嵌入式C语言之-

变量的作用域和生命周期

讲师：叶大鹏

变量的作用域和生命周期

- 变量的作用域，指的是变量能够被使用的范围；针对的是程序编译链接阶段；
- 变量的生命周期，指的是变量创建（分配存储空间）到变量销毁（释放存储空间）之间的时间段（即变量的存在时间）；针对的是程序的执行阶段；
- 按照作用域和生命周期的范围，变量可以分为：

  局部变量、静态局部变量、全局变量、静态全局变量

局部变量的作用域

- 局部变量，指的是定义在函数和代码块{ }之内的变量，包括函数参数：

```
QualityLevel GetCo2Level(void)
{
        int32_t raw = GetRawData();
        int32_t level = raw / 100;
        return (QualityLevel)level;
}
```

变量raw的作用域是整个GetCo2Level函数

局部变量的作用域

- 局部变量，指的是定义在代码块{ }之内的变量：

```
int main(void)
{
        uint32_t sum = 0;
        for (uint8_t i = 1; i <= 100; i++)
        {
                sum += i;
        }
        printf("sum = %d\n", sum);
        return 0;
}
```

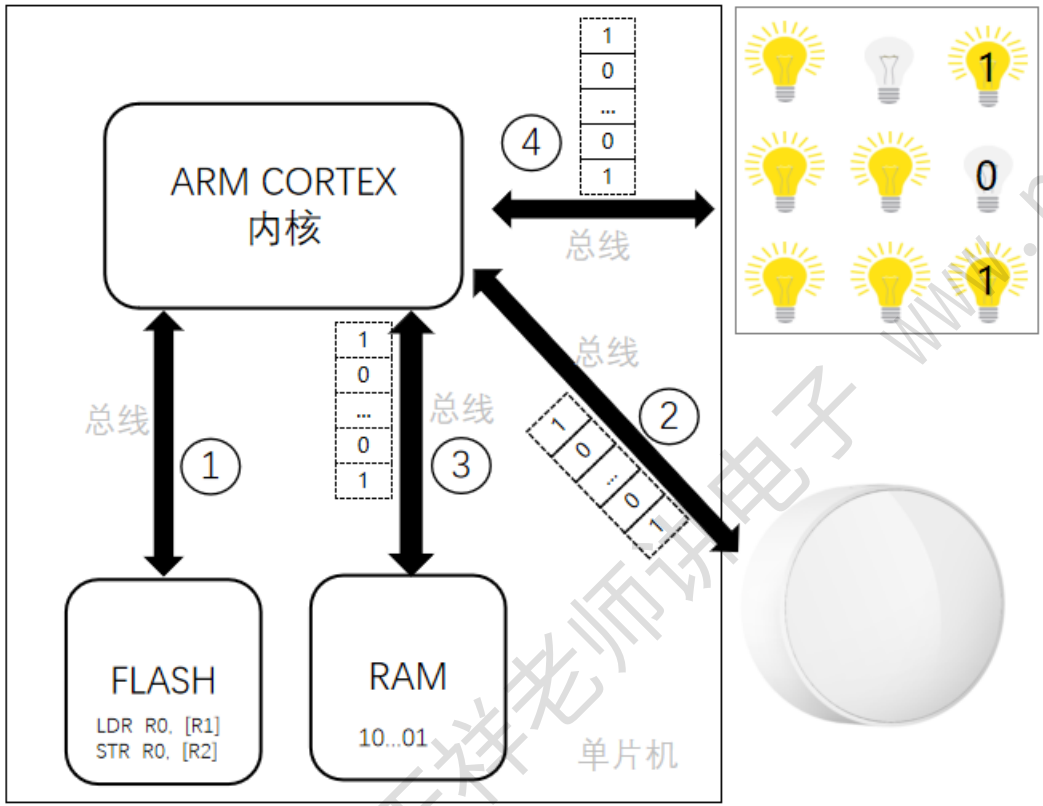
变量i的作用域是整个for循环

变量的生命周期

- 变量的生命周期，指的是变量创建（分配存储空间）到变量销毁（释放存储空间）之间的时间段（即变量的存在时间）；
- 变量的生命周期，针对的是程序的执行阶段。
- 按照作用域和生命周期的范围，变量可以分为：

  局部变量、静态局部变量、全局变量、静态全局变量

程序运行的过程



单片机采集光照强度，动态调节照明亮度：

1.内核通过总线从FLASH中读取指令并响应;

2.内核采集光照传感器的数据并保存在ram中;

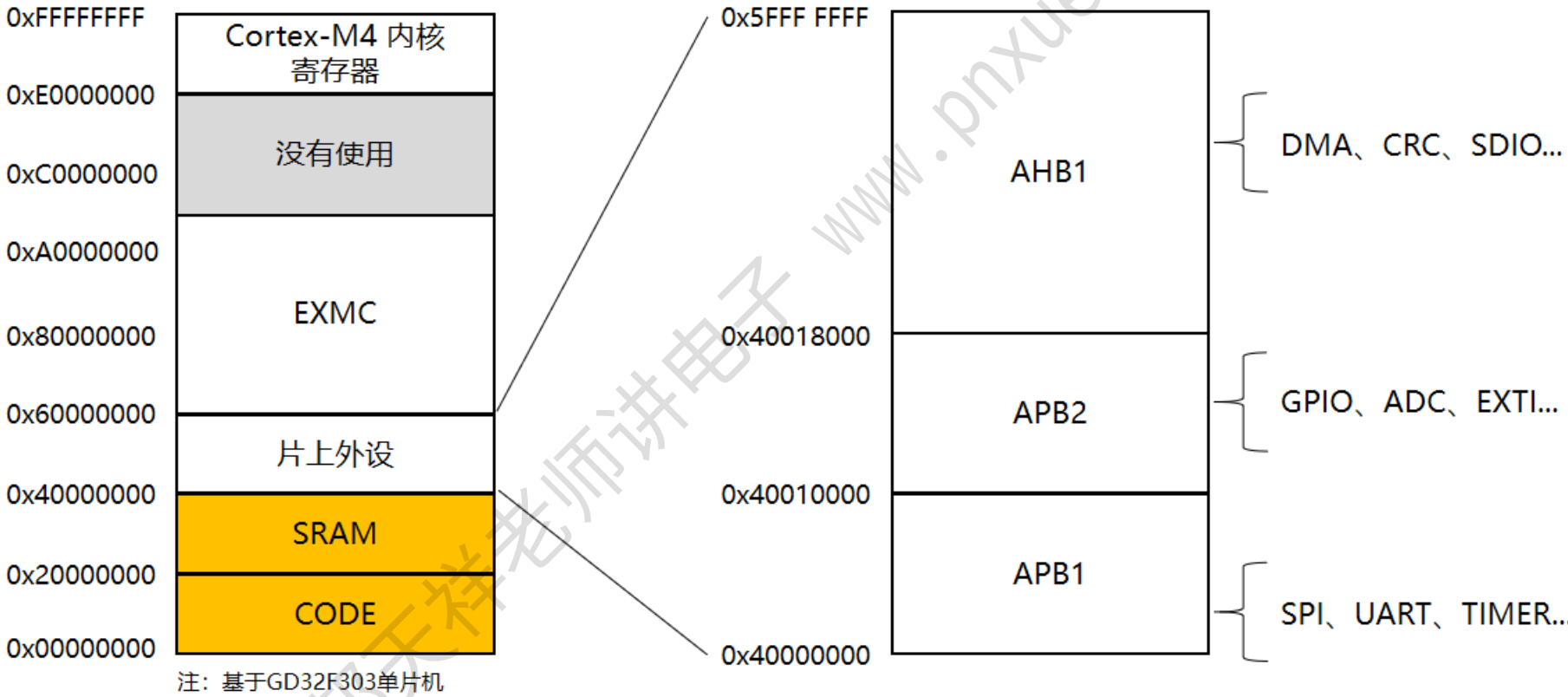
3.内核从ram中读取待处理数据并进行运算;

4.内核将运算结果传送给控制单元进行亮度调节。

- **程序指令保存在flash硬盘中，变量的数值保存在ram内存中。**

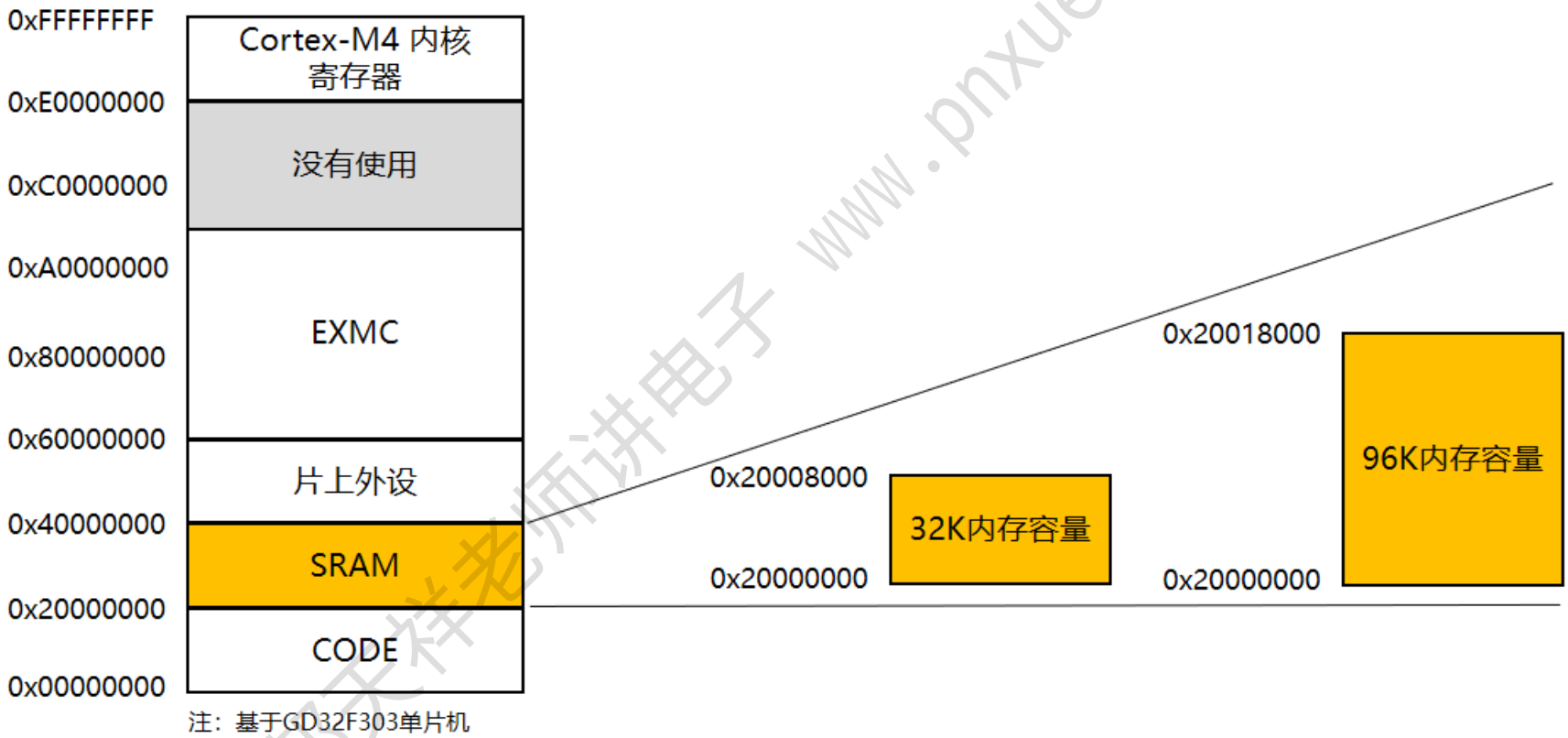
单片机是如何找到寄存器的?

- 每个寄存器都有对应的地址（门牌号），单片机通过地址来访问寄存器，ARM寻址范围4GB，分为多个块，片上外设对应地址范围是0x40000000-0x60000000。



变量的存储空间

- **针对不同型号的单片机，存储容量不同，所以最终内存的寻址范围也不尽相同：**



注：基于GD32F303单片机

变量的存储空间

- 内存根据变量的生命周期和用途会划分为多个部分：

局部变量的生命周期

- 局部变量是存储在栈空间上，它的生命周期是被定义时分配存储空间，销毁时（函数或代码块结束时）释放存储空间。

没有使用
栈，保存函数的参数和局部变量
堆，使用malloc等分配的动态内存
未初始化、初始化为0的全局变量、静态全局变量、静态局部变量
已初始化，并且值为非0的全局变量、静态全局变量、静态局部变量

局部变量的存储空间

```
int main(void)
{
        volatile QualityLevel co2Level;
        co2Level = GetCo2Level();
        DisplayCo2Level(co2Level);

        volatile QualityLevel pm25Level;
        pm25Level = GetPm25Level();
        DisplayPm25Level(pm25Level);
        return 0;
}
```

```
QualityLevel GetCo2Level(void)
{
        volatile int32_t cRaw = GetRawData();
        volatile int32_t cLevel = craw / 100;
        return (QualityLevel)cLevel;
}
```

PN学堂

局部变量的存储空间

```
int main(void)
{
        volatile QualityLevel co2Level;
        co2Level = GetCo2Level();
        DisplayCo2Level(co2Level);

        volatile QualityLevel pm25Level;
        pm25Level = GetPm25Level();
        DisplayPm25Level(pm25Level);
        return 0;
}
```

```
QualityLevel GetPm25Level(void)
{
        volatile int32_t pRaw = GetRawData();
        volatile QualityLevel pLevel;
        ...
        return pLevel;
}
```

THANK YOU!