator | |
| | Output | | |
| | Input / Output | | |
| Return | E_OK | 1 | |
| | E_NOT_OK | 0 | |
| Description | Function responsible for setting the DIO pins values. | | |

| | | | |
|---|---|---|---|
| Function name | LightActuator_GetValue | | |
| Arguments | Input | me | LightActuatorClassType * |
| | | Pointer to one instance of the light actuator | |
| | Output | | |
| | Input / Output | | |
| Return | E_OK | 1 | |
| | E_NOT_OK | 0 | |
| Description | Function responsible for getting the DIO pins values. | | |

- *Buzzer Actuator*

### Datatype Table

### API Design Table

| | | | |
|---|---|---|---|
| Function name | Buzzer_Init | | |
| Arguments | Input | pin | DIO_PIN |
| | | port | DIO_PORT |
| | Output | | |
| | Input / Output | | |
| Return | E_OK | 1 | |
| | E_NOT_OK | 0 | |
| Description | Function responsible for initializing the Buzzer pin. | | |

| Function name | Buzzer_SetValue | | |
|---|---|---|---|
| Arguments | Input | | |
| | Output | | |
| | Input / Output | | |
| Return | E_OK | 1 | |
| | E_NOT_OK | 0 | |
| Description | Function responsible for setting the Buzzer pin value. | | |

| Function name | Buzzer_GetValue | | |
|---|---|---|---|
| Arguments | Input | | |
| | Output | | |
| | Input / Output | | |
| Return | E_OK | 1 | |
| | E_NOT_OK | 0 | |
| Description | Function responsible for getting the Buzzer pin value. | | |

- *BCM*

### Datatype Table

| Name | COM_ChannelType | |
|---|---|---|
| Type | Enumeration | |
| Range | Enumeration | |
| Description | Define available com channel values:<br><br>CAN1_Channel<br>CAN2_Channel<br>UART_Channel<br>I2C_Channel | |

### API Design Table

| Function name | BCM_init | | |
|---|---|---|---|
| Arguments | Input | | |

| | | | |
|---|---|---|---|
| | Output | | |
| | Input / Output | | |
| Return | E_OK | 1 | |
| | E_NOT_OK | 0 | |
| Description | This function is responsible for initializing the different communication modules used in the system, internal or external. | | |

| | | | |
|---|---|---|---|
| Function name | BCM_RxInit | | |
| Arguments | Input | comChannel | COM_ChannelType |
| | | Define which communication channel to setup receive sequence | |
| | Output | | |
| | Input / Output | | |
| Return | E_OK | 1 | |
| | E_NOT_OK | 0 | |
| Description | Function responsible for initializing Rx sequence for the specified Communication channel | | |

| | | | |
|---|---|---|---|
| Function name | BCM_TxInit | | |
| Arguments | Input | comChannel | COM_ChannelType |
| | | Define which communication channel to setup transmit sequence | |
| | Output | | |
| | Input / Output | | |
| Return | E_OK | 1 | |
| | E_NOT_OK | 0 | |
| Description | Function responsible for initializing Tx sequence for the specified Communication channel. | | |

| | |
|---|---|
| Function name | BCM_RxDispatcher |

| | | | |
|---|---|---|---|
| Arguments | Input | comChannel | COM_ChannelType |
| | | Define which communication channel to Create receive event. | |
| | Output | funcPtr | Void * |
| | | Pointer to call back function to be executed when the RX event occur | |
| | Input / Output | | |
| Return | | | |
| Description | Main receive periodic function to start the receive sequence. | | |

| | | | |
|---|---|---|---|
| Function name | BCM_TxData | | |
| Arguments | Input | comChannel | COM_ChannelType |
| | | Define which communication channel to setup transmit sequence | |
| | Output | | |
| | Input / Output | | |
| Return | E_OK | 1 | |
| | E_NOT_OK | 0 | |
| Description | Function responsible for transmitting data over the specified Communication channel.<br>The function should return an error if the channel is not initialized as Tx or an invalid parameter is passed. | | |

- *SOS*

### Datatype Table

| | | |
|---|---|---|
| Name | TaskConfigType | |
| Type | Structure | |
| Elements | pFunc | Void * |
| | Pointer to function to execute when the task is ready | |
| | Delay | Uint8 |

| | | |
|---|---|---|
| | The amount of time in ms to wait before first task ttriggering | |
| | Period | Uint8 |
| | The frequency in ms at which the task will be ready to execute. | |
| Description | Pointer to a new task. | |

## API Design Table

| Function name | SOS_init | | |
|---|---|---|---|
| Arguments | Input | | |
| | Output | | |
| | Input / Output | | |
| Return | E_OK | 1 | |
| | E_NOT_OK | 0 | |
| Description | This function is responsible for initializing the system Timer configurations. | | |

| Function name | SOS_CreateTask | | |
|---|---|---|---|
| Arguments | Input | TaskPtr | TaskConfigType * |
| | Output | | |
| | Input / Output | | |
| Return | E_OK | 1 | |
| | E_NOT_OK | 0 | |
| Description | This function is responsible for creating a new task and reserving its place in stack/memory. | | |

| Function name | SOS_DeleteTask | | |
|---|---|---|---|
| Arguments | Input | TaskPtr | TaskConfigType * |
| | Output | | |
| | Input / Output | | |
| Return | E_OK | 1 | |

| | | |
|---|---|---|
| | E_NOT_OK | 0 |
| Description | This function is responsible for deleting an existing task and freeing its place in stack/memory. | |

| | | |
|---|---|---|
| Function name | SOS_Run | |
| Arguments | Input | |
| | Output | |
| | Input / Output | |
| Return | E_OK | 1 |
| | E_NOT_OK | 0 |
| Description | Main periodic function. responsible for dispatching and running the tasks in queue according to priority and ready flag. | |

- *SWC Message Sender*

Datatype Table

API Design Table

| | | | |
|---|---|---|---|
| Function name | MS_Accept | | |
| Arguments | Input | sensorID | Uint8 |
| | Output | sensorData | Uint8 * |
| | Input / Output | | |
| Return | E_OK | 1 | |
| | E_NOT_OK | 0 | |
| Description | Request data subscription of a specific sensor from the Sensor Observer Server. | | |

| | | | |
|---|---|---|---|
| Function name | MS_SendMsg | | |
| Arguments | Input | MsgID | Uint8 |
| | | Data | Uint8 |
| | Output | | |

| | | |
|---|---|---|
| | Input / Output | |
| Return | E_OK | 1 |
| | E_NOT_OK | 0 |
| Description | Send Msg over BCM | |

| | | | |
|---|---|---|---|
| Function name | MS_Main | | |
| Arguments | Input | | |
| | Output | | |
| | Input / Output | | |
| Return | E_OK | 1 | |
| | E_NOT_OK | 0 | |
| Description | Periodic function to send sensors data on intervals configured by systems requirements | | |

- *SWC Observer*

Datatype Table

| | | |
|---|---|---|
| Name | SensorObserverType | |
| Type | Structure | |
| Elements | SensorID | Enumeration |
| | Id of the sensor instance to be used with the specified sensor module | |
| | SensorData | Uint8 |
| | Reading of the sensor | |
| | NotificationHandler[] | Void()* |
| | Array of subscription list. | |
| Description | Used to add a new instance to the observer server | |

API Design Table

| | | | |
|---|---|---|---|
| Function name | SensorObserver_init | | |
| Arguments | Input | me | SensorObserverType * |

| | | | |
|---|---|---|---|
| | Output | | |
| | Input / Output | | |
| Return | E_OK | 1 | |
| | E_NOT_OK | 0 | |
| Description | Constructor for the sensor instance | | |

| | | | |
|---|---|---|---|
| Function name | SensorObserver_CleanUp | | |
| Arguments | Input | me | SensorObserverType * |
| | Output | | |
| | Input / Output | | |
| Return | E_OK | 1 | |
| | E_NOT_OK | 0 | |
| Description | DeConstructor for the sensor instance | | |

| | | | |
|---|---|---|---|
| Function name | SensorObserver_subscripe | | |
| Arguments | Input | me | SensorObserverType * |
| | Output | | |
| | Input / Output | | |
| Return | E_OK | 1 | |
| | E_NOT_OK | 0 | |
| Description | Add a new client to the subscription list | | |

| | | | |
|---|---|---|---|
| Function name | SensorObserver_unsubscripe | | |
| Arguments | Input | me | SensorObserverType * |
| | Output | | |
| | Input / Output | | |
| Return | E_OK | 1 | |

| | | |
|---|---|---|
| | E_NOT_OK | 0 |
| Description | Remove a client from the subscription list | |

| | | | |
|---|---|---|---|
| Function name | SensorObserver_notify | | |
| Arguments | Input | | |
| | Output | | |
| | Input / Output | | |
| Return | E_OK | 1 | |
| | E_NOT_OK | 0 | |
| Description | Notify subscribed clients of new readings | | |

- *SWC Message Receiver*

### Datatype Table

### API Design Table

| | | | |
|---|---|---|---|
| Function name | MR_CallBack | | |
| Arguments | Input | | |
| | Output | | |
| | Input / Output | | |
| Return | E_OK | 1 | |
| | E_NOT_OK | 0 | |
| Description | Function passed to the BCM to be executed on data reception complete event. It should store the new commands in hared global variables for the Actuator to use later | | |

| | | | |
|---|---|---|---|
| Function name | MR_GetSpeedMsg | | |
| Arguments | Input | | |
| | Output | BuzzerLevel | Uint8 * |
| | Input / Output | | |
| Return | E_OK | 1 | |

| | | | |
|---|---|---|---|
| | E_NOT_OK | 0 | |
| Description | Get the latest speed value received. | | |

| | | | |
|---|---|---|---|
| Function name | MR_GetLightMsg | | |
| Arguments | Input | | |
| | Output | lightLevel | Uint8 * |
| | Input / Output | | |
| Return | E_OK | 1 | |
| | E_NOT_OK | 0 | |
| Description | Get the latest light msg value. | | |

| | | | |
|---|---|---|---|
| Function name | MR_GetSpeedMsg | | |
| Arguments | Input | | |
| | Output | SpeedLevel | Uint8 * |
| | Input / Output | | |
| Return | E_OK | 1 | |
| | E_NOT_OK | 0 | |
| Description | Get the latest speed msg value. | | |

- *SWC Actuator*

### Datatype Table

### API Design Table

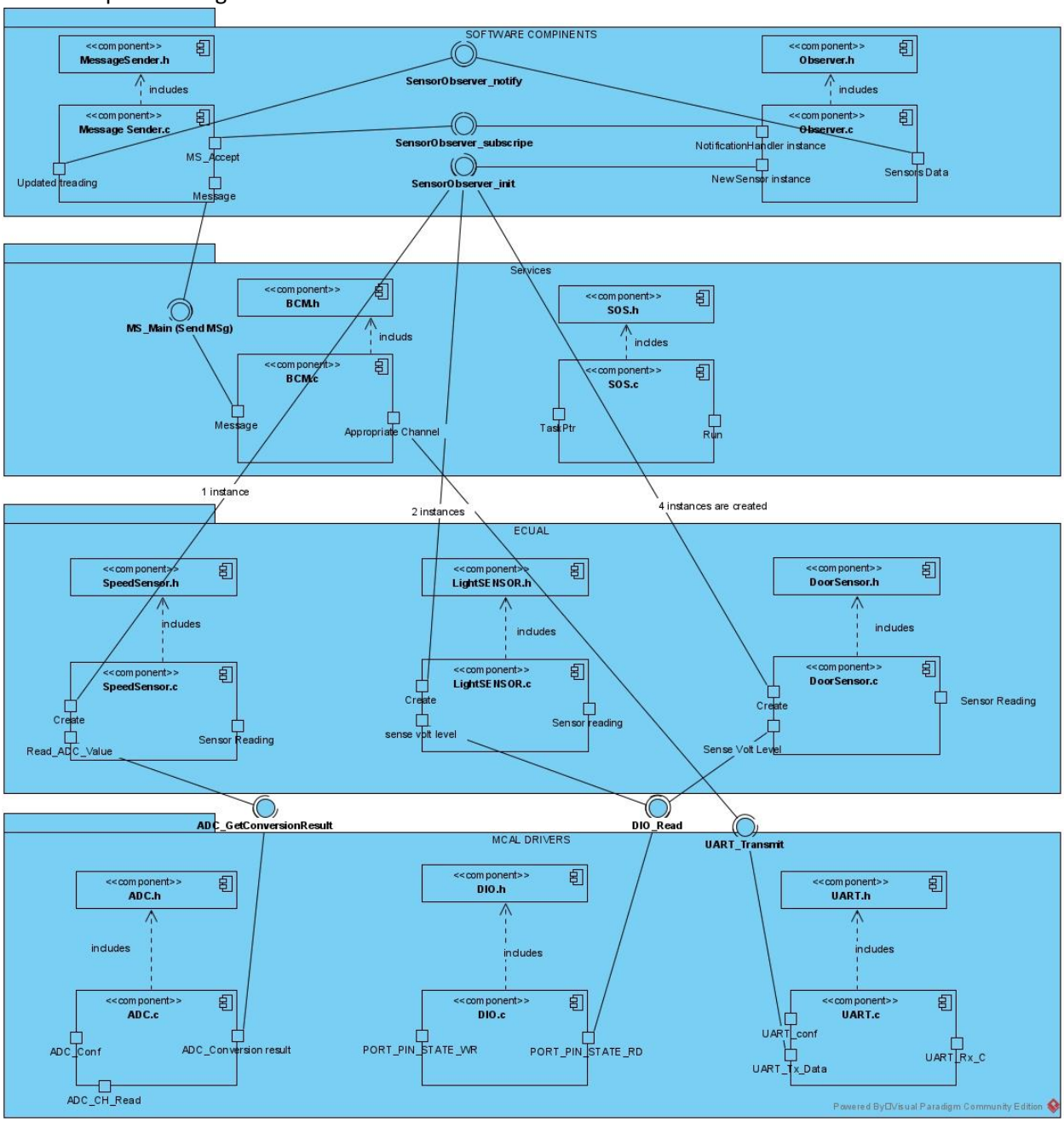
| | | | |
|---|---|---|---|
| Function name | ACT_main | | |
| Arguments | Input | | |
| | Output | | |
| | Input / Output | | |
| Return | E_OK | 1 | |
| | E_NOT_OK | 0 | |

| Description | Periodic function used to update the status of the buzzer and light actuators based on the readings from MR_GetLightCMD && MR_GetBuzzerCMD. |
|---|---|

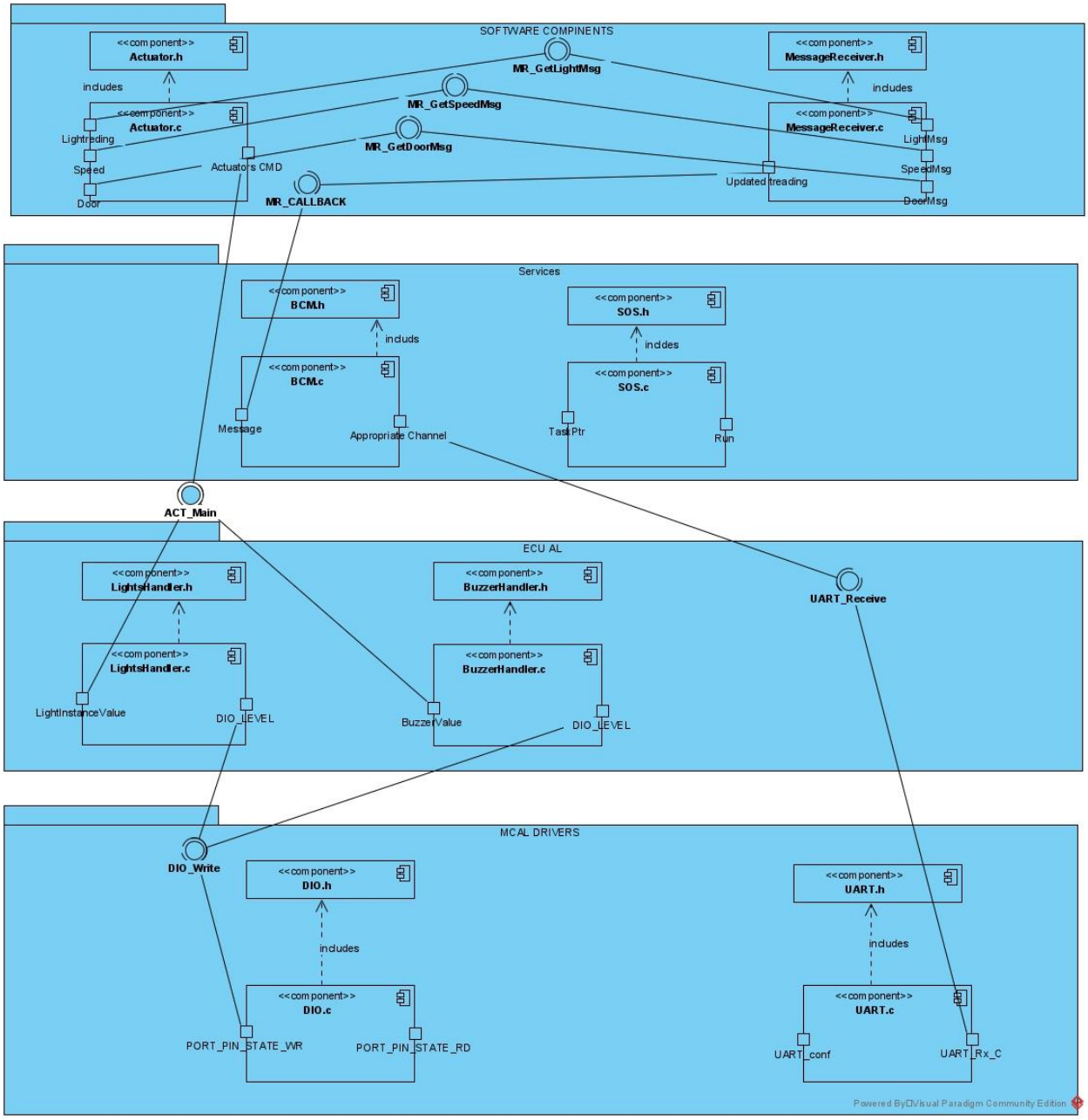
## 3. Dynamic design

### a. Component diagrams

- ECU1 component diagram

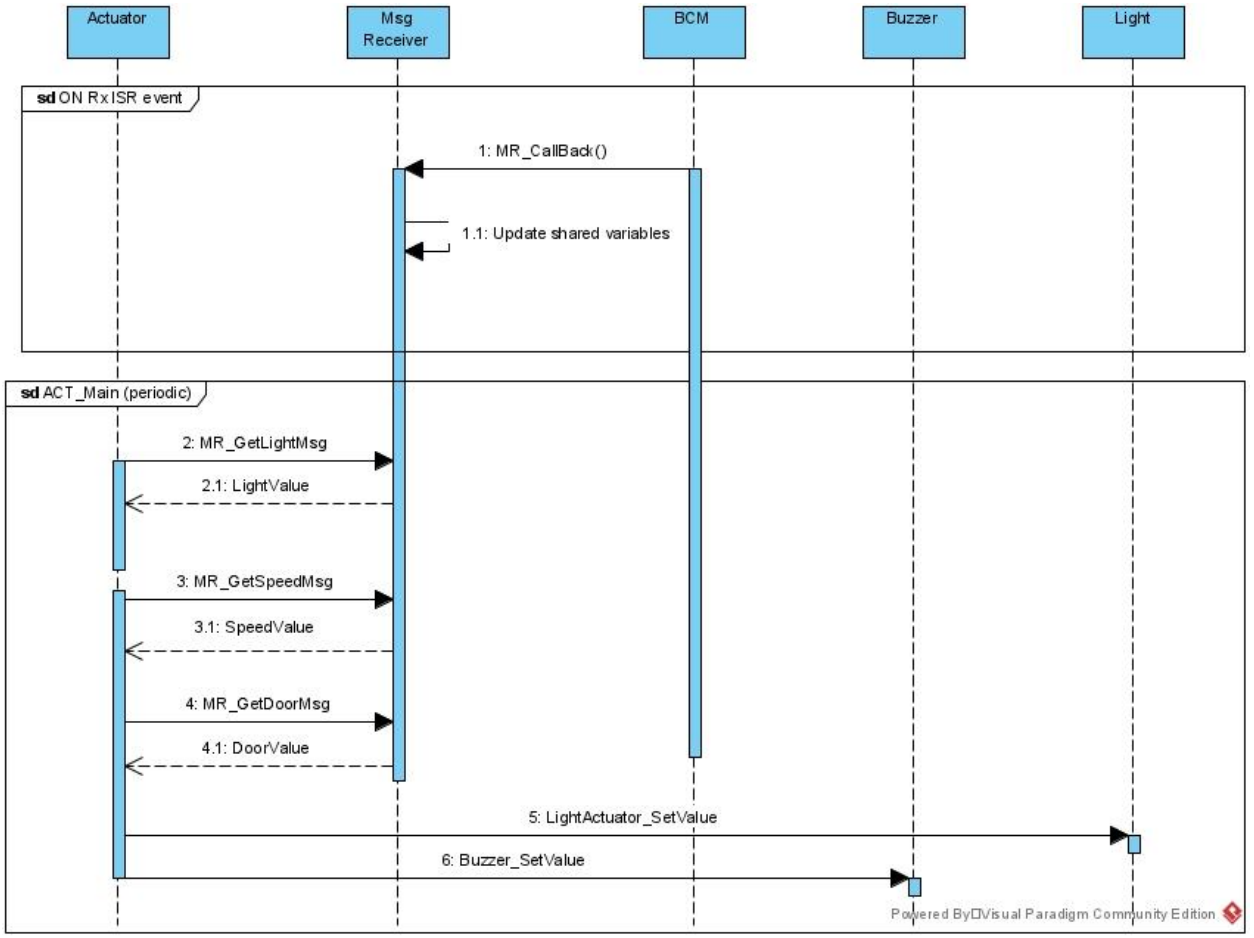


- ECU2 Component Diagram

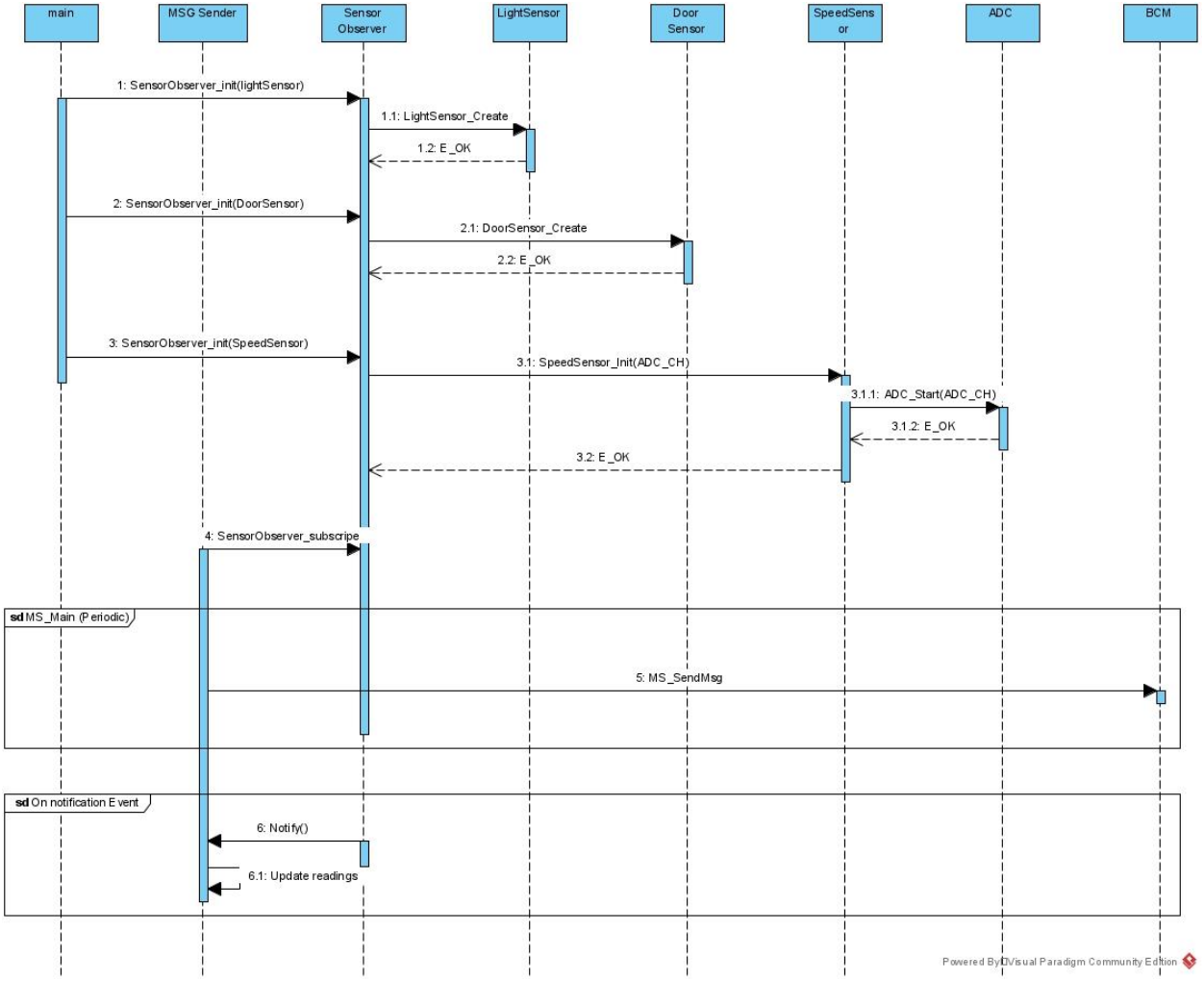


## b. Sequence diagrams

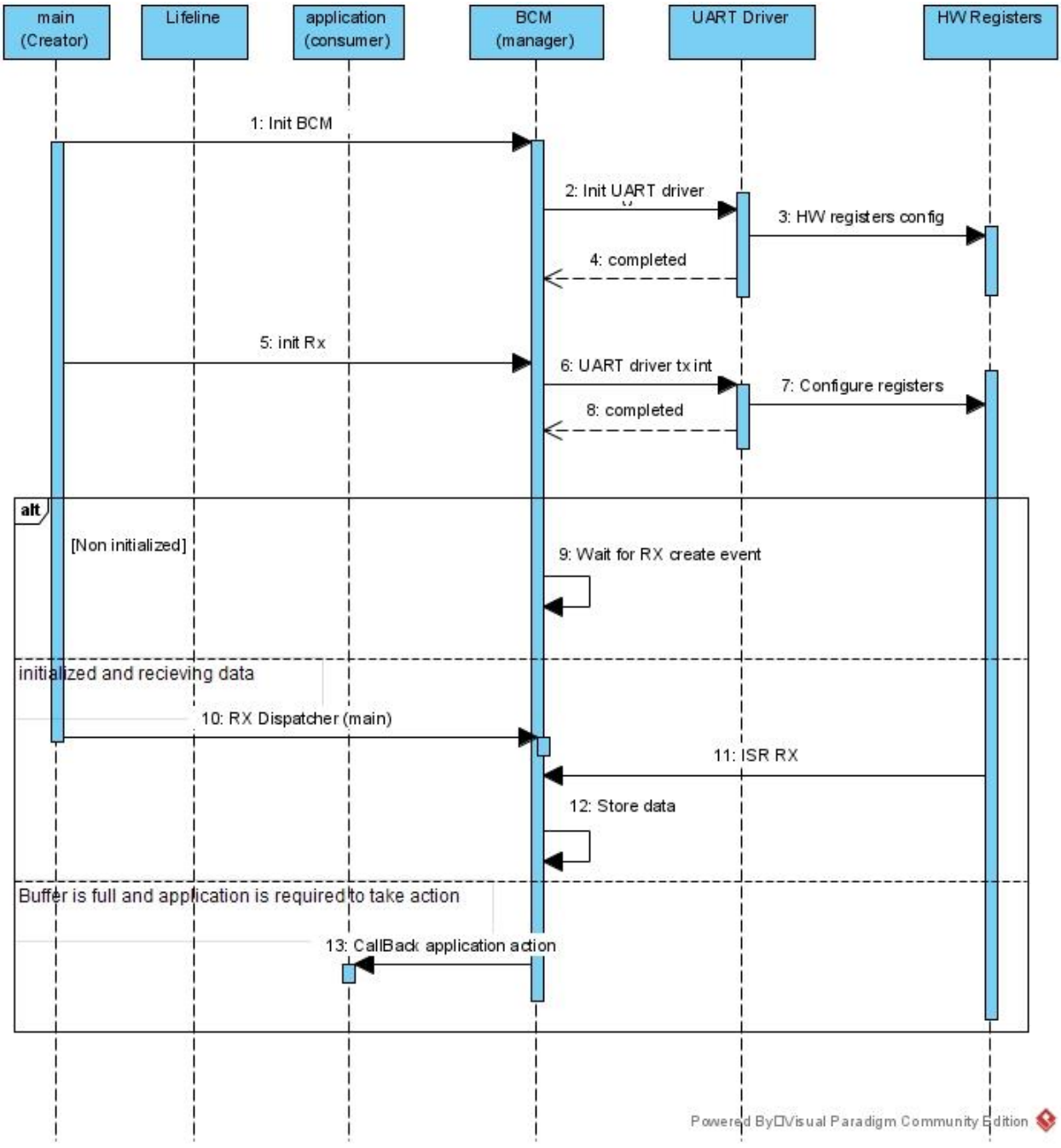
- *Actuator / Receiver sequence*



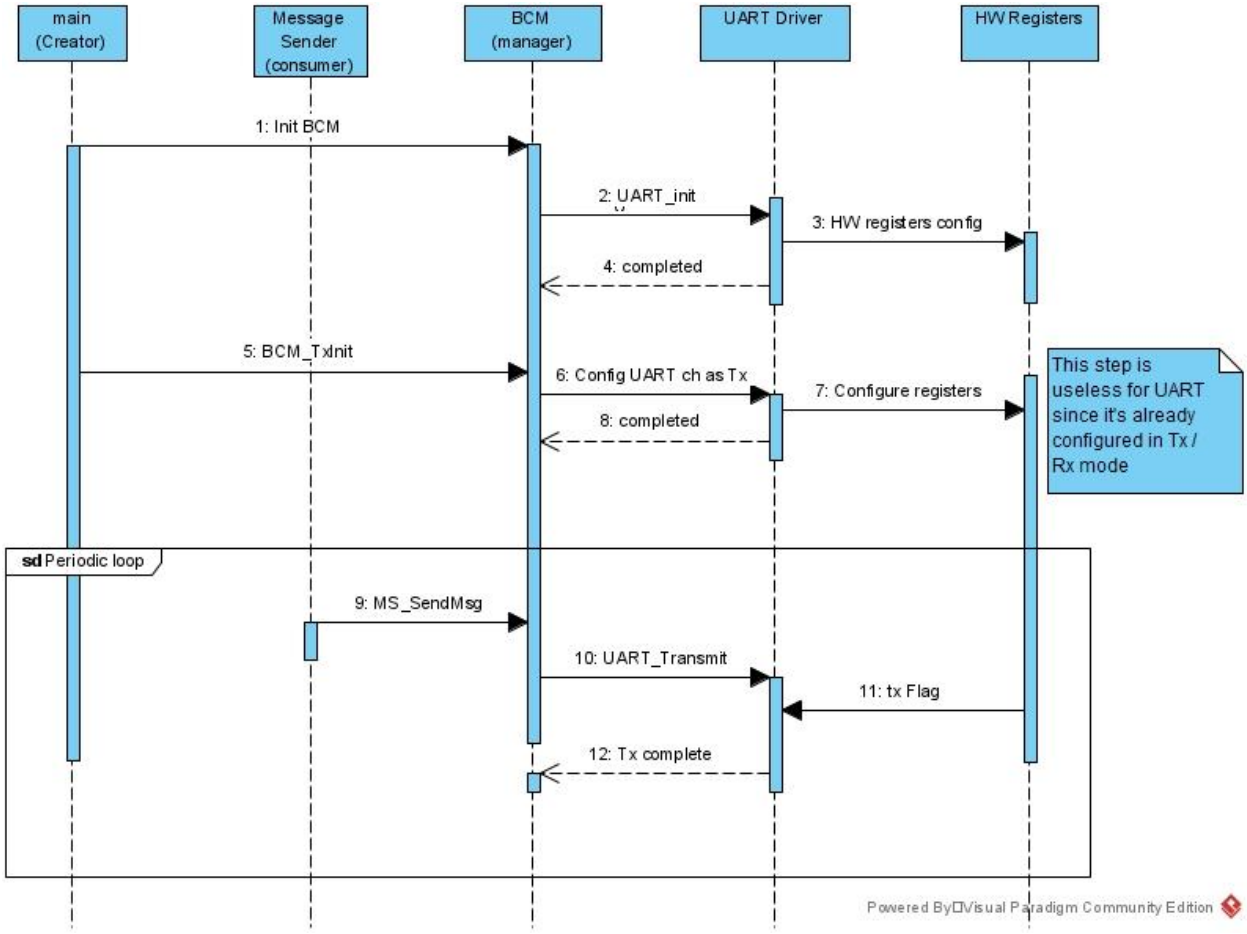
- *Observer / Sender sequence*



- *BCM RX*

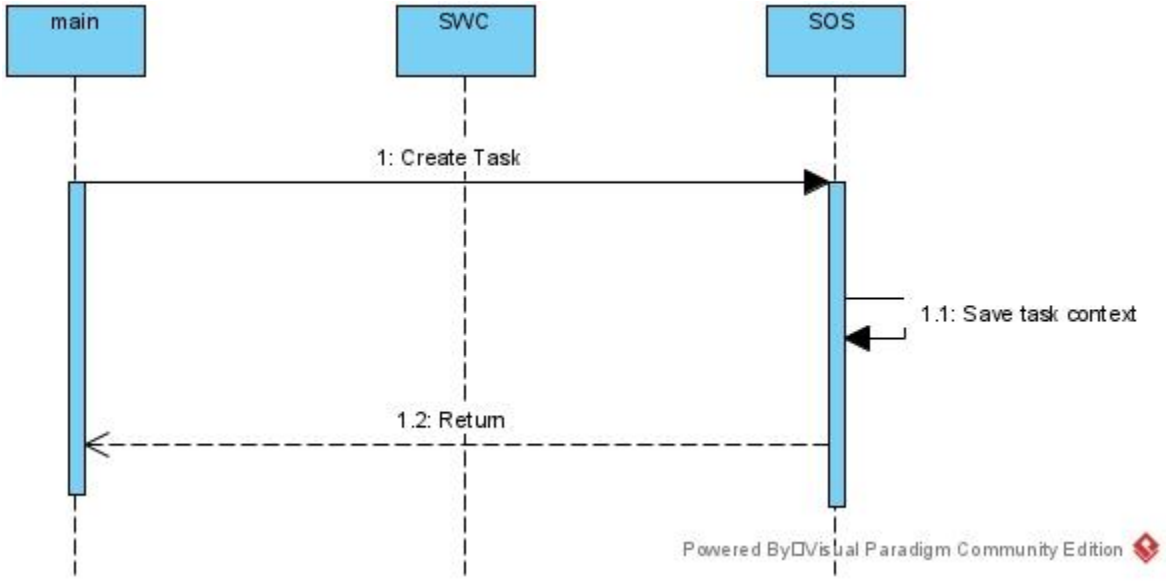

main
(Creator)
Lifeline
application
(consumer)
BCM
(manager)
UART Driver
HW Registers
1: Init BCM
2: Init UART driver
3: HW registers config
4: completed
5: init Rx
6: UART driver tx int
7: Configure registers
8: completed
alt
[Non initialized]
9: Wait for RX create event
initialized and recieving data
10: RX Dispatcher (main)
11: ISR RX
12: Store data
Buffer is full and application is required to take action
13: CallBack application action
Powered By Visual Paradigm Community Edition

- *BCM Tx*

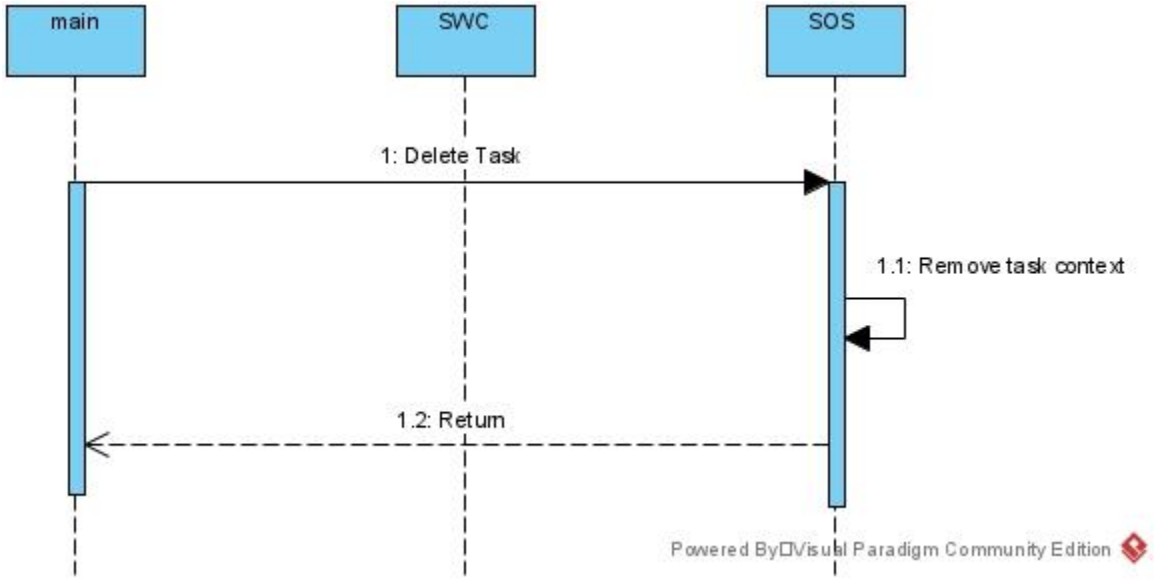



- *SO*S

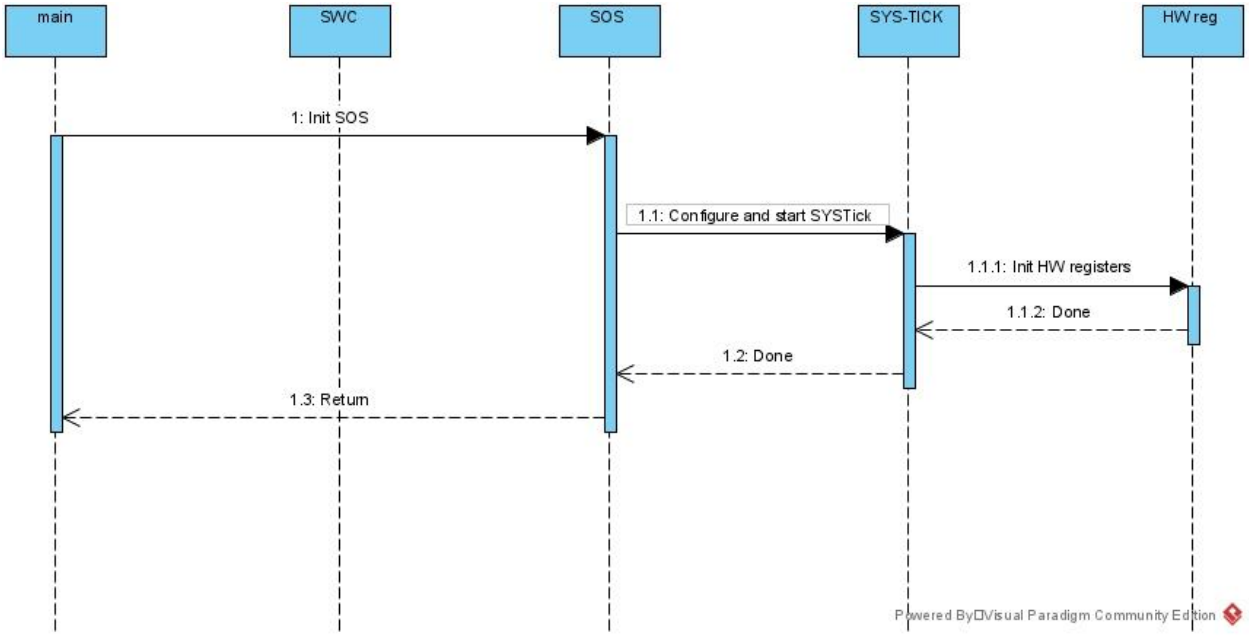
SOS Create Task



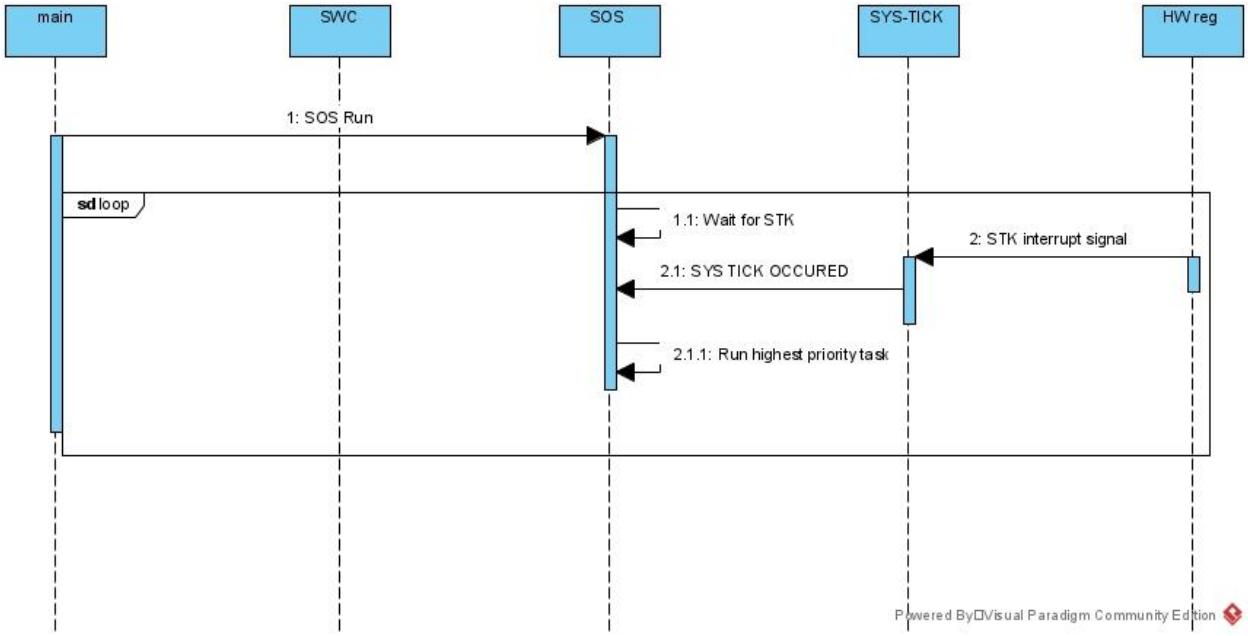
SOS Delete Task




SOS init

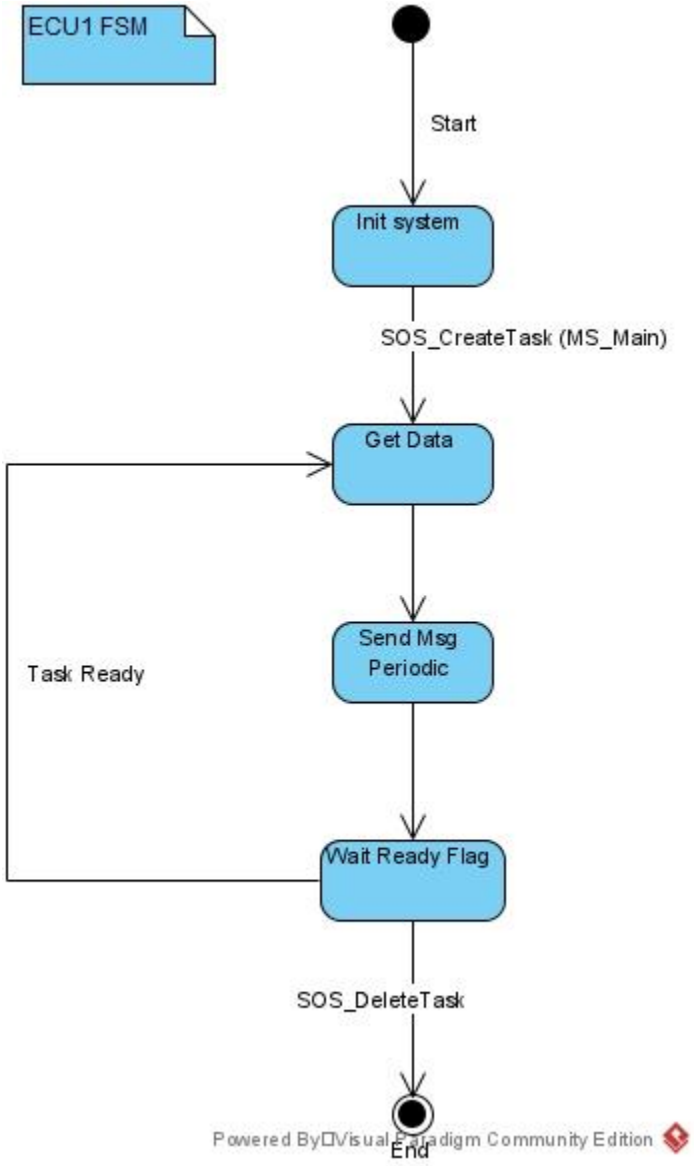


SOS Run

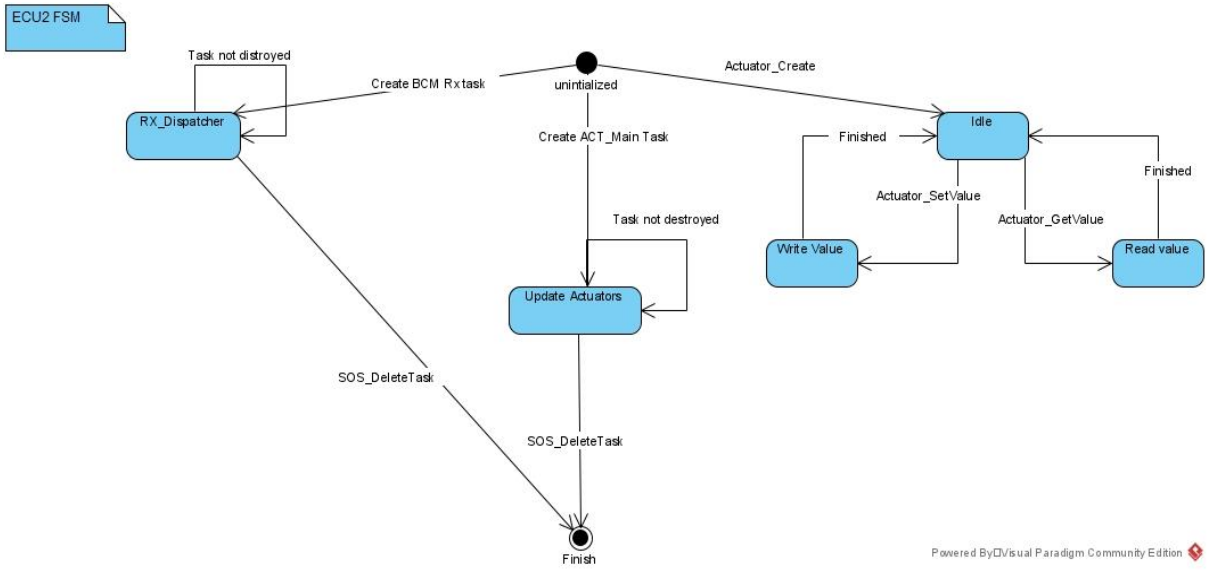

main
SWC
SOS
SYS-TICK
HW reg
1: SOS Run
sd loop
1.1: Wait for STK
2: STK interrupt signal
2.1: SYS TICK OCCURED
2.1.1: Run highest priority task
Powered By Visual Paradigm Community Edition

## c. FSM

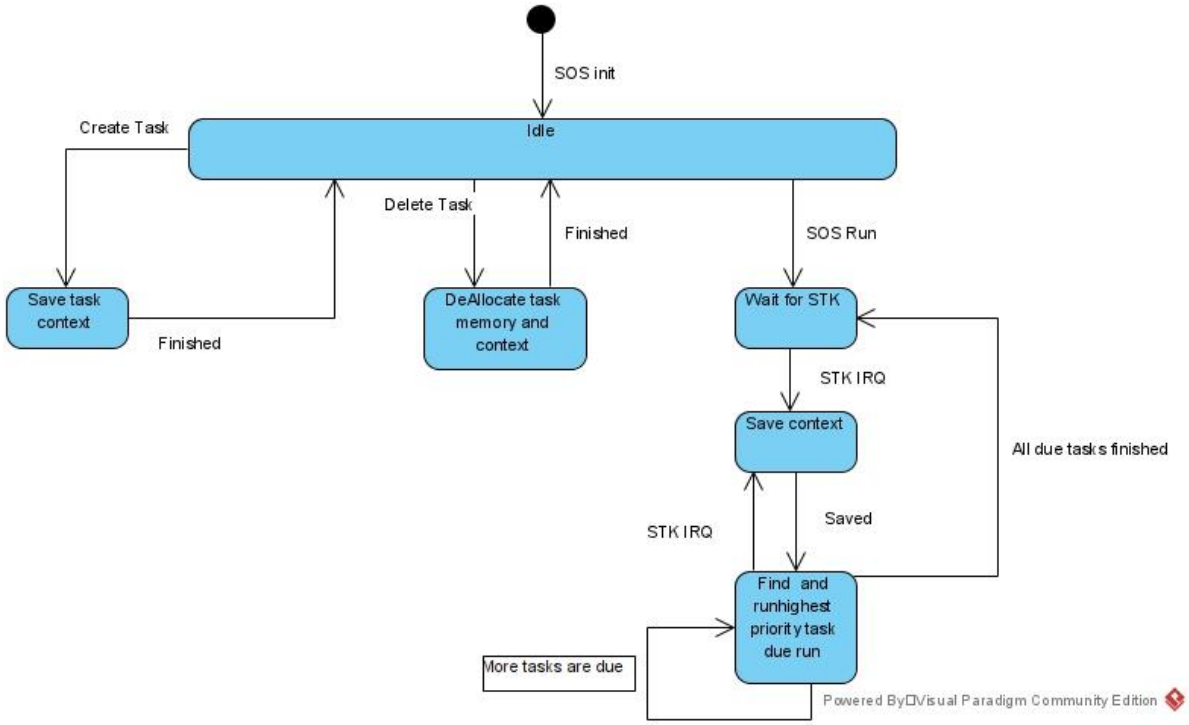
- ECU1


ECU1 FSM
Start
Init system
SOS_CreateTask (MS_Main)
Get Data
Send Msg Periodic
Task Ready
Wait Ready Flag
SOS_DeleteTask
End
Powered By Visual Paradigm Community Edition

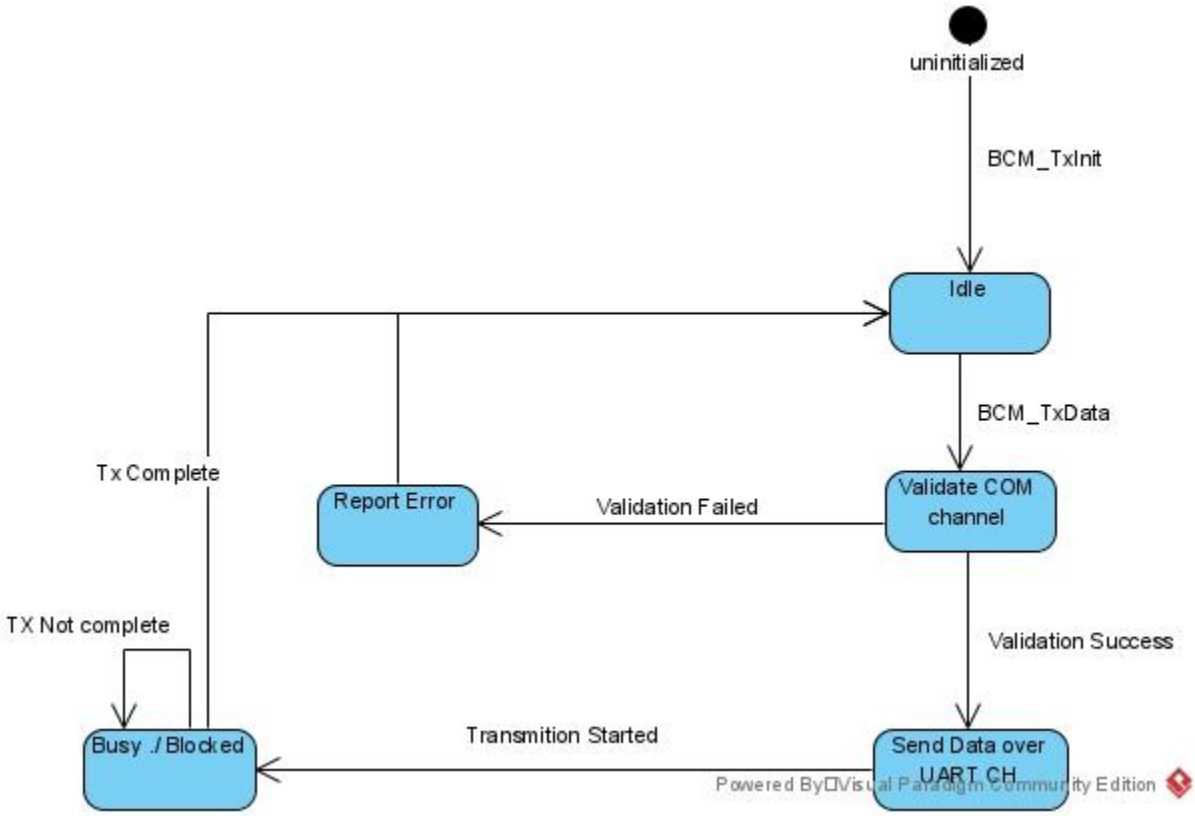
- ECU2


ECU2 FSM
Task not distroyed
Create BCM Rx task
unintialized
Actuator_Create
RX_Dispatcher
Create ACT_Main Task
Finished
Idle
Finished
Actuator_SetValue
Actuator_GetValue
Task not destroyed
Write Value
Read value
Update Actuators
SOS_DeleteTask
SOS_DeleteTask
Finish
Powered By Visual Paradigm Community Edition

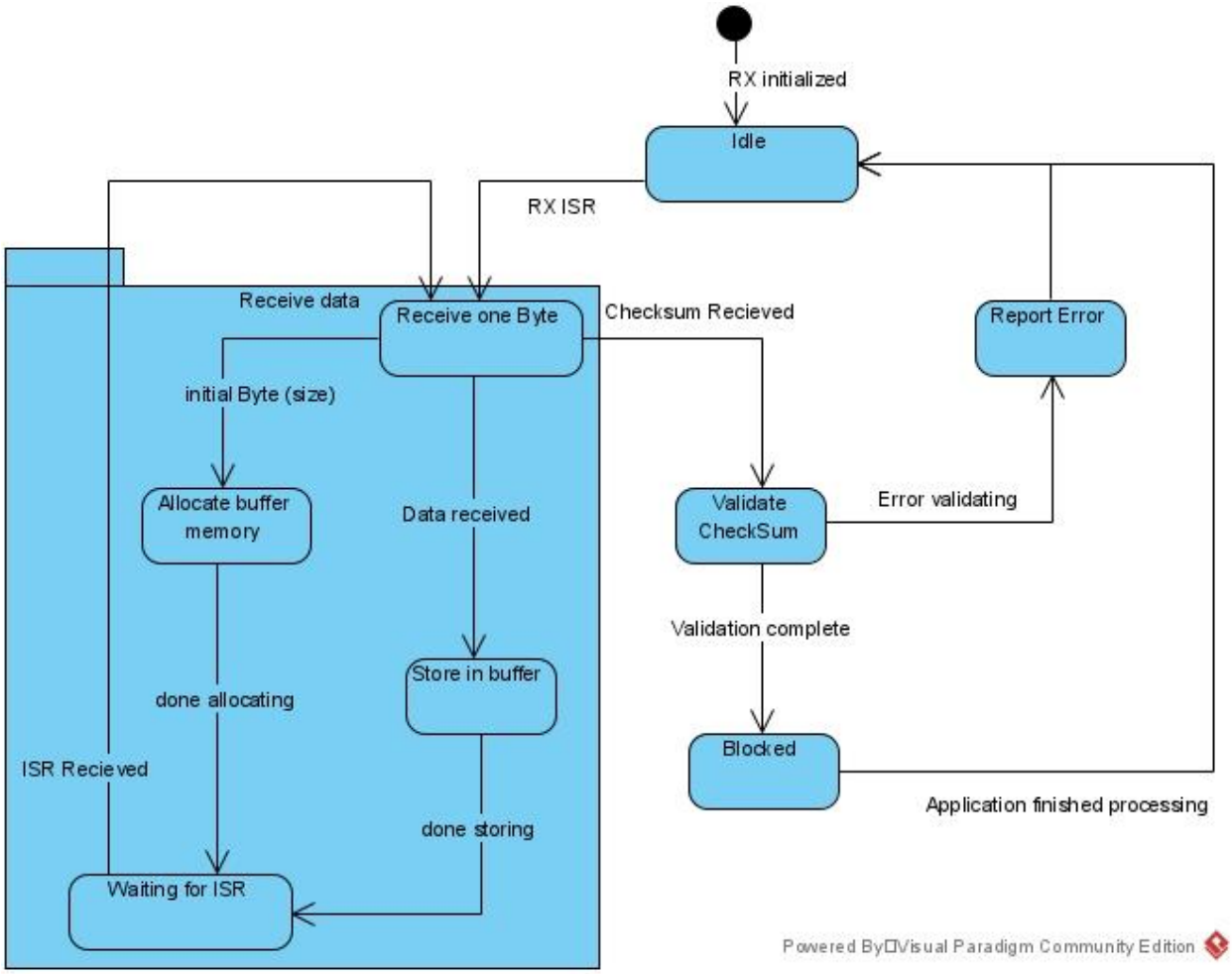
- SOS


SOS init
Idle
Create Task
Delete Task
Finished
SOS Run
Save task context
Finished
DeAllocate task memory and context
Wait for STK
STK IRQ
Save context
All due tasks finished
Saved
STK IRQ
Find and runhighest priority task due run
More tasks are due
Powered By Visual Paradigm Community Edition

- BCM Transmit



- BCM Receiver




## d. Folder Structure

- *ECU1*

- Documentation
- SW
  - BSW
    - MCAL
      - DIO
      - UART
      - ADC
    - ECUAL
      - LightSensor
      - SpeedSensor
      - DoorSensor
    - Services
      - SOS
      - BCM
  - COMMON

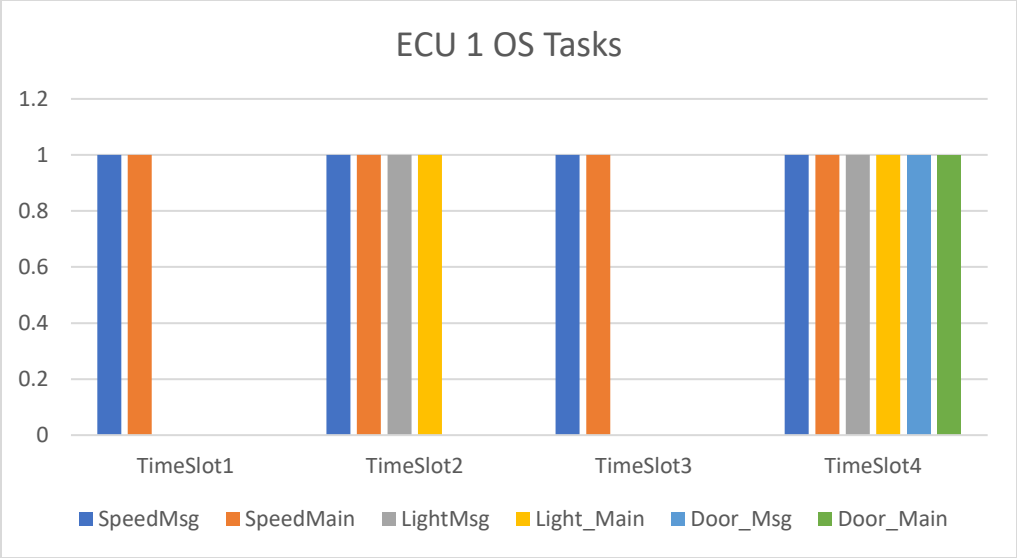
        - STD_Types
        - Bit_Math
        - CompilerAbstraction
        - PlatformTypes
    - SWC
        - MessageSender
        - Observer
- Build
    - MakeFile

- *ECU2*
- Documentation
- SW
    - BSW
        - MCAL
            - DIO
            - UART
        - ECUAL
            - LightsHandler
            - BuzzerHandler
        - Services
            - SOS
            - BCM
    - COMMON
        - STD_Types
        - Bit_Math
        - CompilerAbstraction
        - PlatformTypes
    - SWC
        - MessageReceiver
        - Actuator
- Build
    - MakeFile

## e. CPU Load and Bus Load

### i. ECU1

According to system requirements ECU1 will send at least one message every 5ms Time slots can be represented as follow given that a system tick = 5ms

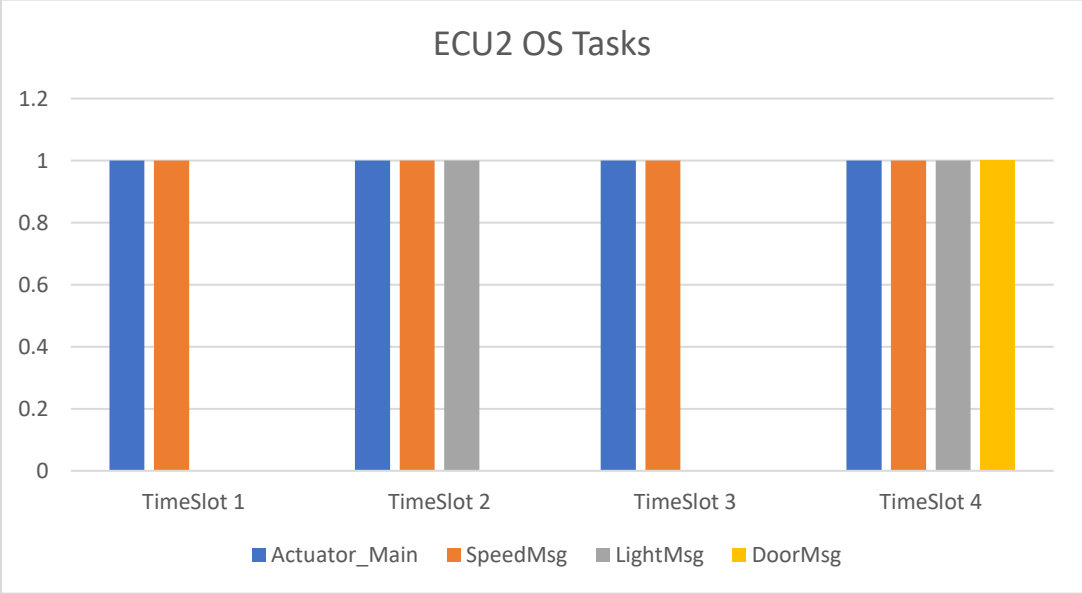



Assuming that each msg is processed in 100 us and sent over UART network in 100us. The Cpu utilization over repetitive 4 time slots = ((400 + 800 + 400 + 1200) us /20 ms)* 100 % = 14% not considering wakeup and sleep events.

And Bus load will only equal the propagation time of messages
= ((100+200+100+300) us / 20 ms )*100% = 7%

### ii. ECU2

According to system requirements ECU2 will receive a message at least every 5 ms thus it can be represented as follows:




It only processes the messages just like ECU1 so the load on CPU and bus is almost the same.