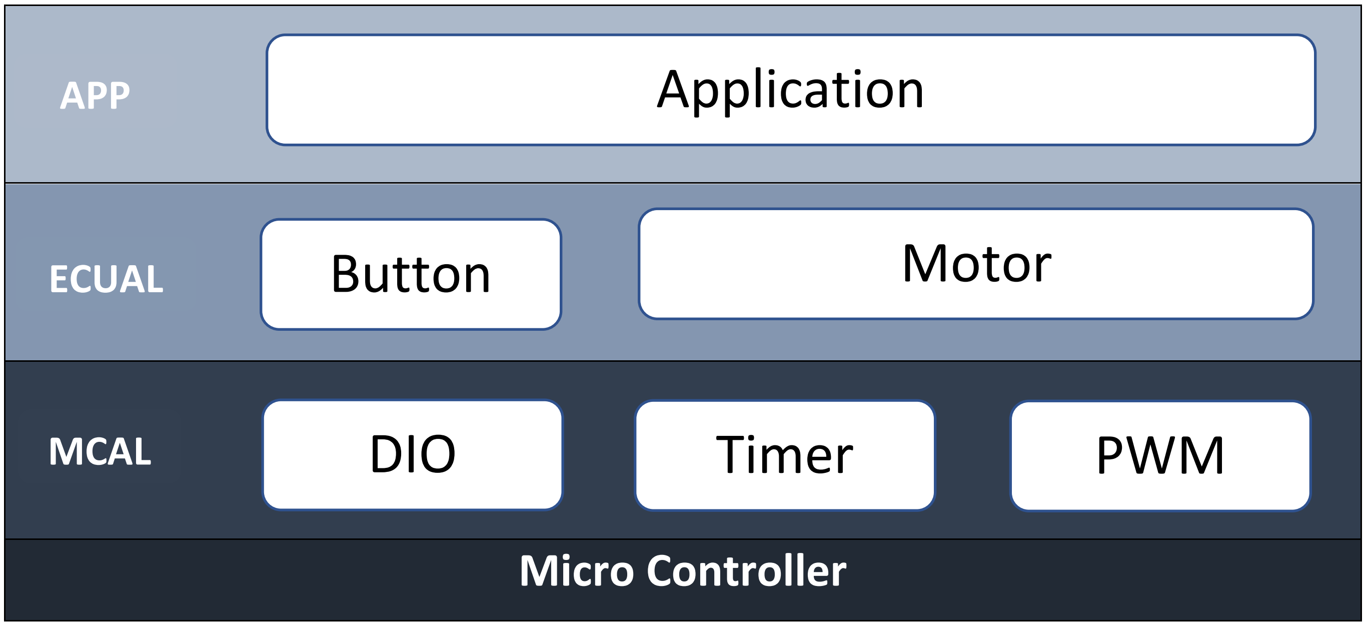
Static Design Task

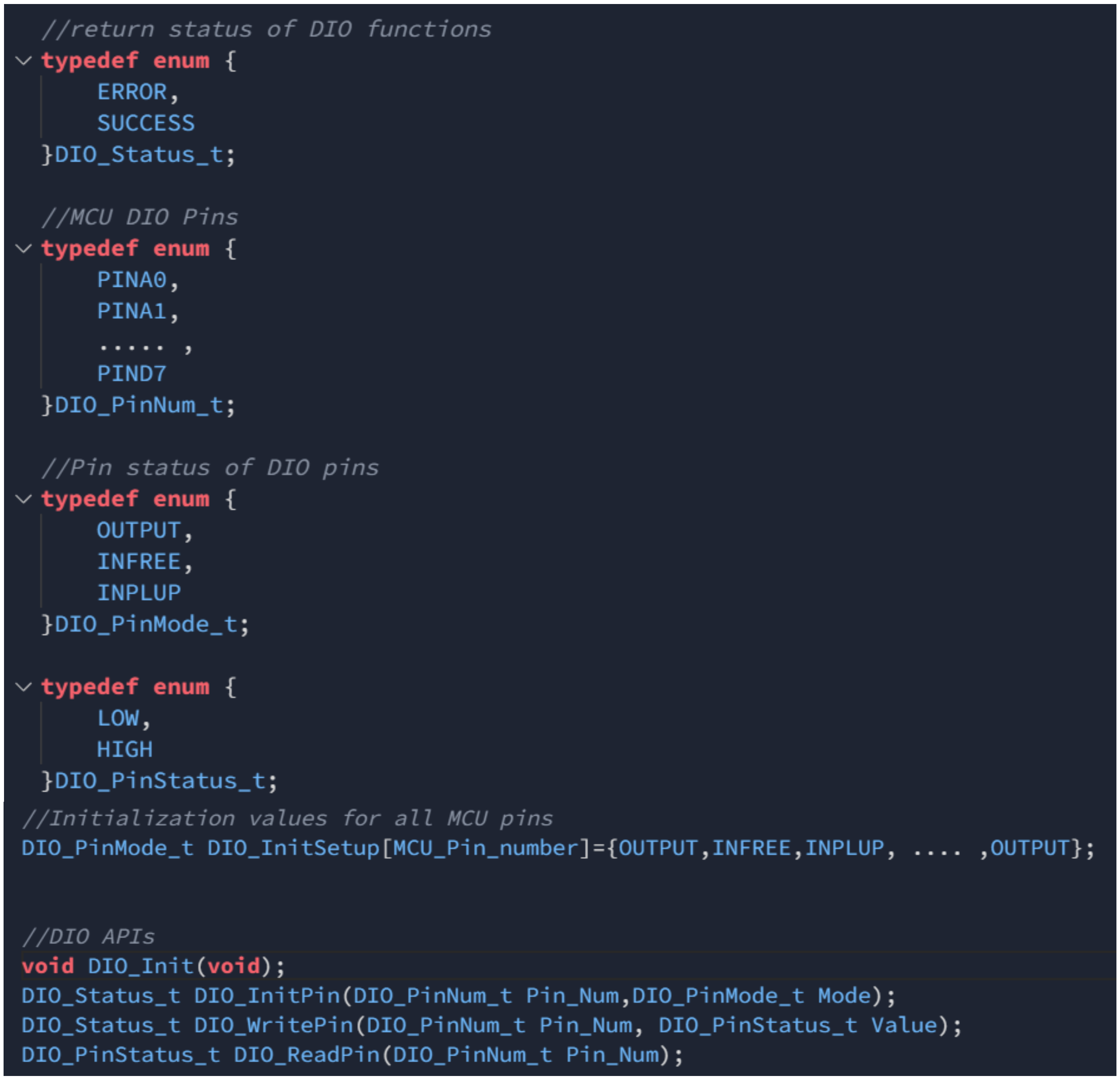
A) Layered Architecture:



B) Defining APIs:

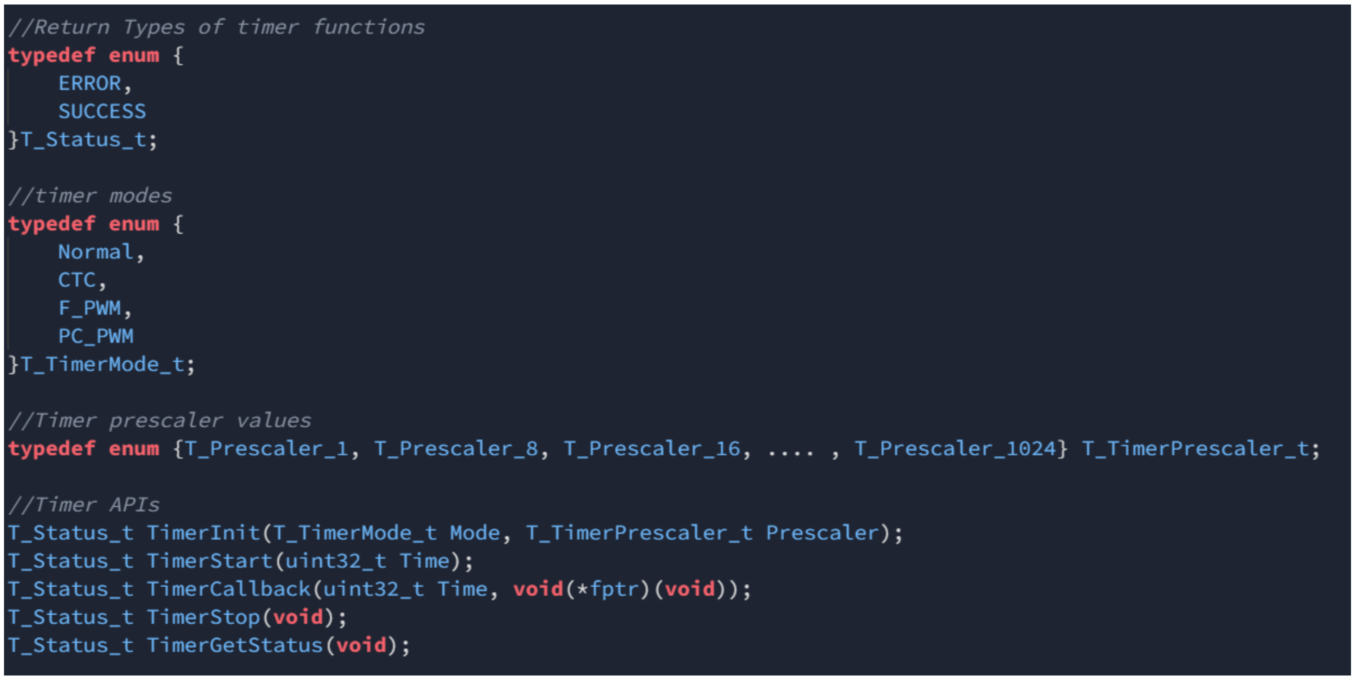
I) MCAL LAYER:

1- DIO APIs:



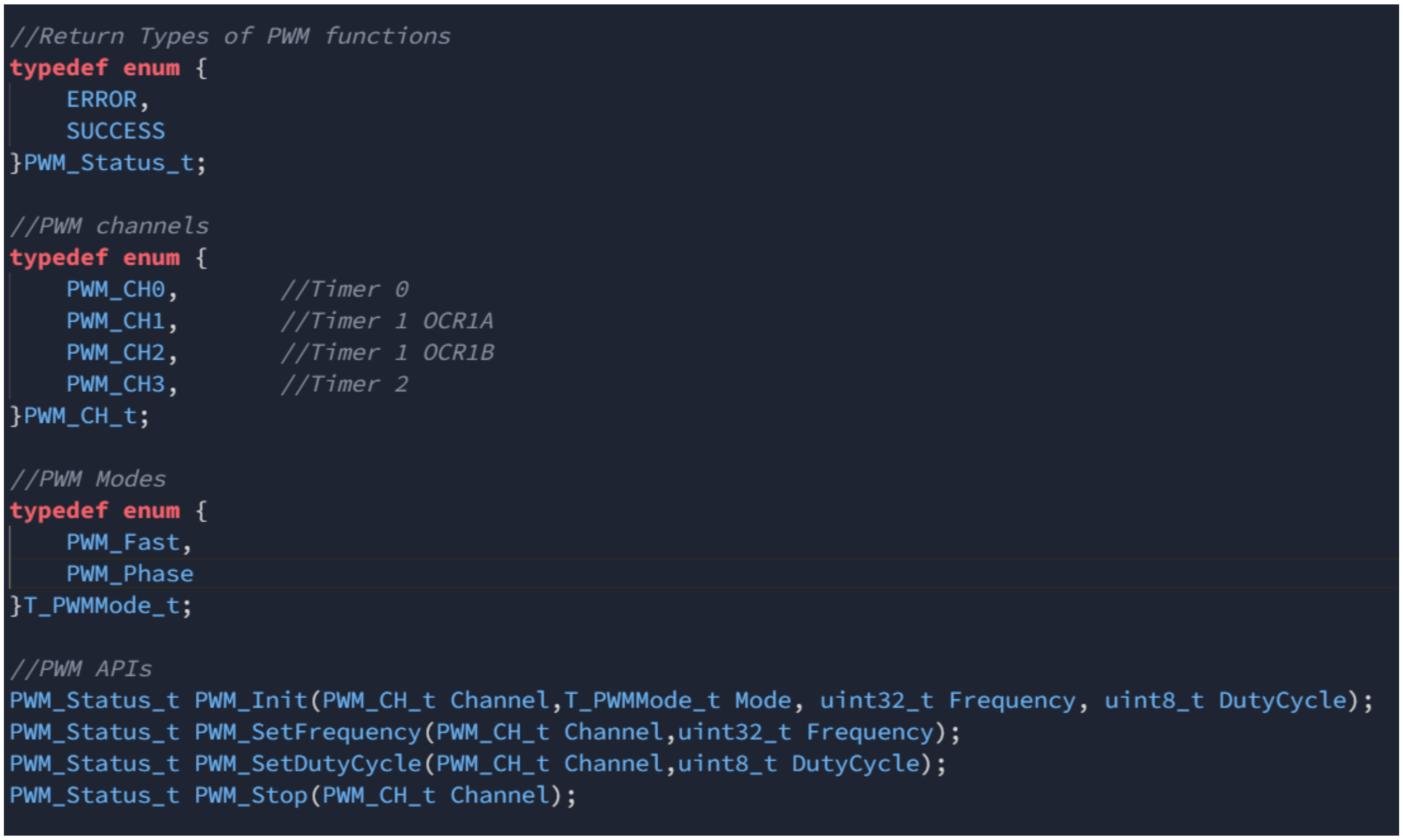
```
//return status of DIO functions
typedef enum {
    ERROR,
    SUCCESS
}DIO_Status_t;

//MCU DIO Pins
typedef enum {
    PINA0,
    PINA1,
    ..... ,
    PIND7
}DIO_PinNum_t;

//Pin status of DIO pins
typedef enum {
    OUTPUT,
    INFREE,
    INPLUP
}DIO_PinMode_t;

typedef enum {
    LOW,
    HIGH
}DIO_PinStatus_t;
//Initialization values for all MCU pins
DIO_PinMode_t DIO_InitSetup[MCU_Pin_number]={OUTPUT,INFREE,INPLUP, .... ,OUTPUT};


//DIO APIs
void DIO_Init(void);
DIO_Status_t DIO_InitPin(DIO_PinNum_t Pin_Num,DIO_PinMode_t Mode);
DIO_Status_t DIO_WritePin(DIO_PinNum_t Pin_Num, DIO_PinStatus_t Value);
DIO_PinStatus_t DIO_ReadPin(DIO_PinNum_t Pin_Num);
```

2- Timer APIs:



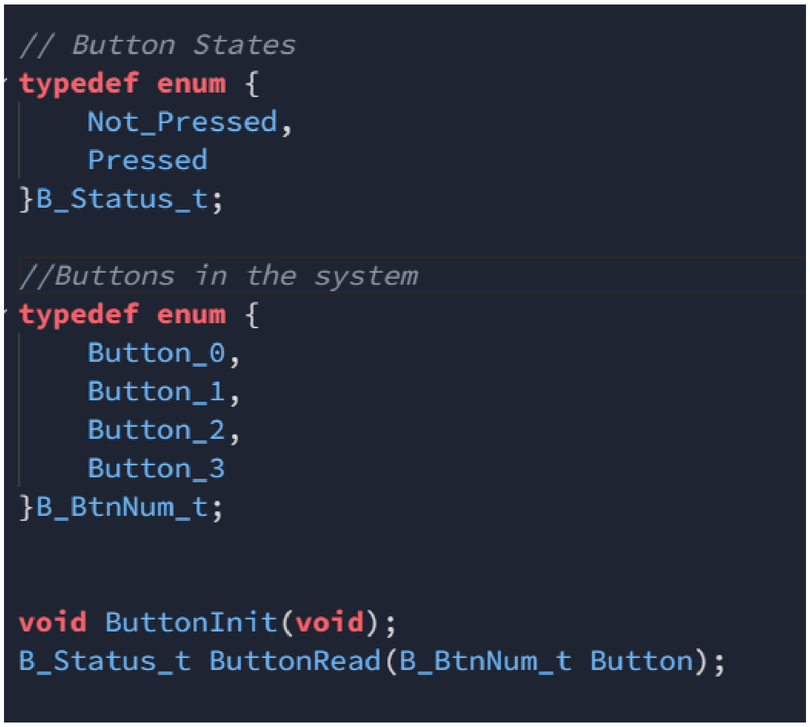
```c
//Return Types of timer functions
typedef enum {
    ERROR,
    SUCCESS
}T_Status_t;

//timer modes
typedef enum {
    Normal,
    CTC,
    F_PWM,
    PC_PWM
}T_TimerMode_t;

//Timer prescaler values
typedef enum {T_Prescaler_1, T_Prescaler_8, T_Prescaler_16, .... , T_Prescaler_1024} T_TimerPrescaler_t;

//Timer APIs
T_Status_t TimerInit(T_TimerMode_t Mode, T_TimerPrescaler_t Prescaler);
T_Status_t TimerStart(uint32_t Time);
T_Status_t TimerCallback(uint32_t Time, void(*fptr)(void));
T_Status_t TimerStop(void);
T_Status_t TimerGetStatus(void);
```

3- PWM APIs:



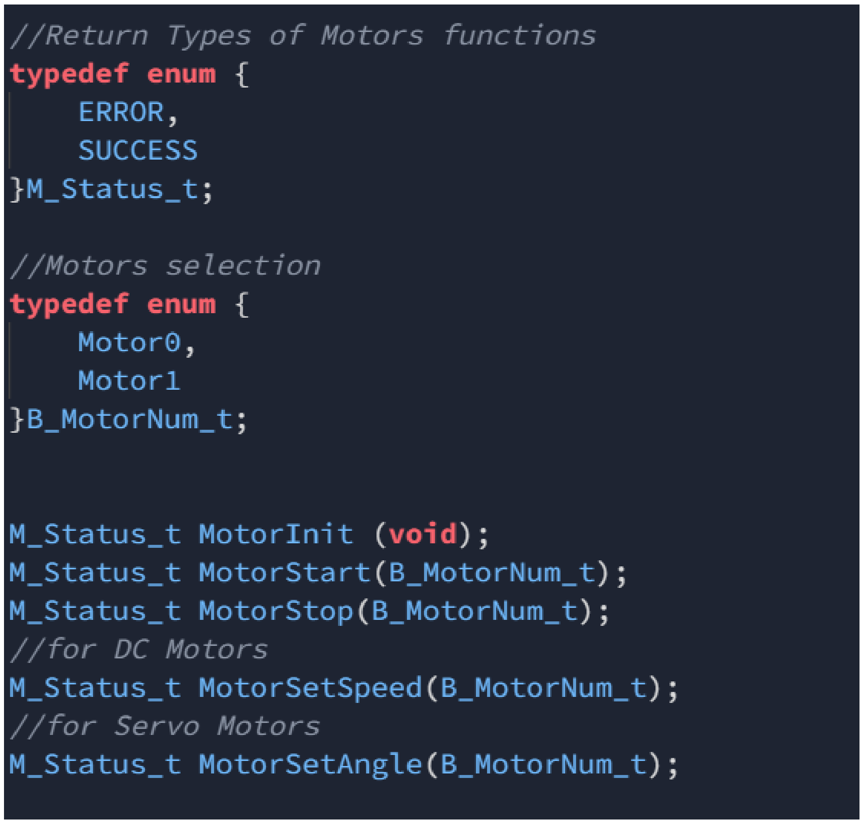
```c
//Return Types of PWM functions
typedef enum {
    ERROR,
    SUCCESS
}PWM_Status_t;

//PWM channels
typedef enum {
    PWM_CH0,        //Timer 0
    PWM_CH1,        //Timer 1 OCR1A
    PWM_CH2,        //Timer 1 OCR1B
    PWM_CH3,        //Timer 2
}PWM_CH_t;

//PWM Modes
typedef enum {
    PWM_Fast,
    PWM_Phase
}T_PWMMode_t;

//PWM APIs
PWM_Status_t PWM_Init(PWM_CH_t Channel,T_PWMMode_t Mode, uint32_t Frequency, uint8_t DutyCycle);
PWM_Status_t PWM_SetFrequency(PWM_CH_t Channel,uint32_t Frequency);
PWM_Status_t PWM_SetDutyCycle(PWM_CH_t Channel,uint8_t DutyCycle);
PWM_Status_t PWM_Stop(PWM_CH_t Channel);
```

1- Button APIs:



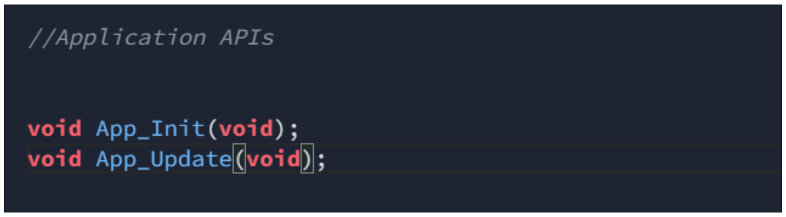
```
// Button States
typedef enum {
    Not_Pressed,
    Pressed
}B_Status_t;

//Buttons in the system
typedef enum {
    Button_0,
    Button_1,
    Button_2,
    Button_3
}B_BtnNum_t;


void ButtonInit(void);
B_Status_t ButtonRead(B_BtnNum_t Button);
```

2- Motor APIs:



```
//Return Types of Motors functions
typedef enum {
    ERROR,
    SUCCESS
}M_Status_t;

//Motors selection
typedef enum {
    Motor0,
    Motor1
}B_MotorNum_t;


M_Status_t MotorInit (void);
M_Status_t MotorStart(B_MotorNum_t);
M_Status_t MotorStop(B_MotorNum_t);
//for DC Motors
M_Status_t MotorSetSpeed(B_MotorNum_t);
//for Servo Motors
M_Status_t MotorSetAngle(B_MotorNum_t);
```

III) Application APIs:



```c
//Application APIs


void App_Init(void);
void App_Update(void);
```