Design document

Project title: Moving car system design

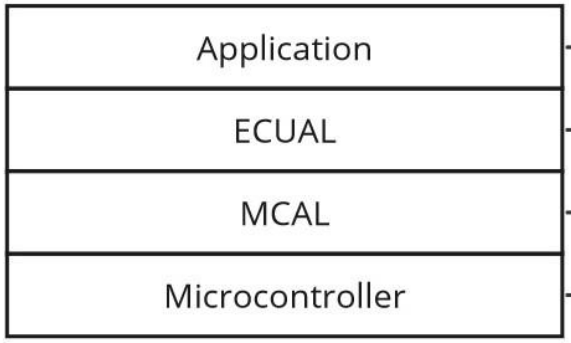
Name: Amr El-Abd

- *Description*

1. **Car Components**:
    1. **Four** motors (**M1**, **M2**, **M3**, **M4**)
    2. **One** button to start (**PB1**)
    3. **One** button for stop (**PB2**)
    4. **Four** LEDs (**LED1**, **LED2**, **LED3**, **LED4**)

2. **System Requirements**:
    1. The car **starts initially** from **0 speed**
    2. When **PB1** is **pressed**, the car will **move forward after 1 second**
    3. The car will move forward to **create the longest side of the rectangle for 3 seconds with 50% of its maximum speed**
    4. After finishing the first longest side the car will **stop for 0.5 seconds**, **rotate 90 degrees to the right**, and **stop for 0.5 second**
    5. The car will move to **create the short side** of the rectangle at **30% of its speed for 2 seconds**
    6. After finishing the shortest side the car will stop for **0.5 seconds**, **rotate 90 degrees to the right**, and **stop for 0.5 second**
    7. Steps **3 to 6** will be **repeated infinitely** until you press the **stop button (PB2)**
    8. **PB2** acts as a **sudden break**, and it has the highest priority.

Layered architecture:



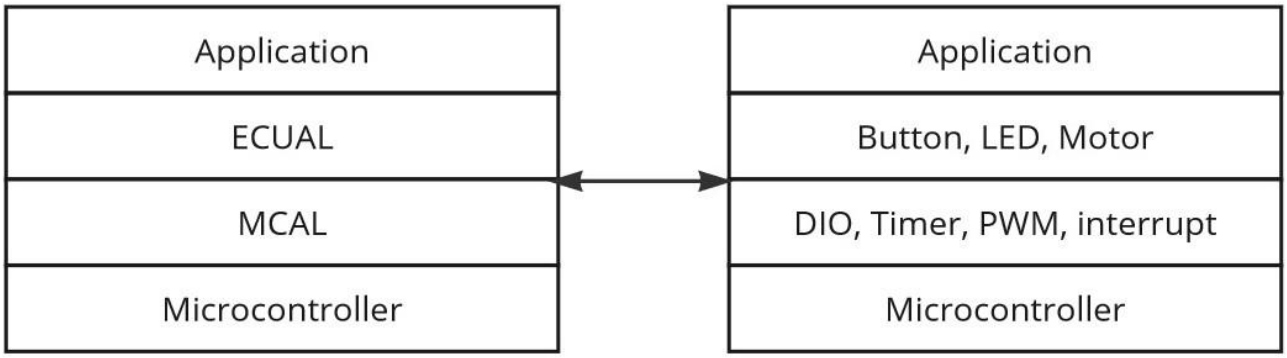
Application Layer: This is the topmost layer of the software stack, which contains the actual application logic. It interacts with the lower layers to perform its tasks. It is responsible for implementing the desired functionality of the system.

ECUAL : This layer provides an abstraction for external devices connected to the microcontroller. The ECUAL layer provides interface to access external devices and hides the implementation details from the application layer.

MCAL (Microcontroller Abstraction Layer): This layer provides an abstraction for the microcontroller hardware. It includes low-level drivers for peripherals. It hides the hardware details and provides a uniform interface to the upper layers.

Microcontroller: This layer represents the physical hardware layer consisting of the microcontroller chip. The microcontroller is responsible for executing the code stored in its memory and controlling the behavior of the system.

system modules/drivers



- Button module will use DIO module to detect its state
- LED module will use DIO module to detect its state
- Motor module will use PWM module to control its speed
- DIO module provides low-level hardware access to the microcontroller
- Timer module can be used by both the PWM module and the Application layer to implement timing functions.
- PWM module can be used by Motor module to control its speed
- Interrupt module can be used by the app module to control the logic

Program APIs:

//DIO.h

```
EN_return DIO_init (EN_ports port, EN_pins pin, EN_direction direction);
EN_return DIO_write (EN_ports port, EN_pins pin, EN_state state);
EN_return DIO_read (EN_ports port, EN_pins pin, uint8_t* value);
```

//LED.h

```
EN_return LED_init (EN_ports port, EN_pins pin);
EN_return LED_on (EN_ports port, EN_pins pin);
EN_return LED_off (EN_ports port, EN_pins pin);
```

//button.h

```
EN_return button_init(EN_ports port, EN_pins pin);
EN_return button_read(EN_ports port,EN_pins pin, EN_state* state);
```

//interrupt.h

```
EN_int__error_t EXI_Enable (EN_int_t Interrupt);
EN_int__error_t EXI_Disable (EN_int_t Interrupt);
EN_int__error_t EXI_Trigger(EN_int_t Interrupt,EN_trig trigger);
void EXI_SetCallBack(EN_int_t Interrupt,void(*ptrf)(void));
```

//pwm.h

```
EN_return PWM_init(EN_ports port, EN_pins pin);
EN_return PWM_set_duty_cycle(EN_ports port, EN_pins pin, uint8_t duty_cycle);
EN_return timer_init();
```

//timer.h

```
void timer_start();
void timer_stop();
uint32_t timer_get_elapsed_time();
EN_return motor_init();
```

//Motor.h

```
void motor_set_state(EN_motor_state state);
void motor_move_forward(uint8_t speed);
void motor_move_backward(uint8_t speed);
void motor_turn_left(uint8_t speed);
void motor_turn_right(uint8_t speed);
```

//program.h

```
void APP_init();
void APP_start();
```