EMBEDDED SYSTEMS ADVANCED COURSE

EMBEDDED SOFTWARE

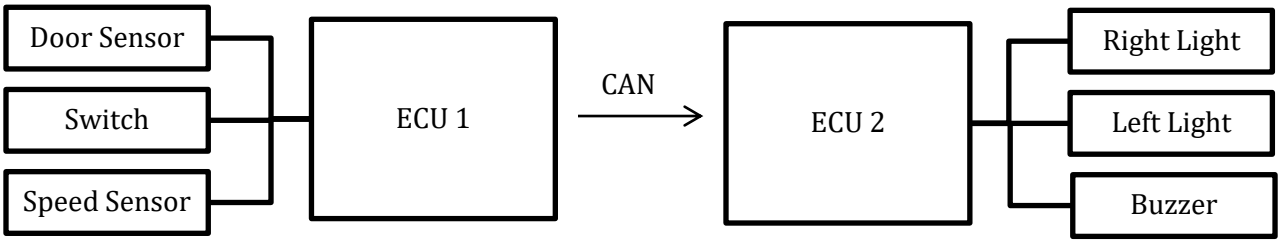
STATIC DESIGN

PROJECT

BY

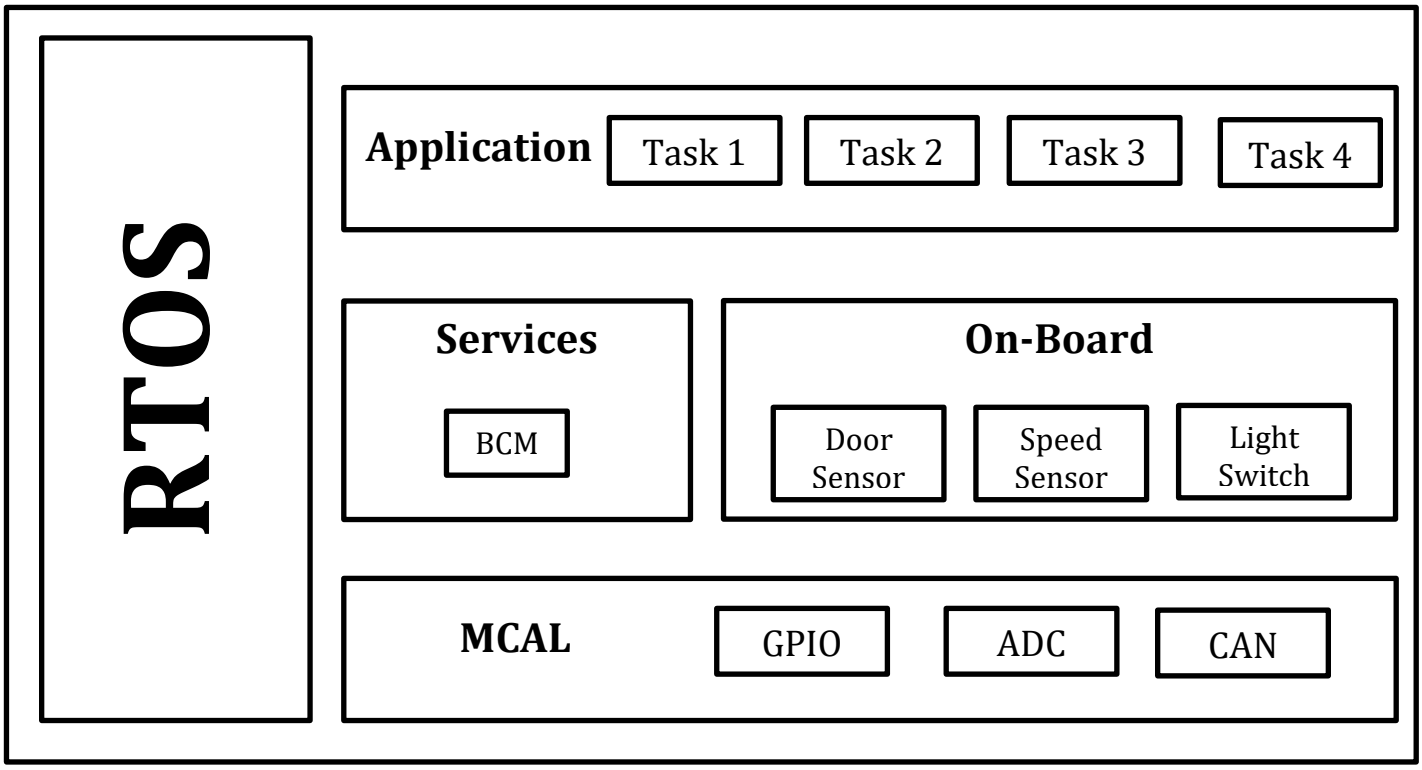
AHMEDSAMEHSAAD99@GMAIL.COM

SYSTEM BLOCK DIAGRAM



ECU #1 – READING SENSORS / SENDING DATA TO CAN

- **Layered Architecture**



- **Module APIs**

MCAL

GPIO	`void GPIO_init(port_t portName, pin_t pinNumber, direction_t direction);`
	`void GPIO_write(port_t portName, pin_t pinNumber, value_t theValue);`
	`value_t GPIO_read(port_t portName, pin_t pinNumber);`
	
	GPIO_init :
	- Initializes the GPIO pin
	- Takes the port name (A, B, C or D) – the pin number (0-7) – the direction either INPUT or OUTPUT
	
	GPIO_write :
	- Writes a digital HIGH or LOW value on the pin
	- Takes the port name (A, B, C or D) – the pin number (0-7) – the value either HIGH or LOW
	
	GPIO_read :
	- Returns the value of the pin which was set as input
	- Takes the port name (A, B, C or D) – the pin number (0-7)

ADC	`void ADC_init(PIN_config_t* configPtr);`
	`ADC_value ADC_read(ADC_port_t portName, ADC_pin_t pinNumber);`
	ADC_init :
	- Initializes the analog to digital converter
	- Takes the configuration pointer which decides which pin will work as ADC.
	
	ADC_read :
	- Returns the value of the ADC pin.
	- Takes the port name (A-B-C-D) and the pin number (0-7)

CAN	`void CAN_init(PIN_config_t* configPtr);`
	`void CAN_sendData(CAN_data_t data);`
	`CAN_data_t CAN_receiveData(void);`
	CAN_init :
	- Initialize the CAN module.
	- Takes the configuration pointer which decides which pin will work as CAN-Tx/Rx
	
	CAN_sendData :
	- Function to send data via the CAN protocol.
	- Takes the data to be sent over.
	
	CAN_receiveData :
	- Function to return the data receied via the CAN protocol.

On-Board

Door_Sensor	`void DOOR_init(DOOR_config_t* configPtr);` `DOOR_state_t DOOR_readSensor(DOOR_sensor_t *theSensor);`

	DOOR_init : - Initializes the door sensor using the configuration pointer passed. - Takes the configuration pointer of the door sensor. DOOR_readSensor : - Reads the sensor state either HIGH or LOW - Takes a pointer to struct of the door sensor specified.

Speed_Sensor	`void SPEED_init(SPEED_config_t* configPtr);`
	`SPEED_value_t DOOR_readSensor(SPEED_sensor_t *theSensor);`
	SPEED_init :
	- Initializes the available speed sensors using the configuration pointer passed.
	- Takes the configuration pointer of the speed sensor.
	
	SPEED_readSensor :
	- Reads the sensor state and returns a value describing the speed.
	- Takes a pointer to struct of the speed sensor specified.

Light_Switch	`void SWITCH_init (SWITCH_config_t* configPtr);`
	`SWITCH_state_t SWITCH_getState(SWITCH_t *theSwitch);`
	SWITCH_init :
	- Initializes the available switches using the configuration pointer passed.
	- Takes the configuration pointer of the speed sensor.
	
	SWITCH_readSensor :
	- Reads the switch state either HIGH or LOW
	- Takes a pointer to struct of the switch specified.

Service

BCM	`void COM_init(COM_config_t *configPtr);`
	`void COM_send(COM_protocol_t theProtocol, COM_data_t theData);`
	`COM_data_t COM_receive(COM_protocol_t theProtocol);`
	COM_init :
	- Initializes the communications that need to be handeled using configuration pointer, describing all types of communication in the system
	- Takes the configuration pointer of the communication protocols.
	
	COM_send :
	- Sends data through a specified communication protocol.
	- Takes a communication protocol (UART, SPI, I2C, CAN etc.) and the data to be sent.
	
	COM_receive :
	- Returns data received through a specified communication protocol.
	- Takes a communication protocol (UART, SPI, I2C, CAN etc.)

Application

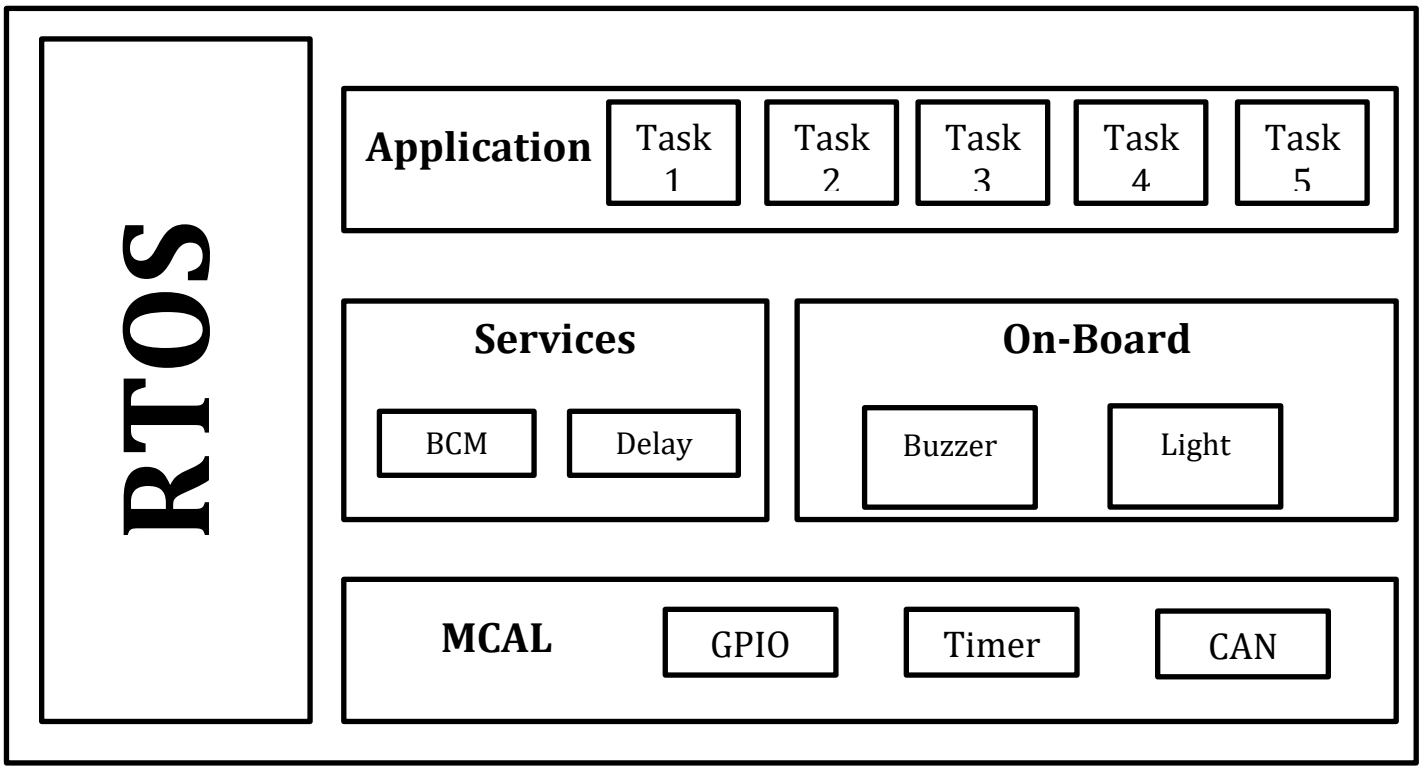
Application Module	`void TASK_getDoorSensorState();`
	`void TASK_getLightSwitchState();`
	`void TASK_getSpeedSensorData();`
	`COM_data_t updateDataFrame(APP_data_t theValue);`
	`void TASK_sendData();`
	
	updateDataFrame:
	- Sends data through a specified communication protocol.
	- Takes a communication protocol (UART, SPI, I2C, CAN etc.) and the data to be sent.

Dividing our project into the following tasks to be implemented:

Task Number	Description	Task Periodicity
Task 1	Reading door sensor	10 ms
Task 2	Reading the switch state	20 ms
Task 3	Reading the speed sensor	5 ms
Task 4	Sending data through CAN bus	5 ms

ECU #2 – RECEIVING DATA FROM CAN / OUTPUT BUZZER-LIGHT

- **Layered Architecture**



- **Module APIs**

MCAL

GPIO	Same as ECU1

Timer	`void TIMER_init(TIMER_config_t* configPtr);`
	`void TIMER_start(TIMER_t theTimer);`
	`void TIMER_stop(TIMER_t theTimer);`
	
	TIMER_init :
	- Initializes the available timers based on the configuration pointer.
	- Takes the configuration pointer for the timers.
	
	TIMER_start :
	- Starts the specified timer.
	- Takes a timer (TIMER0, TIMER1, TIMER2)
	
	TIMER_stop :
	- Stops the specified timer.
	- Takes a timer (TIMER0, TIMER1, TIMER2)

CAN	Same as ECU1

On-Board

Buzzer	`void BUZZER_init (BUZZER_config_t* configPtr);`
	`void BUZZER_setState(BUZZER_t *theBuzzer, BUZZER_state_t theState);`
	
	BUZZER_init :
	- Initializes the available buzzers based on the configuration given.
	- Takes the configuration pointer to the buzzers.
	
	BUZZER_setState :
	- Sets the value of the buzzer either HIGH or LOW.
	- Takes a pointer to the struct of the buzzer, and a state either HIGH or LOW;

Light	`void LIGHT_init(LIGHT_config_t* configPtr);`
	`void LIGHT_setState(LIGHT_t *theLight, LIGHT_state_t theState);`
	
	LIGHT_init :
	- Initializes the available lights based on the configuration given.
	- Takes the configuration pointer to the lights.
	
	LIGHT_setState :
	- Sets the value of the light either HIGH or LOW.
	- Takes a pointer to the struct of the light, and a state either HIGH or LOW;

Service

Delay	`void DELAY_ms(uint32_t milliseconds);` DELAY_ms : - Sets a timed delay for specified milliseconds - Takes the milliseconds to be delayed.

BCM	`void COM_init(COM_config_t *configPtr);`
	`void COM_send(COM_protocol_t theProtocol, COM_data_t theData);`
	`COM_data_t COM_receive(COM_protocol_t theProtocol);`
	Same as ECU1

Application

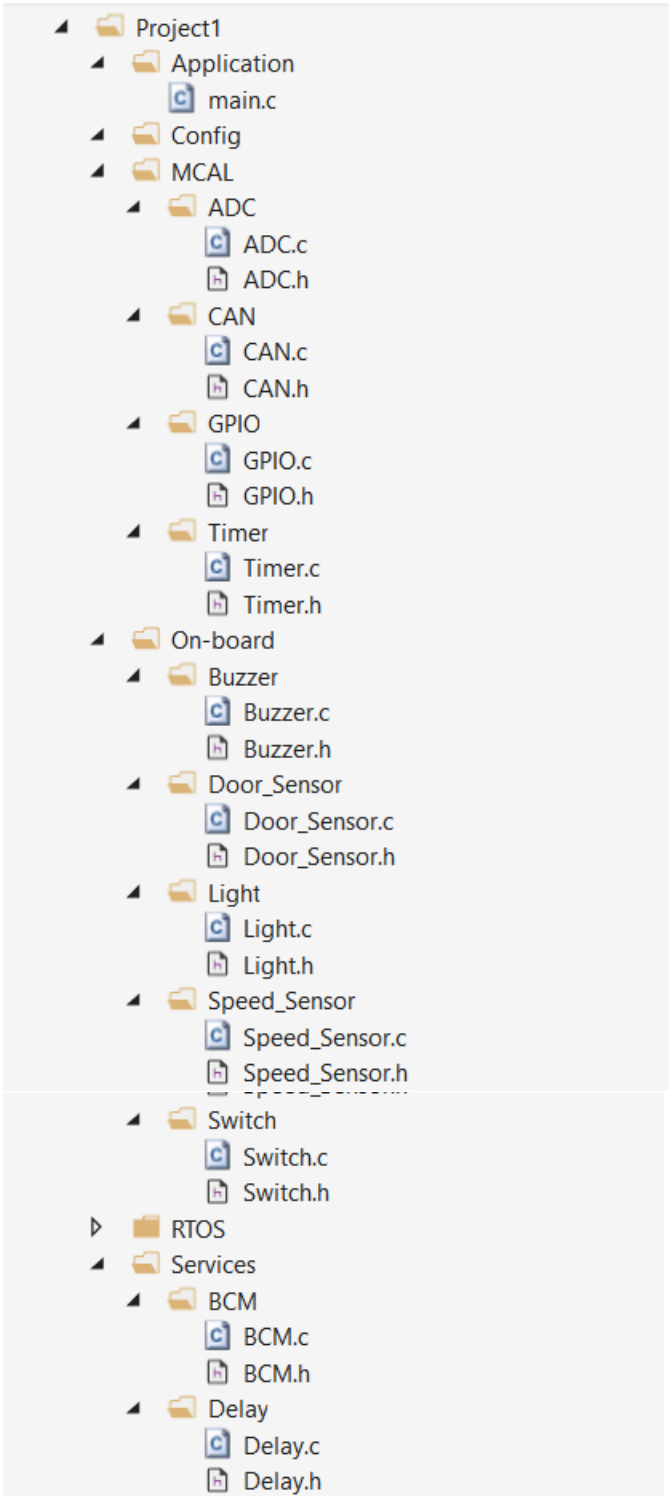
Application Module	`void TASK_receiveData(APP_data_t *dataVar);`
	
	`void buzzerON_LightsOFF(BUZZER_t* theBuzzer, LIGHT_t *theLight);`
	`void buzzerOFF_LightsON(BUZZER_t* theBuzzer, LIGHT_t *theLight);`
	`void lightsOFF_3secs(LIGHT_t *theLight);`
	`void buzzerOFF_LightsON(BUZZER_t* theBuzzer, LIGHT_t *theLight);`

Dividing our project into the a main task to be implemented, and the multiple functions:

Task Name	Description	Task Periodicity
TASK_receiveData	Receive data from CAN bus an put it into a dataVar	5 ms

Function Name	Description
buzzerON_LightsOFF	If DOOR_STATUS == HIGH && SPEED_VAR > 0 Buzzer OFF – lights OFF set LIGHTS_FLAG to LOW
buzzerOFF_LightsON	if DOOR_STATUS == HIGH && SPEED_VAR == 0 OR if SWITCH_STATUS == HIGH && SPEED_VAR > 0 Buzzer OFF – lights ON set LIGHTS_FLAG to HIGH
lightsOFF_3secs	if DOOR_STATUS == LOW && LIGHTS_FLAG == HIGH Lights OFF after 3 seconds
buzzerON_LightsON	If SPEED_VAR == 0 && SWITCH_STATUS == HIGH Buzzer ON, Light ON set LIGHTS_FLAG to HIGH

- **Folder Structure**


Project1
Application
main.c
Config
MCAL
ADC
ADC.c
ADC.h
CAN
CAN.c
CAN.h
GPIO
GPIO.c
GPIO.h
Timer
Timer.c
Timer.h
On-board
Buzzer
Buzzer.c
Buzzer.h
Door_Sensor
Door_Sensor.c
Door_Sensor.h
Light
Light.c
Light.h
Speed_Sensor
Speed_Sensor.c
Speed_Sensor.h
Switch
Switch.c
Switch.h
RTOS
Services
BCM
BCM.c
BCM.h
Delay
Delay.c
Delay.h