1.7 怎样选择设计模式（嵌入式 C 视角）

1. 原则

1. **看变化点**
   - 哪些模块会经常改动?
   - 哪些硬件/算法/协议可能换?
2. **看复用需求**
   - 是否希望相同框架支持多硬件/算法?
   - 是否需要在不同项目中复用模块?
3. **看耦合度**
   - 上层是否依赖底层实现?
   - 是否需要抽象接口来隔离变化?

2. 嵌入式 C 的对应做法

设计模式类型	C 里对应实现	适用场景
策略模式（Strategy）	函数指针 + struct	不同算法可替换，如滤波器、PID
模板方法（Template Method）	框架函数 + 回调	固定流程，可插入变化点
代理 / 委托（Proxy / Delegation）	函数指针/ops委托底层实现	抽象硬件接口、延迟绑定模块
组合模式（Composite）	struct 内嵌或组合	多模块组合，例如三轴平台控制
适配器模式（Adapter）	包装函数/struct	不改上层逻辑，替换底层硬件接口

选择方式总结：

> 先找变化 → 看是否需要复用 → 决定用哪种模式隔离变化

目　的	设计模式	可变的方面
创建	Abstract Factory(3.1)	产品对象家族
	Builder(3.2)	如何创建一个组合对象
	Factory Method(3.3)	被实例化的子类
	Prototype(3.4)	被实例化的类
	Singleton(3.5)	一个类的唯一实例
结构	Adapter(4.1)	对象的接口
	Bridge(4.2)	对象的实现
	Composite(4.3)	一个对象的结构和组成
	Decorator(4.4)	对象的职责，不生成子类
	Facade(4.5)	一个子系统的接口
	Flyweight(4.6)	对象的存储开销
	Proxy(4.7)	如何访问一个对象；该对象的位置
行为	Chain of Responsibility(5.1)	满足一个请求的对象
	Command(5.2)	何时、怎样满足一个请求
	Interpreter(5.3)	一个语言的文法及解释
	Iterator(5.4)	如何遍历、访问一个聚合的各元素
	Mediator(5.5)	对象间怎样交互、和谁交互
	Memento(5.6)	一个对象中哪些私有信息存放在该对象之外，以及在什么时候进行存储
	Observer(5.7)	多个对象依赖于另外一个对象，而这些对象又如何保持一致
	State(5.8)	对象的状态
	Strategy(5.9)	算法
	Template Method(5.10)	算法中的某些步骤
	Visitor(5.11)	某些可作用于一个（组）对象上的操作，但不修改这些对象的类

1.8 怎样使用设计模式（嵌入式 C 视角）

1. 浏览模式 → 确认适用性

- 先看这个模式是否解决你的问题
- 仅在真的需要灵活性/复用时使用

```
// 例：传感器算法可能变化 → Strategy 模式适用
```

2. 理解结构 → 找参与者

- struct + 函数指针表 = 模式中的“类和对象”
- 组合 + 委托 = 模式中的“协作”

3. 看示例 → 找实现思路

- 参考 struct + ops + 委托实现
- 了解数据和函数指针如何交互

---

4. 映射到你的模块

- 给 struct/函数起业务相关名字
- 把模式角色映射到实际硬件/算法模块

```
filter_if_t KalmanFilter;   // Strategy 角色
sensor_t sensor = { .filter = &KalmanFilter };
```

---

5. 定义接口和数据结构

- 定义 struct
- 定义函数指针（接口）
- 组合可替换模块

```
typedef struct {
    filter_if_t* filter;
} sensor_t;
```

---

6. 定义操作（命名与职责）

- 上层调用统一接口
- 内部委托给具体实现

```
int sensor_read(sensor_t* s) {
    int v = read_hardware();
    return s->filter ? s->filter->apply(v) : v;
}
```

---

7. 实现操作 → 完成模式落地

- 填充函数指针
- 完成协作逻辑

```
int value = sensor_read(&sensor);
```

- 替换 filter → 上层不动

---

8. 使用注意事项

- 只在真正需要灵活性/复用时使用
- 避免过度增加间接层，尤其是性能敏感的嵌入式模块

一句话总结流程：

> **确认模式 → 理解结构 → 映射模块 → 定义接口 → 委托实现 → 上层调用 → 替换模块时不改上层。**