# Embedded SW Design

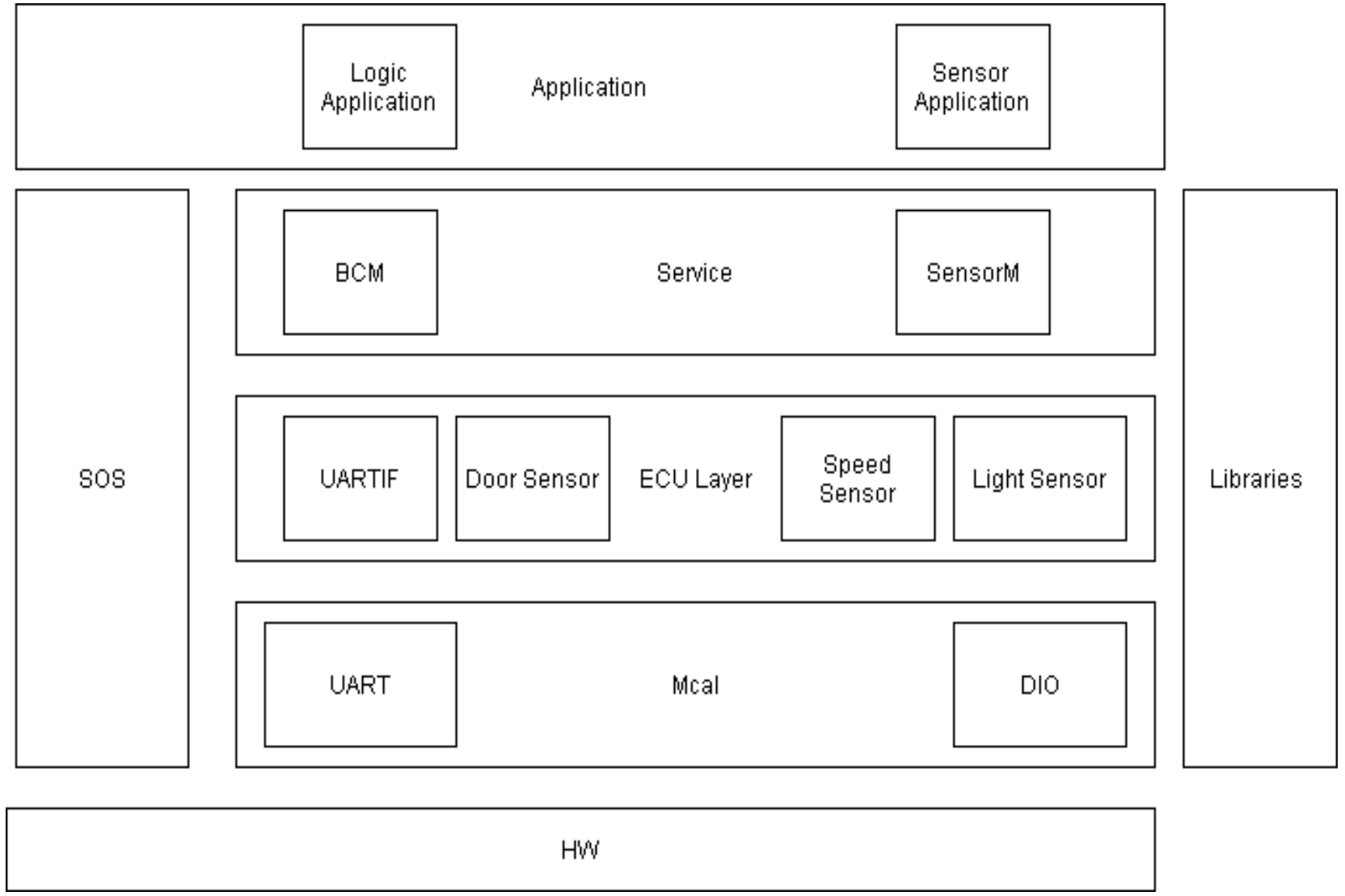
FINAL PROJECT

# System Description

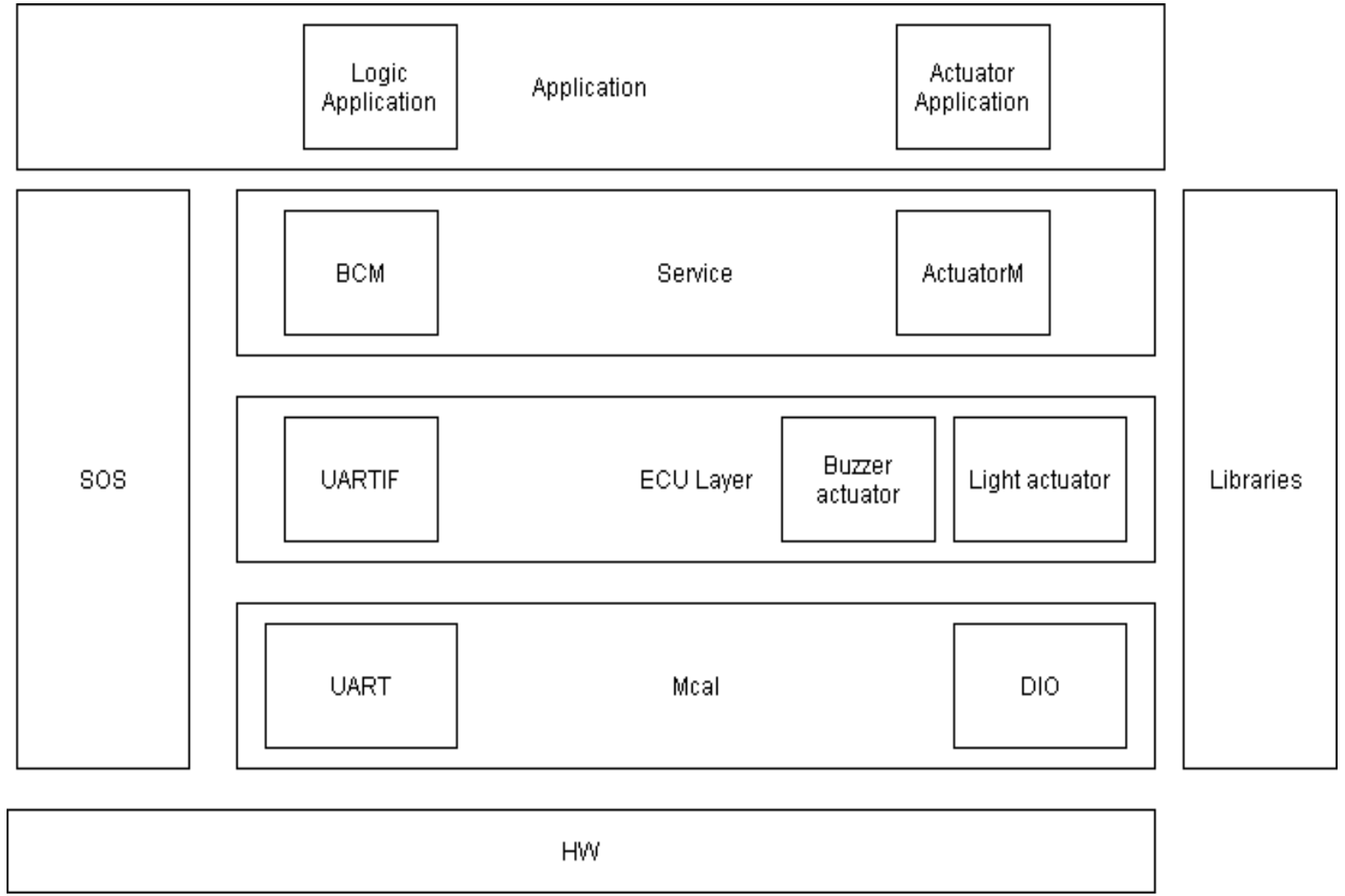
This system is supposed to take reading from 3 sensors on ECUA and send the reading results to ECUB by UART communication that take action based on these reading which are setting and resetting buzzer and light.

# Static Analysis

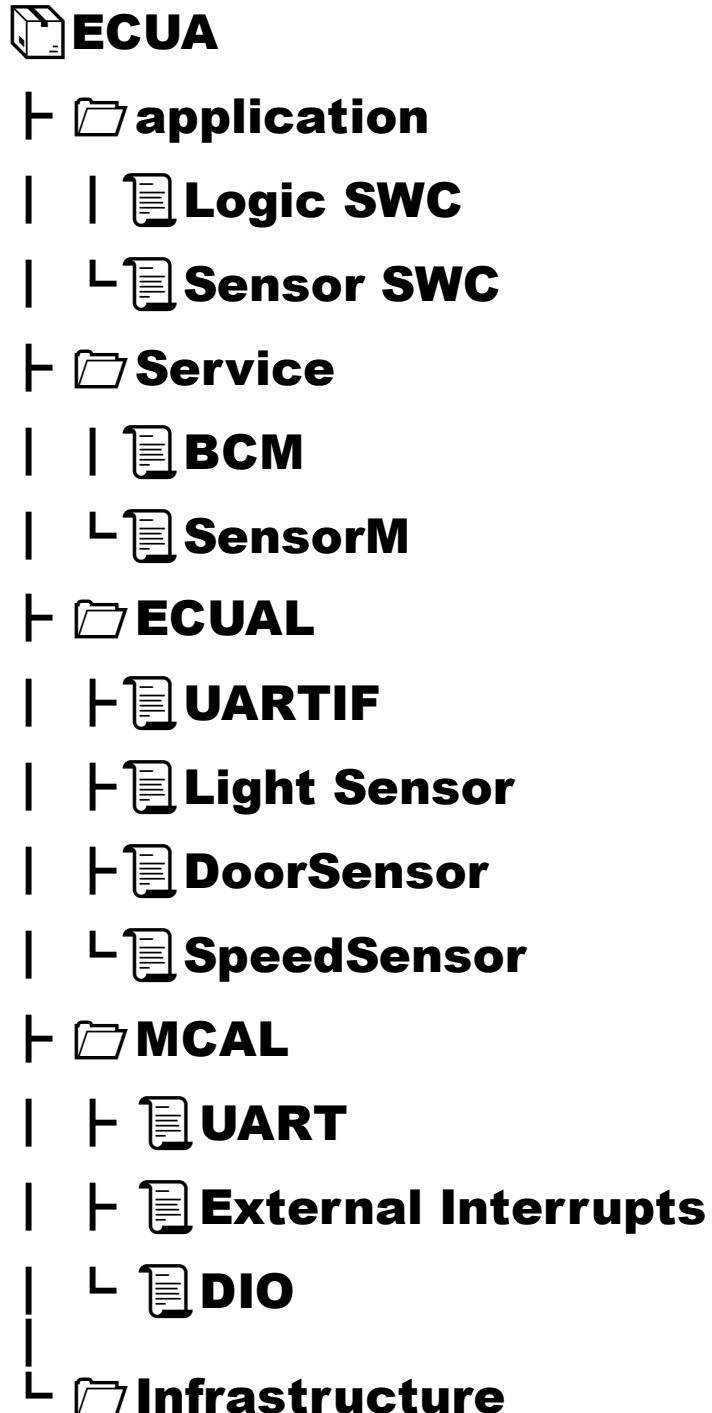
# Layered Architecture ECUA



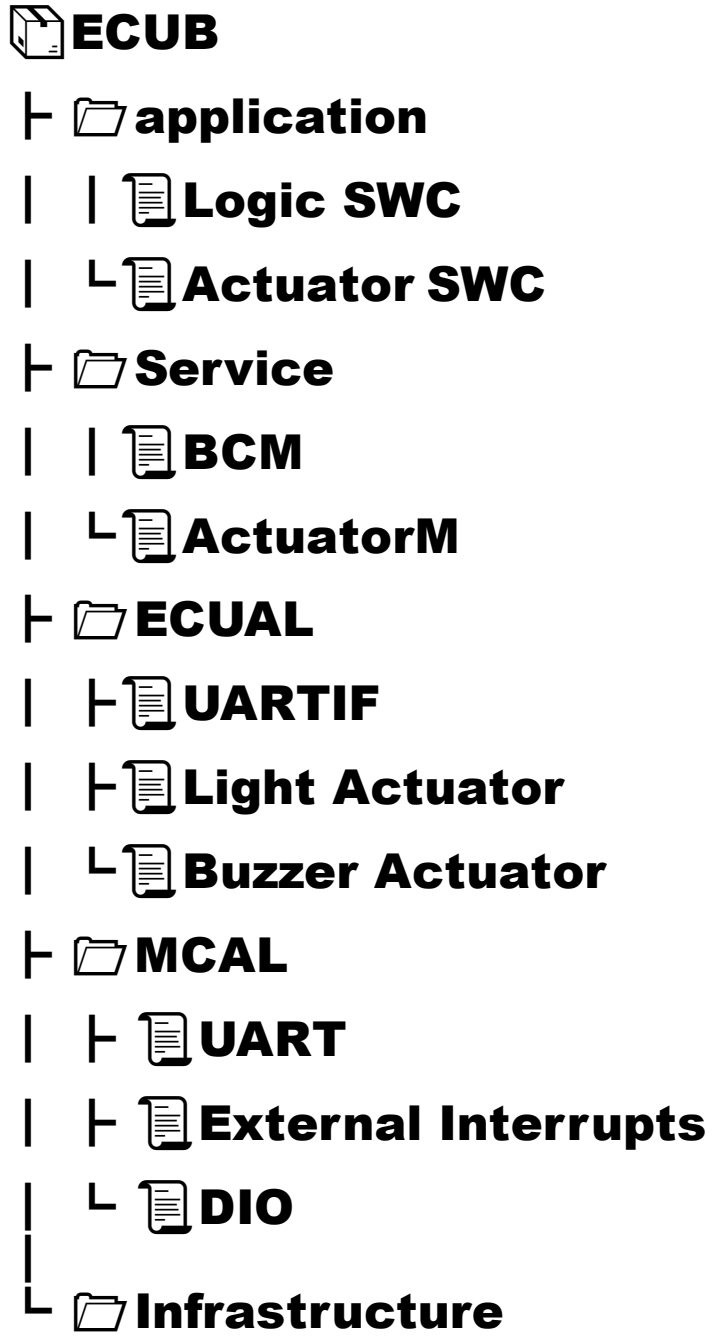
# Layered Architecture ECUB



# Folder Structure ECUA



# Folder Structure ECUB



# SW Data Type Tables

# DIO Data types :

| Name | DIO_PinType |
|---|---|
| Type | Enumeration |
| Range | 0-31 |
| Description | it define the pin number used in the API |

| Name | DIO_LevelType |
|---|---|
| Type | Enumeration |
| Range | 0-1 |
| Description | : it define the level used on the pin. |

# BCM Data types :

| Name | BCM_ConfigType |
|---|---|
| Type | Struct |
| Range | none |
| Description | Configuration of the BCM. |
| Content | BCM_BufferSize<br>BCM_BuadRate |

| Name | BCM_DataType |
|---|---|
| Type | uint8 |
| Range | none |
| Description | data sent or received by BCM. |

# SensorM Data types :

| Name | SensorM_ConfigType |
| --- | --- |
| Type | Struct |
| Range | none |
| Description | Configuration of the SensorM. |
| Content | SensorM_NumSensors<br>SensorM_Mode |

# ActuatorM Data types :

| Name | ActuatorM_ConfigType |
| --- | --- |
| Type | Struct |
| Range | none |
| Description | Configuration of the ActuatorM. |
| Content | ActuatorM_NumSensors<br>ActuatorM_Mode |

# SW Layers

# MCAL Layer

# DIO APIs:

- DIO_Init()
- DIO_Read()
- DIO_Write()

# DIO_cfg.h:

```
typedef enum {
   PORTA,PORTB
}DIO_PortType;

typedef enum{
PINA0,PINA1,PINA2,PINA3,PINA4,PINA5,PINA6,PINA7,PINB0,PINB1,PINB
2,PINB3,PINB4,PINB5,PINB6,PINB7
}DIO_PinType;

typedef uint8 DIO_DataType

typedef struct{
DIO_PortType Port;
DIO_PinType Pin;
DIO_DataType Data;

}DIO_ConfigType;
```

# Dio.h:

```
#include "DIO_cfg.h"

DIO_Init(DIO_ConfigType *Cfg) ;

DIO_Read(DIO_PinType Pin ,DIO_DataType *  copy_u8Data);

DIO_Write(DIO_PinNum Pin,DIO_DataType copy_u8Data);
```

# APIs Pseudo Code:

```
#include "DIO.h"

error_status DIO( DIO_ConfigType cfg){

    Configure the Port and Pin Directions;
}
error_status DIO_Read(DIO_PinType Pin,DIO_DataType * readValue){

    *readValue=Portx;
}
error_status DIO_Write(DIO_PinType Pin,DIO_DataTypereadValue){

    Portx=writeValue;
}
```

# APIs Description:

| Function name: | DIO_Init |
| --- | --- |
| Arguments: | • Input: Pin( DIO_PinType ).<br>• Output: none.<br>• Input/output: none. |
| Return | E_OK(0),E_NOK(1) |
| Description | initialize the DIO module |
| Synchronous / Asynchronous | synchronous |
| Reentrant/Non-Reentrant | Non-Reentrant |

# APIs Description:

| Function name: | DIO_Read |
| --- | --- |
| Arguments: | • Input: Pin(DIO_PinType)<br>• Output: Level(DIO_LevelType)<br>• Input/output: none |
| Return | E_OK(0),E_NOK(1) |
| Description | Reads the pin value |
| Synchronous / Asynchronous | synchronous |
| Reentrant/Non-Reentrant | Reentrant |

# APIs Description:

| Function name: | DIO_Write |
|---|---|
| Arguments: | • Input: Pin ( DIO_PinType ), Level ( DIO _LevelType ).<br>• Output: none.<br>• Input/output: none. |
| Return | E_OK(0),E_NOK(1) |
| Description | Set the Pin value . |
| Synchronous / Asynchronous | synchronous |
| Reentrant/Non-Reentrant | Reentrant |

# UART APIs:

- UART_Init()
- UART_Start()
- UART_Stop()

# UART_cfg.h:

```
typedef enum {
   UART0,UART1
}UART_ChannelType;

typedef enum{
PRE_1,PRE_8,PRE_256
}UART_PrescalerType;

typedef uint32 UART_Freq ;

typedef struct{
UART_ChannelType Channel;
UART_Freq freg;
UART_PrescalerType Pre;

}UART_ConfigType;
```

# UART.h:

```
#include "UART_cfg.h"


UART_Init();

UART_Send();

UART_Receive();
```

# APIs Pseudo Code:

```
#include "UART.h"
error_status UART_Init(Config){

    Configure the UART;
}
error_status UART_Send(void){

Check buffer full;
Put data on Uart Send Regester ;
}
error_status UART_Receive(void){

Check buffer empty;
Get data from Receive Buffer;
}
```

# APIs Description:

| Function name: | UART_Init |
| --- | --- |
| Arguments: | Input:Config (UART_ConfigType) |
| Return | E_OK(0),E_NOK(1) |
| Description | Intialize the UART |
| Synchronous / Asynchronous | synchronous |
| Reentrant/Non-Reentrant | Non-Reentrant |

# APIs Description:

| Function name: | UART_Send |
|---|---|
| Arguments: | Input:UART_DataType |
| Return | E_OK(0),E_NOK(1) |
| Description | Send Data |
| Synchronous / Asynchronous | synchronous |
| Reentrant/Non-Reentrant | Reentrant |

# APIs Description:

| Function name: | UART_Receive |
|---|---|
| Arguments: | Output:UART_DataType |
| Return | E_OK(0),E_NOK(1) |
| Description | Receive Data |
| Synchronous / Asynchronous | synchronous |
| Reentrant/Non-Reentrant | Reentrant |

# ECU Layer

- Light Sensor
- Door Sensor
- Speed Sensor
- Buzzer actuator
- Light actuator
- UART interface

# xSensor APIs:

- xSensor_Init()
- xSensor_Start()
- xSensor_Stop()

# xSensor_cfg.h:

```
typedef enum {
   X0,X1,X2,X3
}xSensor_ChannelType;


typedef uint32 Sensor_Speed

typedef struct{
xSensor_ChannelType Channel;
xSensor_Speed freg;

}xSensor_ConfigType;
```

# XSensor header Files:

```
#define "xSensor_cfg.h"

xSensor_Init()

 xSensor_Start()

xSensor_Stop()
```

# APIs Pseudo Code:

```
error_status xSensor_Init( Config){

    Configure the Sensor;
}
error_status
      xSensor_Start(void){   start
      Sensor;
}
error_status
    xSensor_stop(void){   Stop
    Sensor;
}
```

# APIs Description:

| Function name: | xSensor_Init |
| --- | --- |
| Arguments: | • Input: Config(xSensor_ConfigType).<br>• Output: none.<br>• Input/output: none. |
| Return | E_OK(0),E_NOK(1) |
| Description | Init Sensor |
| Synchronous / Asynchronous | synchronous |
| Reentrant/Non-Reentrant | Non-Reentrant |

# APIs Description:

| Function name: | xSensor_Start |
|---|---|
| Arguments: | none |
| Return | E_OK(0),E_NOK(1) |
| Description | readSensor |
| Synchronous / Asynchronous | synchronous |
| Reentrant/Non-Reentrant | Reentrant |

# APIs Description:

| Function name: | xSensor_Stop |
|---|---|
| Arguments: | none |
| Return | E_OK(0),E_NOK(1) |
| Description | Stops the Sensor |
| Synchronous / Asynchronous | synchronous |
| Reentrant/Non-Reentrant | Reentrant |

# Service Layer

- BCM
- SensorM
- ActuatorM

# BCM APIs:

- BCM_Init
- BCM_Send
- BCM_Setup_RxBuffer
- BCM_RxMainFunction
- BCM_TxMainFunction

# BCM_cfg.h:

Typedef uint8 ComM_DataType

#define BCM_BUFFER_SIZE 1000

#define BCM_HEADER_SIZE 2

# BCM header Files:

- uint8_t BCM_Init();
- uint8_t BCM_Send();
- uint8_t BCM_Setup_RxBuffer();
- uint8_t BCM_RxDispatcher();
- uint8_t BCM_TxDispatcher();

# APIs Pseudo Code:

```
uint8_t BCM_Init(){
   UART_init(&uartConfig);
   UART_TX_SetCallBack(functionTx);
   UART_RX_SetCallBack(functionRx);
   return 1;
}
uint8_t BCM_Send(uint8_t * arr,uint32_t size){

   UART_sendByte(ptBuffer[tPostion]);

}
uint8_t BCM_Setup_RxBuffer(uint8_t * arr,uint32_t size){
        arr=internalBuffer;

}
uint8_t BCM_RxDispatcher(){
   if(rPostion==rSize-1) return 1;
   else return 0;
}
uint8_t BCM_TxDispatcher(){

   if(tPostion==tSize-1) return 1;
   else return 0;
}
```

# APIs Description:

| Function name: | BCM_Init |
|---|---|
| Arguments: | none |
| Return | E_OK(0),E_NOK(1) |
| Description | Initialize BCM |
| Synchronous / Asynchronous | synchronous |
| Reentrant/Non-Reentrant | Non-Reentrant |

# APIs Description:

| Function name: | BCM_SetupRxBuffer |
| --- | --- |
| Arguments: | Output: BCM_DataType Data |
| Return | E_OK(0),E_NOK(1) |
| Description | Receive Data |
| Synchronous / Asynchronous | synchronous |
| Reentrant/Non-Reentrant | Reentrant |

# APIs Description:

| Function name: | BCM_Send |
|---|---|
| Arguments: | Input:BCM_dataType Data |
| Return | E_OK(0),E_NOK(1) |
| Description | Sends Data |
| Synchronous / Asynchronous | synchronous |
| Reentrant/Non-Reentrant | Reentrant |

# MotorM: APIs:

- **ActuatorM_Init()**
- **ActuatorM_SetValue()**

# MotorM_cfg.h:

```
typedef uint32     actuator_ID
#define NUMBER_OF_ACTUATORS 2
Typedef uint32     actuator_Status
```

# ActuatorM header Files:

#include "ActuatorM_Cfg.h"

ActuatorM_Init()

ActuatorM_SetValue()

# APIs Pseudo Code:

```
#include "ActuatorM.h"

error_status ActuatorM_Init( void){

    Configure the Actuator.
}
error_status ActuatorM_SetValue(){

  Set actuator value.
}
```

# APIs Description:

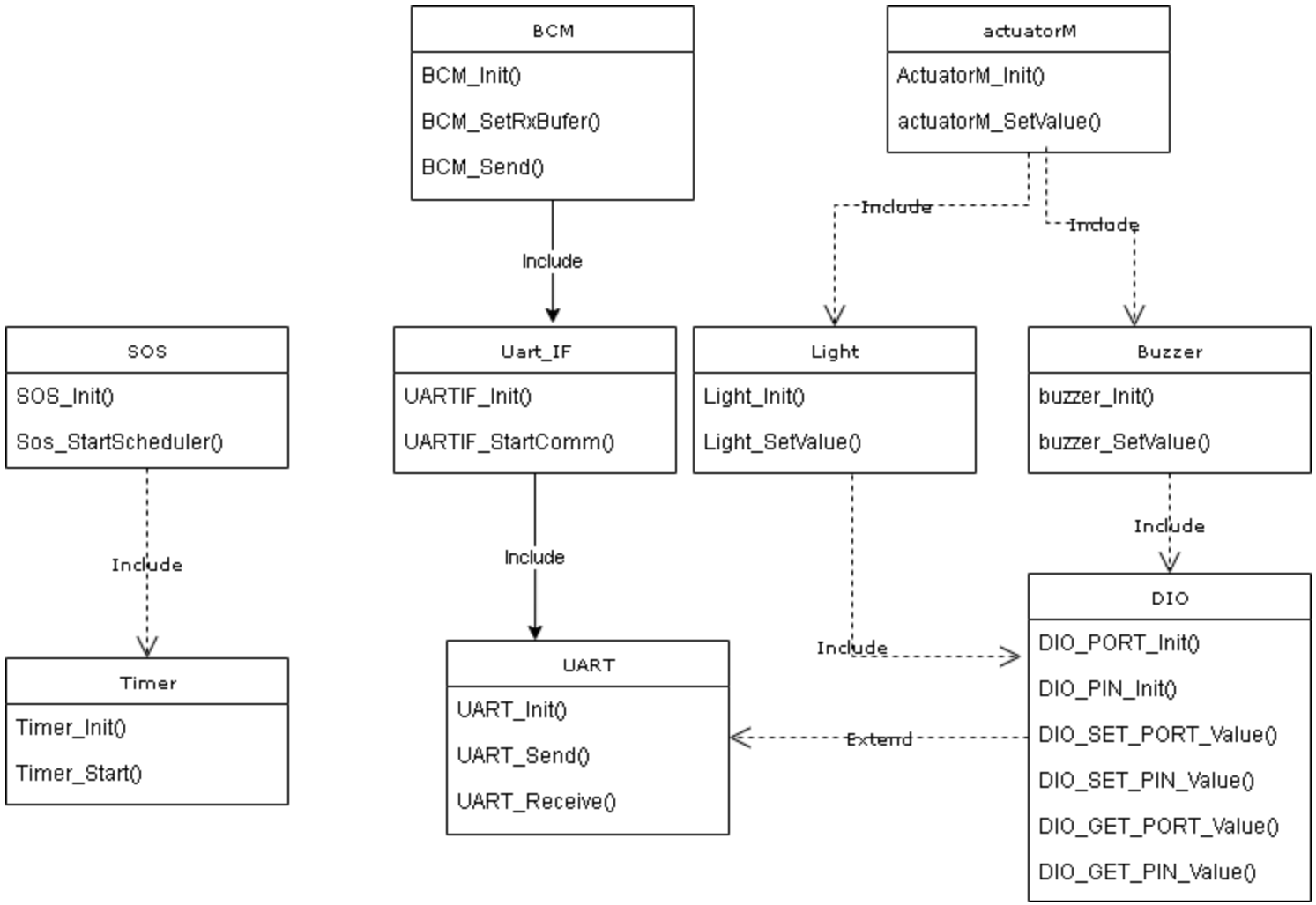
| Function name: | ActuatorM_Init() |
|---|---|
| Arguments: | none |
| Return | E_OK(0),E_NOK(1) |
| read the pin value | Init actuator manager |
| Synchronous / Asynchronous | synchronous |
| Reentrant/Non-Reentrant | Reentrant |

# APIs Description:

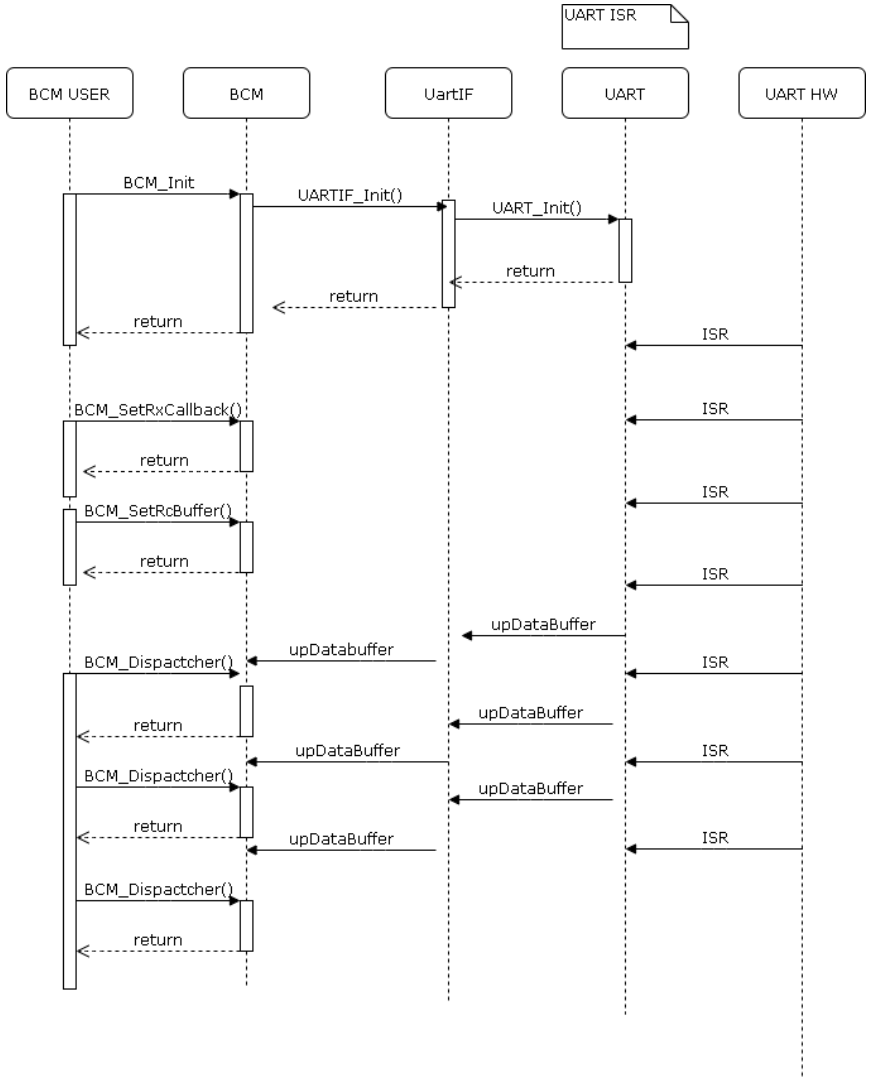
| Function name: | Actuator_SetValue |
| --- | --- |
| Arguments: | • Input: status(ActuatorM_DataType).<br>• Output:none.<br>• Input/output: none. |
| Return | E_OK(0),E_NOK(1) |
| Description | Set the value of actuator |
| Synchronous / Asynchronous | synchronous |
| Reentrant/Non-Reentrant | Reentrant |

# Dynamic Analysis

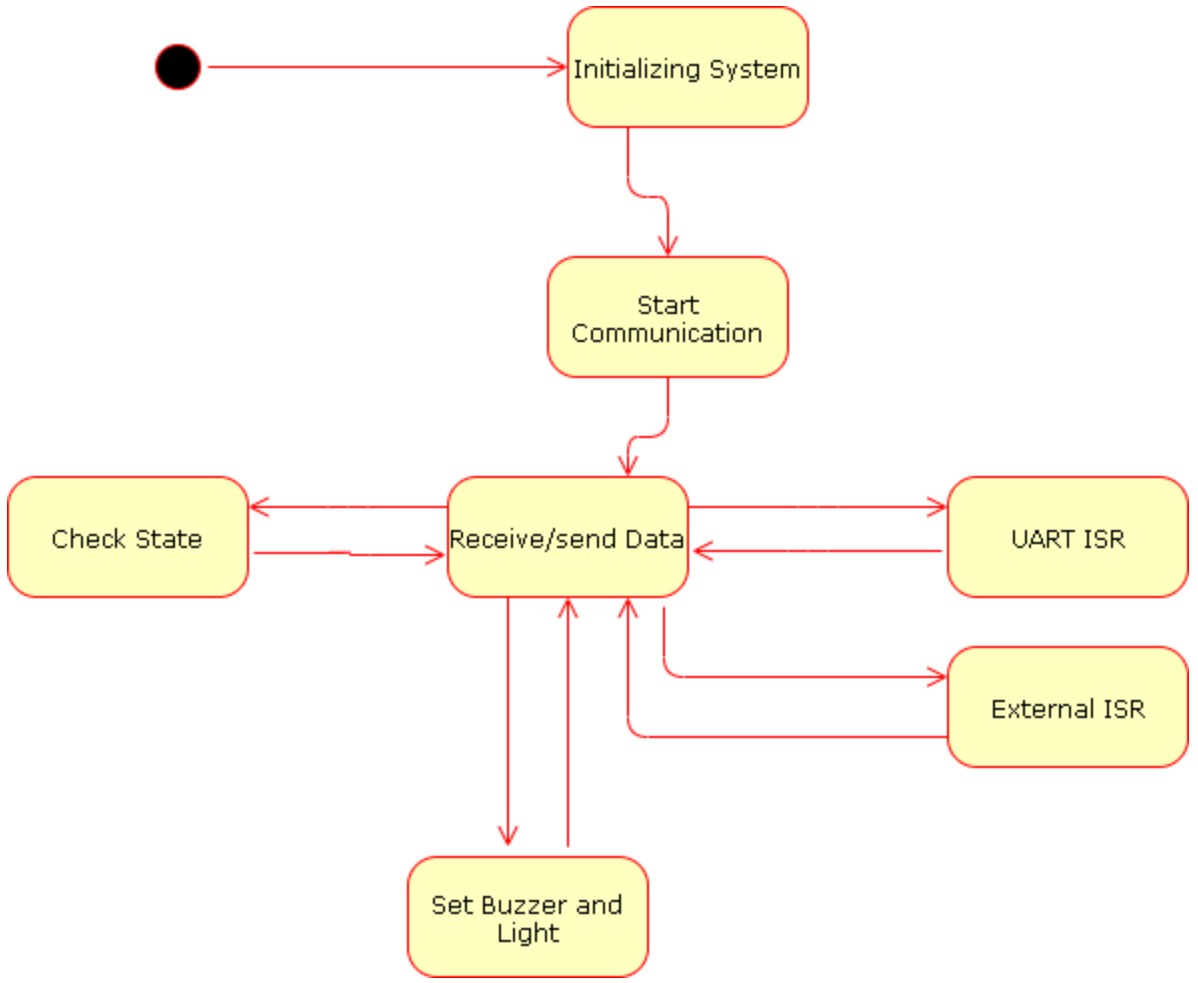
# Class Diagram



# Sequence Diagram



# State Diagram



# Design Patterns

- Interrupt Pattern : UART , Light Sensor, Speed Senosr , Door Sensor.
- Hardware Proxy Pattern : UART_IF.
- Debouncing Pattern : Light Sensor, Speed Senosr , Door Sensor.
- Observer Pattern:Buzzer actuator ,Light actuator.

# Tasks ECUA

- LogicTask

(Periodicity:50ms,Deadline:50ms,Execution Time:10ms,Priority:1).

- SensorTask

(Periodicity:20ms,Deadline:20ms,Execution Time:5ms,Priority:2).

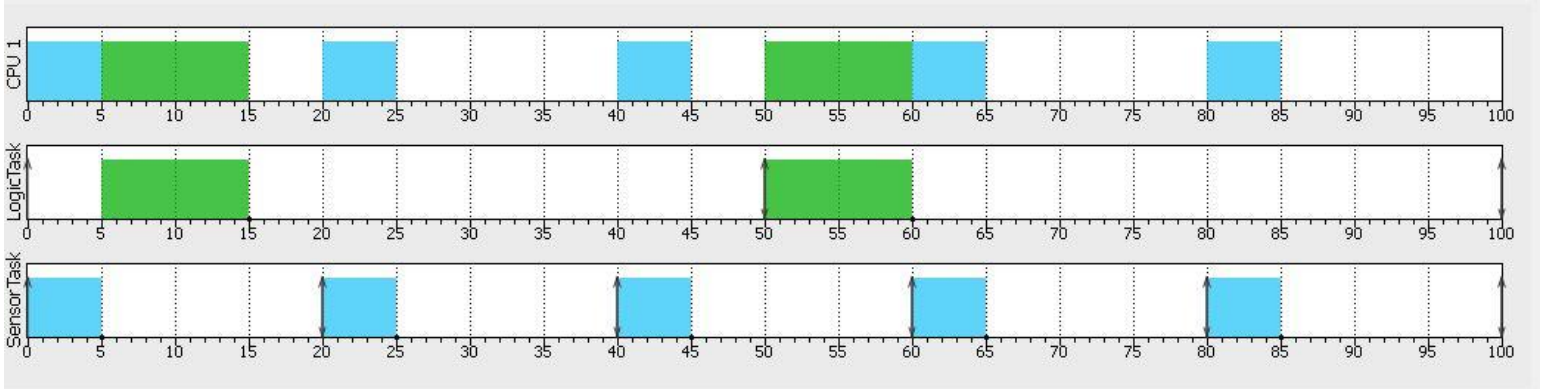
# CPU Load ECUA

Observation Window:

from 0.00 to 100.00 ms

| | Total load | Payload |
|---|---|---|
| CPU 1 | 0.4500 | 0.4500 |
| Average | 0.4500 | 0.4500 |

# ECUA Gantt Chart



# Tasks ECUB

- LogicTask

(Periodicity:30ms,Deadline:30ms,Execution Time:10ms,Priority:2).

- ActuatorTask

(Periodicity:20ms,Deadline:20ms,Execution Time:5ms,Priority:1).

# CPU Load ECUB

Observation Window:

from 0.00 to 100.00 ms

| | Total load | Payload | System load |
|---|---|---|---|
| CPU 1 | 0.6500 | 0.6500 | 0.0000 |
| Average | 0.6500 | 0.6500 | 0.0000 |

# ECUB Gantt Chart