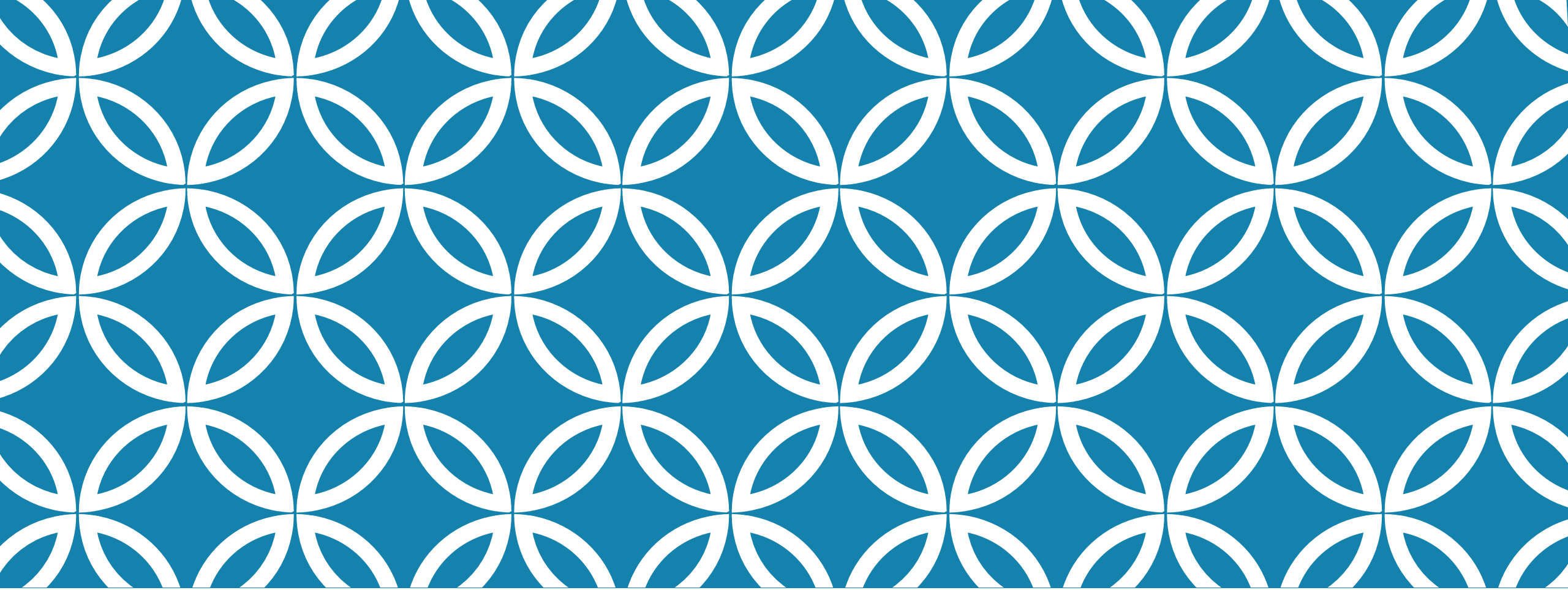


STATIC ARCHITECTURE

EMBEDDED SYSTEMS

OUTLINES

- Introducation
- Modular Programming
- Layerd Archticture
- Folder Structure
- SOLID Principiles
- Steps to make static archticture

INTRODUCTION

- Static architecture **describes** the system **components, interfaces** without any clear description of the system flow in action.

- It uses **modular programming, layered architecture**, and **SOLID principles** to achieve better design.

MODULAR PROGRAMMING

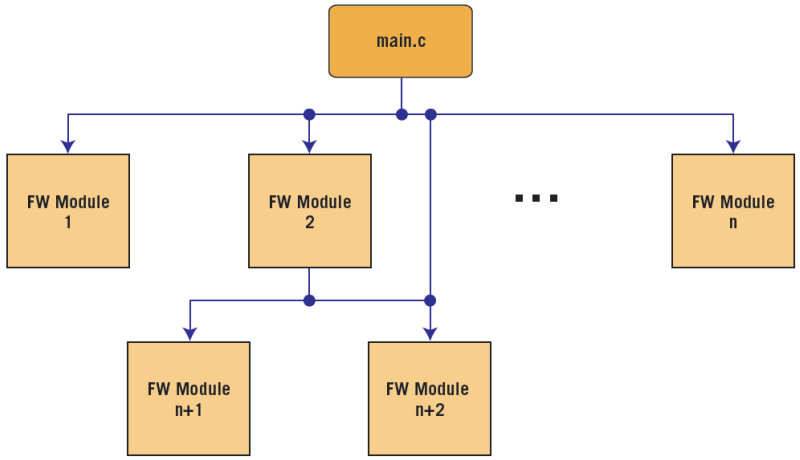
It is a **software design technique** that is intended to **separate and isolate** an **application into small units** that performs a **unique functionality**.

This unit is called a **driver in embedded systems.**

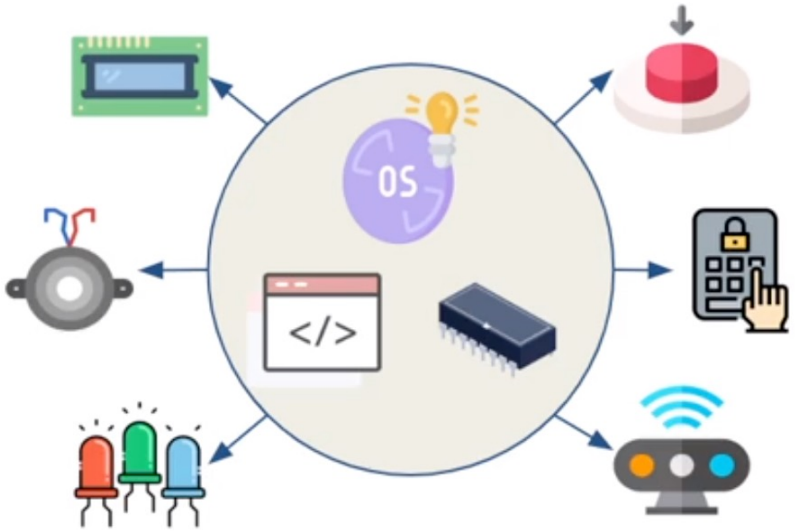
Benefits of modular programming:

- Increase application readability.
- Easier to detect errors.
- Easier to modify and enhance your code.
- Increase code reusability.
- Collaboration.

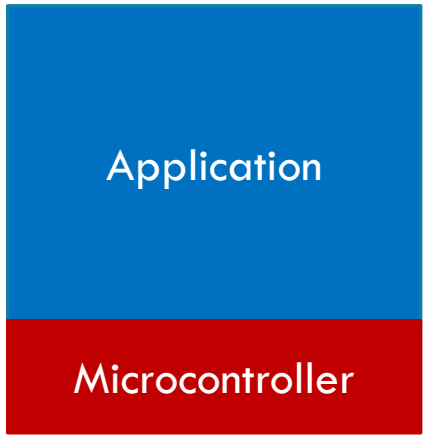
A firmware architecture with modular programming.

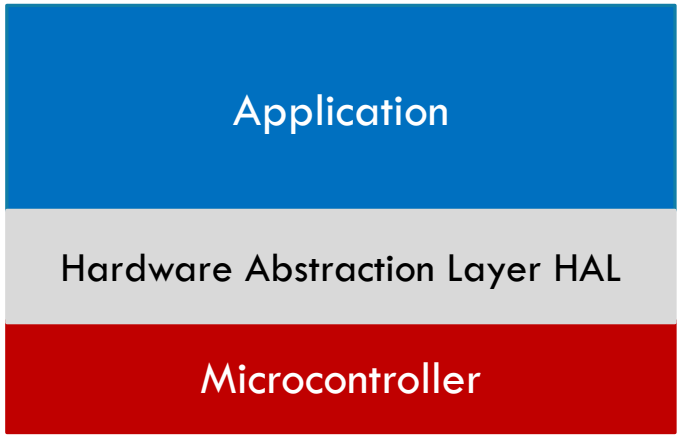


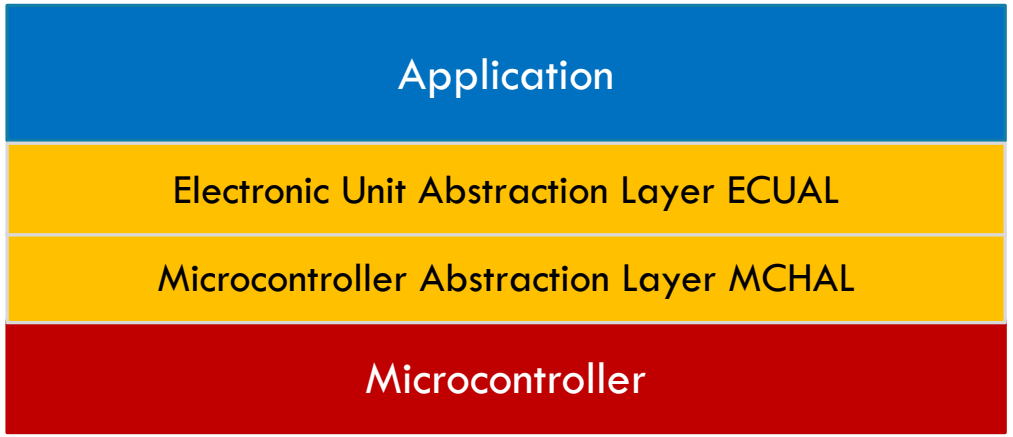
LAYERD ARCHTECTURE



- It is one of the software architecture patterns.
- It is the representation of the system as layers.
- Each layer describe a part of the system.
- Each layer must be abstracted (isolated) from the other layers.

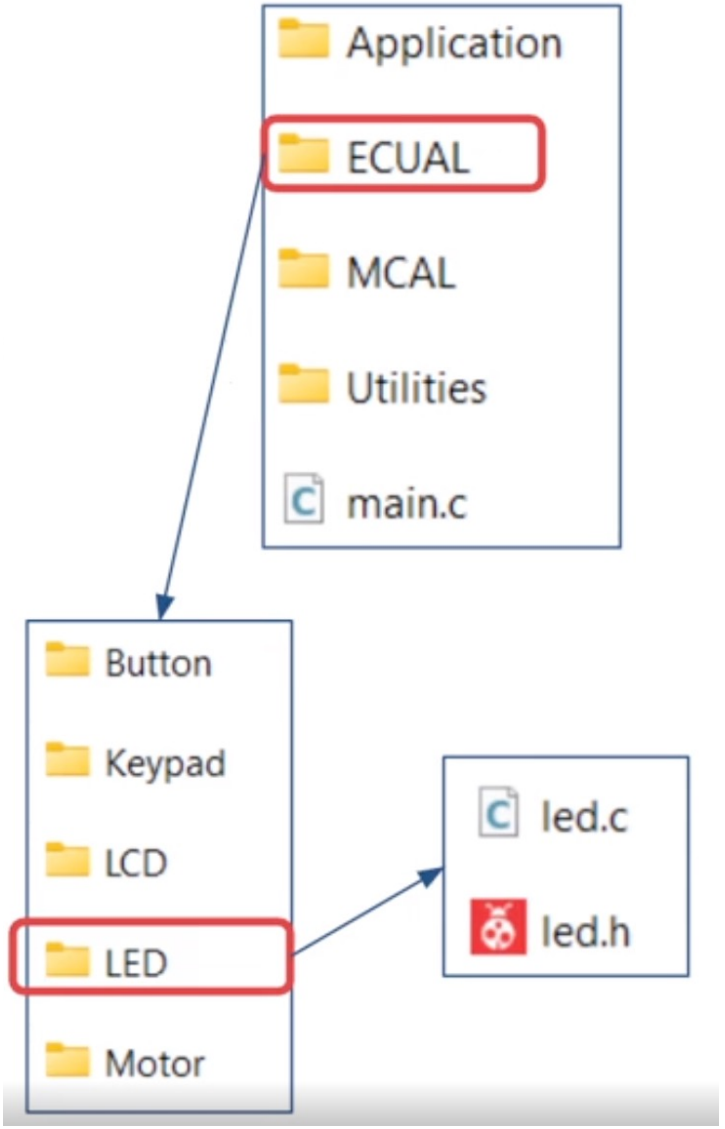






FOLDER STRUCTURE

- You must **organize your project folders** in a way to have **clear modularity** and **abstraction.**
- Prepare your folders:
  - Create folder for each layer.
  - In each layer folder create drivers' folders.
  - Each driver folder contains at least two files, driver_name.c and driver_name.h.
  - You may add another folders, like utilities.
  - You may add any number of header files in each driver if you need.



SOLID PRINCIPILES

S: Single Responsibility.
Each module must be responsible for one thing only.

I: Interface Segregation.
Module user shouldn't be forced to depend upon interfaces that they don't use.

L: Liskov Substitution.
Each module can be substituted with another module that delivers the same functionality.

O: Open/Close.
Each module must be open for extension and close for modifications.

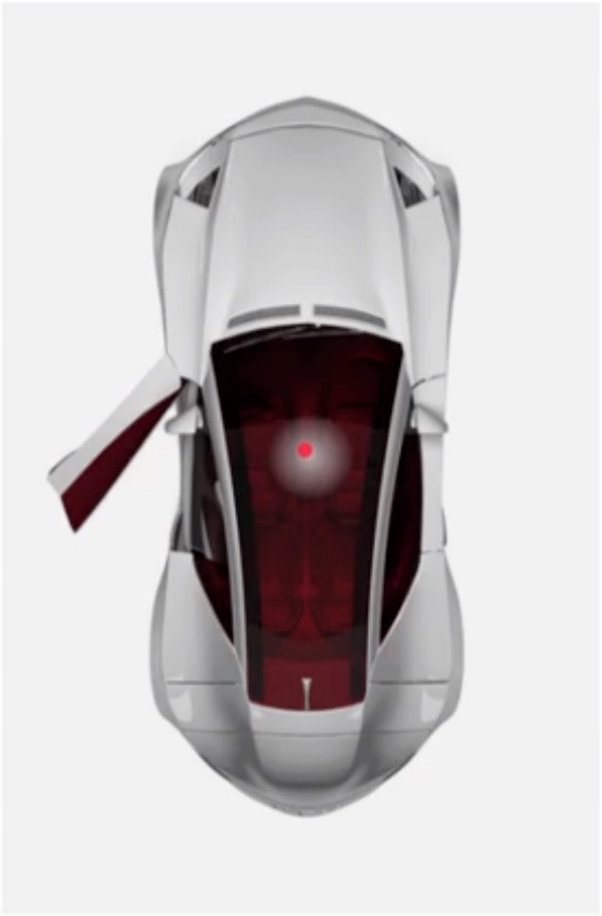
D: Dependency Inversion.
Higher level modules shouldn't depend on lower level modules.
Details should depend on Abstraction.

STEPS TO MAKE STATIC ARCHTICTURE

- **Split** your system into layers.
- **Determine** system modules/Drivers.
- **Decide** which module/Driver will become in which layer.
- **Write the APIs** for each module that will provide specific functionalities for the upper layers.

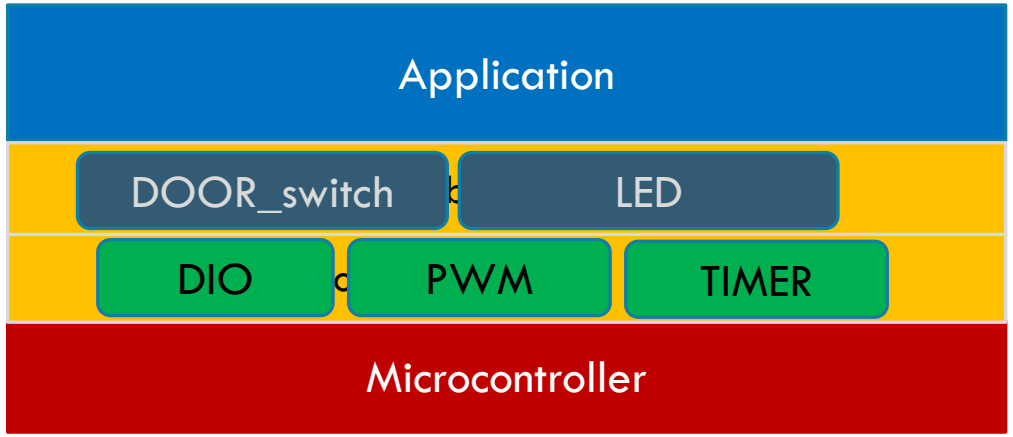
EXAMPLE

- This system supposed to change the light intensity according to the door motion.
- When the door is opened or closed the light will be on with 100% intensity.
- Then after one second the light intensity will be reduced by 50%.
- Then after another second the lights will be off.



EXAMPLE

- Split your system into layers
  - Microcontroller
  - MCAL
  - ECUAL
  - Application
- Divide your system into drivers
  - DIO, PWM, TIMER, DOOR_switch ,and LED
- Decide which driver will become in which layer.



EXAMPLE

Write the APIs for each module that will provide specific functionalities for the upper layers.

- DIO APIs

```
void DIO_init(ST_DIO_config_t* configurations);
void DIO_write(uint8_t port,EN_pins pin, uint8_t data);
void DIO_read(uint8_t port,EN_pins pin, uint8_t *data);
void DIO_toggle(uint8_t port,EN_pins pin);
```

- PWM APIs

```
void PWM_init(ST_PWM_config_t* configurations);
void PWM_start(EN_frequency_t frequency, EN_duty_t dutyCycle);
void PWM_stop(void);
```

- TIMER APIs

```
void TIMER_init(ST_TIMER_config_t* configurations);
void TIMER_start(uint64_t ticks);
void TIMER_read(uint8_t *value);
void TIMER_set(uint8_t value);
void TIMER_checkStatus(uint8_t *status);
```

EXAMPLE

Write the APIs for each module that will provide specific functionalities for the upper layers.

LED API

```
EN_LED_status_t led_init(ST_LED_config_t *led);
EN_LED_status_t led_turn_on(ST_LED_config_t *led);
EN_LED_status_t led_turn_off(ST_LED_config_t *led);
EN_LED_status_t led_toggle(ST_LED_config_t *led);
```

SWITCH API

```
EN_DSW_error_t dsw_init(const ST_dsw_t* dsw);
EN_DSW_error_t dsw_read_state( ST_dsw_t *dsw, EN_DSW_state_t *dsw_state);
```