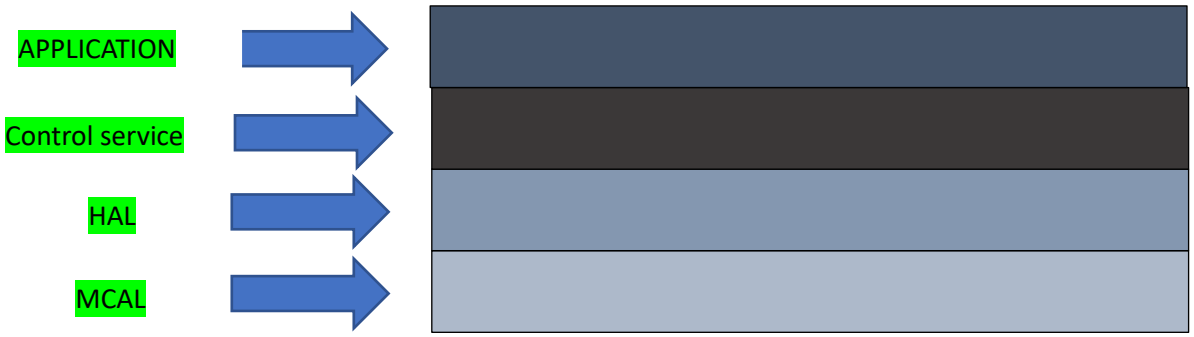
1-Layered architecture:

MCAL: layered includes all the microcontrollers peripheral drivers.

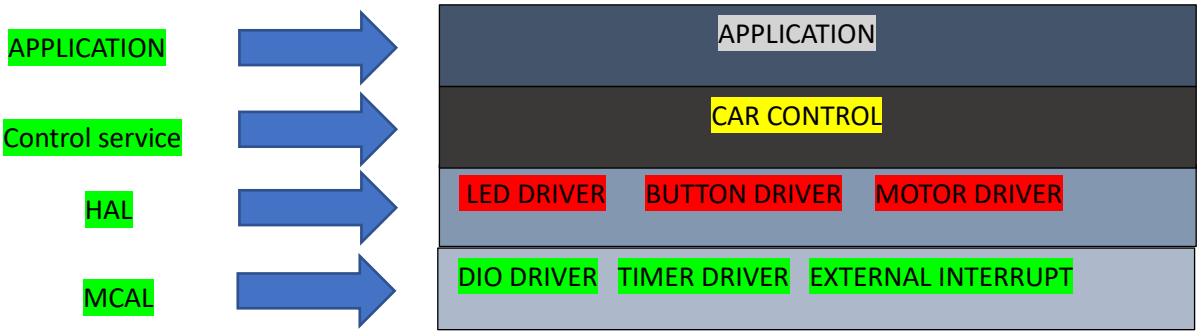
HAL layered: includes all the external hardware peripherals drivers.

CONTROL SERVICE: layered abstracted from the below layers and interact with application directly.

APPLICATION layered: contain the application logic and abstracted from the hardware layers.



2-System modules:



3-Modules APIs

DIO APIs:

Typedef enum EN_direction_t {input, output} EN_direction_t;

Typedef enum EN_vlevel_t {low, high} EN_vlevel_t;

typedef enum EN_ports_t {porta=0,portb,portc, portd} EN_ports_t;

typedef enum EN_pins_t {pin0=0, pin1 ,pin2 ,pin3 ,pin4 ,pin5 ,pin6 ,pin7} EN_pins_t;

typedef enum EN_error_t {ok, not_ok} EN_error_t;

/*description: this function initializes the dio driver

*Arguments: port, pin number, direction of the pin

*Return: ok if function fine, Not_ok if something wrong happens

*/

EN_error_t DIO_INIT (EN_ports_t port, EN_pins_t, EN_direction_t direction);

/*description: this function write high or low on specific pin

*arguments: port ,pin number, output level of the pin

*return: ok if function fine, Not_ok if something wrong happens

*/

EN_error_t DIO_WRITE (EN_ports_t port, EN_pins_t, EN_vlevel_t output);

/*description: this function read input on a specific pin

*Arguments: port, pin number

*Return: ok if function fine, Not_ok if something wrong happens

*/

EN_error_t DIO_READ (EN_ports_t port, EN_pins_t);

TIMER APIs:

typedef enum EN_error_t {ok, not_ok} EN_error_t;

Typdef enum EN_modes_t {normal, ctc, fast_pwm, phase_correct} EN_modes_t;

Typedef struct ST_timer_t

{

EN_modes_t mode;

Uint16_t prescaller;

} ST_timer_t;

/*description: this function initializes the timer driver

*Arguments: pointer to timer_config

*Return: ok if function fine, Not_ok if something wrong happens

*/

EN_error_t timer_init (ST_timer_t timer_config);

/*description: this function start the timer with choose prescaller

*Arguments: take the pre_scaller

*Return: ok if function fine, Not_ok if something wrong happens

*/

EN_error_t timer_start (uint16_t prescaller);

/*description: this function stops the timer

*Arguments: void

*Return: void

*/

void timer_stop (void);

/*description: this function set the call back function

*Arguments: pointer to the callee function

*Return: ok if function fine, Not_ok if something wrong happens

*/

EN_error_t timer_ovf_callbackfunc (void (*timer_ovr_ptr) (void));

EN_error_t timer_comp_callbackfunc (void (*timer_comp_ptr) (void));

/*service routine*/

ISR (timer_ovf)

{

Timer_ovf_Ptr ();

}

ISR (timer_comp)

{

Timer_comp_Ptr ();

}

EXTERNAL INTERRUPT APIs :

typedef enum EN_error_t {ok, not_ok} EN_error_t;

Typedef enum EN_edge_t {rising_edge, falling _edge, any_chang, high_level} EN_edge_t;

Typedef enmu EN_external_interrupt {int0, int1, int2} EN_external_interrupt;

Typdef struct ST_interrupt_t

{

EN_external_interrupt interrupt;

EN_edge_t edge;

} ST_interrupt_t;

/*description: this function set the external interrupt configuration

*Arguments: pointer to external interrupt config

*Return: ok if function fine, Not_ok if something wrong happens

*/

EN_error_t external_interrupt_init (ST_interrupt_t interrupt_config);

/*description: this function enable the external interrupt

*Arguments: external interrupt enum

*Return: ok if function fine, Not_ok if something wrong happens

*/

EN_error_t external_interrupt_enable (EN_external_interrupt interrupt);

```
/*description: this function set call back function
*Arguments: pointer to the caller function
*Return: ok if function fine, Not_ok if something wrong happens
*/
EN_error_t external_interrupt0_callbackfunc (void (ext_int0_ptr)(void));
EN_error_t external_interrupt1_callbackfunc (void (ext_int1_ptr)(void));
EN_error_t external_interrupt1_callbackfunc (void (ext_int1_ptr)(void));
/*service routines*/
ISR (ext_int0)
{ext_int0_ptr ();
}
ISR (ext_int1)
{ext_int1_ptr ();
}

ISR (ext_int2)
{ext_int2_ptr ();
}
```

LED APIs

```
Typedef struct ST_led_t
{
EN_ports_t led port;
EN_pins_t led pin;
} ST_led_t;
```

/*description: this function init the led

*Arguments: pointer to struct led

*Return: ok if function fine, Not_ok if something wrong happens

*/

EN_error_t led_init (ST_led_t led);

/*description: this function turn on the led

*Arguments: pointer to struct led

*Return: ok if function fine, Not_ok if something wrong happens

*/

EN_error_t led_on (ST_led_t led);

/*description: this function turn off the led

*Arguments: pointer to struct led

*Return: ok if function fine, Not_ok if something wrong happens

*/

EN_error_t led_off (ST_led_t led);

BUTTON APIs

Typedef EN_button_states_t {pushed, released};

/*description: this function init the button

*Arguments: port , pin number

*Return: void

*/

Void button_init (EN_port_t port, EN_pin_t pins);

```
/*description: this function read button state
*Arguments: port , pin number
*Return: the button state
*/
EN_button_states_t button_read (EN_port_t port, EN_pin_t pins);
```

MOTOR APIs

```
Typedef struct ST_motor_t
{
 EN_ports_t port;
EN_pins_t pin_number_1;
EN_pins_t pin_number_2;
}
/*description: this function init the motor
*Arguments: pointer to struct motor
*Return: void
*/
Void motor_init (ST_motor_t motor);

/*description: this function control the speed of the motor
*Arguments: speed
*Return: void
*/
Void motor_speed(uint8_t speed);
```

/*description: this function stops the motors

*Arguments: void

*Return: void

*/

Void motor_stop(void);

/*description: this function rotates the motor to the right

*Arguments: pointer to struct motor, speed

*Return: void

*/

Void motor_reverse_right(uint8_t speed, ST_motor_t motor);

/*description: this function rotate the motor to the left

*Arguments: pointer to struct motor,speed

*Return: void

*/

Void motor_reverse_left(uint8_t speed, ST_motor_t motor);

CAR CONTROL APIs

Typedef enum EN_carmoving_t {forword, backword, reverse_right, reverse_left , stop}EN_carmoving;

/*description: this function init the car control

*Arguments: void

*Return: void

*/

Void car_init(void);

/*description: this function move the car forword

*Arguments: speed

*Return: void

*/

Void car_forword (uint8_t speed);

/*description: this function rotates the car to the right

*Arguments: speed

*Return: void

*/

Void car_rotated_right (uint8_t speed);

/*description: this function moves the car forward

*Arguments: void

*Return: void

*/

Void car_stop (void)