Smart Hardware Design

Conception de matériel intelligent

Diseño de hardware inteligente

スマートハードウェア設計

智能硬件设计

第二章 智能硬件的处理单元

Design de hardware inteligente

تصميم الأجهزة الذكية

Slimme hardwareontwerpen

Σχεδίαση έξυπνου υλικού

大连理工大学-朱明

Progettazione di hardware intelligente

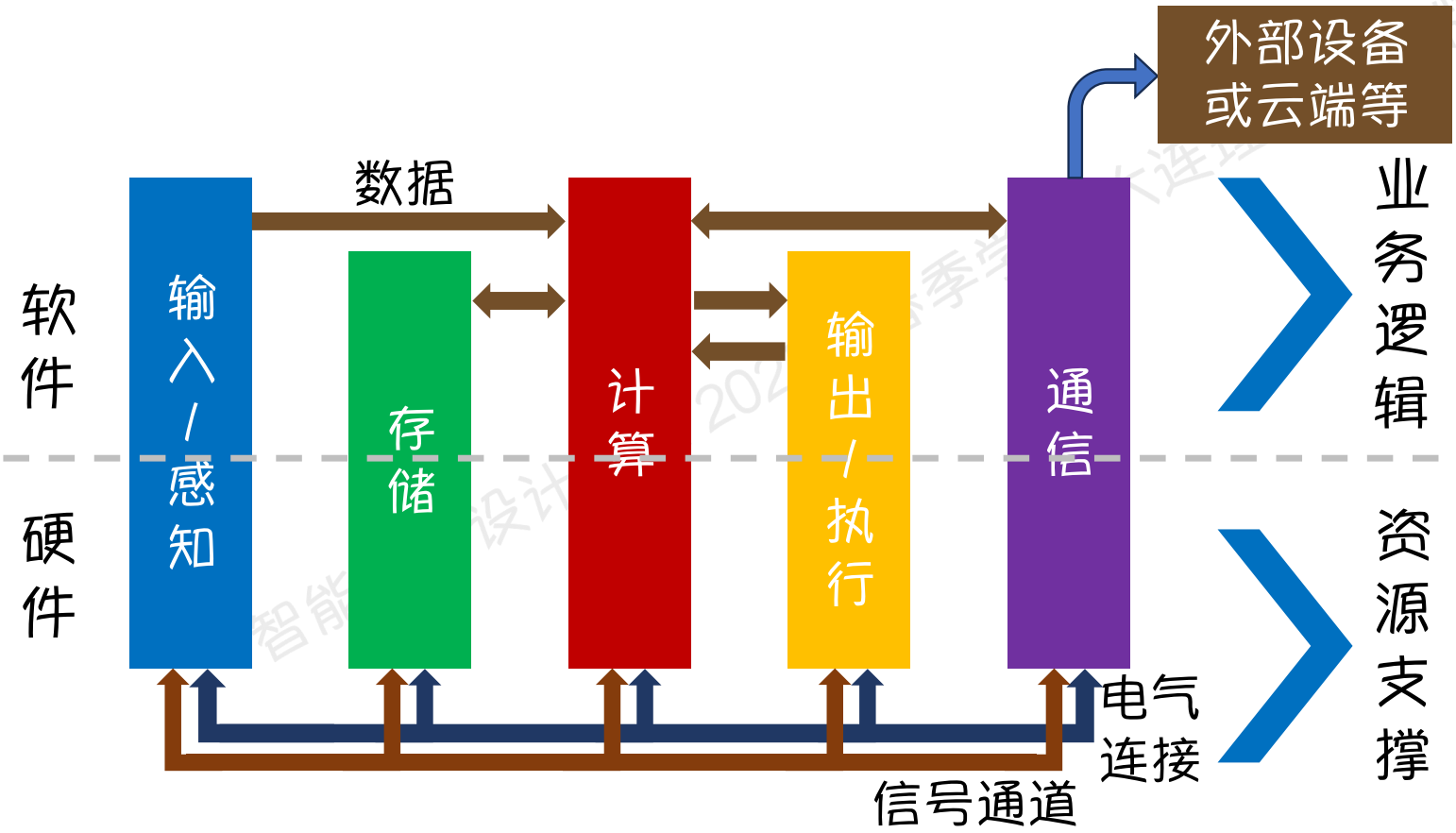
스마트 하드웨어 설계

Smart-Hardware-Design

Проектирование умного оборудования

2.0 思考回顾

- [2.0.0] 智能硬件的软硬件关系图



智能硬件系统软硬件关系

1 硬件层面：电气互联互通，可以有共通的信号通道(信号传输线路)，支撑软件运行
2 软件层面：有严格的数据传输流向和关联关系，构建业务逻辑功能
3 计算部件处于系统的核心位置

2.1 通用计算部件

[2.1.1] 常规信息系统的通用计算部件-处理器

- CPU(中央处理单元，Central Processing Unit)的主要指标
  - 时钟频率：以赫兹(Hz)为单位，表示每秒钟能完成的指令周期数
  - 核心数量：每个核心可以独立执行任务，提升并行处理能力
  - 线程数量：线程是CPU执行指令的最小单位，多线程技术允许每个核心同时执行多个线程，从而提高处理效率
  - 缓存：缓存(L1、L2、L3)存储了CPU频繁访问的数据和指令，较大的缓存可以有效减少CPU访问内存的延迟，提升性能
  - 指令集架构：指令集架构是CPU能够理解和执行的指令集合，常见的指令集架构包括x86、ARM、RISC-V、LoongArch等
  - 热设计功耗(TDP)：CPU在满负载运行时产生的最大热功率

台式机和服务器上的CPU普遍以性能为主，使用市电供电，不关心CPU功率

2.1 通用计算部件

- [2.1.1] 常规信息系统的通用计算部件-处理器
  - CPU(中央处理单元，Central Processing Unit)的主要指标
  - 国外：
  - 国产：

自主可控

台式机和服务器上的CPU普遍以性能为主，使用市电供电，不关心CPU功率

2.1 通用计算部件

- [2.1.1] 常规信息系统的通用计算部件-处理器
  - GPU(图形处理单元，Graphics Processing Unit)的主要指标
    - 核心数量：CUDA核心(Nvidia显卡)数量是衡量GPU并行处理能力的重要指标，较高的核心数量意味着GPU可以同时处理更多的任务
    - 显存容量：显存的容量决定了GPU能够存储多少图形数据和中间计算结果。较大的显存容量对于高分辨率图形和大规模数据计算非常重要
    - 内存带宽：内存带宽表示GPU与显存之间的数据传输速率，带宽越高，GPU在处理图形数据时能够更快速地读取和写入数据
    - 时钟频率：GPU的时钟频率决定了其每秒钟能够完成的计算周期数，较高的时钟频率通常意味着更快的计算速度
    - 热设计功耗(TDP)：GPU在满负载运行时产生的最大热功率

图形工作站和服务器上的GPU以性能为主，使用市电供电，不关心GPU功率

2.1 通用计算部件

- [2.1.1] 常规信息系统的通用计算部件-处理器
  - CPU和GPU的共同特点
    - 性能高
      - 多核：CPU一般8核或更高
        GPU一般上千核或更高
      - 复杂：丰富的指令集和结构
    - 功耗高：
      - 单处理器功耗：几十W或更高
    - 体积大：
      - 处理器面积：数百mm^2或更高
    - 外围电路复杂：
      - 大容量存储、大功率电源等

Intel NUC12SNKI72
整机体积：
230×180×60 mm^3
电源功率：
330W

GeForce RTX 4090
显卡体积：
337×137×66 mm^3
显卡TDP：
450W
显卡不能独立工作

2.1 通用计算部件

- [2.1.1] 常规信息系统的通用计算部件
  - CPU和GPU的共同特点
    - 性能高
      - 多核：CPU一般8核或更高
        GPU一般上千核或更高
      - 复杂：丰富的指令集和结构
    - 功耗高：
      - 单处理器功耗：几十W或更高
    - 体积大：
      - 处理器面积：数百mm^2或更高
    - 外围电路复杂：
      - 大容量存储、大功率电源等

优势

为系统提供强大的运算处理性能

在小型智能硬件领域的劣势

无需处理复杂数据	性能过剩
小电源或电池供电	功耗过高
附着在其他物品上	体积过大
结构简单可靠性高	系统复杂

？

2.1 通用计算部件

[2.1.1] 常规信息系统的通用计算部件-处理器

- CPU和GPU的共同特点
  - 性能高
    - 多核：CPU一般8核或更高
      GPU一般上千核或更高
    - 复杂：丰富的指令集和结构
  - 功耗高：
    - 单处理器功耗：几十W或更高
  - 体积大：
    - 处理器面积：数百mm^2或更高
  - 外围电路复杂：
    - 大容量存储、大功率电源等

优势

为系统提供强大的运算处理性能

在小型智能硬件设备的劣势

无需处理复杂数据	性能过剩
小电源或电池供电	功耗过高
附着在其他物品上	体积过大
结构简单可靠性高	系统复杂

CPU/GPU很好，只是不适合

2.1 通用计算部件

●[2.1.2] 小型智能硬件的计算需求

基本需求(智能锁为例)	常规问题	具有一定专用性的计算部件
处理简单的数据 ①密码数据 ②指纹数据 ③面部图像 ④指令数据 ⑤控制逻辑	性能过剩 功耗过高 体积过大 系统复杂	性能略低、结构简单 ①处理密码、指纹和图像等常规数据 ②执行用户指令和机构控制功能
有限的供电 ①电池供电		较低至超低的运行功耗和待机功耗
附着于门上 ①体积小巧		体积小、集成度高 ①体积小巧、外围电路简单 ②可集成有存储、输出和通信等功能
简单且可靠 ①结构简单 ②定制化低 ③工作稳定		

处理器特点

- 运算能力满足需求，有针对性的应用在不同智能硬件设备
- 能针对处理任务的需求，工作在较低功耗甚至超低功耗下
- 芯片体积小、外围电路简单，有简单的外围电路即可工作
- 集成有常用的存储功能，以及通信和基本的输入输出部件

前面介绍的
处理器-Processor

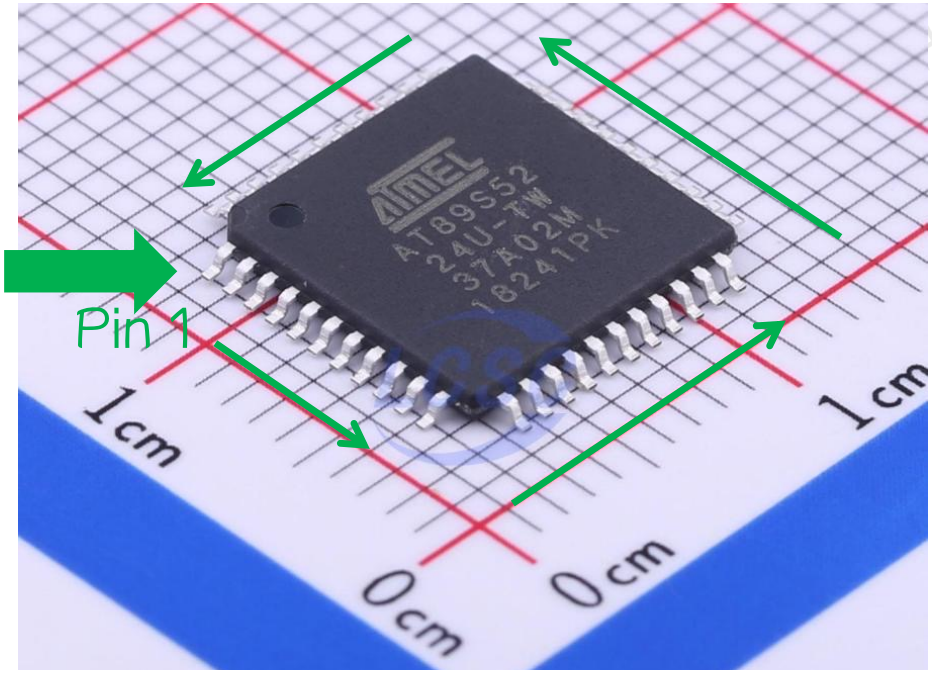
本页的缩减小的
处理器如何取名

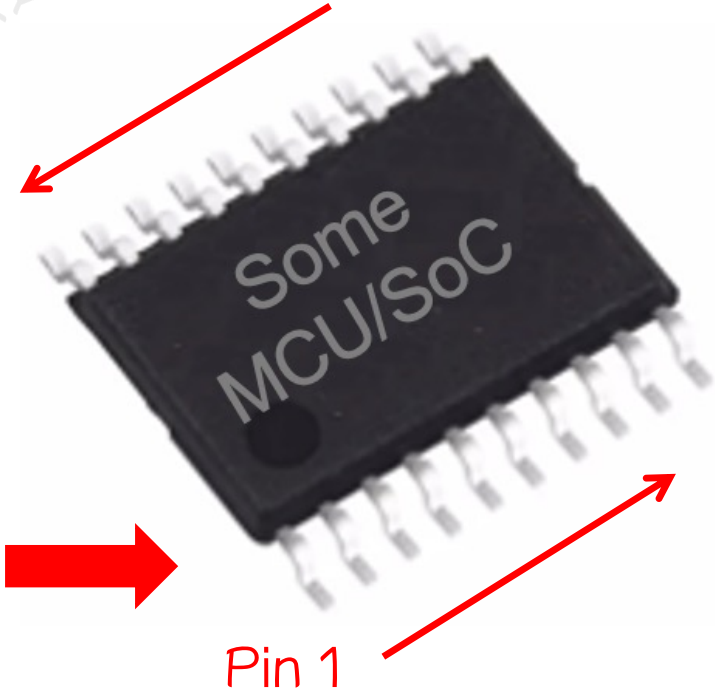
2.2 智能硬件的计算部件

- [2.2.1] 智能硬件常用的计算部件
  - 两类智能硬件常用的计算部件：MCU和SoC
    - MCU(Microcontroller Unit)：一种小型的单片计算机，通常用于嵌入式系统中，集成了处理器核心(CPU)、内存、外设接口以及基本的外设模块
    - MCU用于执行简单的控制和计算任务，适合用于低功耗、低成本和小规模功能的应用
    - SoC(System on Chip)：一种将整个系统的多个功能集成到单个芯片上的技术。SoC不仅包含一个或多个CPU核心，还可以集成图形处理单元(GPU)、通信模块(Wi-Fi、蓝牙、LTE等)、存储单元、音频、视频处理单元，甚至是AI加速器等。
    - SoC适用于需要高性能和多功能的复杂系统，如智能手机、平板电脑、智能电视、智能家居的核心设备等

2.2 智能硬件的计算部件

- [2.2.1] 智能硬件常用的计算部件
  - 两类智能硬件常用的计算部件：MCU和SoC外观上就是芯片
    - 芯片：各种各样的外观和大小，但是都有相同的外观规律





2.2 智能硬件的计算部件

●[2.2.2] 计算部件的性能对比

● MCU、SoC、CPU概况对比(2024年)(数值区分不绝对)

常见分类方法

	MCU <-|-> SOC <-|-> CPU
运行频率	100KHz ~ 200MHz ~ 2GHz
核心数量	单核 双核 多核
缓存大小	无 ~ 256KB ~ 32MB
核心尺寸	$1mm^2$ ~ $100mm^2$ ~ $200mm^2$
能耗水平	~uW ~1W ~10W
总线位宽	<8位><16位><32位> <64位>

2.2 智能硬件的计算部件

●[2.2.2] 计算部件的性能对比

● MCU、SoC、CPU概况对比(2025年)

SoC与MCU的一种区别观点

	内部功能模块(硬件)																	
速度	较低					中							较高			高		
类型	GPIO	ADC	PWM	I2C	UART	SPI	I2S	CAN	SDIO	Wi-Fi	BT	4G 5G	ISP	USB	SATA	DIMM	DMI	PCIE
MCU	■	■	■	■	■	■	■	■	■					■				
SoC	■	■	■	■	■	■	■	■	■	■	■	■	■	■				■
CPU														■	■	■	■	■

■ 一般具备 ■ 部分具备，功能不固定，仅供参考

	操作系统	开发/运行语言
MCU	裸机(Bare-metal)、实时操作系统(RTOS)两大类 如FreeRTOS、RT-Thread、uC/OS-II/III、CMSIS RTOS等	汇编语言、C语言、C++等
SoC	Android、Linux、iOS、OpenHarmony、HarmonyOS等	常用语言普遍支持
CPU	Windows、Linux、MacOS等	常用语言普遍支持

*特别注意：I2C和I2S严格写法为I2C和I2S，课程中为展示清晰，不采用上标

2.2 智能硬件的计算部件

- [2.2.3] 常用的MCU-按照位宽进行分类-[8位]

三种主流处理器核心

- 8位MCU的特点：8位位宽、KB级别存储，较少内部功能模块
- 常见的8位MCU：主频不超过20MHz，多周期12T
  - 8051架构：最经典的8位MCU，128B RAM、4KB ROM
    - Intel 8051：最早的8051 MCU
    - AT89系列(ATMEL公司)：8051为基础，提升了集成度和功能
  - PIC系列：Microchip公司(已被收购)
    - PIC16系列(如PIC16F877A)，目前仍在使用
  - AVR系列： ATMEL公司
    - ATmega系列(如ATmega328P)，仅存的广泛使用的八位MCU

旧 → 新

传统的8051等8位MCU运行速度低、指令效率低，内部功能模块少，且功耗偏高
基本内部功能模块：定时器、串行通信、中断和ISP等，支持标压5V或低压3.3V运行

2.2 智能硬件的计算部件

- [2.2.3] 常用的MCU-[8位]
  - 国产8051 MCU：STC8051系列：性能大幅度提升
    - STC89/90系列(举例说明)，多周期6T/12T
      - STC90C516AD：40MHz，61KB/6352B，增加ADC(模数转换)
    - STC10/11/12系列(举例说明)，单周期1T
      - STC11L60XE： 35MHz，60KB/1280B， ISP等传统功能
      - STC12LE5630AD：35MHz，30KB/756B，ADC，PWM(脉宽调制)
    - STC15系列和STC8系列：略
    - STC32系列(举例说明)，单周期1T，32位8051内核
      - STC32G12K128：36MHz，128KB/12KB，DMA、CAN、LIN、USB、SPI、I2C、PWM、在线仿真等高级功能

8位MCU一般应用在非智能型的设备控制领域，小家电领域(电磁炉、微波炉等)

2.2 智能硬件的计算部件

[2.2.4] 常用的MCU-[16位]

两种主流处理器核心

- 16位MCU的特点：除性能提升之外，更丰富的外设功能
- 常见的16位MCU：主频一般不超过100MHz
  - MSP430系列：最早的16位MCU之一
    - MSP430F149：8MHz、60KB/2KB、UART、SPI、PWM、ADC等
    - MSP430F5631：20MHz、192KB/16KB、 UART、SPI、PWM、ADC、USB、I2C等
  - PIC24F系列：
    - PIC24FJ512GL410：32MHz、512KB/32KB、 UART、SPI、LCD、ADC、DAC、PWM、I2S、I2C

16位MCU运行速度提升，指令效率大幅提升，广泛使用3.3V工作电压，集成更多功能模块
16位MCU主要应用在一些电池供电设备、有控制算法需求的家电(洗衣机、空调等)

2.2 智能硬件的计算部件

- [2.2.5] 常用的MCU-[32位]
  - 32位MCU的特点：性能强大、外设全面，SoC趋势
  - 主流内核一：ARM Cortex-M内核
    - STM32系列MCU：市场上主流的32位MCU产品，部分内置FPU

型号	主频 MHz	FLASH KB	RAM KB	32位 定时器	16位 定时器	ADC	UART	SPI	I2C	DMA	DAC	CAN	I2S	以太网	USB	看门狗	RTC
STM32F070RB	48	128	16		8	1		2	2						■	■	■
STM32F107VC	72	256	64		7	2	2	3	1	■	2	2	2	■	■	■	■
STM32F398VE	72	512	80	1	10	4	2	2	3	■		1	2			■	■
STM32G4A1VE	170	512	112	1	11	3	2	3	3	■	4	2	2		■	■	■
STM32F779NI	216	2048	512	2	12	3	4	6	4	■	2	3	3	■	■	■	■
STM32H757ZI	480	2048	1024	2	12	3	4	5	4	■	2	2	3	■	■	■	■

    - PIC32系列MCU、NXP LPC系列MCU：略

2.2 智能硬件的计算部件

- [2.2.5] 常用的MCU-[32位]
  - 32位MCU的特点：性能强大、外设全面，SoC趋势
  - 主流内核二：RISC-V内核
    - GD32VF103系列MCU：国产MCU

型号	主频 MHz	FLASH KB	RAM KB	32位 定时器	16位 定时器	ADC	UART	SPI	I2C	DMA	DAC	CAN	I2S	以太网	USB	看门狗	RTC
GD32VF103RB	108	128	32		5	2	5	3	2	■	2	2	2		■	2	■
GD32VF103R8	108	64	20		5	2	5	3	2	■	2	2	2		■	2	■
GD32VF103R6	108	32	10		3	2	2	1	1	■	2	2			■	2	■
GD32VF103R4	108	16	6		3	2	2	1	1	■	2	2			■	2	■
GD32VF103VB	108	128	32		5	2	5	3	2	■	2	2	2		■	2	■
GD32VF103V8	108	64	20		5	2	5	3	2	■	2	2	2		■	2	■

兆易创新(Gigadevice)与芯来科技(Nuclei System Technology)
面向物联网及其它超低功耗场景应用自主联合开发的一款商用RISC-V处理器

2.2 智能硬件的计算部件

●[2.2.5] 常用的MCU-[32位]

- 32位MCU的特点：性能强大、外设全面，SoC趋势
- 主流内核二：RISC-V内核
  - Hi3065系列MCU：国产MCU，内置FPU

型号	主频 MHz	FLASH KB	RAM KB	32位 定时器	16位 定时器	ADC	UART	SPI	I2C	DMA	DAC	CAN	I2S	以太网	USB	看门狗	RTC
Hi3061M	150	128	32	4		1	4	2	2	■	2	1				■	
Hi3061H	200	152	16	3		2	3		1	■	1					■	
Hi3065H	200	152	16	3		3	3	1	1	■	3	1				■	

基于海思公司(Hisilicon)自研RISC-V内核的高性能实时控制专用MCU

- 其他RISC-V内核的MCU：CH32Vxxx

CH32V003F4U6	48	16	2		2	1	1	1	1							■	
CH32V37VCT6	144	256	64		8	2	8	3	2	■	2	2	2	■	■	■	

2.2 智能硬件的计算部件

- [2.2.5] 常用的MCU-[32位]
  - 32位MCU的特点：性能强大、外设全面，SoC趋势
  - 内核三：Tensilica Xtensa内核
    - ESP32-P4系列MCU：国产MCU，内置FPU
      - 双核MCU：HP核400MHz，LP核40MHz
      - 支持外扩RAM：16/32MB PSRAM

型号	主频 MHz	FLASH KB	RAM KB	32位 定时器	16位 定时器	ADC	UART	SPI	I2C	DMA	DAC	CAN	I2S	以太网	USB	看门狗	RTC
ESP32-P4NRW16	400 40	128	768 32		■	2	6	5	3	■			4	■	■	■	
ESP32-P4NRW32	400 40	152	768 32		■	2	6	5	3	■			4	■	■	■	

乐鑫公司(Espressif)的产品更多带有Wi-Fi、BT等无线通信功能，课程将其归类为SoC

2.2 智能硬件的计算部件

[2.2.6] 常用的SoC

- 回顾：SoC与MCU的主要区别是什么？
- SoC的基本分类方法：按照性能进行分类

分类级别	核心数量	最高主频	内存(可支持)	GPU	基本外设(I2C、UART等)	代表处理器
超高性能	10+	2GHz+	16GB+	■	■	Apple M1/2/3/4，Tesla HW3/4，高通骁龙8cx Gen3等
高性能	8+	1GHz+	4GB+	■	■	Apple A系列，高通骁龙8系列，华为Kirin系列等
中等性能	4+	1GHz+	1GB+	■	■	高通骁龙600系列，MediaTek Helio P系列等
弱性能	1或1+	100MHz+	1MB+		■	海思Hi386x系列、ESP32系列、紫光展锐V5663等

- 超高性能：个人计算机、数据中心、自动驾驶、复杂机器人控制等
- 高性能：智能手机、平板电脑、边缘计算设备、游戏主机等
- 中等性能：智能电视、平板电脑、边缘计算设备、机顶盒等
- 弱性能：智能家居设备、无线传感器、低功耗物联网等

课程关注点

2.2 智能硬件的计算部件

● [2.2.6] 常用的SoC-[32或64位]

● 弱性能SoC举例：ESP32-S/C/H系列

型号	主频 MHz	FLASH KB	RAM KB	基本外设	无线通信
ESP32-S3	双 240	384	512	SPI、UART、I2C、I2S、PWM、ADC、SD/MMC等	2.4GHz Wi-Fi (IEEE 802.11b/g/n)、Bluetooth 5 (LE)
ESP32-S2	240	128	320	SPI、UART、I2C、I2S、PWM、LCD 接口、Camera 接口、ADC、DAC、触摸传感器等	2.4GHz Wi-Fi (IEEE 802.11b/g/n)
ESP32-C6	160	320	512	SPI、UART、I2C、I2S、PWM、TWAI等	2.4GHz Wi-Fi 6 (802.11ax)、Bluetooth 5 (LE)、Zigbee、Thread
ESP32-C5		320	384	202501时还未见公开资料	2.4/5GHz Wi-Fi 6 (802.11ax)、Bluetooth 5 (LE)、Zigbee、Thread
ESP32-C3	160	384	400	SPI、UART、I2C、I2S、PWM、USB、ADC等	2.4GHz Wi-Fi (IEEE 802.11b/g/n)、Bluetooth 5 (LE)
ESP32-C2	120	576	272	SPI、UART、I2C、PWM、ADC等	
ESP32-H2	96	120	320	SPI、UART、I2C、I2S、PWM、IrDA等	Bluetooth 5 (LE)、IEEE 802.15.4
ESP32-D0WD-V3	双 240		520	SPI、UART、I2C、I2S、SD/MMC、ADC等	2.4GHz Wi-Fi、Bluetooth、Bluetooth LE

2.2 智能硬件的计算部件

- [2.2.6] 常用的SoC-[32或64位]
  - 弱性能SoC举例：紫光展锐SoC(结构接近中端，但性能不足)

型号	主频 MHz	FLASH KB	RAM KB	基本外设	无线通信
UNISOC 5981	160	4096 MAX	8192 MAX	SPI、UART、I2C、I2S、PWM、ADC、SD/MMC、USB等	2.4GHz Wi-Fi (IEEE 802.11b/g/n)
UNISOC V5663	442 416	32768 MAX	8192 MAX	SPI、UART、I2C、I2S、PWM、ADC、SD/MMC、USB3.0等	2.4/5GHz Wi-Fi (IEEE 802.11b/g/n/ac)、Bluetooth 5

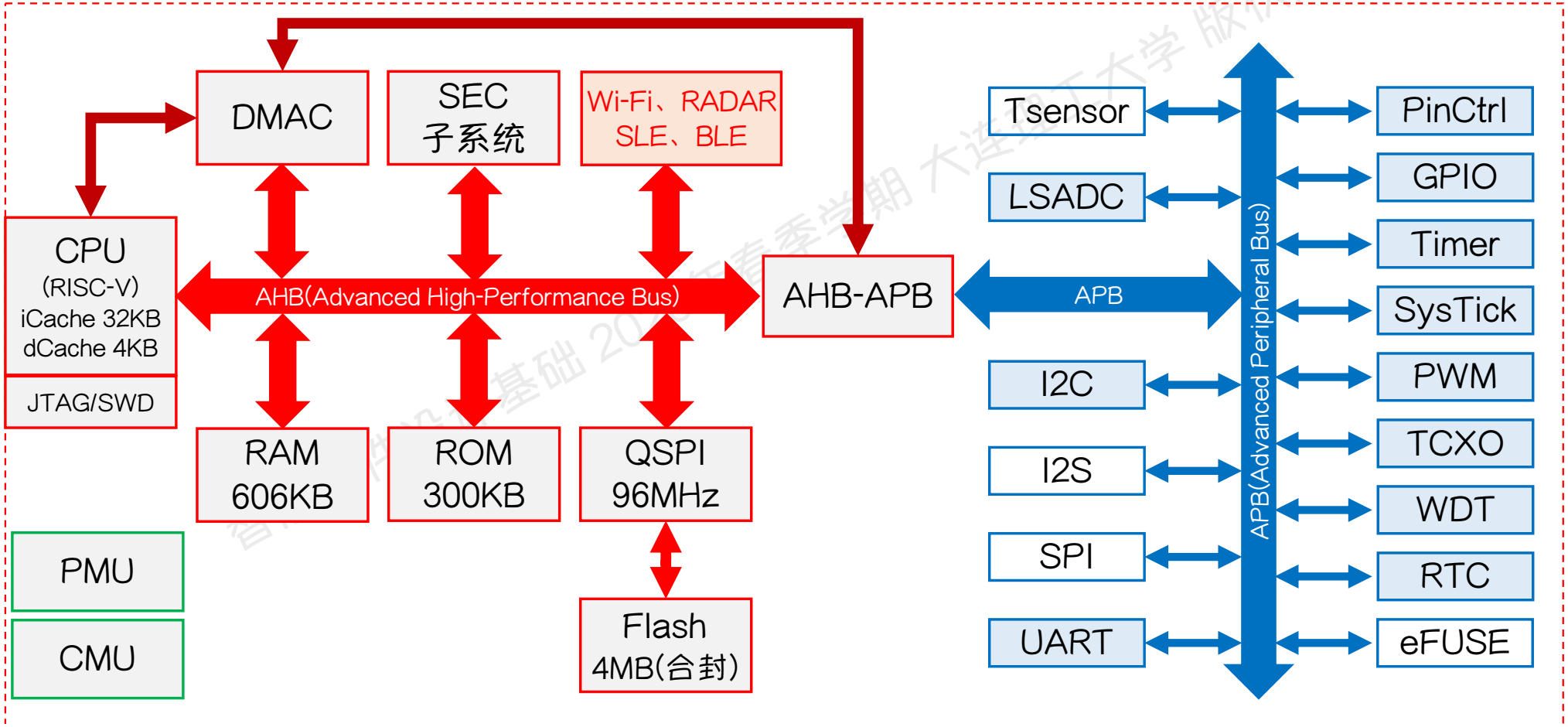
  - SoC举例：华为海思Hi3861和WS63 SoC

课程平台

型号	主频 MHz	FLASH KB	RAM KB	基本外设	无线通信
Hi3861	160	2048 288	352	SPI、UART、I2C、I2S、PWM、ADC、SD/MMC等	2.4GHz Wi-Fi (IEEE 802.11b/g/n)
WS63 解决方案	240	4096 300	606	SPI、UART、I2C、I2S、PWM、ADC等	2.4GHz Wi-Fi (IEEE 802.11b/g/n/ax)、Bluetooth 5.4(LE)、星闪Sparklink Low Energy(SLE) 1.0

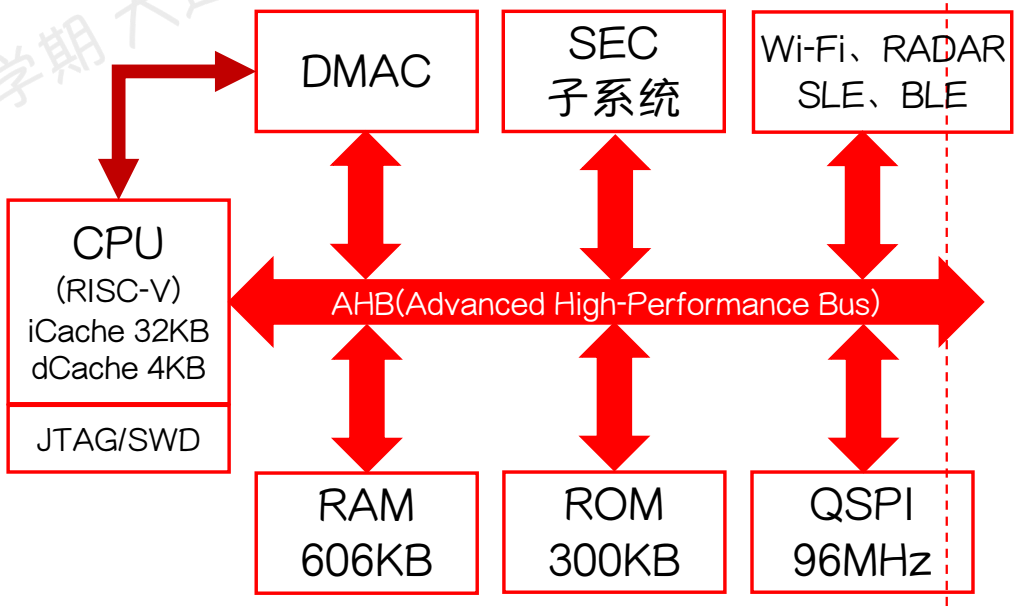
2.3 SoC内部结构

- [2.3.1] 星闪SoC WS63的结构



2.3 SoC内部结构

- [2.3.1] 星闪SoC WS63的结构
  - SoC内部的总线结构
    - 总线(Bus)是处理器内部功能模块传输信息的公共通道
    - 传统MCU/SoC(8051/PIC等)总线
      - 数据总线(Data Bus)传输数据，数据总线的宽度(位数)决定了处理器一次可以传输的最大数据量
      - 地址总线(Address Bus)传输地址，表达数据源地址或目的地址，地址总线的宽度决定了系统的寻址能力
      - 控制总线(Control Bus)传递控制信号，包括读/写信号、时钟信号、中断信号等



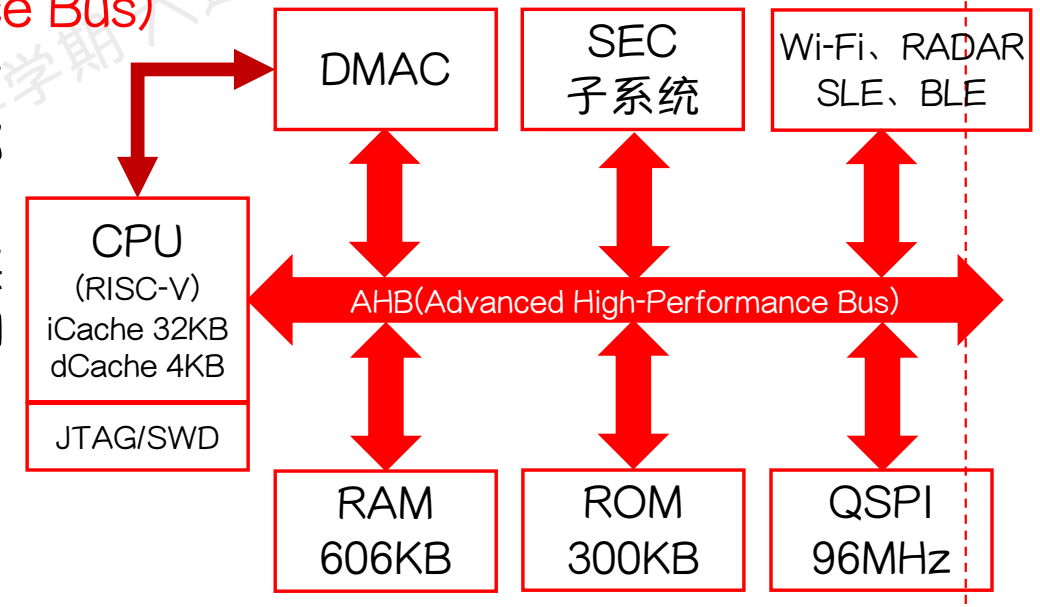
计算机组成原理课程中会详细介绍上述内容

Q：尝试在WS63内部找到左侧总线？

2.3 SoC内部结构

- [2.3.1] 星闪SoC WS63的结构
  - AMBA总线形式-MCU/SoC主流
    - AMBA(Advanced Microcontroller Bus Architecture)高级微处理器总线架构
    - AHB(Advanced High-performance Bus)
      - AHB可以将微控制器(CPU)、高带宽的片上RAM、高带宽的外部存储器接口、DMA总线控制器，以及各种AHB接口的控制器等连接起来，构成一套独立的完整的SoC系统
      - 单通道总线，不能并行读写
      - AHB：高速、高性能

AMBA是一种架构
定义多种总线形式



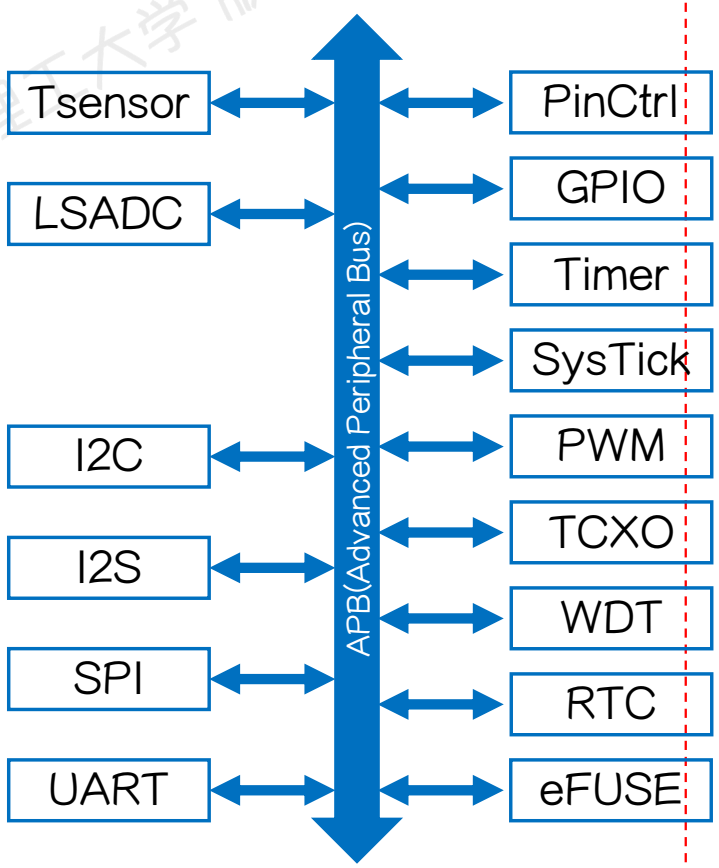
计算机组成原理课程中会详细介绍CPU工作原理

A：WS63是基于AMBA架构的总线

2.3 SoC内部结构

- [2.3.1] 星闪SoC WS63的结构
  - 是否SoC的所有部件都需要高速总线
    - 低速外设连接高速总线会拉低系统性能
    - 有低速外设的SoC一般都要有低速总线
    - APB(Advanced Peripheral Bus)
      - 低功耗精简接口总线，可以连接多种不同低速外设；主要应用在低带宽的外设上，如GPIO、UART、I2C、WDT等
      - 单通道总线，不能并行读写
      - APB：低速、性能相对较弱

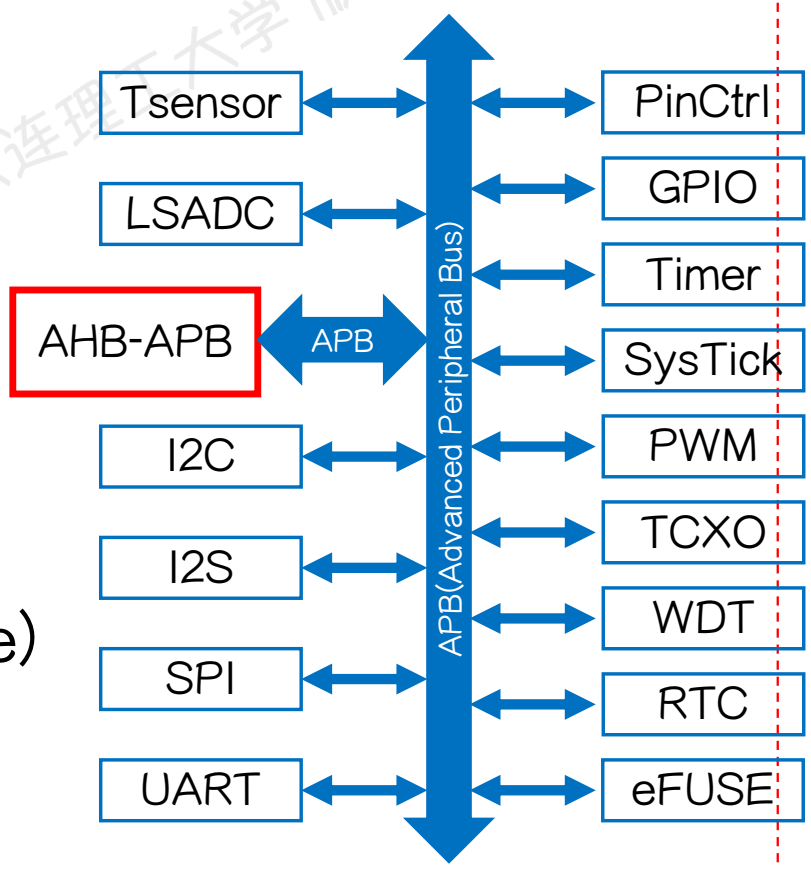
Q：GPIO等外设如何被CPU访问和控制



2.3 SoC内部结构

- [2.3.1] 星闪SoC WS63的结构
  - AHB与APB连接的Bridge
    - AHB：高速、高性能、复杂协议
    - APB：低速、低功耗、简单协议
    - Bridge的主要作用：相互转换
      - 不同速度模块的通信转换：CPU与外设
      - 协议的转换：复杂协议与简单协议
      - 地址映射：AHB地址与APB地址
  - 其他AMBA架构的总线
    - AXI(Advanced eXtensible Interface)
    - ACE(AXI Coherency Extensions)
    - CHI(Coherent Hub Interface)

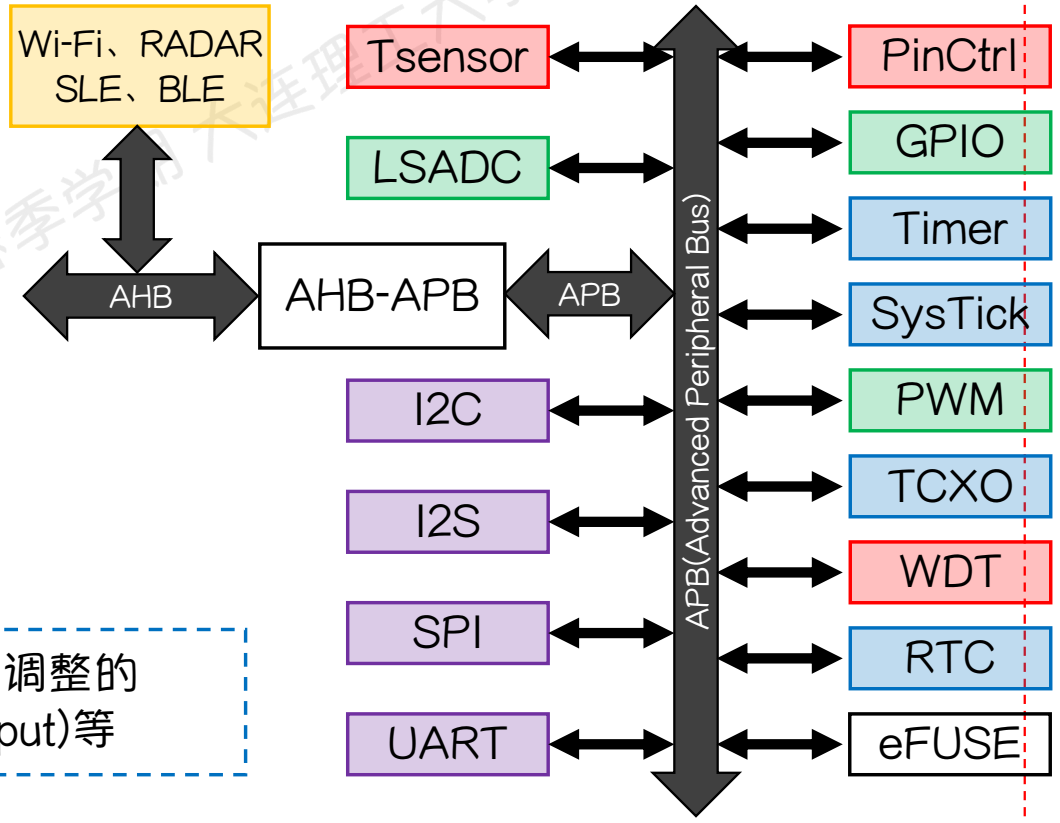
不能直接连接



2.3 SoC内部结构

- [2.3.2] 星闪SoC WS63的外设模块
  - 外设模块：SoC的各种硬件功能单元，完成特定的任务。
  - 与外部功能直接相关的模块
    - 简单信号：GPIO等
    - 有线通信：I2C、SPI等
    - 无线通信：Wi-Fi等
    - 时间控制：Timer等
    - 内部控制：WDT等
    - 敏感存储：eFUSE

部分模块的用途是可以根据用户设定调整的
如GPIO(General Purpose Input/Output)等



2.4 SoC的控制方式

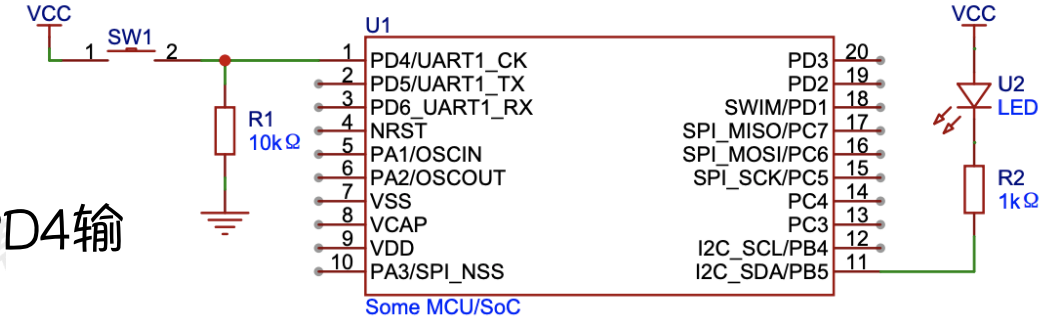
- [2.3.3] SoC的控制方式
  - 硬件决定性能，软件定义功能
  - 假定场景
    - U1是可以正常工作的SoC
    - SoC中运行的程序，可检测PD4输入，也可以控制PB5输出
    - SW1是按键，与PD4的关系如表
    - U2是LED，与PB5的关系如表

目标

按键按下 > SoC检测输入 SoC控制输出 > LED点亮

按键松开 > SoC检测输入 SoC控制输出 > LED熄灭

软件与硬件沟通的桥梁是什么？



按键状态	PD4输入电压	PD4输入逻辑
按键松开	0V	低电平(0)
按键按下	VCC(3.3V)	高电平(1)

PB5输出逻辑	PB5输出电压	LED状态
低电平(0)	0V	点亮
高电平(1)	VCC(3.3V)	熄灭

构建一段代码，来实现这一过程

2.4 SoC的控制方式

[2.3.3] SoC的控制方式

按键-SoC-LED控制的代码段

```
#define PD4 (*((volatile unsigned int *)0x40011404))
#define PB5 (*((volatile unsigned int *) 0x40010C08))

int main() {
    while(1) {
        if(PD4 == 1) {   ← 检测点(循环检测)
            //
        } else {
            //
        }
    }
}
```

软件与硬件沟通的桥梁是什么？

按键按下 > SoC检测输入 SoC控制输出 > LED点亮

按键松开 > SoC检测输入 SoC控制输出 > LED熄灭

按键状态	PD4输入电压	PD4输入逻辑
按键松开	0V	低电平(0)
按键按下	VCC(3.3V)	高电平(1)

PB5输出逻辑	PB5输出电压	LED状态
低电平(0)	0V	点亮
高电平(1)	VCC(3.3V)	熄灭

已经结课的C语言的知识

2.4 SoC的控制方式

- [2.3.3] SoC的控制方式
  - 按键-SoC-LED控制的代码段

```
#define PD4 (*((volatile unsigned int *)0x40011404))
#define PB5 (*((volatile unsigned int *) 0x40010C08))

int main() {
    while(1) {
        if(PD4 == 1) {
            PB5 = 0;
        } else {
            PB5 = 1;
        }
    }
}
```

软件与硬件沟通的桥梁是什么？

按键按下 > SoC检测输入 SoC控制输出 > LED点亮

volatile：变量易变化，编译器不要优化，每次都需要重新读取
(unsigned int *)：32位长度无符号型的指针，相当于p

#define PD4 *p
PD4为p指针指向的位置的内容，访问了特定内存地址的数据

Q：尝试分析功能
①if(PD4 == 1) { //balabala... }
②PB5=1

按键按下	VCC(3.3V)	高电平(1)

逻辑	PB5输出电压	LED状态
(0)	0V	点亮
(1)	VCC(3.3V)	熄灭

2.4 SoC的控制方式

[2.3.3] SoC的控制方式

- 按键-SoC-LED控制的代码段

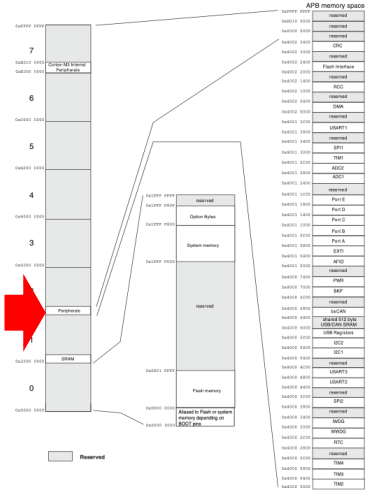
```
#define PD4 (*((volatile unsigned int *)0x40011404))
#define PB5 (*((volatile unsigned int *) 0x40010C08))

int main() {
    while(1) {
        if(PD4 == 1) {      //按键被按下
            PB5 = 0;        //LED点亮
        } else {            //按键没有被按下
            PB5 = 1;        //LED熄灭
        }
    }
}
```

对SoC的控制是通过对寄存器的读写实现的

软件与硬件沟通的桥梁是什么？

特定的内存地址通过硬件电路
与外设模块建立关联(不详述)

SoC与硬件有关联功能的特定的内存地址
特殊功能寄存器(Special Function Register)



①在硬件开发领域，特殊功能寄存器(SFR)一般也被简称为寄存器
②SFR在Memory Map中只占很小的内存空间，但控制了全部外设模块和大部分SoC内部功能
③SFR就是内存地址，很重要，需要记住吗？

2.4 SoC的控制方式

- [2.3.3] SoC的控制方式
  - 寄存器访问 vs API访问
    - 通过PD4、PB5类似的别名定义，对寄存器进行访问的

```
#define GPIO_CTRL_REG  (*(volatile unsigned int *)0x40021000)
GPIO_CTRL_REG = 0x01;
```

第一层封装
无需记住
寄存器地址

问题①：32位SoC的寄存器有32位长度，只控制一个引脚的功能就很浪费
问题②：32位寄存器的每一位都会表示不同硬件功能，很难记忆

    - 通过上层封装的函数接口，以抽象的方式间接访问寄存器
      - API通常由硬件厂商开发提供，隐藏系统的底层操作细节

```
HAL_GPIO_WritePin(GPIO_PORT, GPIO_PIN, GPIO_PIN_SET);
```

第二层封装
无需记住
寄存器名字

只要连接外设模块的名字，需要设置的属性即可，不需要记寄存器和定义

2.4 SoC的控制方式

[2.3.3] SoC的控制方式

寄存器访问的优势与劣势

优势	劣势
操作直接，执行效率高，无额外开销	对寄存器的配置细节要求较高，容易出错
适合高性能或实时性要求的系统	不易移植到其他平台或 SoC，硬件依赖性强
完全控制硬件功能和细节	难以维护，代码可读性和可维护性较差
不需要依赖厂商提供的库或 API	缺少对复杂功能的封装，开发效率较低

API访问的优势与劣势

优势	劣势
简化硬件操作，开发速度快，易于理解和使用	可能引入额外的开销，性能略低于直接寄存器访问
代码可移植性高，支持多个硬件平台	封装的复杂度可能限制某些底层硬件的高级功能
维护和调试简单，适合大规模团队开发	依赖于厂商的库或 API，若不支持则可能受限
隐藏底层细节，减少开发人员的学习成本	对于实时性要求高的场景可能不够高效

课程主要以API访问为主，部分时效性的实践案例会采用寄存器访问

2.4 SoC的控制方式

- [2.3.3] SoC的控制方式
  - 适用场景 - 寄存器访问
    - 实时性要求高：如驱动定时器、快速响应中断
    - 嵌入式开发入门或特殊需求：需要完全控制硬件的寄存器级配置
    - 资源受限的微控制器：使用寄存器操作可以减少代码大小
    - 无需厂商API支持：如某些简单或定制化的 oC
  - 适用场景 - API访问
    - 快速开发：需要在短时间内完成功能验证或产品开发
    - 跨平台开发：如基于STM32 HAL库或带有操作系统的项目
    - 团队协作：多人开发项目需要代码易读性和可维护性
    - 复杂外设控制：如 USB、以太网等，API 封装通常更高效和稳定

2.5 本章作业

- [2.5.0] 作业与思考

1. 智能硬件系统通过电气连接，为硬件设备之间建立了什么通道
2. 为何常规CPU和GPU很难应用在小型智能硬件系统上
3. MCU与SoC主要区别是什么
4. SoC按照性能进行分类，可以分为哪几类
5. 什么是总线(Bus)，传统MCU包括那三类内部总线
6. 简述现代AMBA架构中，AHB和APB的作用和区别
7. 简述API访问硬件和寄存器访问硬件的适用场景
8. 思考：SoC如何通过外设与外界进行信息交互(数据传输)

GPIO	UART	I2C	LSADC	PWM
Wi-Fi、RADAR、SLE、BLE		Timer	WDT	TCXO