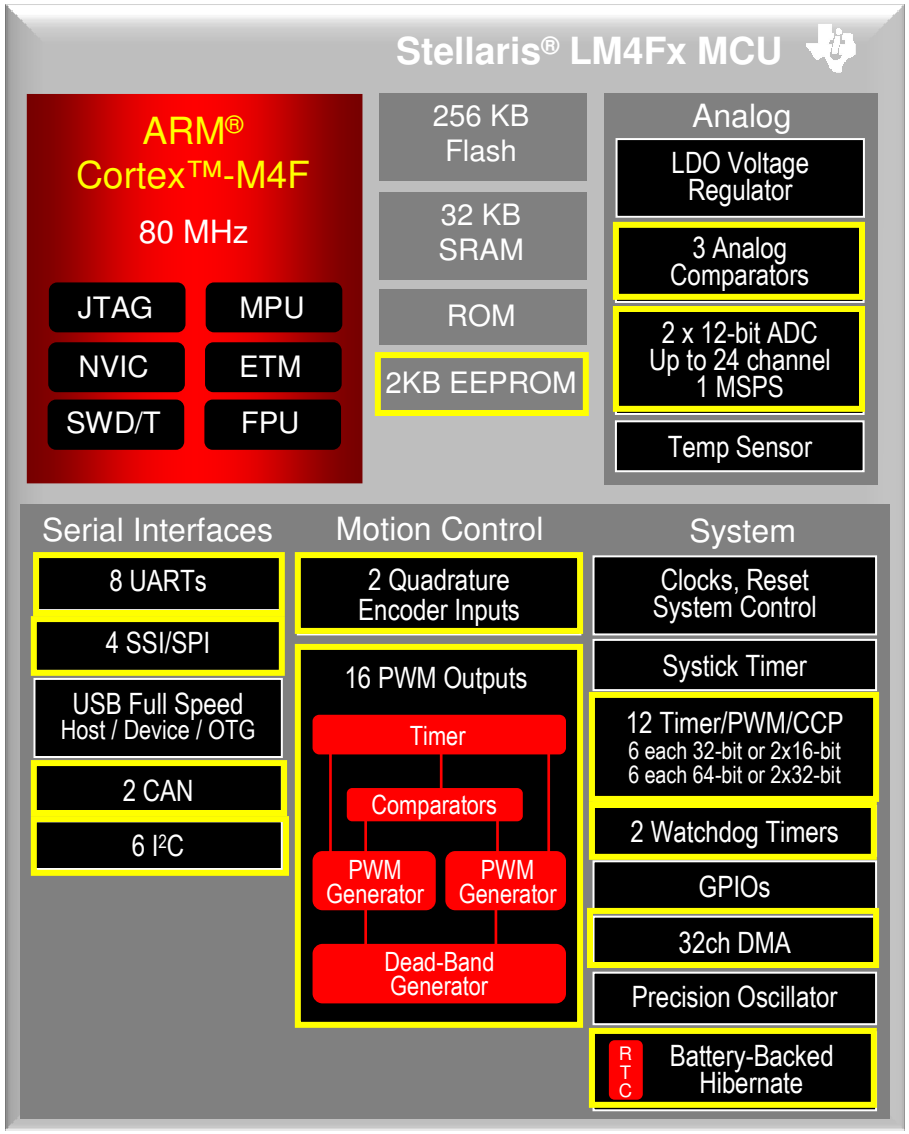
Stellaris® ARM® Cortex™-M4F Blizzard Peripheral 概述

Agenda

- **Stellaris LM4F Blizzard**系列通用规范
- **ARM® Cortex™-M4F** 特点
- 其它系统特点
  - 低功耗
  - 看门狗定时器
- 定时器和**GPIO**
- 模拟外设
- 连通性
- 运动控制外设

Stellaris® LM4F Blizzard 通用规范

Stellaris® ARM® Cortex™-M4F *Blizzard*



连通性

- **CAN, USB H/D/OTG, SPI, I2C, UARTs**

集成高性能模拟外设

- 双 **12-bit ADC,** 速度**1 MSPS**
- 三个模拟比较器

卓越的低功耗性能

- 低达 **370 uA/MHz**
- 从低功耗模式下唤醒只需**500µs**
- 低功耗模式下，带**RTC**功能，电流低至**1.7uA**

全面的 **roadmap**

- 更高速度
- 更大容量
- 超低功耗

Stellaris® LM4F Blizzard 技术规范

- 基于ARM® Cortex™M4F内核
  - 符合IEEE 754的单精度浮点运算单元
  - 嵌入式跟踪宏单元ETM（Embedded Trace Macrocell)
- 系统时钟频率高达80 MHz
- 最多24个定时器 (12个16-bit和12个32-bit)
  - 6个16/32 bit 定时器 (1个32位可变成2个16位定时器)
  - 6个32/64 bit 定时器 (1个64位可变成2个32位定时器)
- 2个PWM模块,每个PWM模块包含4个PWM生成器
- 2个QEI模块
- 支持DMA模式

内部存储器

- **256 KB Flash存储器，支持40 MHz以下的单周期访问；预取指功能提高40MHz以上频率的性能。**
- **32 KB单周期SRAM，带Bit-banding功能**
- **内部ROM预先固化StellarisWare软件：**
  - **Stellaris Peripheral Driver Library**
  - **Stellaris Boot Loader**
  - **AES密码表**
  - **CRC校验功能**
- **2KB EEPROM**

ARM™ Cortex®-M4F 内核特点

ARM® Cortex™-M4F

- **What's new**
  - 符合**IEEE 754** 单精度浮点运算单元 **(FPU)**
  - 优化的单指令多数据指令**SIMD** （对**16-bit**数据）
  - **32 x 32** 单时钟周期乘法累加**(MAC)**单元，带**64-bit**位结果
  - 饱和算法
  - 嵌入式跟踪宏单元**ETM**
- 优点
  - 使控制环具有更高的精度 **(**例如，电机应用**)**
  - 更快的信号处理能力
  - 更易与**MATLAB**与**LabVIEW**之类工具结合
  - 利用**ETM**更为方便的进行调试和代码优化

ARM® Cortex™-M4 外设

- 浮点单元 **FPU**
- 嵌入式跟踪宏单元**ETM**
- **JTAG**边界扫描和在线编程
- 系统定时器**Systick**
- 嵌套向量中断控制器**NVIC**
- 存储器保护单元**MPU**

浮点单元FPU

- 符合**IEEE 754**
- **32-bit**指令集，具有单精度数据操作能力
- 乘累加功能提高运算精度
- 硬件支持转换，加，减，乘（乘累加），除，平方根
- **32**个专门的**32**位单精度寄存器，也可以按照**16**个双**word**寄存器方式寻址。
- 可将**FPU**关闭以降低功耗

嵌入式跟踪宏单元
Embedded Trace Macrocell

- **ETM**可用来记录处理器在执行时的状态
- 可以在调试状态下全速执行代码，并且中断和定时器正常工作
  - 允许编程器记录中断服务程序的发生时间，以及如何发生
  - 观察时效性指令是怎样被执行的
- **Keil** 和**IAR**可以增加相关硬件来支持此功能
- 优点：
  - 可以准确看到处理器的时间是花在哪些地方
  - 选择性的对时间花费较多的函数进行优化
  - 跟方便的对中断相关问题进行调试

JTAG

- **JTAG**
  - 符合工业标准的基于边界扫描原理的在线测试接口
  - 在线**FLASH**编程
- 并行**JTAG TAP**
  - 允许访问芯片的**JTAG**用作边界扫描或者访问**Cortex-M4 JTAG**用来**Debug**
- **SWD/SWT**
  - 只需**2**个引脚即可进行调试

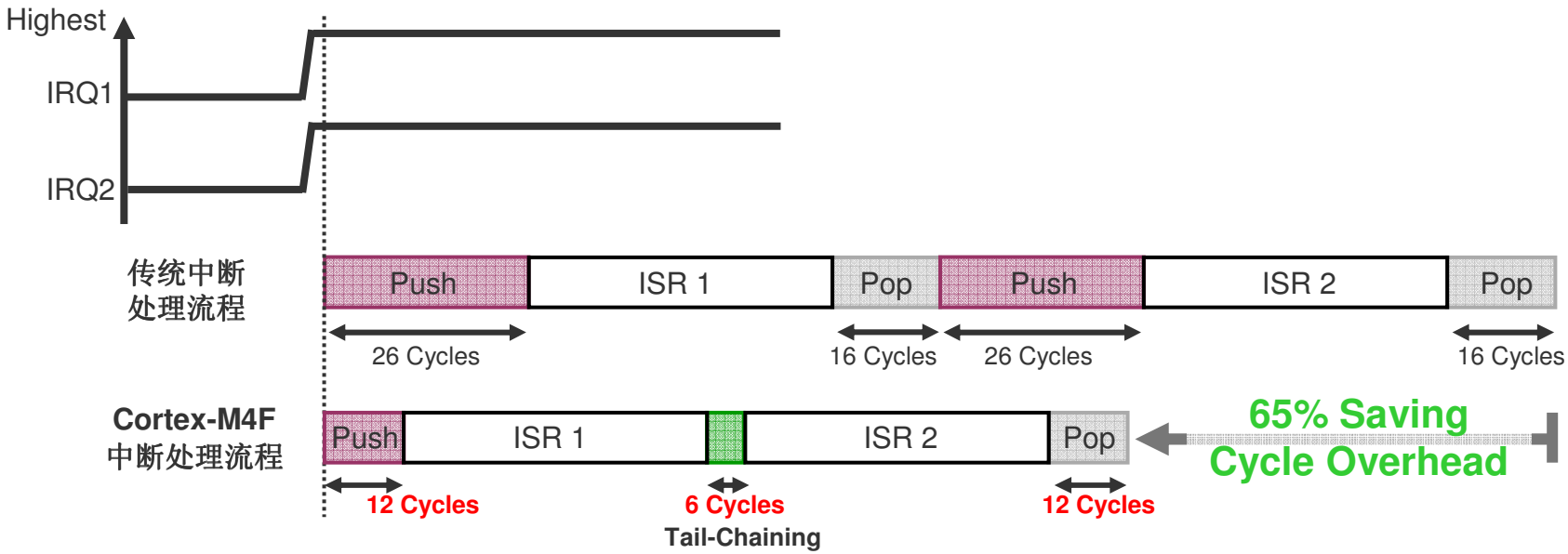
系统定时器 SysTick

- **24-bit** 写清除，递减，减至**0**装载计数器
- **LM4F**的新特点：
  - **SysTick**可以在**PIOSC/4**时钟下运行，而非原先的系统时钟
- 可配置用于
  - **RTOS**的基准定时器
  - 基于系统时钟的高速报警定时器
  - 可变频率的报警或信号定时器
  - 简单用来测量时间的计数器
  - 内部时钟源

嵌套向量控制器
NVIC

- 处理异常和中断
- 8个可编程的优先级，优先组别
- 自动现场保护和恢复
- 自动读取向量表入口
- 抢占式/嵌套中断
- 尾链

中断响应 – 尾链 Tail Chaining



- 从**IRQ1**到**ISR1**需**12**个周期
- 从**ISR1**退出到进入**ISR2**需**6**个周期
- 从**ISR2**返回需**12**个周期

存储器保护单元

- 特点
  - **8**个保护区域(禁止访问, 读/写, 只读)，范围**32B ~ 4GB**
  - 禁止访问
  - 对于重叠的**region,**以优先级高（**region**号大）的**region**的属性为准
  - 当访问与**MPU**配置的属性不匹配时，会产生**fault**中断
- 优点
  - 实行优先规则
  - 分区处理
  - 执行访问准则

Stellaris LM4F Blizzard 系统特点

DMA, HIB, WDT

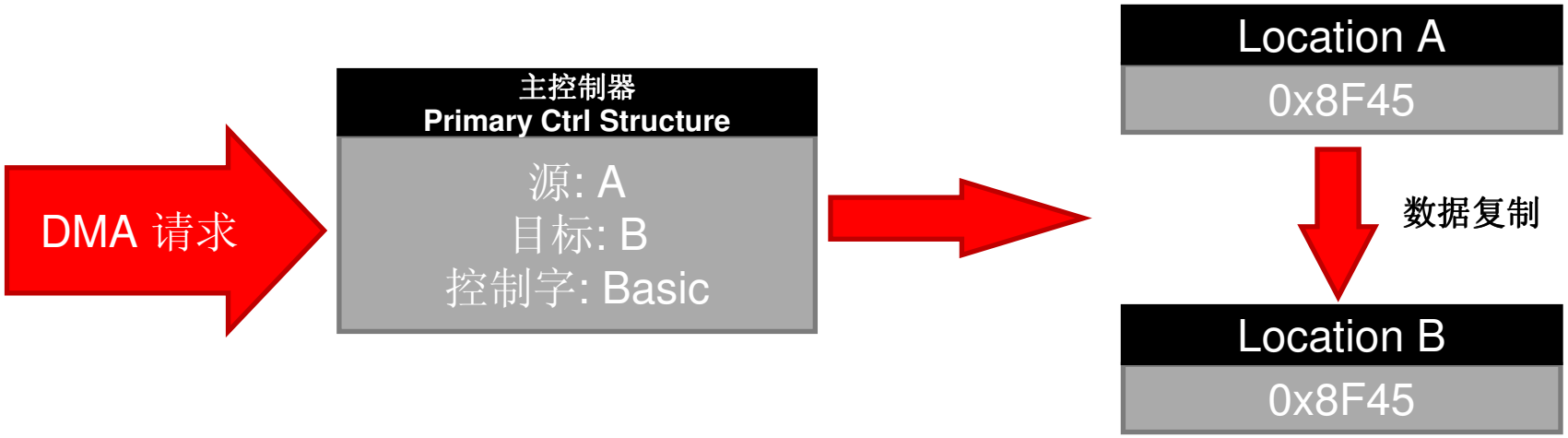
DMA

- 32个通道可配置的uDMA控制器
- 对所支持的外设有专门的通道
  - 每个通道都有接收和发送路径供双向外设使用
  - 多种数据尺寸，2级优先级，可屏蔽设备请求
- 传输完成产生中断，每个通道有独立的中断
- 内核总是拥有总线访问的优先权
- 多种传输模式:
  - 基本传输
  - 自动传输
  - Ping-pong传输（针对外设的连续数据流）
  - Scatter-gather传输: 有传输请求时，按照一个可编程的清单将数据从不同地方传输到某个目标位置
- DMA 请求支持的外设:
  - UART, Timer, USB, ADC, SSI, GPIO

DMA 请求

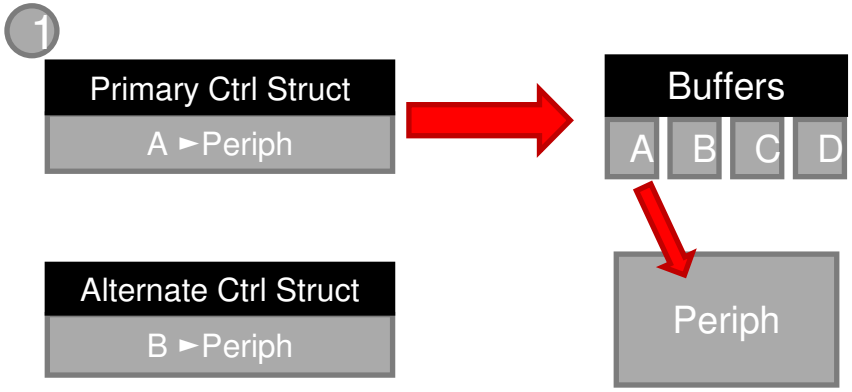
- DMA请求由软件或外设发出送至DMA模块，由DMA通道控制器（ctrl structure)执行
- DMA通道控制器(ctrl structure)包含：源地址，目标地址和如何传输数据（控制字control word）
- 控制字(control word)包含传输模式，以及数据传输的其他信息
- 每个通道都有一个主要控制器(Primary ctrl structure)和一个备份控制器(Alternate ctrl structure)

DMA 基本传输和自动传输

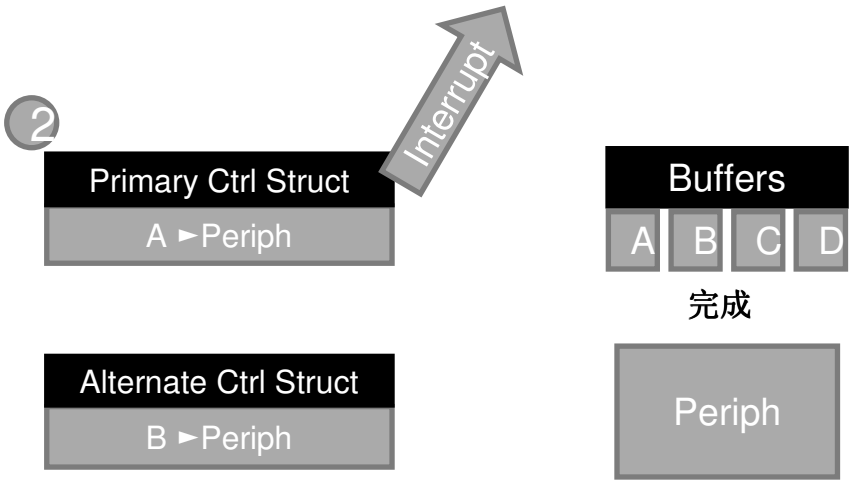


- 基本传输（**Basic Transfer**）是当请求有效时，将数据从一个地址传输到另一个地址。并且只有在请求有效的情况下会存在数据传输。这种方式适用于那些在整个传输过程中都始终保持请求有效的外设。
- 自动传输（**Auto Transfer**）与基本传输类似。但是当请求结束后，仍然会完成这次传输。比较典型的是软件请求，软件请求不必在整个传输过程中都保持有效。

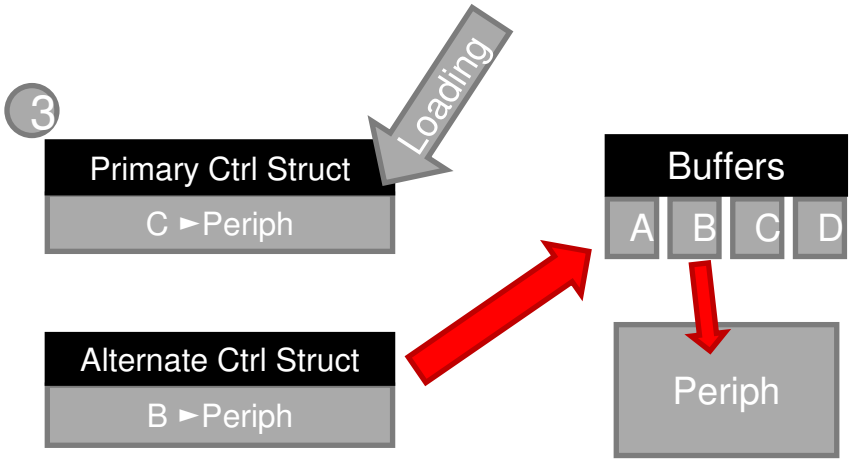
DMA Ping-Pong传输



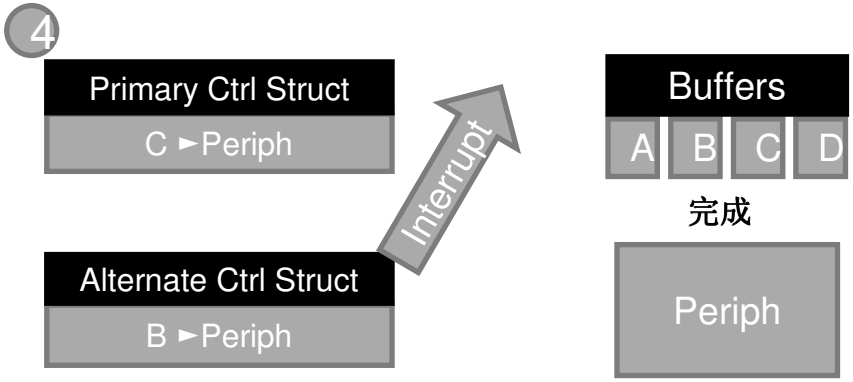
当副控制器空闲（或正在装载准备下次传输）时，主控制器正在执行传输



主控制器发送中断请求，表示它的这次传输完成。

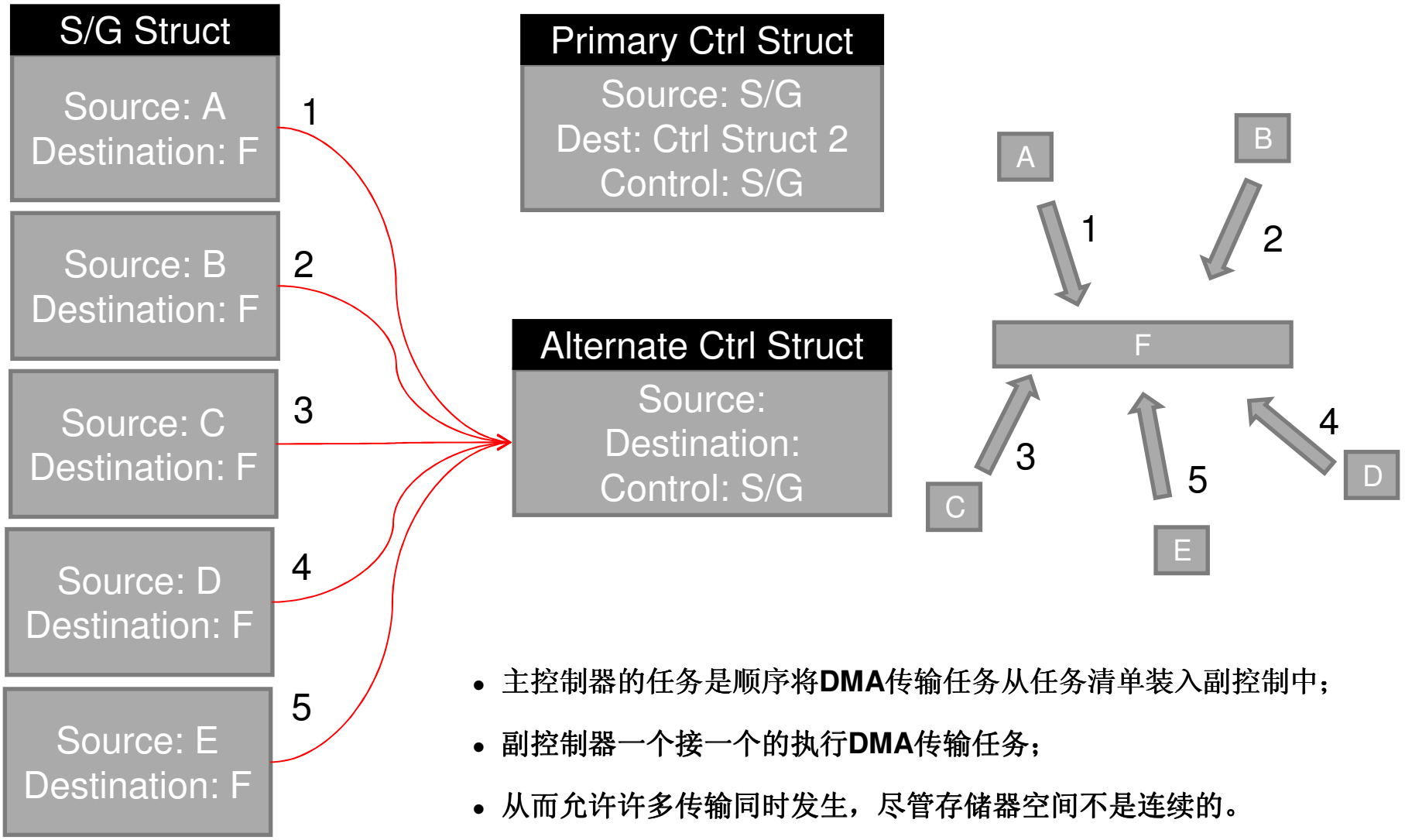


当主控制器装载准备下次传输时，副控制器执行传输



副控制器发送中断请求，表示它的这次传输完成。
重新回到第一步.

Scatter-Gather 传输



- 主控制器的任务是顺序将**DMA**传输任务从任务清单装入副控制中；
- 副控制器一个接一个的执行**DMA**传输任务；
- 从而允许许多传输同时发生，尽管存储器空间不是连续的。

HIB和低功耗模式

Stellaris Blizzard系列提供以下三种低功耗模式：

- 睡眠模式 **Sleep Mode**
- 深度睡眠模式 **Deep-Sleep Mode**
- 休眠模式 **Hibernation**

睡眠模式和深度睡眠模式

睡眠模式 **Sleep Mode**

- 关闭主处理器时钟
- 当收到中断请求时，被唤醒

深度睡眠模式 **Deep Sleep**

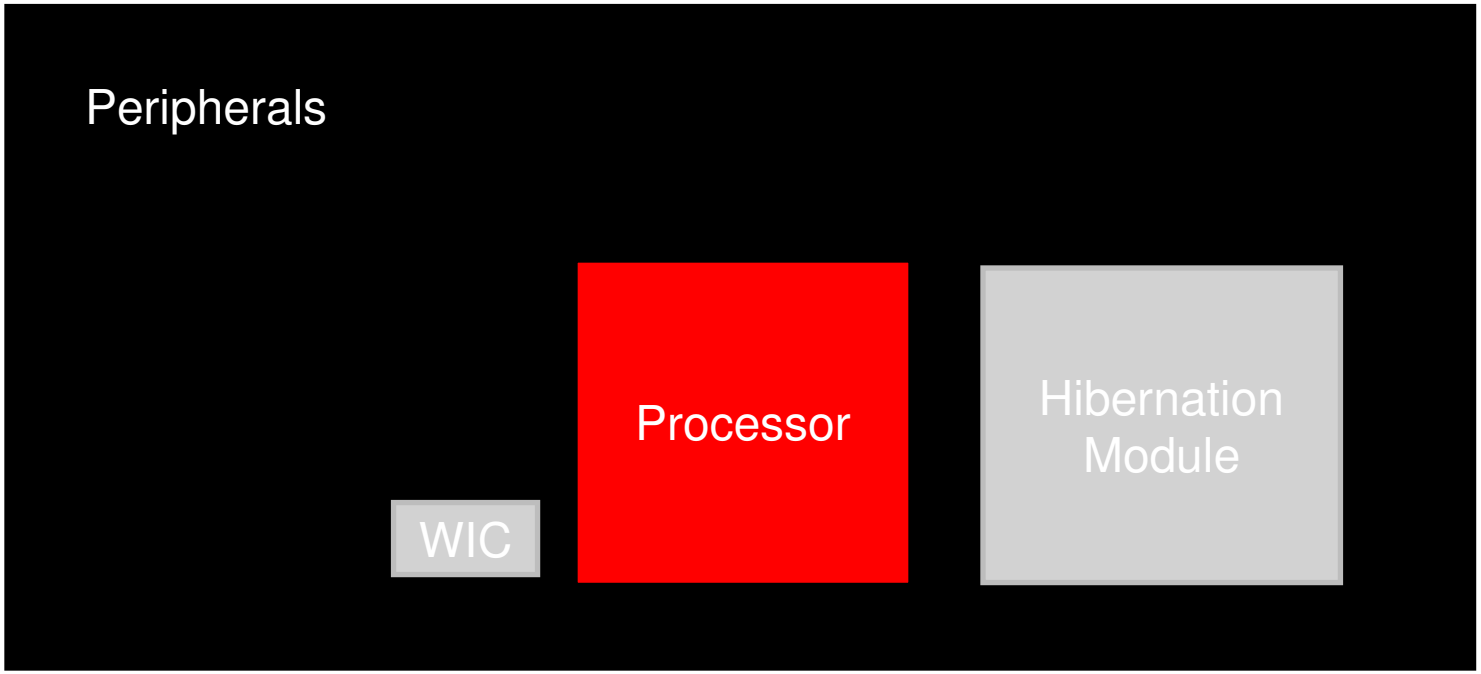
- 关闭系统时钟，**PLL,**和**Flash**
- 使能唤醒中断控制器，当有中断发生时将处理器从深度睡眠模式唤醒

休眠模块（HIB）

通过关闭和恢复电源供给来降低功耗。可以将除它自己以外的任何片上资源的供电切断

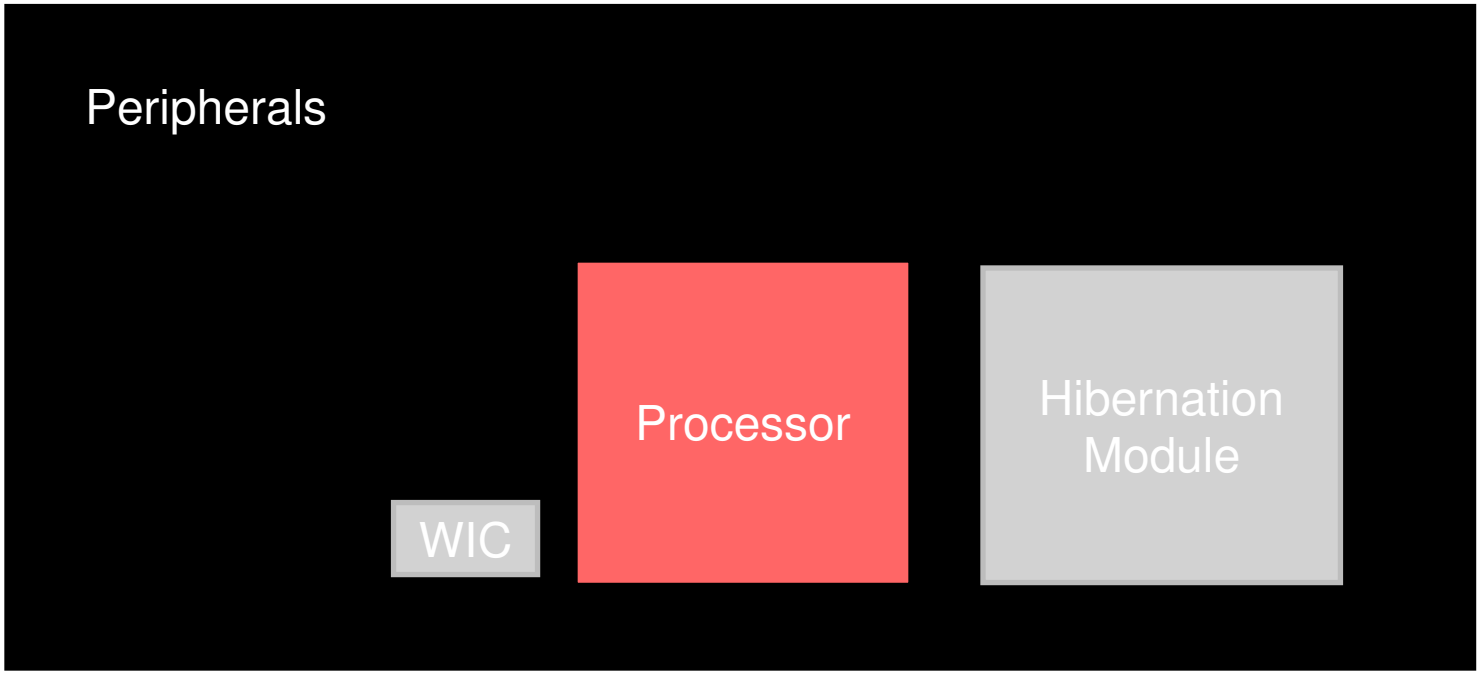
- **32**位实时秒计数器，分辨率**1/32,768**
- 专门的引脚用来通过外部信号唤醒**MCU**
- **GPIO**引脚的状态仍然保持不变，**16**个 **32-bit**的非易失性寄存器的可以在休眠模式下存储系统状态
- 可编程唤醒中断
  - **RTC**匹配
  - 外部唤醒
  - 电池电压低
- 休眠模块有独立的时钟和电源供给

正常运行模式



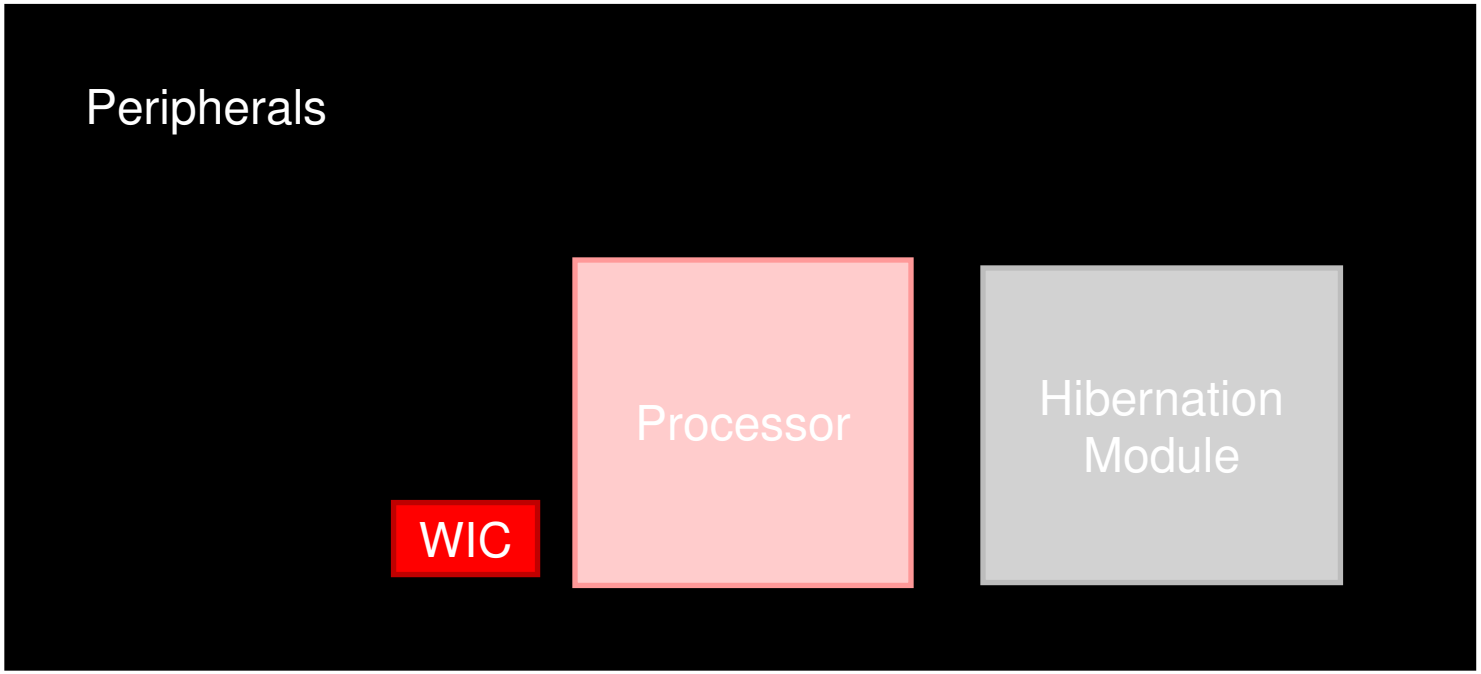
处理器和外设均供电
睡眠，深度睡眠和休眠相关部分都处于断电状态

睡眠模式 Sleep



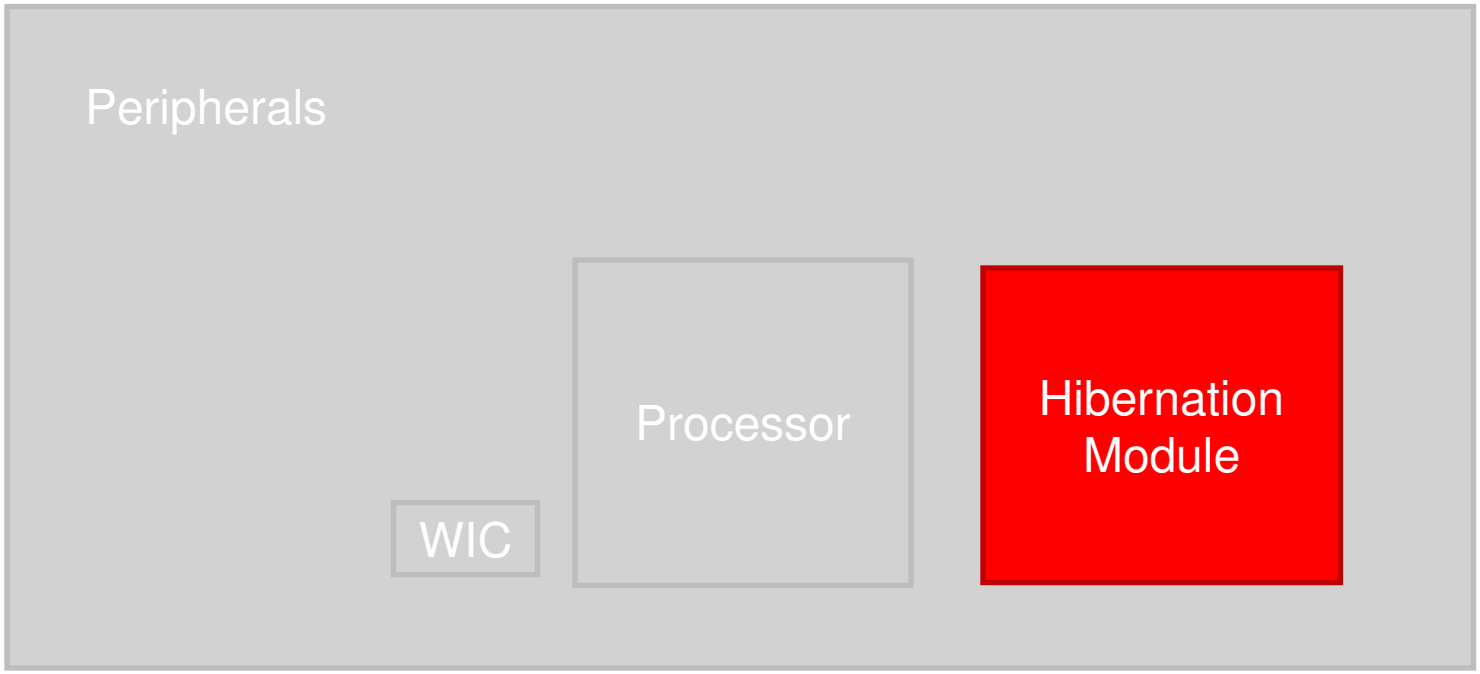
处理器进入低功耗状态，停止执行指令直到收到一个足够重要的中断请求为止

深度睡眠模式 Deep Sleep



处理器进入极低功耗状态，部分功能（如**Systick**）被完全关断。
唤醒中断控制器**WIC**获取中断请求，判断是否需要唤醒处理器。

休眠模式 HIB



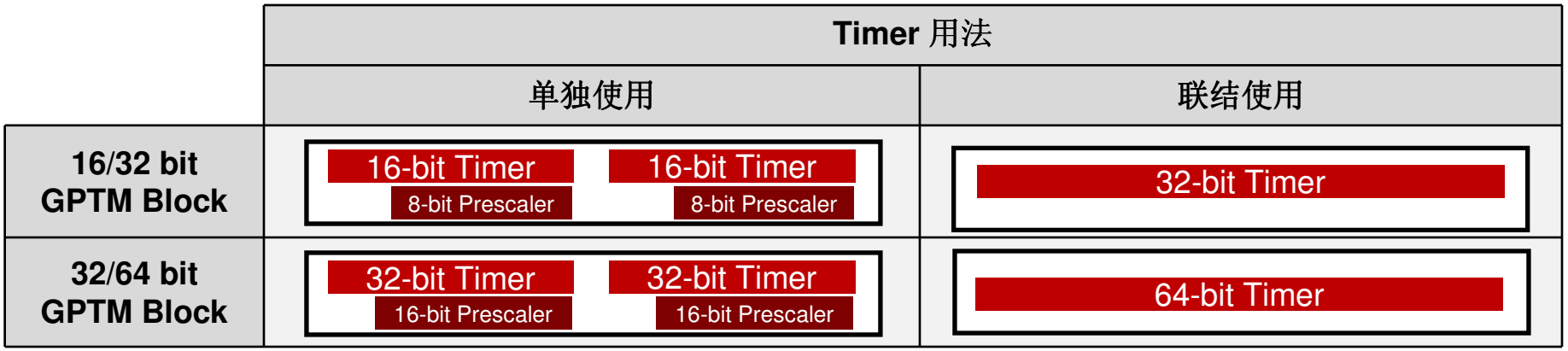
片上大部分电源被切断，包括处理器和所有外设。**HIB**模块由独立的供电和时钟供给以继续工作，直到满足唤醒条件。

看门狗定时器

- 两个32位递减计数器，可以产生可屏蔽或不可屏蔽中断
- 一个看门狗定时器使用系统时钟，另一个的时钟由内部PIOSC供给
- 可根据需要将配置锁定，以防止定时器配置信息被误改写
- 在软件调试时，用户可以将看门狗定时器暂停
- 未及时清除中断时，会产生系统复位
- 作用
  - 从软件错误中恢复
  - 从外部设备故障中恢复，或未能达到预期效果时恢复
  - 当主振荡器损坏或故障时，将系统复位并继续工作

Stellaris LM4F Blizzard Timers 和 GPIO

单次（One-Shot）/周期（Periodic） 定时器模式



- 向上计数或向下计数模式
- 等待触发模式Wait-for-Trigger
- µDMA 触发模式
- 单次（One-Shot） 定时器模式
  - 定时器满足条件后停止计数并且清除标志位
- 周期（Periodic）定时器模式
  - 计数器从0开始向上计数至装载值，重新自动装载，并从0开始重新计数
  - 计数器从预装载值开始递减并重新装载预装载值

实时时钟定时器模式 (Real-Time Clock)

	Timer用法	
	单独使用	联结使用
16/32 bit GPTM Block	N/A	32-bit Timer
32/64 bit GPTM Block	N/A	64-bit Timer

- 只支持向上计数
- 复位后，计数器值为0x01
- 输入时钟(CCP0)必须是32.768kHz，并且被分频为1Hz
- 当计数器值与预装载值匹配时，GPTM开始连续计数直到硬件复位或由软件禁止
- 当定时器值达到最终计数值，定时器重新从0开始计数

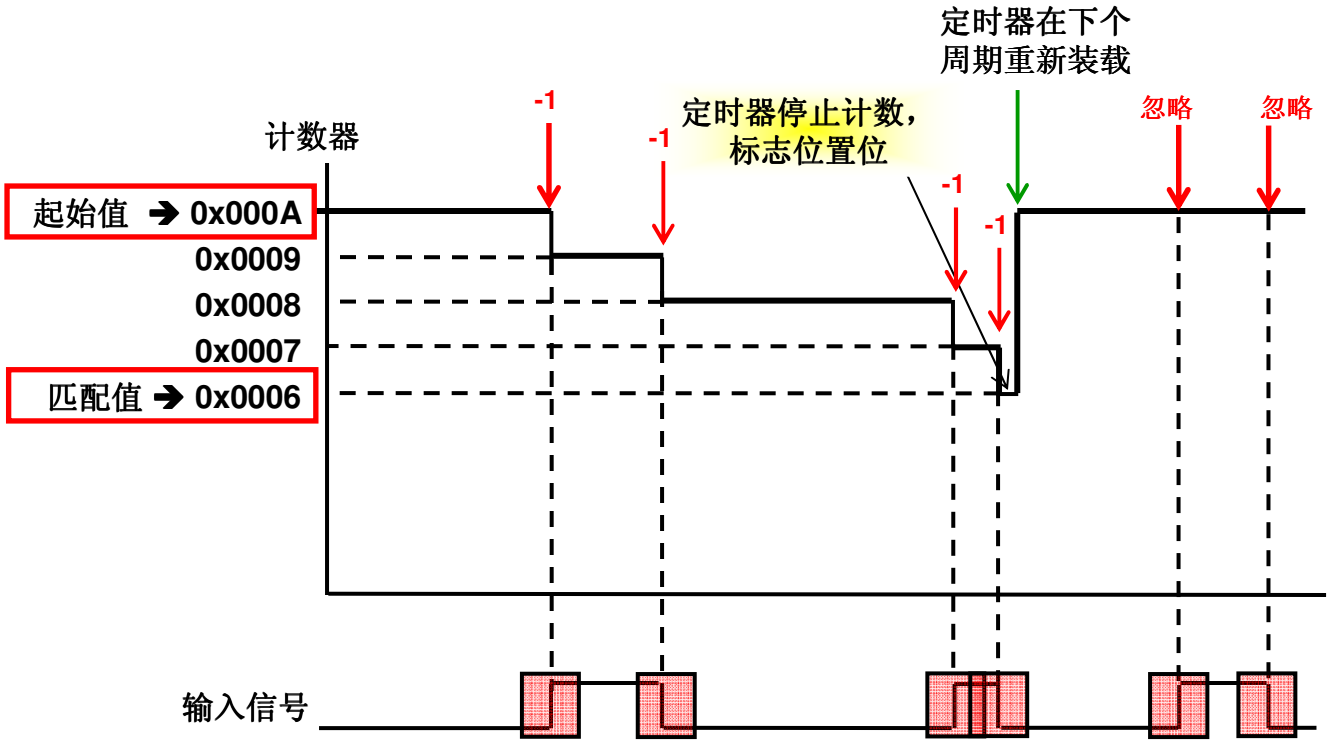
输入边沿计数模式 （Input Edge-Count）

	Timer 用法	
	单独使用	联结使用
16/32 bit GPTM Block	24-bit Timer　Optional Prescaler	N/A
32/64 bit GPTM Block	48-bit Timer　Optional Prescaler	N/A

- 最大输入频率为 ¼ 的系统频率
- 定时器配置为**24**位或**48**位向上或者向下计数模式
- 向下计数：起始值需要初始化
- 向上计数：由**0**开始计数
- 可以捕获下面三种类型的事件
  - 上升沿
  - 下降沿
  - 边沿（包括上升和下降）
- 可以产生中断，触发**ADC**，或**uDMA**传输

输入边沿计数模式是如何工作的？

- 达到匹配值后，计数器重新装载起始值并停止计数



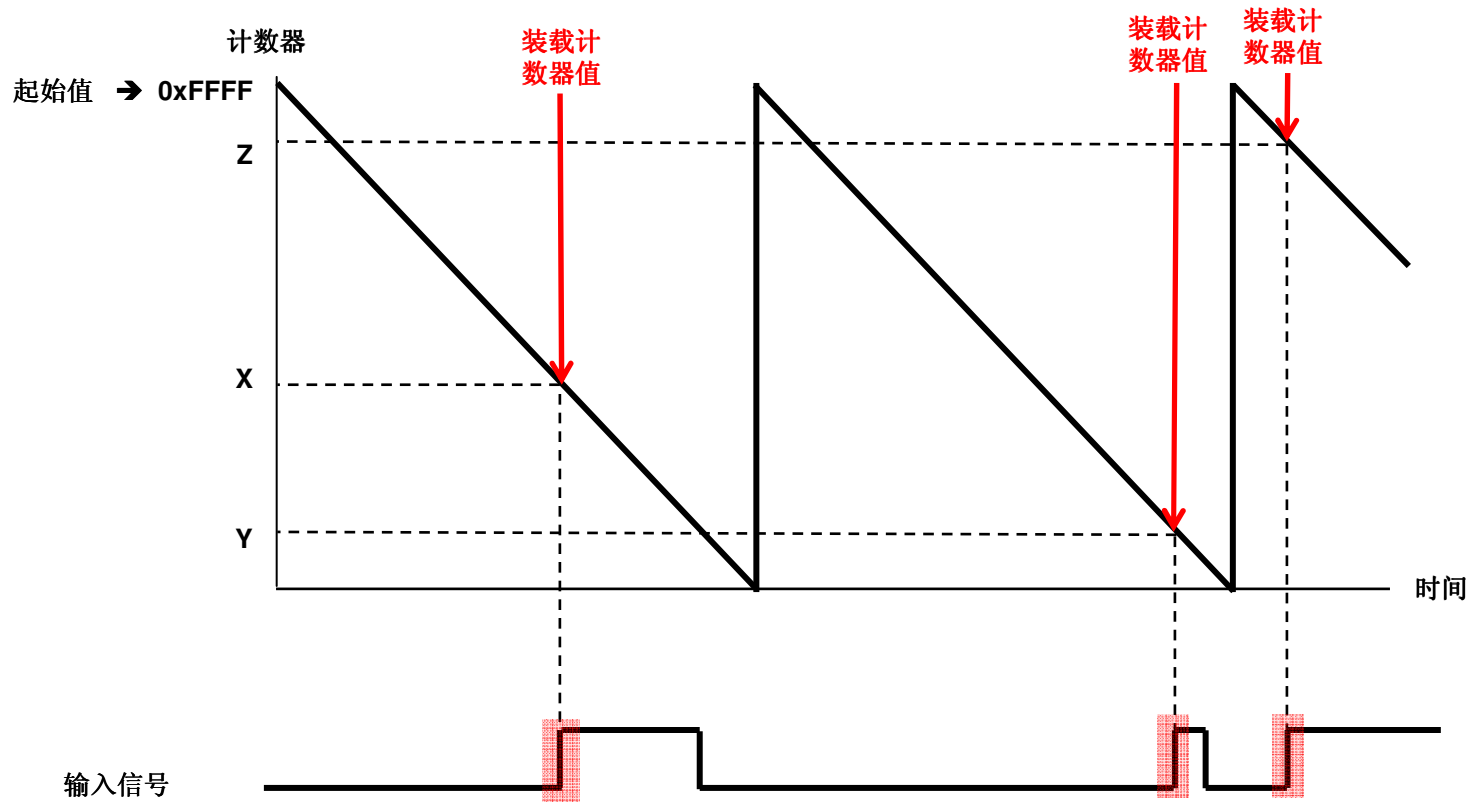
输入边沿时间模式（Input Edge-Time）

	Timer 用法	
	单独使用	联结使用
16/32 bit GPTM Block	24-bit Timer　Optional Prescaler	N/A
32/64 bit GPTM Block	48-bit Timer　Optional Prescaler	N/A

- 最大输入频率为 ¼ 的系统频率
- 定时器配置为24位或48位向上或者向下计数模式
- 向下计数：起始值需要初始化
- 向上计数：由0开始计数
- 可以捕获下面三种类型的事件
  - 上升沿
  - 下降沿
  - 边沿（包括上升和下降）
- 可以产生中断，触发ADC，或uDMA传输
- 捕获事件后，定时器不会停止计数

输入边沿时间模式是如何工作的?

- 若配置成只捕获上升沿
- 每当捕获到上升沿时间，会装载计数器值



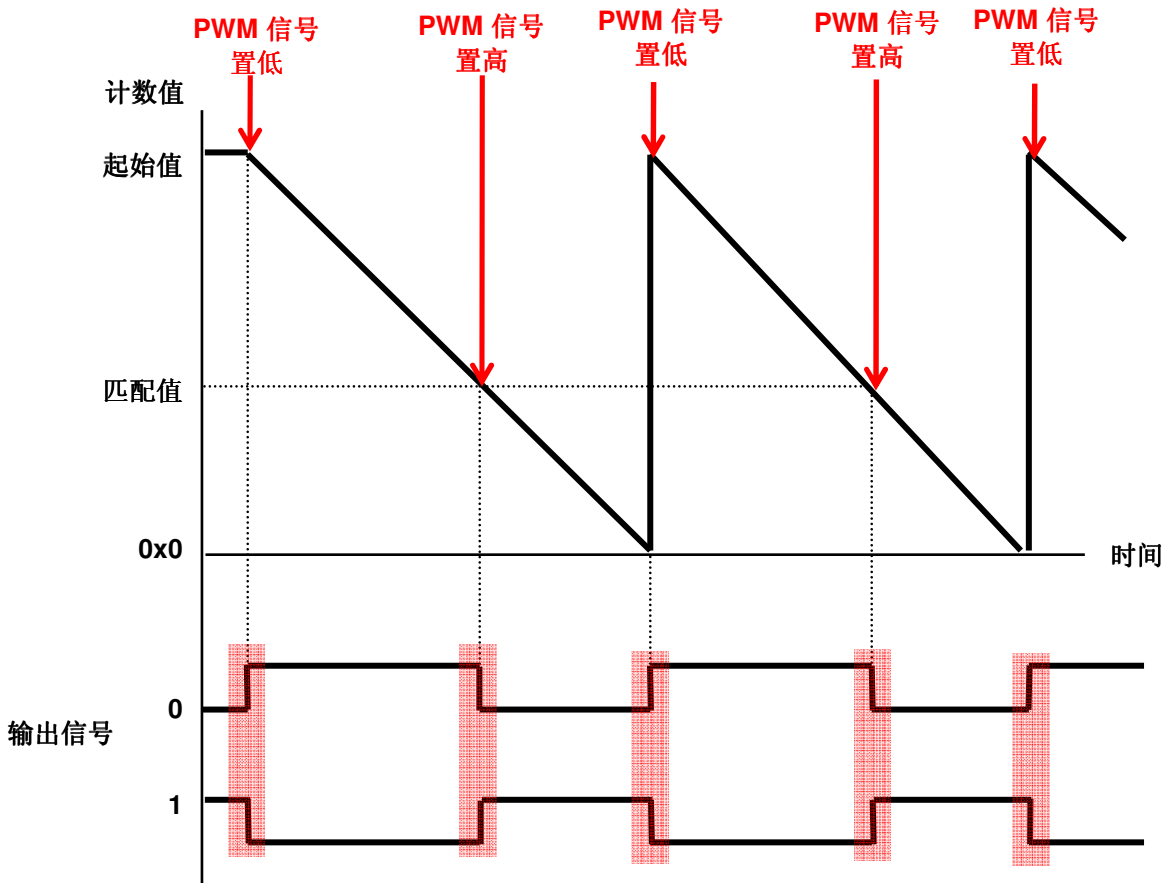
PWM模式

	Timer 用法	
	单独使用	联结使用
16/32 bit GPTM Block	24-bit Timer	N/A
32/64 bit GPTM Block	48-bit Timer	N/A

- 定时器配置成**24**位或**48**位向下计数模式，并设定起始值
- 向下计数直至**0x00**
- 等待触发模式
- 在周期模式中，下一个计数周期重新装载起始值
- 可以在以下时间产生中断
  - 上升沿
  - 下降沿
  - 边沿

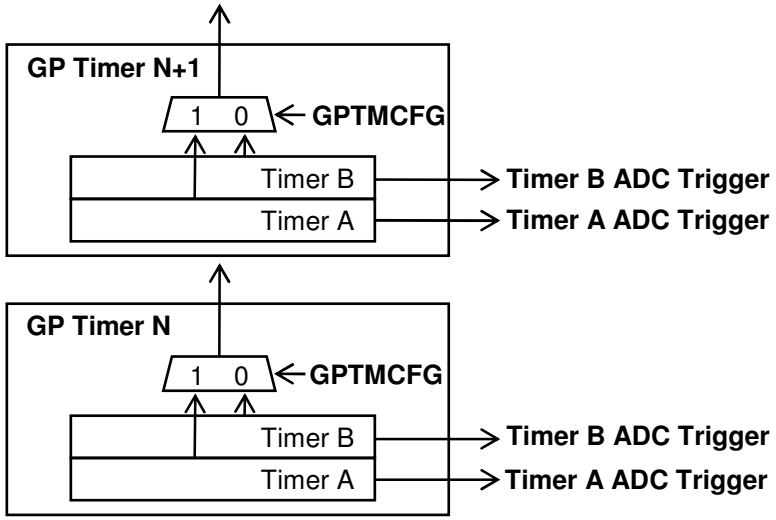
PWM 模式

- 计数器达到起始值时，**PWM**信号置低
- 计数器等于匹配值时，**PWM**信号置高
- 可根据需要将输出**PWM**信号取反



等待触发模式（Wait-for-Trigger）

- 支持级联
- 如果**Timer A** 工作在**32**位模式，那么将触发下一个定时器模块的**Timer A**
- 如果**Timer A** 工作在**16**位模式，那么将触发同一个定时器模块的**Timer B**
- 仅在单次触发，周期触发，和**PWM**模式有效



同步 GP Timer模块

- 同步所选择的定时器同时开始计数
- 当定时器被同步时不会产生中断
- 在连结模式下，仅需对**Timer A**进行配置

Mode	Count Dir	Time Out Action
32-bit One Shot	-	N/A
32-bit periodic	Down	Count value = ILR
	Up	Count value = 0
32-bit RTC	Up	Count value = 0
16-bit One Shot	-	N/A
16-bit Periodic	Down	Count value = ILR
	Up	Count value = 0
16-bit Edge-Count	Down	Count value = ILR
	Up	Count value = 0
16-bit Edge-Time	Down	Count value = ILR
	Up	Count value = 0
16-bit PWM	Down	Count value = ILR

GPIO

- 可编程端口定位
- 任意GPIO都可以作为外部中断源
- 通过AHB总线，引脚翻转速度高达系统时钟速度
- 所有GPIO具有5V兼容性
- 可编程驱动能力
  - 2, 4, 8 mA 或 8 mA 带斜率控制
- 可编程弱上拉，弱下拉，开漏，数字输入

GPIO 地址Mask

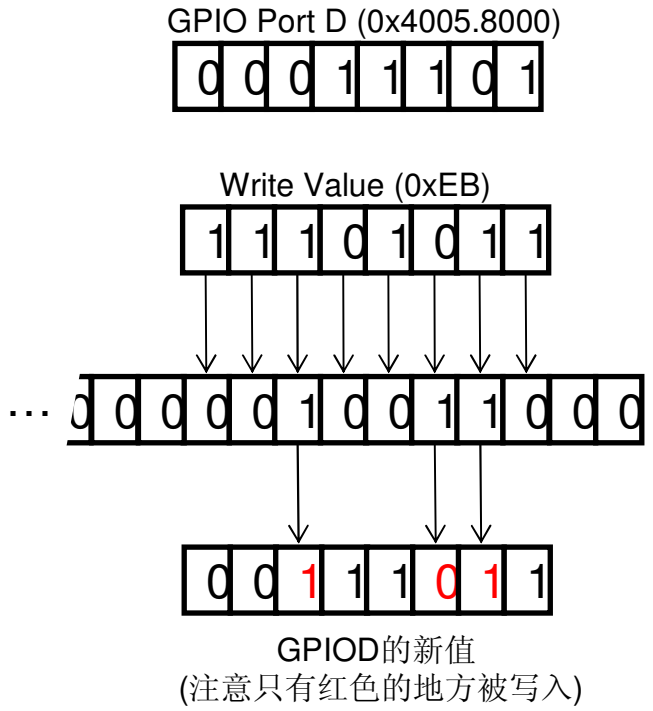
Bit-band区的存储器有一个基地址（存放实际的数据）和一定范围的偏移地址。偏移地址是用来在写入和读出某寄存器时作为bit-mask使用

需要配置GPIOD的寄存器 (0x4005.8000):

写入值： 0xEB

无需直接写GPIOD的寄存器，而是向0x4005.8098写入相应的值. 右图中的Bits 9:2就变成了你所需要写的值的屏蔽值。

只有屏蔽值为1的位置的内容才会被改变



Stellaris LM4F Blizzard 模拟外设

Comparator 和 ADC

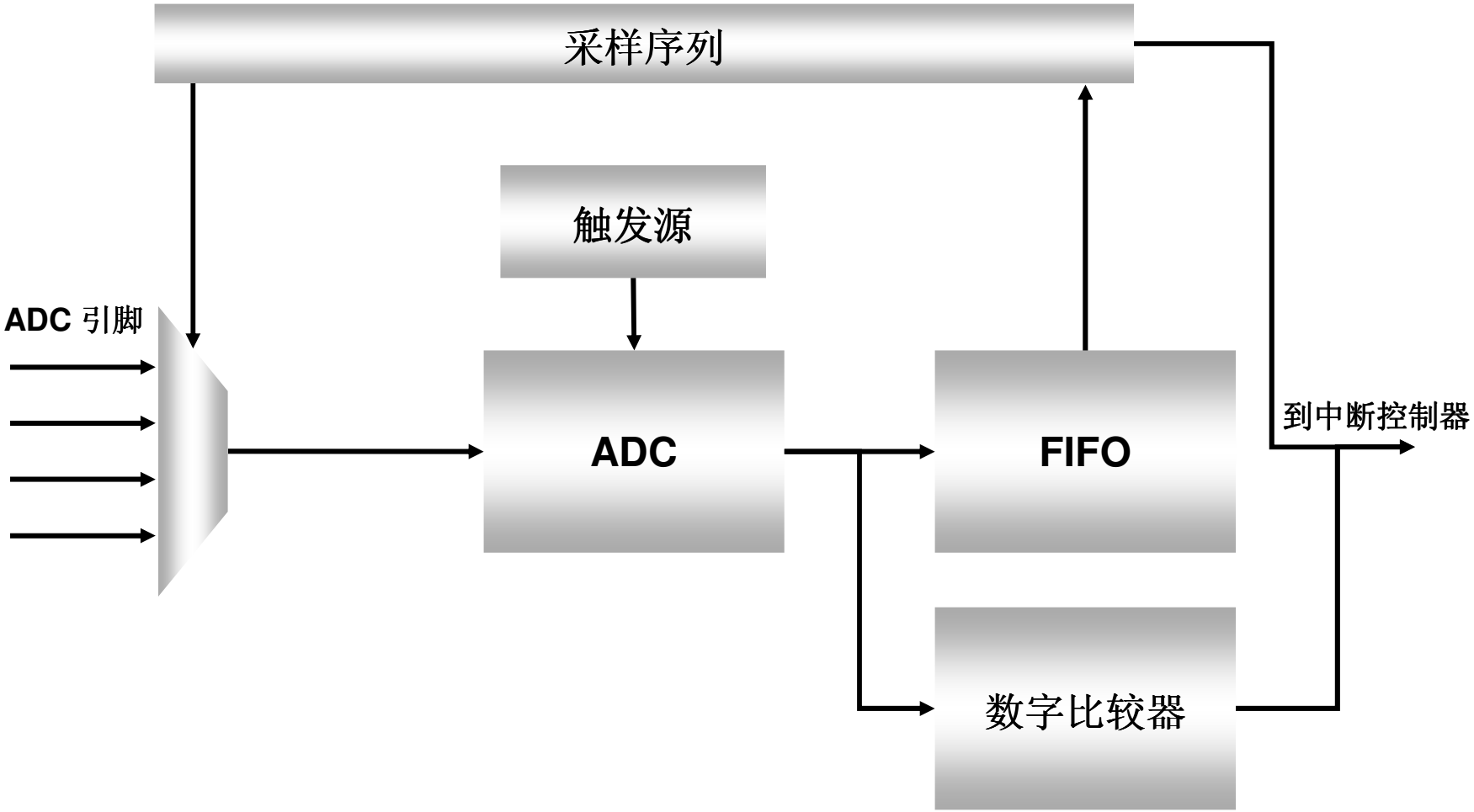
ADC

- 2个独立的 ADC 模块
- 最多24个共享的模拟输入通道
- 12-bit精度
- 支持单端和差动输入方式
- 内置片上温度传感器
- 单通道最大速率1MSPS
- 灵活的触发方式 (定时器,模拟比较器, PWM, GPIO, 软件触发)
- 硬件均值：最高64次采样硬件均值，提高精度
- 利用 µ DMA传输转换结果
  - 对每个采样序列有专门传输通道
  - ADC采用burst传输

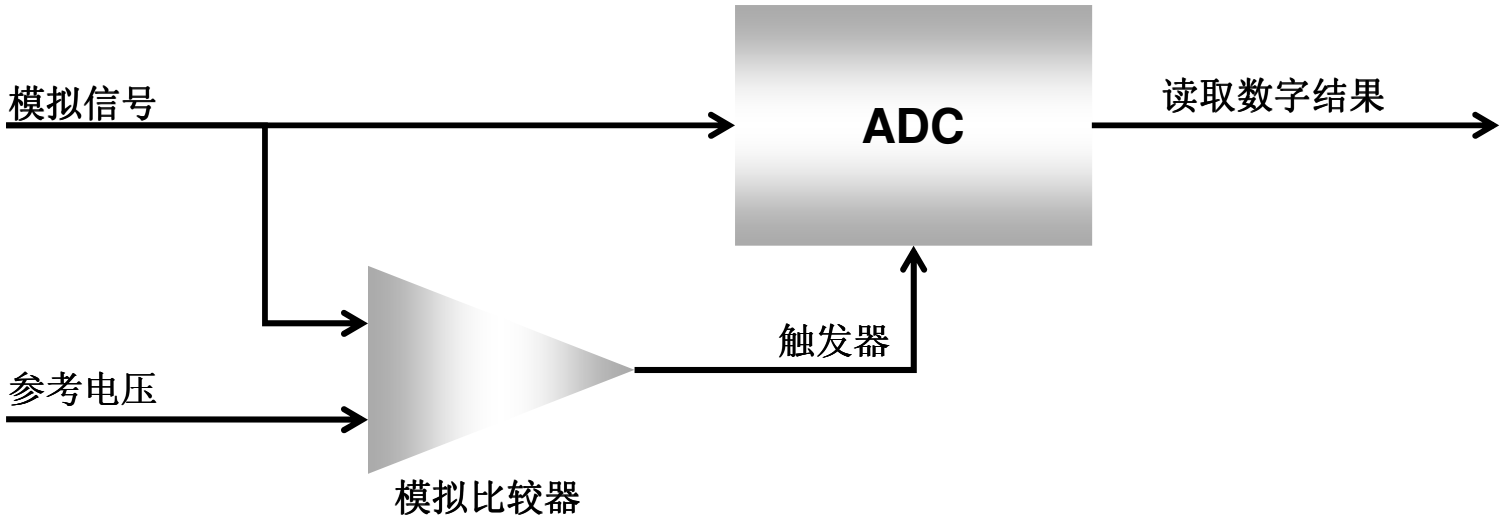
模拟比较器

- 最多3个模拟比较器
- 多种参考源
  - 单独的外部引脚
  - 共享的外部引脚
  - 共享的内部参考电压
- 可产生中断
- 可用来触发ADC
  - 用于阈值检测或不频繁发生的超出阈值范围的事件检测
  - 节省MCU花费在ADC上的时间
  - 更快的发现输入信号是否超出阈值
  - ADC触发器独立于任何中断

ADC 框图



模拟比较器框图



模拟比较器也可用作**ADC**触发源

- 方便用户在传感器超过一定阈值后监控传感器值
- 只有当电压在要求的范围内时，才会开始进行转换

模拟信号测量

- 共享的输入
  - 两个**ADC**模块同时连接到所有**24**个模拟引脚，可以分别独立进行**AD**转换
- 序列
  - **ADC**可以配置成由一个信号触发后，根据设定的采样序列连续进行多个采样
  - 采样序列的大小可以是**1, 4, 8.**
  - 转换结果放在寄存器中，可以通过**DMA**方式传输到目标位置，以节省**CPU**访问**ADC**模块而造成的开销
- 采样序列，共享输入引脚和**DMA**方式的应用可以在有效监控模拟信号的同时又不需要太多占用**MCU**资源和接口

Stellaris LM4F Blizzard 通讯外设

SSI, I²C, UART, USB, CAN

同步串行接口 (SSI)

- 最多4个 SSI模块
- 可编程的接口功能
  - Freescale SPI 和 MICROWIRE接口
  - TI的SSI接口
- 可编程的时钟速率和预分频器，SSI slave时钟频率最高为1/6系统时钟
- 独立的发送和接收FIFO,16位宽度，8级深度
- 可编程的数据帧大小：4~16位
- 内部loopback测试模式用来诊断和调试
- 可通过uDMA进行高效传输
  - 发送和接收有独立的通道
  - 当有数据到达FIFO产生单次接收请求；当FIFO包含4个数据则产生Burst请求
  - 当在FIFO内有剩余空间时，产生单次发送请求；当发送FIFO剩余空间4个时则产生Burst请求

I²C

- 最多6个I²C模块
- 支持主/从模式
- **Simultaneous master and slave operation**
- 主机仲裁，时钟同步，多主通讯，7位寻址模式
- 共有4种I²C模式
  - 主发送，主接收
  - 从发送，从接收
- 通讯速率100 Kbps / 400 Kbps
- 主从中断
  - 作为master时，当完成一次发送或接收时产生中断
  - 作为slave时，当数据发送完成或收到master请求时产生中断

UART （1/2）

- 全系列带8个UART
- 每个UART具有
  - 分离的发送和接收
  - 可编程的FIFO长度
  - FIFO触发等级 1/8, 1/4, 1/2, 3/4, 7/8
  - 可编程的波特率发生器，最高速率达10Mbps
  - 标准异步通讯位：start, stop，parity
  - 起始位出错检测
  - Line-break发生和检测
- 完全可编程的串行接口：
  - 5, 6, 7, or 8 数据位
  - Even, odd, stick, no-parity
  - 1 或 2 停止位
- IrDA编解码功能
  - 可编程作为红外信号或UART输入/输出
  - 支持红外IrDA SIR编解码功能，半双工模式下最高速率115.2 Kbps
  - 支持常规模式3/16和低功耗模式(1.41-2.23 μs) 位周期

UART （2/2）

- 支持ISO 7816 Smard carder 接口
- 支持LIN模式
- 支持EIA-485 9-bit 模式
- 标准FIFO等级和发送结束中断
- 利用uDMA进行高效传输
  - 独立的发送和接收通道
  - 当FIFO接收到数据时产生接收单次传输请求，当FIFO达到预设深度时，产生Burst传输请求
  - 当FIFO有空间时产生发送单次传输请求，当FIFO空间达到预设深度，产生Burst传输请求

USB

集成MAC+PHY

- 支持USB 2.0 Full Speed (12 Mbps) 通讯
- 不同的芯片分别具有 OTG/Host/Device 功能
- 传输: 控制传输，中断传输，批量传输和同步传输
- 16个端点
  - 0和1端点用于控制传输 (one in, one out)
  - 其余14个端点可由软件进行配置
- 4 KB 专门的端点存储器空间
  - 支持DMA方式
  - 端点可以定义成双缓冲方式，具有1023byte的包尺寸

USB-IF 兼容

- TI是USB Implementers Forum成员
- 完全符合 USB-IF 认证标准
- 用户可以使用TI Stellaris VID & PID

CAN

- 最多2个独立CAN控制器
- 符合 CAN 2.0 A/B
- 位速率最高 1Mb/s
- 32个消息对象，每个消息对象有自己的ID mask
- 可屏蔽中断
- 有禁止自动重发模式（ TTCAN）
- 可编程的loop-back模式用于自测

Stellaris LM4F Blizzard 运动控制

PWM, QEI

脉宽调制模块 PWM (1/2)

双PWM模块，每个PWM带4个PWM发生器。每个发生器包含：

- 1个16-bit计数器
  - 向下或上/下计数模式
  - 输出频率由16位的自动装载值配置
  - 装载值可以同步更新
  - 计数器值达到0和装载值时产生输出信号
- 双数字比较器
  - 比较器值可以同步更新
  - 匹配时产生输出信号
- PWM发生器
  - 输出信号由计数器和比较器输出的结果决定
  - 产生2个独立的PWM信号

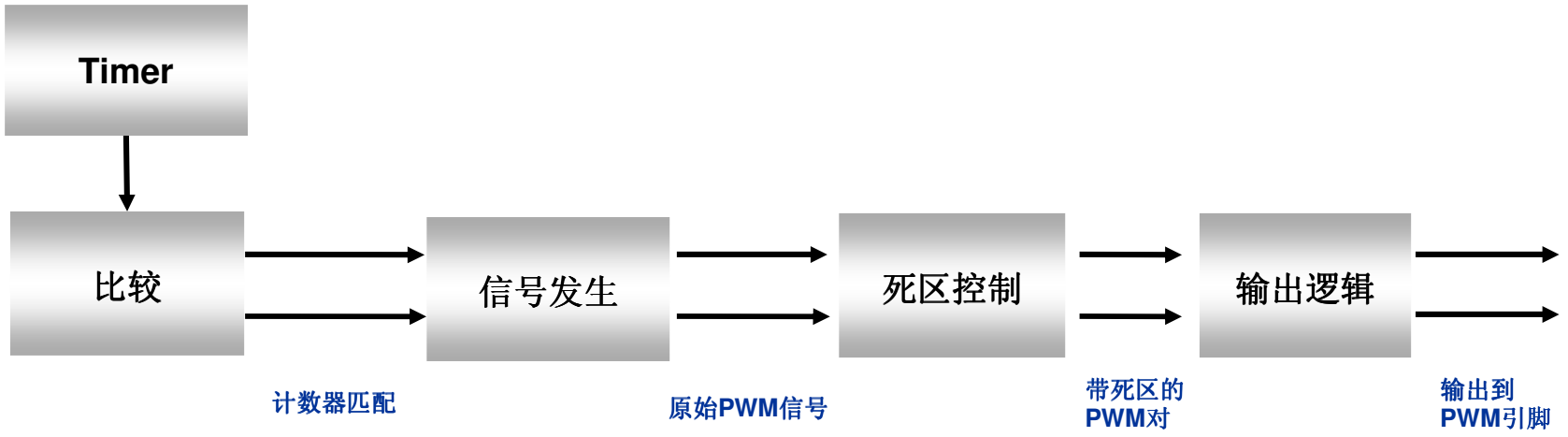
脉宽调制模块 PWM (2/2)

- 死区发生器
  - 在2路PWM信号中插入可编程的死区时间，用于驱动半桥
  - 可以被旁路，只输出原始的PWM信号
- 输出控制单元
  - 每路PWM信号都有相应的使能位
  - 每路PWM信号都有可配置的输出反向控制
  - 每路PWM信号都有可选的出错处理
  - 与定时器同步
  - 与定时器/比较器的更新同步
  - **Interrupt status summary of the PWM generator blocks**

PWM 模块框图

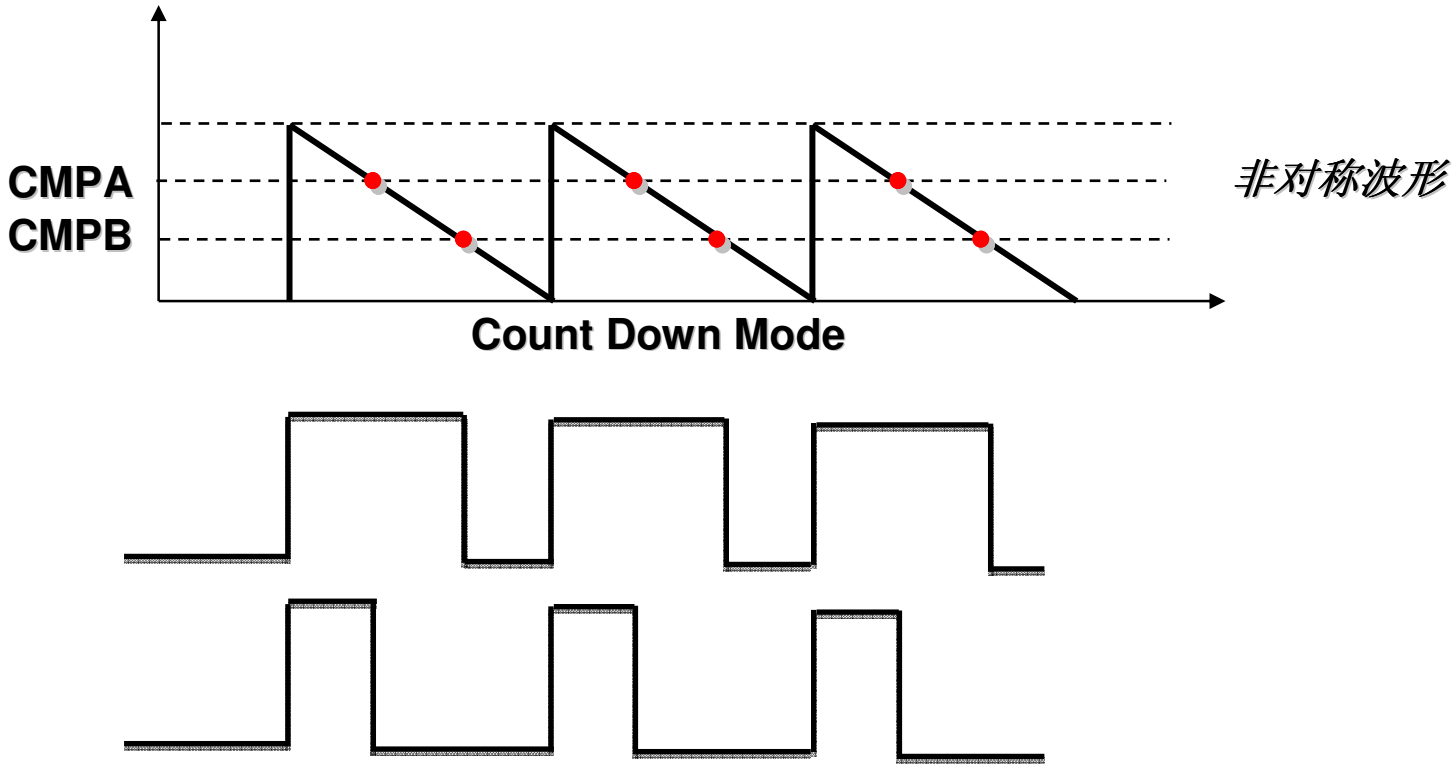
PWM模块包含

- 1个定时器
- 1个可配置的比较模块
- 3阶段的信号发生和控制逻辑单元

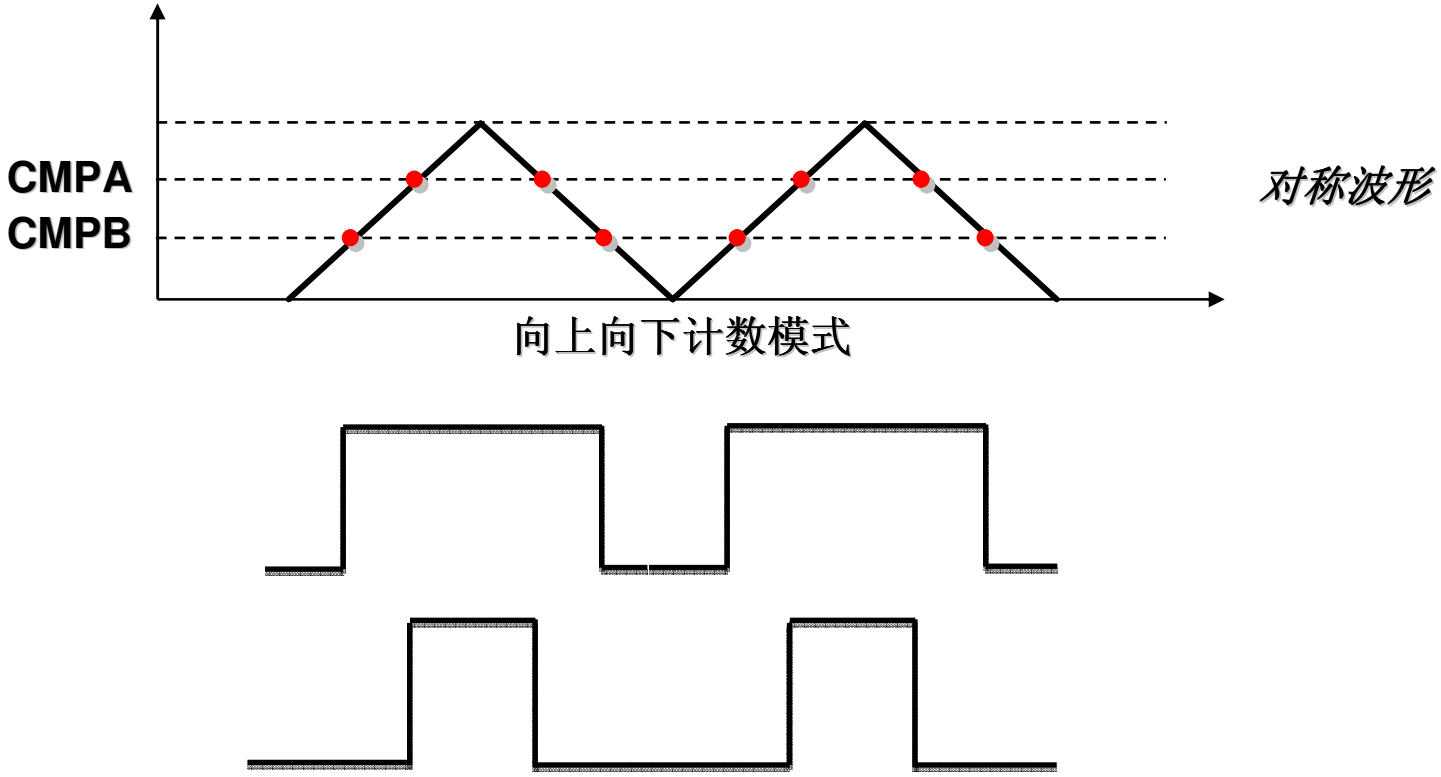


PWM 比较事件波形

- 1个定时器比较匹配事件产生的信号可以用来产生PWM的上升沿或者下降沿
- 每个计数器根据比较事件的配置最多产生2路PWM信号



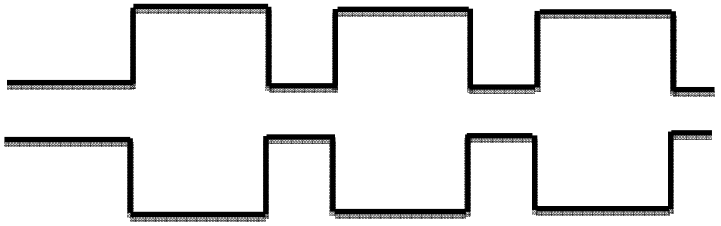
PWM 比较事件波形



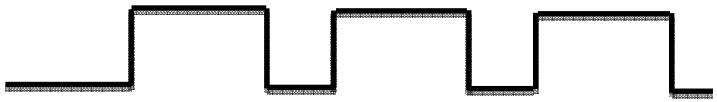
上下计数模式适用于产生对称波形，中心对齐波形

PWM 死区和输出逻辑

计数器产生的PWM波形可以选择不插入死区时间后，产生相应的互补波形。

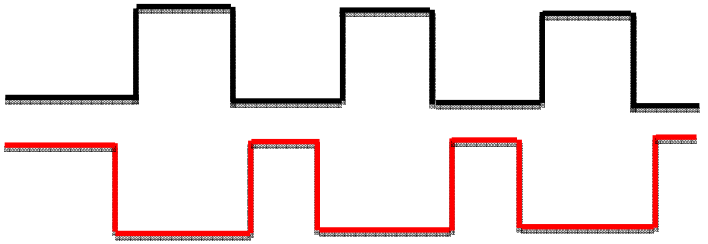


无死区控制



原始波形

原始PWM波形插入死区时间，产生相应的互补波形适合电机控制应用



插入死区时间

Fault-condition中断

- **Fault condition**指控制器必须停止正常**PWM**输出，并将输出信号置于安全的状态

- 两种必要的情况：
  - 控制器停止，不能继续进行电机控制所必须的及时的计算(控制器停止由内部**debugger**引起）
  - 检测到外部错误或故障事件（由 **FAULTn** 引脚引起)

正交编码接口 QEI

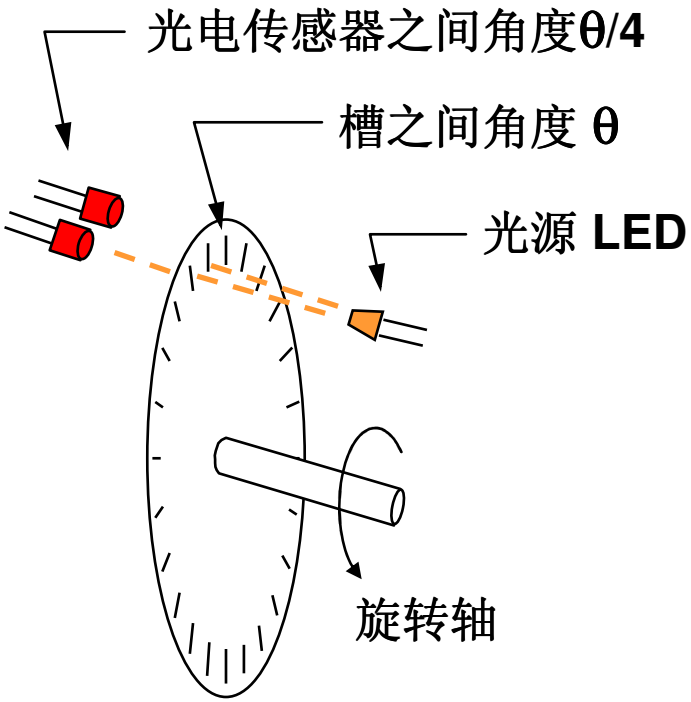
与常规的双通道增量式编码器相比，正交编码器接口将角位移转换成脉冲信号。通常适用于运动控制应用，获得电机的位置，速度和方向信息

Stellaris QEI 特点

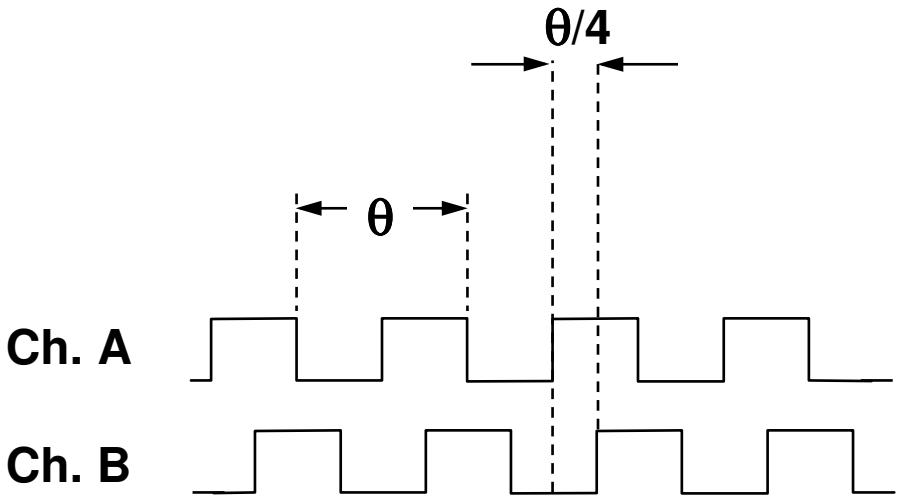
- 包含双QEI接口
- 位置积分器获取编码器位置
- 输入信号可编程的滤波配置
- 利用内部Timer获得速度信息
- 中断发生:
  - 索引脉冲
  - 速度定时器时间结束
  - 方向改变
  - 检测到正交编码出错

正交编码接口 QEI

数字角位移传感器



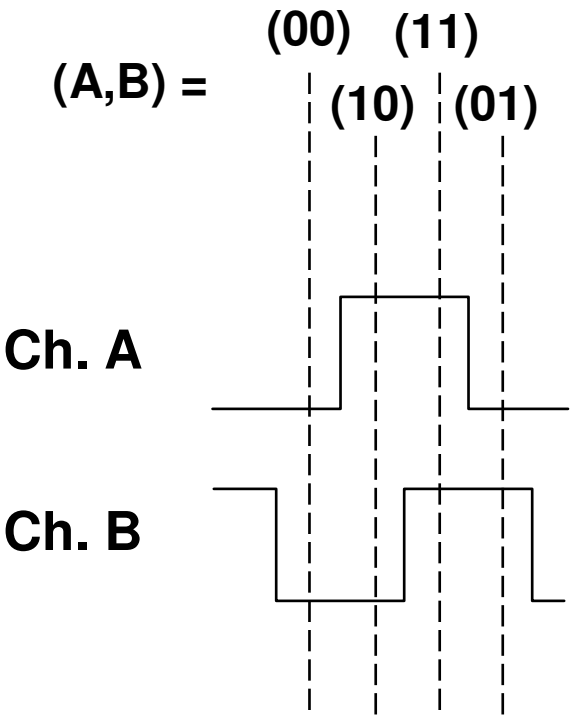
增量式光电编码器

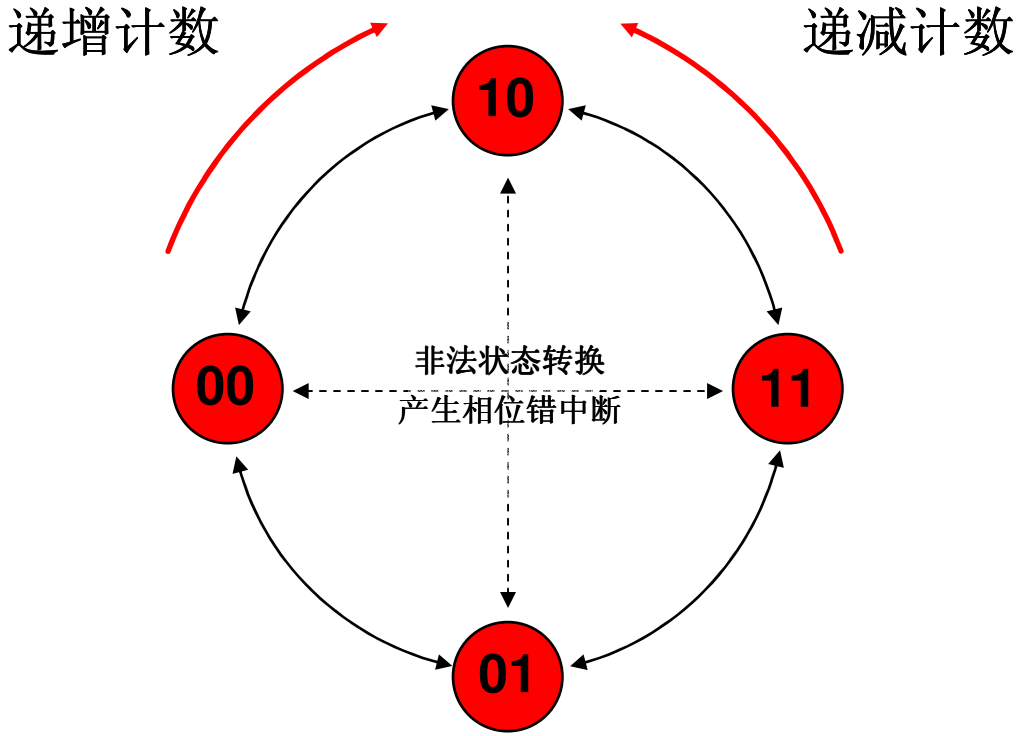


光电传感器的正交输出波形

正交编码接口 QEI

位置分辨率 θ/4



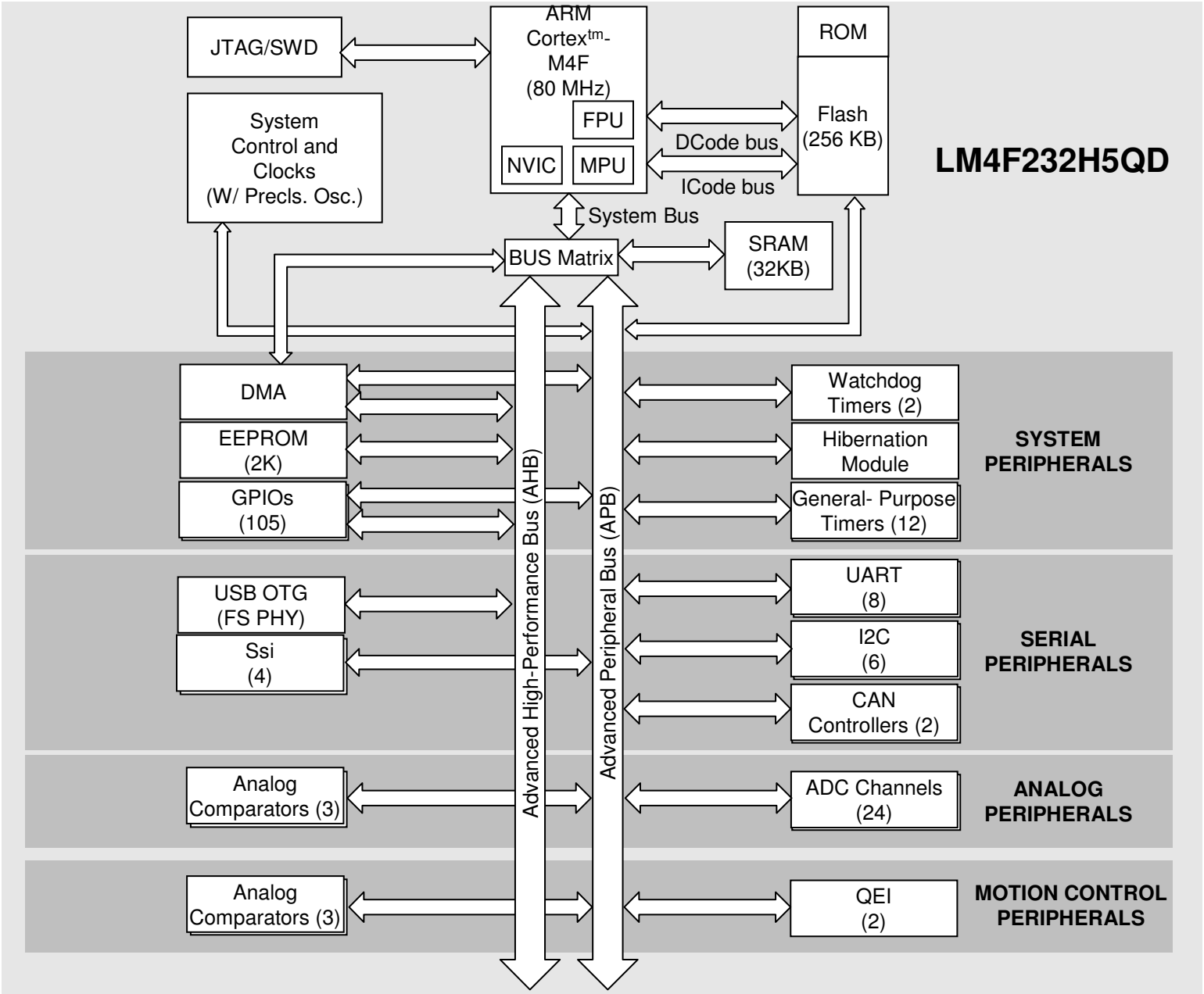


正交编码器状态机

Q&A

Reserve Slides

Stellaris architecture

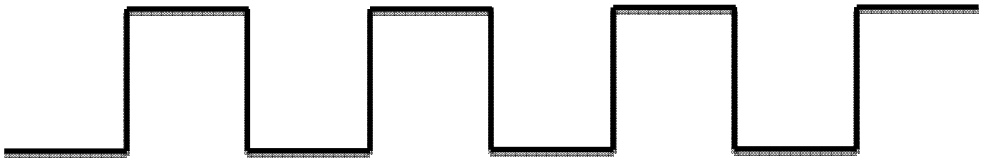


Bit-banding

- Memory aliasing system that provides a more efficient way to change single bits in memory. (Atomic operations vs. read-modify-write)
- Bit-Banding regions exist for SRAM and some peripherals as well.
- Stellaris bit-banding is implemented with bit-masks, meaning that arbitrary groups of bits in a single memory location may be changed without penalty.
- Bit-banded reads are possible as well: only the selected bits will be read, and all others will appear as zeroes.

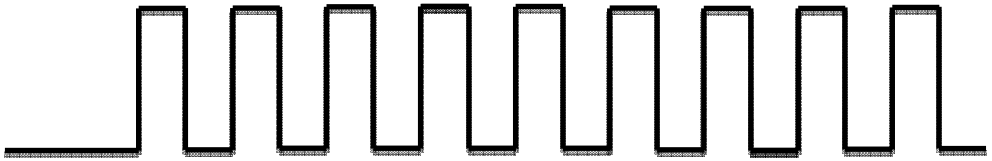
Bit-Banding Example: Writing to GPIOs

Without Bit-banding



f = clk/10

With Bit-banding



f = clk/4

Without bit-banding, a read-modify-write sequence is required to toggle a single pin without affecting others. Bit-banding provides the same end behavior in only one instruction, yielding higher max toggle speeds.