Static architecture

By: Yehia M. Abu Eita

Outlines

- Introduction
- Modular programming
- Layered architecture
- Folder structure
- SOLID principles
- Steps to make your static design

Introduction

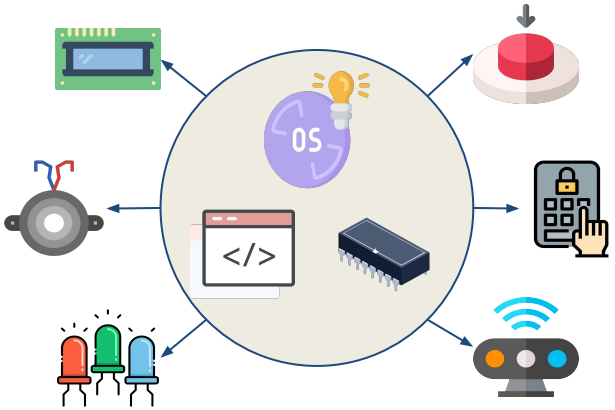
- Static architecture **describes** the system **components**, **interfaces** without any clear description of the system flow in action.
- It uses **modular programming**, **layered architecture**, and **SOLID principles** to achieve better design.

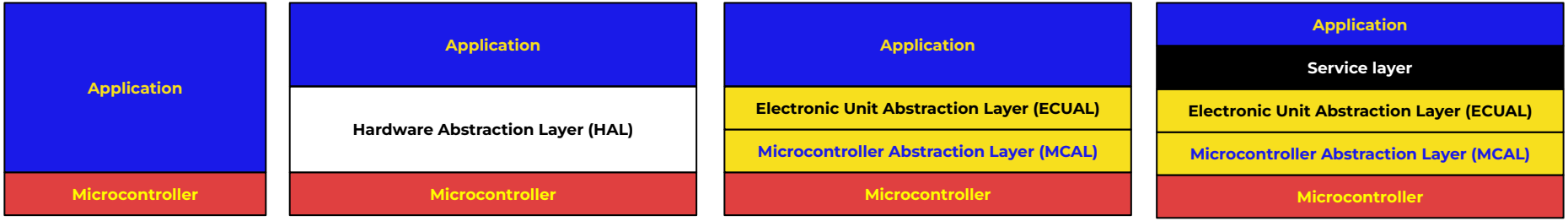
Modular programming

- 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.**

Layered architecture

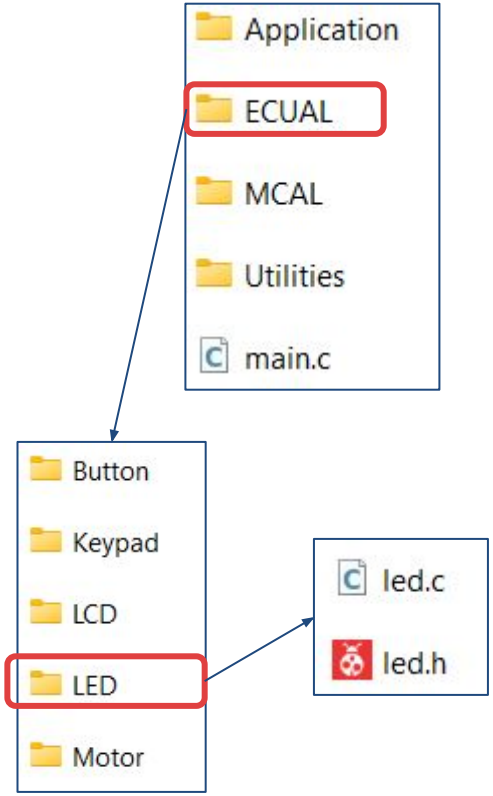
- It is one of the **software architecture patterns.**
- It is the **representation** of the system as **layers**.
- Each layer describe a part of the system.
- Each layer must be **abstracted (isolated) from the other layers.**





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 principles

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.

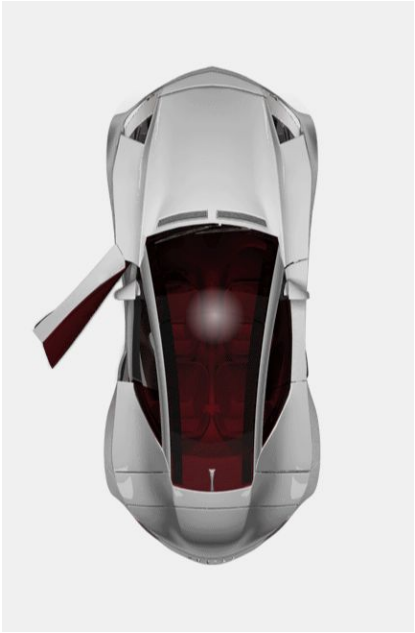
D: Dependency Inversion.
Higher level modules shouldn't depend on lower level modules. Details should depend on Abstraction.

Steps to make your static design

- **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.

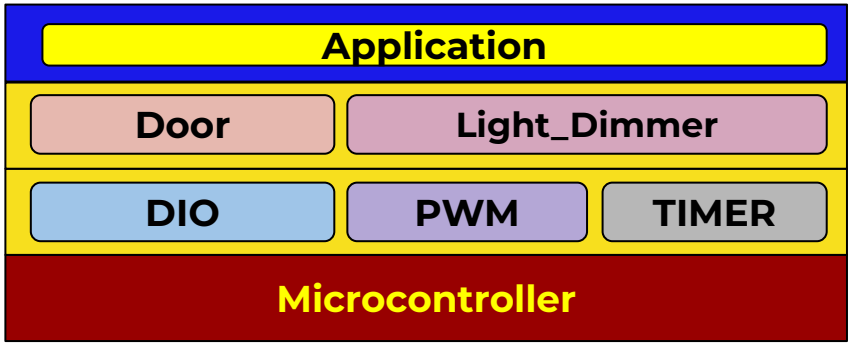
Steps to make your static design

- 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.



Steps to make your static design

- Split your system into layers
  - The system may be divided to 4 layers.
  - Microcontroller
  - MCAL
  - ECUAL
  - Application
- Divide your system into drivers
  - DIO, PWM, TIMER, Door, and Light_Dimmer
- Decide which driver will become in which layer.



Steps to make your static design

- **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);
```

Summary

- Now you are familiar with static architecture
- Nothing is wrong in design, it is just bad, good, or very good
- Remember, as much as you apply the SOLID principles as good as design you will have
- Remember, a driver is that thing which controls the devices programmatically