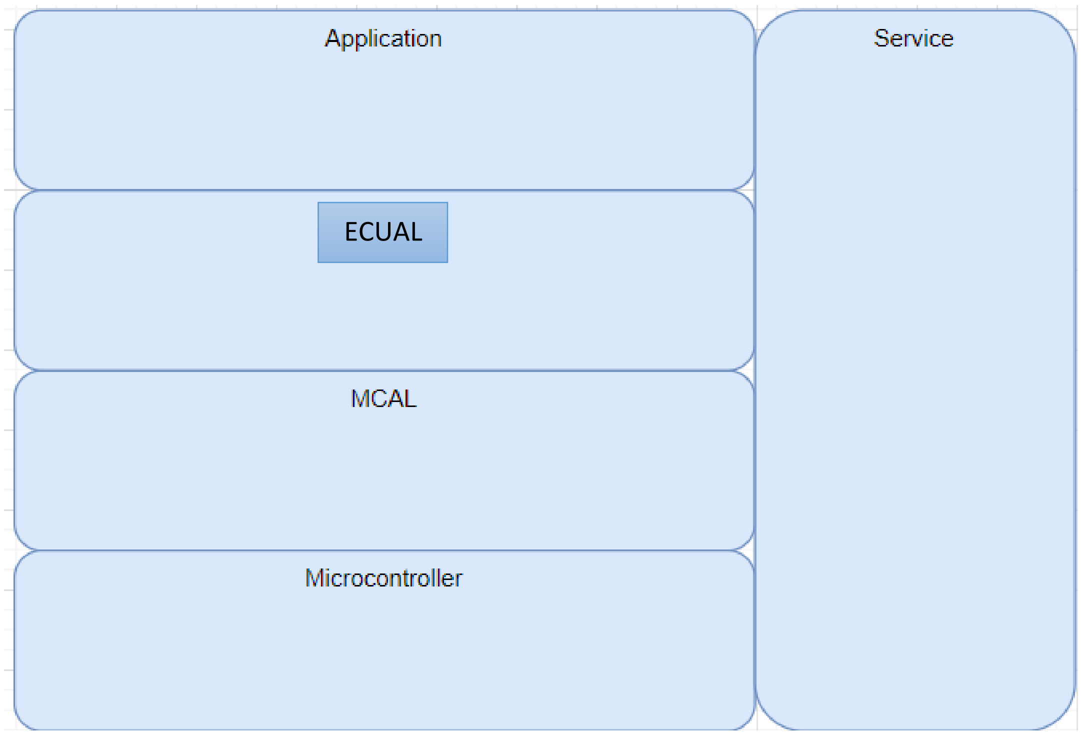
Layered architecture



1-Application Layer: This layer includes the main logic and rules for the car's behavior. It handles the input events received from the presentation layer and calculates the appropriate actions for the car, such as moving forward, stopping, rotating, and repeating the steps.

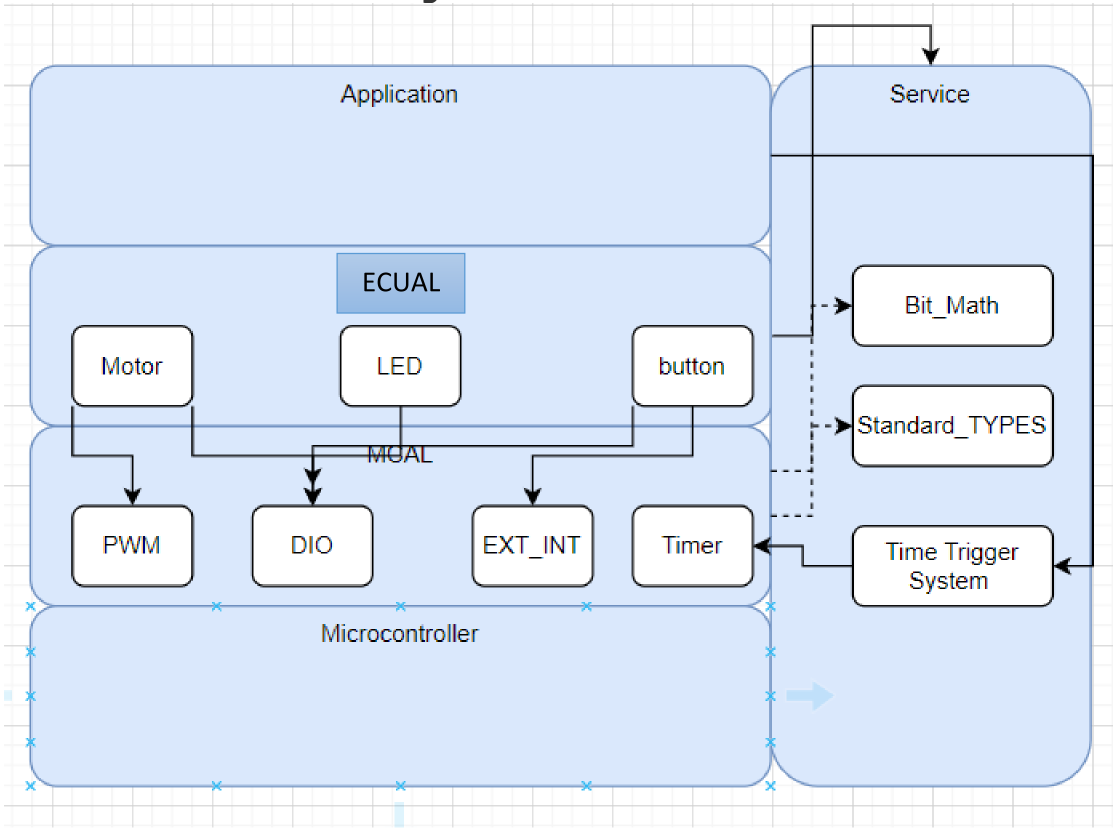
2-ECUAL: This layer includes the domain-specific components and entities of the car, such as the four motors and the car's state. It provides the functionality for controlling the car's movement and state based on the input events received from the application layer.

3- MCAL: This includes device drivers that are specific to the hardware of the microcontroller being used in the system. These drivers are responsible for configuring and controlling the various peripherals of the microcontroller, such as GPIO.

4-Microcontroller Layer: This layer is responsible for managing the operation of the microcontroller itself. It includes functions like initializing the microcontroller's clock and other internal resources, setting up interrupts, and managing the memory map of the microcontroller. This layer is closely tied to the hardware implementation of the system and provides a foundation for the higher-level layers.

5-Serives Layer: This layer The Service Layer in the layered architecture of the system consists of modules that provide higher-level functionalities and abstractions for managing timing requirements and performing bit-level operations. Such as Standard Types.

System modules



1-Physical Modules:

DIO (Digital Input/Output): This module deals with the digital input and output operations, such as reading and writing to digital pins of a microcontroller or a microprocessor. It may include functions for setting pin direction, reading and writing digital values, and handling interrupts related to digital pins.

Motor: This module is responsible for controlling the motors in the car, such as M1, M2, M3, and M4 as mentioned in the system requirements. It may include functions for setting motor speed, direction, and handling motor control signals.

Button: This module deals with the buttons in the system, such as PB1 and PB2 as mentioned in the system requirements. It may include functions for detecting button presses and handling button-related events.

EXT_INT (External Interrupt): This module handles external interrupts, which are signals from external sources that can trigger interrupts in the microcontroller or microprocessor. It may include functions for configuring and handling external interrupts from external devices, such as buttons or sensors.

PWM (Pulse Width Modulation): This module deals with generating PWM signals, which are used for controlling the intensity of an output signal, such as controlling the speed of motors or the brightness of LEDs. It may include functions for configuring and controlling PWM signals.

LED he LED module is responsible for controlling the LEDs (LED1, LED2, LED3, LED4) mentioned in the system requirements. It may include functions for setting the LED states (e.g., ON or OFF), controlling LED brightness or color (if applicable), and handling any other operations related to LED control.

2-Supporting Modules:

Timer: This module deals with timer operations, such as configuring and handling timers in the microcontroller or microprocessor. It may include functions for setting timer intervals, handling timer interrupts, and measuring time**.**

BIT_MATH: This module provides functions for performing bitwise operations, such as AND, OR, XOR, and shifting, which are commonly used for manipulating individual bits in registers or memory locations.

Standard Types: This module includes standard data types, such as integer types, floating-point types, and Boolean types, which are used for representing data in a standardized way across the system.

Time Trigger System: Based on the system requirements, the "Time Trigger System" is likely a higher-level module that coordinates the timing and sequence of various operations in the system, such as starting and stopping the car, rotating the car, and controlling the speed of the car based on time triggers. It may include functions for managing time-based events, coordinating the timing of different operations, and controlling the overall behavior of the system based on time-based triggers. The specific implementation of this module will depend on the system requirements and design decisions made in the project.

APIs

1-DIO (Digital Input/Output) Module:

- DIO_init(uint8_t portNumber, uint8_t pinNumber, uint8_t direction):

Description: This API initializes the direction of a specific Digital Input/Output (DIO) pin.

Parameters:

portNumber: The number of the port to which the DIO pin belongs (e.g., PORTA, PORTB, etc.).

pinNumber: The number of the pin within the specified port.

direction: The desired direction of the pin (INPUT or OUTPUT).

Return value:

E_OK: If the initialization is successful.

WRONG_PORT_NUMBER: If an invalid port number is provided.

WRONG_PIN_NUMBER: If an invalid pin number is provided.

WRONG_DIRECTION: If an invalid direction is provided.

-DIO_write(uint8_t portNumber, uint8_t pinNumber, uint8_t value):

Description: This API writes a digital value (HIGH or LOW) to a specific DIO pin.

Parameters:

portNumber: The number of the port to which the DIO pin belongs.

pinNumber: The number of the pin within the specified port.

value: The digital value to be written to the pin (HIGH or LOW).

Return value:

E_OK: If the write operation is successful.

WRONG_PORT_NUMBER: If an invalid port number is provided.

WRONG_PIN_NUMBER: If an invalid pin number is provided.

WRONG_VALUE: If an invalid value is provided.

-DIO_toggle(uint8_t portNumber, uint8_t pinNumber):

Description: This API toggles the state (HIGH to LOW or vice versa) of a specific DIO pin.

Parameters:

portNumber: The number of the port to which the DIO pin belongs.

pinNumber: The number of the pin within the specified port.

Return value:

E_OK: If the toggle operation is successful.

WRONG_PORT_NUMBER: If an invalid port number is provided.

WRONG_PIN_NUMBER: If an invalid pin number is provided.

-DIO_read(uint8_t portNumber, uint8_t pinNumber, uint8_t *value):

Description: This API reads the digital value (HIGH or LOW) from a specific DIO pin.

Parameters:

portNumber: The number of the port to which the DIO pin belongs.

pinNumber: The number of the pin within the specified port.

value: Pointer to a variable where the read value will be stored.

Return value:

E_OK: If the read operation is successful.

WRONG_PORT_NUMBER: If an invalid port number is provided.

WRONG_PIN_NUMBER: If an invalid pin number is provided.

value: The read digital value (HIGH or LOW) will be stored in the memory location pointed to by the value parameter. Note: The value parameter is an output parameter, and the value read from the pin will be updated in the memory location pointed to by this parameter.

Button module:

- BUTTON_init(uint8_t buttonPort, uint8_t buttonPin)

Description: Initializes the BUTTON module by setting the direction of the specified button pin as input.

Parameters:

buttonPort: The port number of the button pin to be initialized.

buttonPin: The pin number of the button pin to be initialized.

Returns:

E_OK: If the initialization is successful.

BUTTON_INIT_ERROR: If there is an error during button initialization.

- BUTTON_read(uint8_t buttonPort, uint8_t buttonPin, uint8_t *buttonState)

Description: Reads the current state of the specified button pin.

Parameters:

buttonPort: The port number of the button pin to be read.

buttonPin: The pin number of the button pin to be read.

buttonState: Pointer to a variable that will hold the read button state (0 for released, 1 for pressed).

Returns:

E_OK: If the button state is read successfully.

BUTTON_READ_ERROR: If there is an error during button state reading.:

Timer Module:

- TIMER_init(uint8_t timerUsed)

Description: Initializes the specified timer for use.

Parameters:

timerUsed: The timer number to be initialized.

Returns:

E_OK: If the timer initialization is successful.

WRONG_TIMER_NUMBER: If the provided timer number is invalid.

-TIMER_setTime(uint8_t timerUsed, uint32_t desiredTime)

Description: Sets the desired time for the specified timer.

Parameters:

timerUsed: The timer number for which the time is to be set.

desiredTime: The desired time value to be set for the timer, in milliseconds.

Returns:

E_OK: If the time setting is successful.

WRONG_TIMER_NUMBER: If the provided timer number is invalid.

WRONG_DESIRED_TIME: If the provided desired time value is invalid.

- TIMER_start(uint8_t timerUsed)

Description: Starts the specified timer.

Parameters:

timerUsed: The timer number to be started.

Returns:

E_OK: If the timer is started successfully.

WRONG_TIMER_NUMBER: If the provided timer number is invalid.

- TIMER_stop(uint8_t timerUsed)

Description: Stops the specified timer.

Parameters:

timerUsed: The timer number to be stopped.

Returns:

E_OK: If the timer is stopped successfully.

WRONG_TIMER_NUMBER: If the provided timer number is invalid.

- TIMER_setCallBack(void (*funPtr) (void))

Description: Sets a callback function to be called when the timer expires.

Parameters:

funPtr: A pointer to the callback function that will be called when the timer expires. This function should have a void return type and no parameters. It will be executed in the interrupt context.

Returns: None.

Motor Module:

- MOTOR_init(void);

Description: Initializes the motor module. This API sets up the necessary configuration for the motor control, such as initializing hardware resources, setting default parameters, and preparing the motor for operation.

Parameters: None.

Returns:

E_OK if the motor module is initialized successfully.

INIT_ERROR if there is an error during initialization.

- MOTOR_setDirection(uint8_t direction);

Description: Sets the direction of the motor rotation.

Parameters:

direction: The desired direction of the motor rotation. This can be one of the predefined constants or values that represent the desired direction (e.g., forward, backward, etc.).

Returns:

E_OK if the motor direction is set successfully.

WRONG_DIRECTION if the provided direction value is invalid.

- MOTOR_speed(uint8_t setSpeed);

Description: Sets the speed of the motor.

Parameters:

setSpeed: The desired speed of the motor. This can be a value that represents the desired speed level or percentage of the maximum speed.

Returns:

E_OK if the motor speed is set successfully.

WRONG_SPEED if the provided speed value is invalid.

- MOTOR_start(void);

Description: Starts the motor rotation.

Parameters: None.

Returns:

E_OK if the motor starts successfully.

STARTING_ERROR if there is an error during motor starting.

- MOTOR_stop(void);

Description: Stops the motor rotation.

Parameters: None.

Returns:

E_OK if the motor stops successfully.

STOPPING_ERROR if there is an error during motor stopping.

Note

EXT_INT (External Interrupt) Module:

- EXT_INT_init(uint8_t intNumber):

Description: This API initializes the external interrupt specified by intNumber. It configures the interrupt edge triggering mode and enables the interrupt for the corresponding interrupt number. Parameters:

intNumber: The external interrupt number to be initialized (0, 1, or 2). Returns:

E_OK: If the external interrupt was initialized successfully.

WRONG_INTERRUPT_NUMBER: If an invalid interrupt number was provided.

- EXT_INT_setCallBackInt0(void (*funPtr)(void)):

Description: This API sets the callback function to be executed when an interrupt occurs on external interrupt 0 (INT0).

Parameters:

funPtr: A function pointer to the callback function to be executed.

Returns: None.

- EXT_INT_setCallBackInt1(void (*funPtr)(void)):

Description: This API sets the callback function to be executed when an interrupt occurs on external interrupt 1 (INT1).

Parameters:

funPtr: A function pointer to the callback function to be executed.

Returns: None.

- EXT_INT_setCallBackInt2(void (*funPtr)(void)):

Description: This API sets the callback function to be executed when an interrupt occurs on external interrupt 2 (INT2).

Parameters:

funPtr: A function pointer to the callback function to be executed.

Returns: None.

PWM (Pulse Width Modulation) Module:

- PWM_init(uint8_t pwmPort, uint8_t pwmPin);

Description: This function initializes the PWM module for a specific port and pin on the microcontroller.

Parameters:

pwmPort: An unsigned 8-bit integer representing the port number to which the PWM pin is connected.

pwmPin: An unsigned 8-bit integer representing the pin number to which the PWM signal is generated.

Return value:

E_OK: If the PWM module is initialized successfully.

WRONG_PORT_NUMBER: If an invalid port number is provided.

WRONG_PIN_NUMBER: If an invalid pin number is provided.

-PWM_setDutyCycle(uint8_t pwmPort, uint8_t pwmPin, uint8_t desiredDutyCycle);

Description: This function sets the duty cycle of the PWM signal for a specific port and pin.

Parameters:

pwmPort: An unsigned 8-bit integer representing the port number to which the PWM pin is connected.

pwmPin: An unsigned 8-bit integer representing the pin number to which the PWM signal is generated.

desiredDutyCycle: An unsigned 8-bit integer representing the desired duty cycle of the PWM signal, ranging from 0 to 255.

Return value:

E_OK: If the duty cycle is set successfully.

WRONG_PORT_NUMBER: If an invalid port number is provided.

WRONG_PIN_NUMBER: If an invalid pin number is provided.

WRONG_DUTY_CYCLE: If an invalid duty cycle value is provided.

- PWM_start(uint8_t pwmPort, uint8_t pwmPin);

Description: This function starts the generation of the PWM signal on a specific port and pin.

Parameters:

pwmPort: An unsigned 8-bit integer representing the port number to which the PWM pin is connected.

pwmPin: An unsigned 8-bit integer representing the pin number to which the PWM signal is generated.

Return value:

E_OK: If the PWM signal generation is started successfully.

WRONG_PORT_NUMBER: If an invalid port number is provided.

WRONG_PIN_NUMBER: If an invalid pin number is provided.

- PWM_stop(uint8_t pwmPort, uint8_t pwmPin);

Description: This function stops the generation of the PWM signal on a specific port and pin.

Parameters:

pwmPort: An unsigned 8-bit integer representing the port number to which the PWM pin is connected.

pwmPin: An unsigned 8-bit integer representing the pin number to which the PWM signal is generated.

Return value:

E_OK: If the PWM signal generation is stopped successfully.

WRONG_PORT_NUMBER: If an invalid port number is provided.

WRONG_PIN_NUMBER: If an invalid.