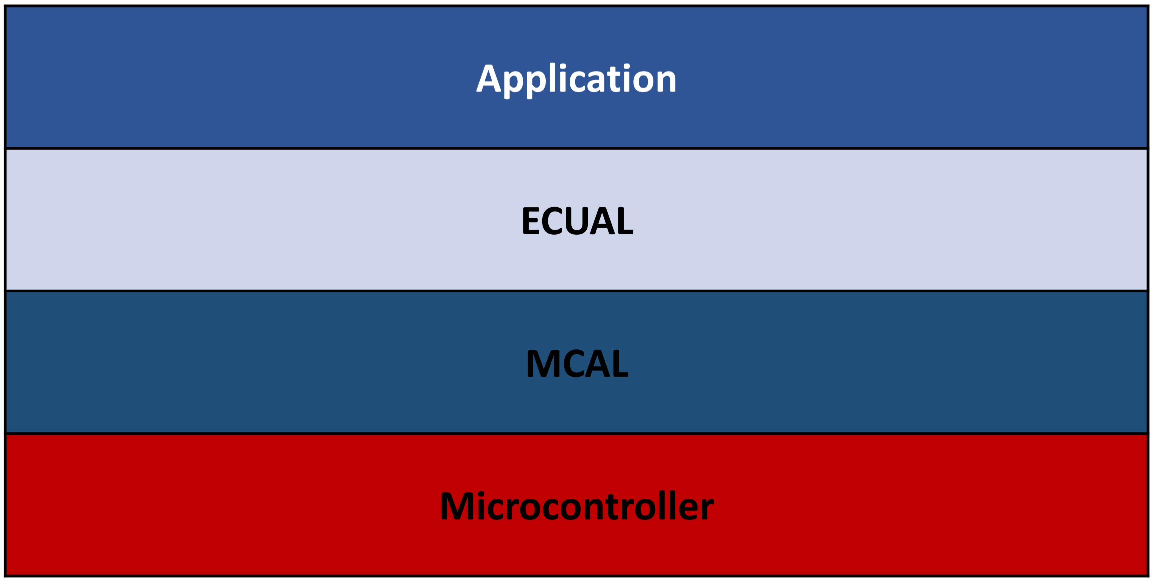
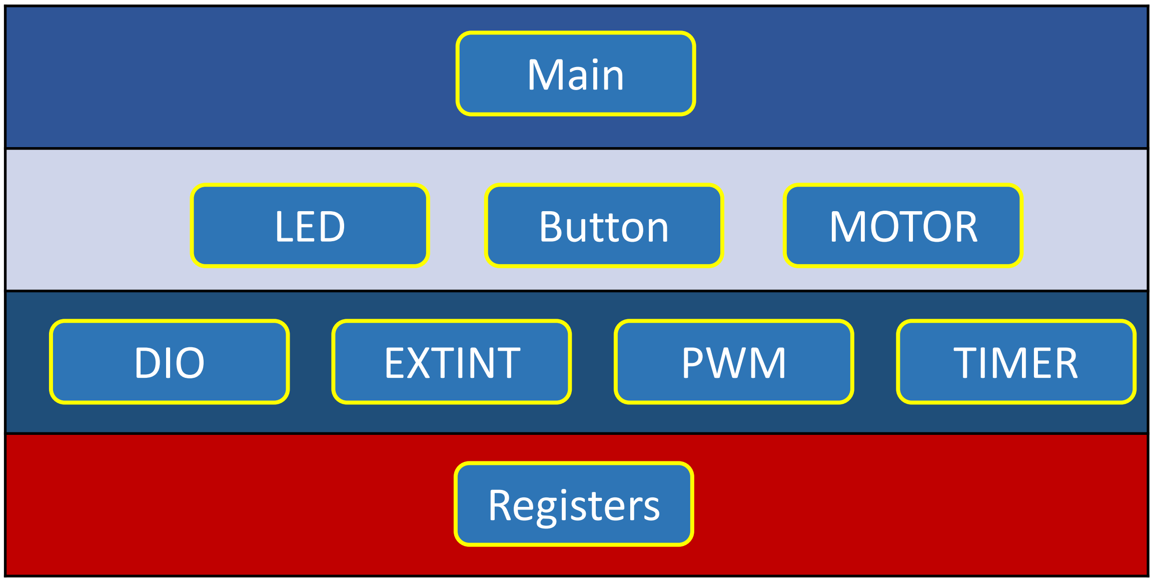
Moving Car Design

By Sharpel Malak

1 : Layered Architecture :



2 : System modules



3 : Project Modules APIs

3.1 - DIO

```
// DIO TYPEDEFS
typedef enum EN_DIO_ERROR{
   DIO_OK=0,
   DIO_NOT_OK
}EN_DIO_ERROR;
typedef enum EN_DIO_PINS{
   DIO_PIN0=0,
   DIO_PIN1,
   DIO_PIN2,
   DIO_PIN3,
   DIO_PIN4,
   DIO_PIN5,
   DIO_PIN6,
   DIO_PIN7,
}EN_DIO_PINS;
typedef enum EN_DIO_PORTS{
   DIO_PORTA=0,
   DIO_PORTB,
   DIO_PORTC,
   DIO_PORTD
}EN_DIO_PORTS;
typedef enum EN_DIO_DIRECTION{
   INPUT=0,
   OUTPUT
}EN_DIO_DIRECTION;
typedef enum EN_DIO_LEVEL{
   LOW=0,
   HIGH
}EN_DIO_LEVEL;

// DIO FUNCTIONS PROTOTYPES

// Description : This function initialize PIN and set it's direction
//     ARGS        : take PIN Number and PORT Number and Direction (INPUT,OUTPUT)
//     return      : return DIO_OK if the PIN initializes correctly, DIO_NOT_OK otherwise
EN_DIO_ERROR DIO_init(EN_DIO_PINS pinNumber,EN_DIO_PORTS portNumber,EN_DIO_DIRECTION direction);

// Description : This function write on PIN and set it's level
// ARGS        : take PIN Number and PORT Number and level (LOW,HIGH)
// return      : return DIO_OK if the PIN level sets correctly, DIO_NOT_OK otherwise
EN_DIO_ERROR DIO_write(EN_DIO_PINS pinNumber,EN_DIO_PORTS portNumber,EN_DIO_LEVEL level);

// Description : This function toggles PIN level
// ARGS        : take PIN Number and PORT Number
// return      : return DIO_OK if the PIN toggles correctly, DIO_NOT_OK otherwise
EN_DIO_ERROR DIO_toggle(EN_DIO_PINS pinNumber,EN_DIO_PORTS portNumber);

// Description : This function reads PIN level and store it in the variable
// ARGS        : take PIN Number and PORT Number and pointer to the variable
// return      : return DIO_OK if the PIN value stored correctly , DIO_NOT_OK otherwise
EN_DIO_ERROR DIO_read(EN_DIO_PINS pinNumber,EN_DIO_PORTS portNumber, uint8_t * value);
```

3.2 - EXTINT

```
// EXTINT MACROS
#define GLOBAL_INTERRUPT_ENABLE     1
#define GLOBAL_INTERRUPT_DISABLE    0
#define INT_TRIGGER_LOW_LEVEL       0
#define INT_TRIGGER_RISING_EDGE     1
#define INT_TRIGGER_FALLING_EDGE    2
#define INT_TRIGGER_ANYLOGICCHANGE  3

#define GLOBAL_INTERRUPT_STATE       GLOBAL_INTERRUPT_ENABLE
#define EXTERNAL_INTERRUPT0_TRIGGER  INT_TRIGGER_RISING_EDGE
#define EXTERNAL_INTERRUPT1_TRIGGER  INT_TRIGGER_FALLING_EDGE

//  remember this interrupt source has only two modes rising edge  and falling edge
#define EXTERNAL_INTERRUPT2_TRIGGER  INT_TRIGGER_FALLING_EDGE

void SET_GLOBAL_INTERRUPT(void);
void EXT_INTERRUPT0_init(void);
void EXT_INTERRUPT1_init(void);
void EXT_INTERRUPT2_init(void);
```

3.3 - Timer

```
void Timer0_Init(void);

void Timer0_Start(void);

void Timer0_Stop(void);

void Timer0_SetDelay(uint32 Delay_ms);
```

3.4 - PWM

```
void PWM_Init(void);

void PWM_SETSPEED(pinNumber,portNumber,speed);
```

3.5 - LED

```
typedef enum EN_LED_Error_t
{
   LED_OK = 0,
   LED_NOT_OK
}EN_LED_Error_t;

EN_LED_Error_t LED_init(EN_DIO_PINS pinNumber,EN_DIO_PORTS portNumber);
EN_LED_Error_t LED_on(EN_DIO_PINS pinNumber,EN_DIO_PORTS portNumber);
EN_LED_Error_t LED_off(EN_DIO_PINS pinNumber,EN_DIO_PORTS portNumber);
EN_LED_Error_t LED_toggle(EN_DIO_PINS pinNumber,EN_DIO_PORTS portNumber);
```

3.6 - Button

```
// Button typedefs

typedef enum EN_BTN_Error_t
{
   BTN_OK = 0,
   BTN_NOT_OK

}EN_BTN_Error_t;

EN_BTN_Error_t Button_init(EN_DIO_PINS pinNumber,EN_DIO_PORTS portNumber);
EN_BTN_Error_t Button_read(EN_DIO_PINS pinNumber,EN_DIO_PORTS portNumber,uint8_t
*value);
```

3.7 - Motor

```
typedef enum
{
   DIR_CLOCK_WISE,
   DIR_ANTI_CLOCK_WISE
}DcMotor_Dir;

void DcMotor_Init(void);

void DcMotor_SetDir(DcMotor_Dir dir);

void DcMotor_SetSpeed(uint8 speed);

void DcMotor_Start(void);

void DcMotor_Stop(void);
```