嵌入式系统

北京邮电大学
计算机学院

戴志涛

嵌入式系统的开发与调试

- 嵌入式系统的交叉开发方式
- 嵌入式系统的开发工具
- 嵌入式系统调测方法与技术
- STM32软件开发方法

嵌入式系统开发

- 通用计算机
  - 具有完善的人机界面，增加一些开发程序和环境即可进行对自身的开发
- 嵌入式系统
  - 本身不具备自主开发能力，必须有一套开发工具和环境才能进行开发
  - 交叉开发——工具和环境一般是基于通用计算机上的软硬件及外部设备

交叉开发

- 宿主机：用于开发的机器（运行编辑器、编译器、仿真器、调试器、模拟器....）
- 目标机：程序运行的机器

嵌入式系统的开发工具

- 程序设计工具
  - 程序设计语言编译器(CompilerTools)
- 调试工具
  - 实时在线仿真器ICE(In-Circuit Emulator)
  - ROM仿真器
  - ROM监视器
  - 源程序模拟器（Simulator）
  - ……

程序设计语言编译器

- 汇编语言
  - 充分利用硬件的特性
  - 尽可能地提高时空效率
- 专门为嵌入式环境设计的高级语言
  - Intel PLM语言
- 标准程序设计语言的嵌入式版本
  - 嵌入式C语言

嵌入式系统程序设计语言的选择

- C语言：经典嵌入式系统事实上的标准
  - Why not Java or Python?
    - 可以精确控制处理器执行指令
      - C：中级语言?
    - 对ROM、RAM的要求低，可以降低系统成本
- 汇编语言?
  - 直接对硬件操作
    - 要理解编译器是否编译正确，需要理解其所生成的机器代码
  - 有时需要通过汇编版本的函数提高性能
  - 系统引导代码（bootloader）

程序设计语言编译器

- 汇编语言
  - 充分利用硬件的特性
  - 尽可能地提高时空效率
- 专门为嵌入式环境设计的高级语言
  - Intel PLM语言
- 标准程序设计语言的嵌入式版本
  - 嵌入式C语言
  - 嵌入式C++语言
  - Java ME（Java Platform，Micro Edition，J2ME）
- 交叉编译：编译器本身在通用平台上运行，但生成的二进制代码在嵌入式处理器上运行

嵌入式系统的开发与调试方法

- 嵌入式系统的开发与调试方法
  - 硬件仿真(Emulation)
  - 功能仿真(Simulation，模拟)
- 软件和硬件结合
  - 借助于“正确”的软件来测试硬件
  - 借助于“正确”的硬件来测试软件
- 不同的阶段，调测的内容、手段和使用的工具不尽相同
  - 产品开发阶段：排错
  - 产品开发后期及生产和现场运行阶段：测试

Debugging Toolset

- 在线仿真器（In-Circuit Emulator，ICE）
- ROM仿真器（ROM Emulator）
- ROM监视器（ROM monitor，Debug Monitor）
- 片上调试（On-Chip Debug）器
  - BDM
  - JTAG
- 指令系统模拟器（Instruction Set Simulator，ISS）
- 逻辑分析仪（Logic Analyzer）

In-circuit emulators (ICE)

Host computer runs emulator control software
- Provides run control
- Displays real time trace at source level
- Loads overlay memory with object code
- High-speed link to emulation chassis

Probe head contain emulation microprocessor
- Substitutes for, or disables target microprocessor
- Contains run control circuitry and cable buffers
- May contain memory mapping hardware

Target system

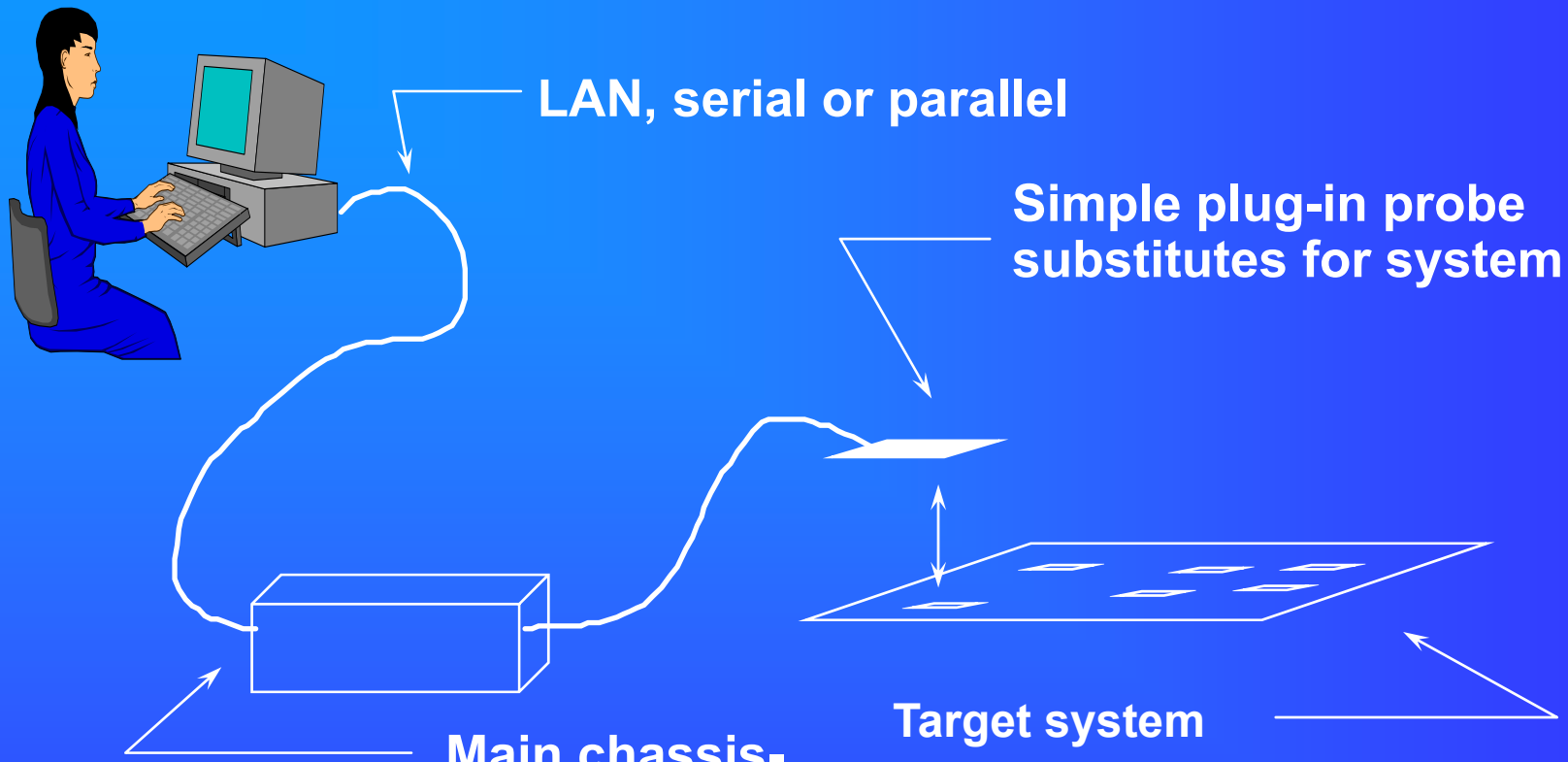
Main chassis
- Contains emulation (overlay memory)
- Trace analysis hardware and trace memory
- Performance analysis hardware
- Power supply
- Control and communications

A typical engineer with emulator

ICE的特点

- 优点
  - 集嵌入式系统连接、观察和控制所需的所有功能于一身
    - 确保对微处理器的控制
    - 仿真存储器取代目标系统ROM
    - 跟踪程序流程
    - 实现系统测量功能
  - 通过主机上的单一用户接口与系统紧密集成
- 缺点
  - 价格昂贵
  - 连接困难，尤其对表贴CPU
  - 可能造成系统不稳定
  - 仿真器的软件和硬件需要一定的CPU资源
  - 不适于高层代码调试

ROM Emulator



ROM Emulator 的优点

- 廉价
- 兼容于不同的存储器配置
- 代码高速下载至目标板
- 实时跟踪ROM代码的运行
- 可在ROM中设置断点

ROM Emulator 的缺点

- 目标板内存稳定后才能使用
- 只有代码在常规ROM中才能使用
- 只有直接在ROM中执行程序时才能实时跟踪
  - 许多目标系统由于性能的原因，将代码拷贝到RAM中运行

Remote Debugger

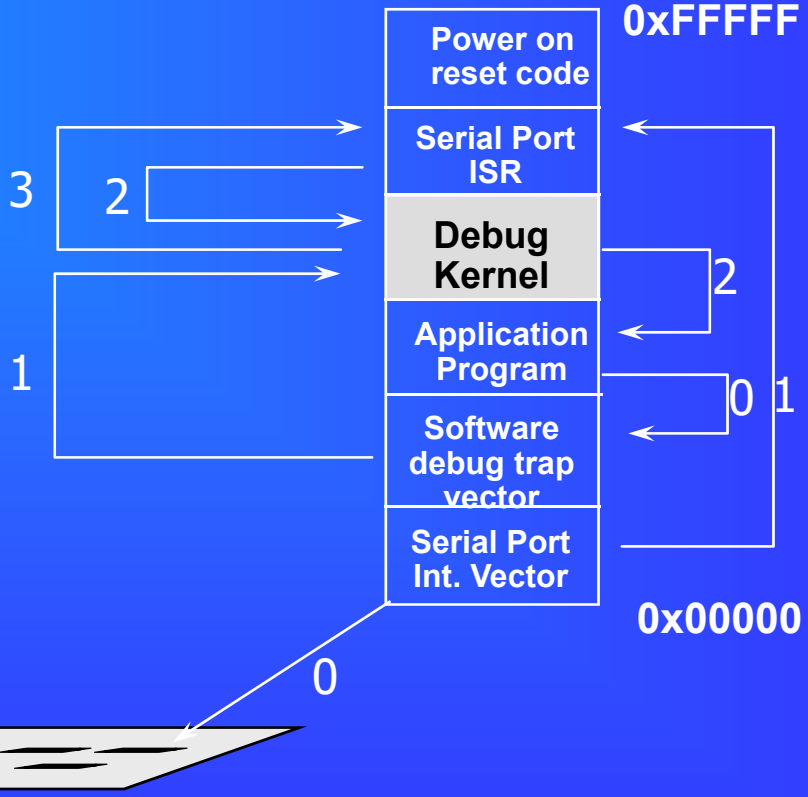
- Front-end runs on host computer and provides user interface
- Backend runs on target processor and communicates with the front-end over communication link
  - Backend is known as **debug monitor** and provides low level control of target processor

ROM monitor/Debug monitor

HOST-BASED DEBUGGER PROGRAM
- Knowledge of source files
- Knowledge of object files
  - > Symbol Table
  - > Cross reference files

SERIAL COMM LINK/
ETHERNET LINK

SYSTEM ROM
CODE PARTITION



ROM monitor的优缺点

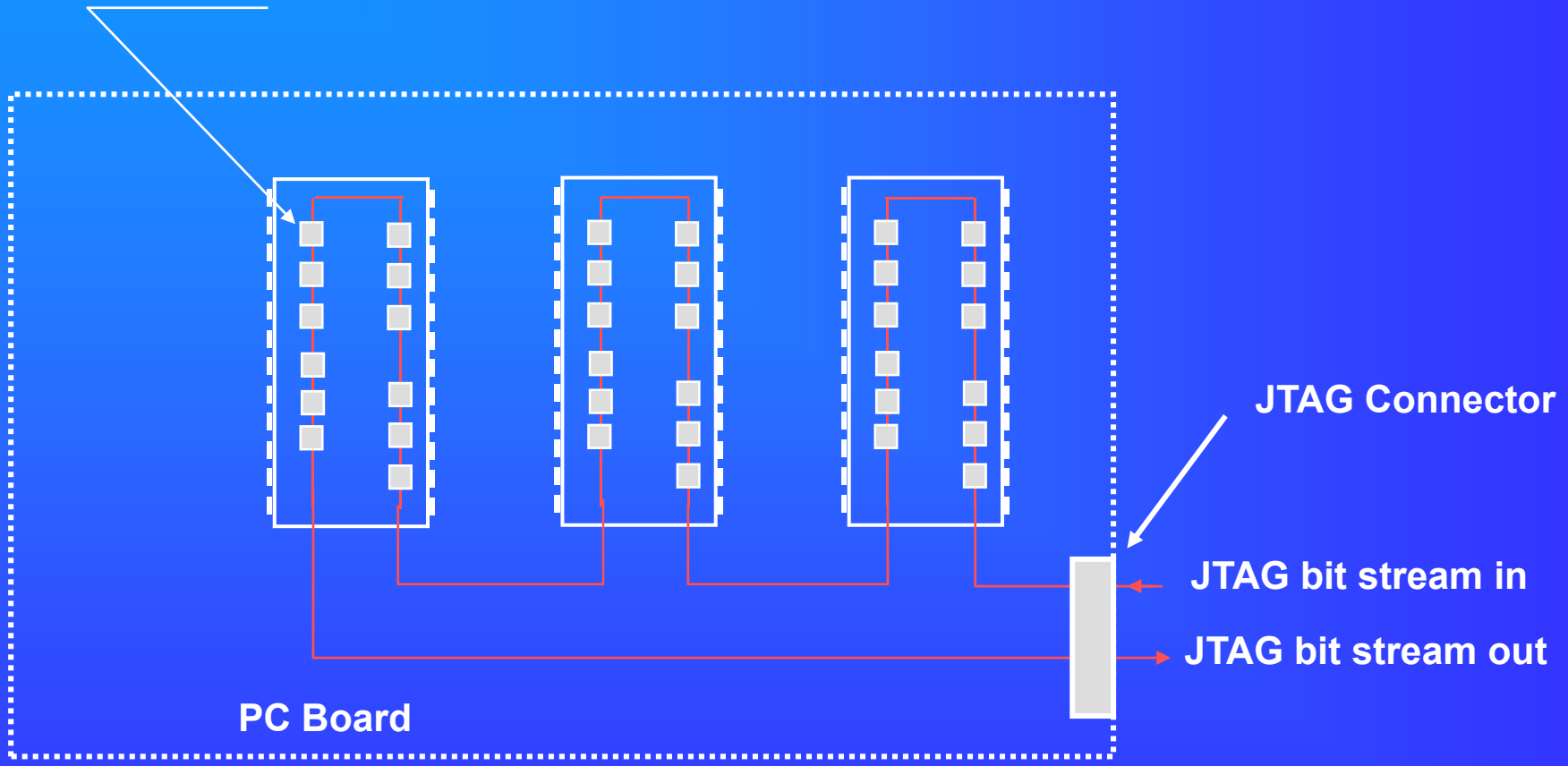
- 优点：
  - 最廉价的一种方式
  - 适于在硬件最小系统稳定后调试软件和外围硬件
- 缺点：
  - 依赖于目标系统上工作正常的存储器
  - 不适用于最开始的软硬件调试
  - 非“实时”
  - 系统性能受影响
  - 必须恰当安排中断优先级
  - 不能单步运行或设置断点

片上调试器

- 在微处理器内部嵌入额外的支持开发调试的控制模块
- 主机的调试器可以通过处理器内部特设的通信接口访问各种资源（寄存器、存储器等）并执行指令
  - 目标板上的CPU无需被替换取出
  - 主机上的调试软件和目标板上的CPU内部的调试微码借助集成在处理器上的串行通信链路进行通信
  - 允许程序跟踪、设置断点、单步运行、读写寄存器和内存单元等
- 接口方式
  - JTAG (Joint Test Action Group)

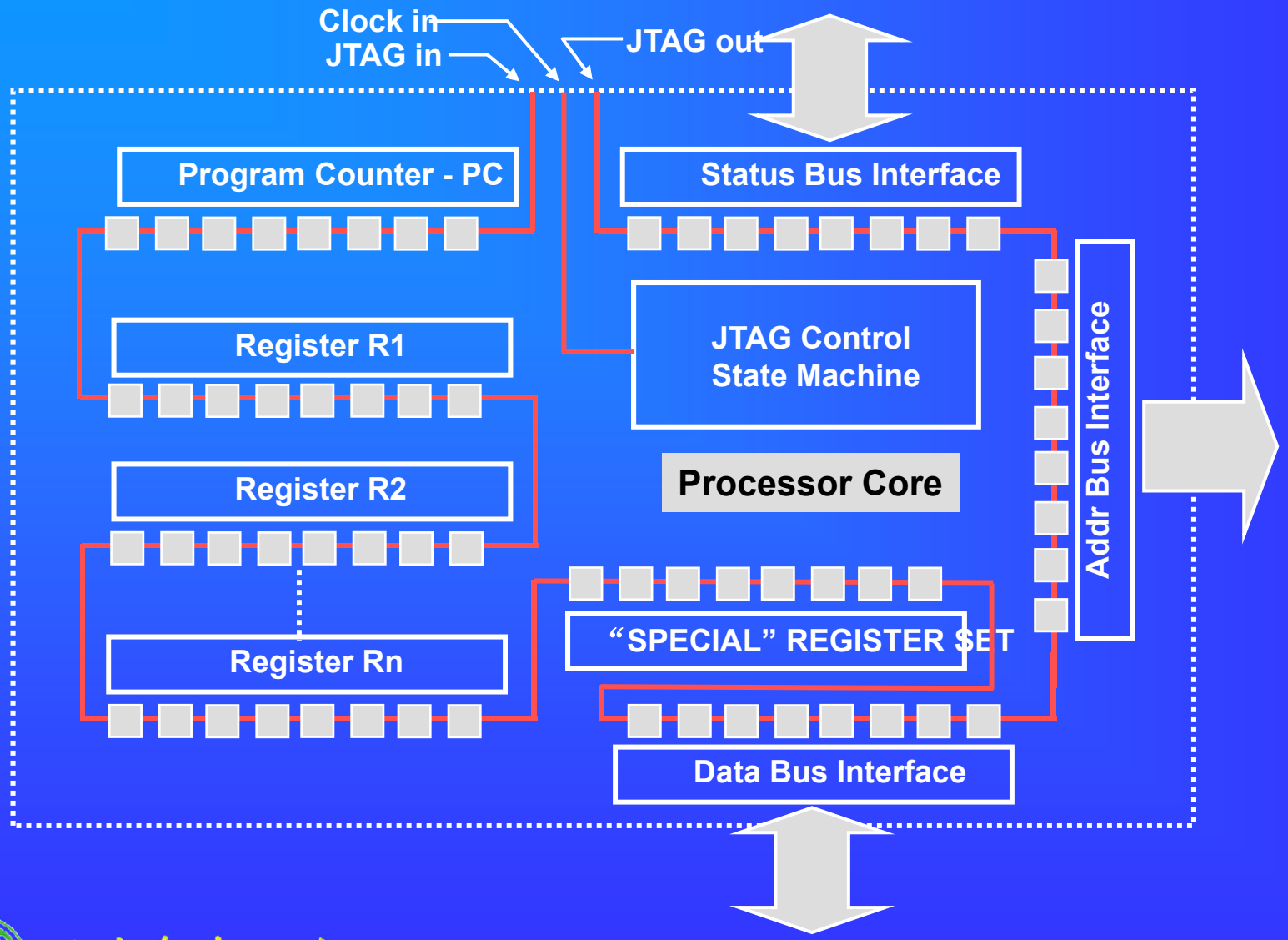
JTAG Loop

Each JTAG cell “sniffs” the state of the corresponding output bit of the IC



Bit stream forms one long shift-register

JTAG-based debug kernel



ARM JTAG调试器实例

ADT IDE For ARM

JTAG

Evaluation Board

片上调试器的优缺点

- 优点：
  - 低成本的标准调试接口，独立于芯片的变种
  - 一个低价设备支持所有JTAG兼容器件
  - 不占用目标平台的通信端口
  - 无需修改目标操作系统
  - 能调试目标操作系统的启动过程
- 缺点：
  - 速度低
  - 软件工作量增加

指令系统模拟器

- 在广泛使用的、人机接口完备的工作平台上通过软件手段模拟执行为某种嵌入式处理器内核编写的源程序的测试工具
- 两种类型
  - 功能精确：仅实现指令系统（纯软件）
  - 周期精确：精确保持处理器每个周期的行为特性
- 模拟器软件独立于处理器硬件，一般与编译器集成在同一个环境中

指令系统模拟器的优缺点

- 优点：
  - 廉价
  - 有效的源程序检验和测试工具
  - 用于项目初期，硬件未工作之前就可以调试软件
- 缺点
  - 在指令执行时间、中断响应、定时器等方面很可能与实际处理器有相当的差别
  - 无法仿真嵌入式系统在应用系统中的实际执行情况

仿真与模拟

- 硬件仿真（Emulation）
  - 高速
  - 价格昂贵
  - 在调试嵌入式软件之前需要搭好外围硬件电路
  - 很难区分软件和硬件的错误
- 模拟（功能仿真，Simulation）
  - 仿真速度低
  - 仿真结果与实际运行的结果的差异大
  - 只能对软件进行调试，不能对硬件进行调试

硬件辅助调试工具

- 示波器
  - 用于检查硬件任意位置的任何电气信号
  - 只能检查几个输入（例如4路）
  - 无触发逻辑

硬件辅助调试工具

- 逻辑分析仪
  - 优点
    - 强大的测量和触发能力
    - 可用于任何处理机
    - 高达200通道，大于1GHz的数据率
    - 可同时监视整个数字系统
  - 缺点
    - 价格昂贵
    - 不能检查模拟信号
    - 严格被动，不能控制处理机
    - 复杂
    - 不能与软件开发环境很好地集成
    - 必须配合其他工具使用

嵌入式系统调测方法与技术

- 开发的最初期
  - 借助仿真器或片上调试器进行硬件测试和启动代码调试
- 在目标板上设计测试接口
  - LED（或蜂鸣器）
  - 调试用串口
  - 调试用以太网端口

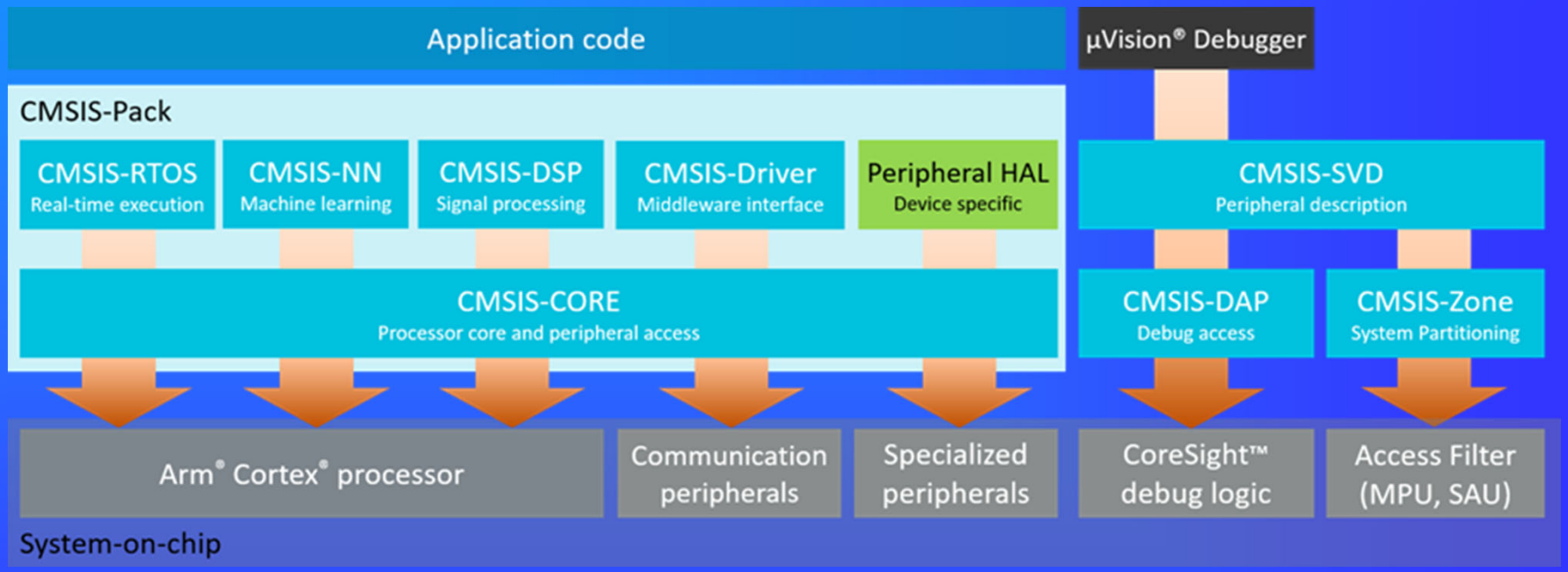
STM32软件开发方法

- 直接访问寄存器操作硬件
- STM32标准外设库
- STM32Cube和LL API

CMSIS简介

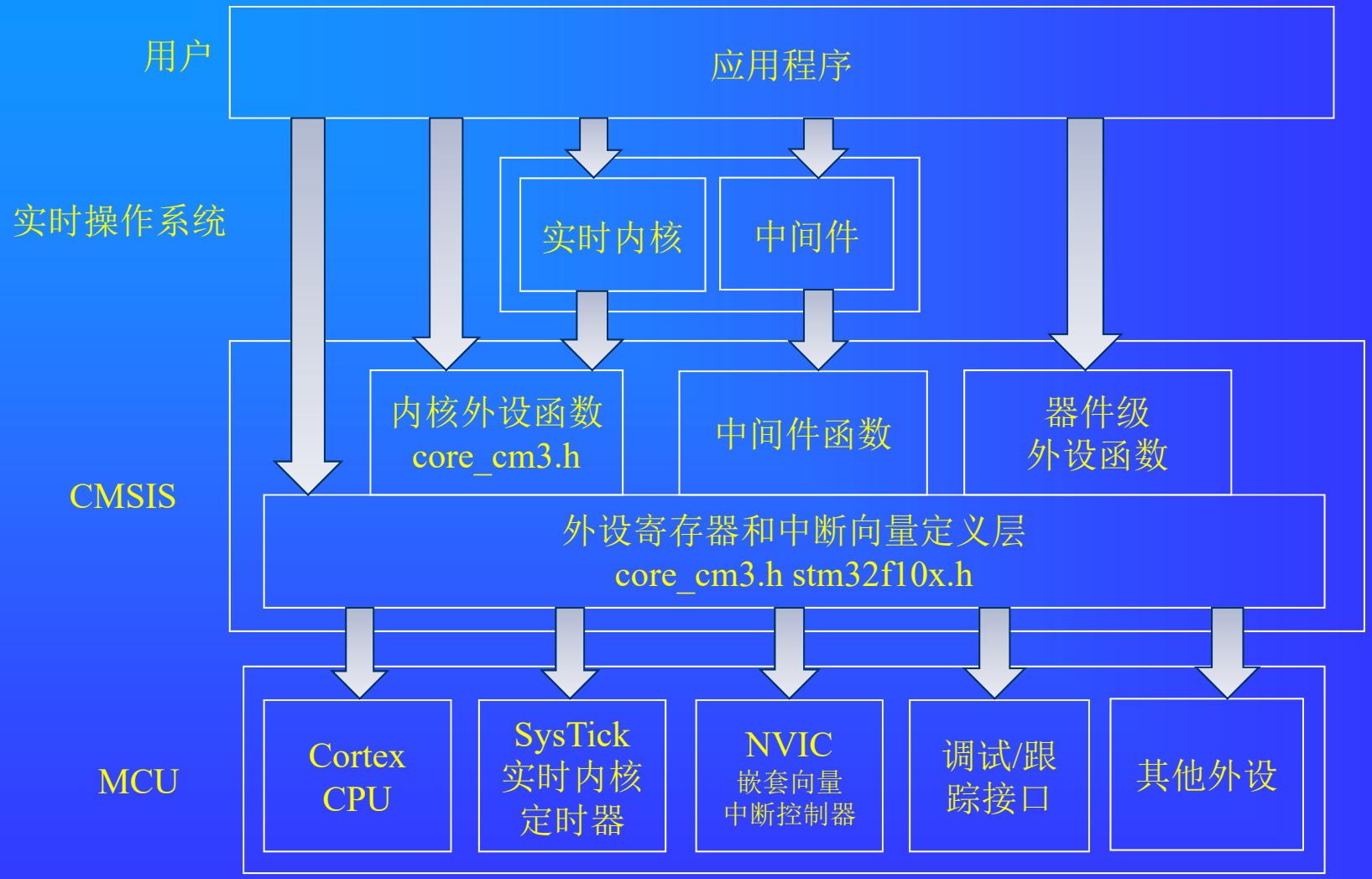
- Cortex Microcontroller Software Interface Standard
- Cortex微控制器软件接口标准
- 2008年由ARM提出的专门针对Cortex-M系列处理器的软件标准
  - 后扩展至Cortex-A5/A7/A9
- ARM、编译器厂家、半导体厂家共同遵循
- ARM和芯片厂商提供通用的API接口访问CORTEX内核及专用外设，以降低开发和移植成本
- 基于Cortex-M系列处理器的软件代码可以复用

基于CMSIS标准的软件架构



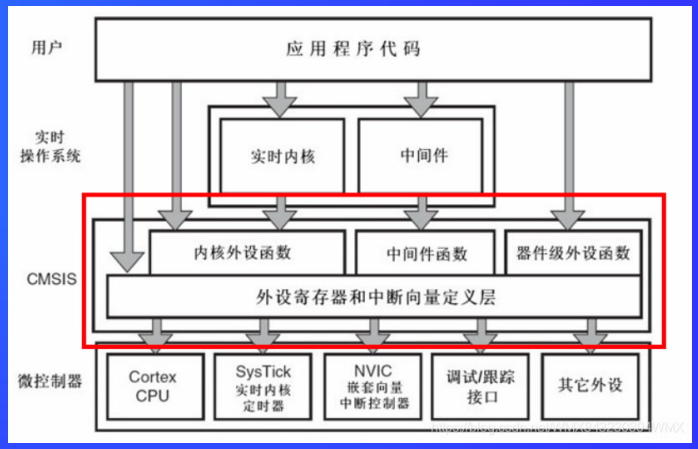
2021 Version V5.5.0

基于CMSIS标准的软件架构



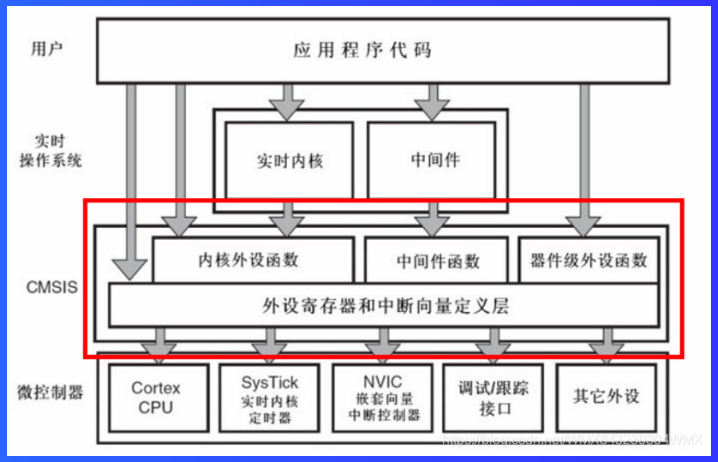
CMSIS V2.0的3个基本功能层

- 内核外设访问层（CPAL）
  - Core Peripheral Access Layer
  - 定义处理器片内寄存器地址及功能函数
    - 例：对内核寄存器、NVIC、调试子系统的访问
    - 一些对特殊用途寄存器的访问被定义成内联函数或内嵌汇编形式
  - 由ARM公司提供
- 中间件访问层（MWAL）
  - Middleware Access Layer
  - 定义访问中间件的通用API
  - 由ARM公司提供，芯片厂商根据需要更新



CMSIS的3个基本功能层

- 内核外设访问层（CPAL）
  - Core Peripheral Access Layer
  - 定义处理器片内寄存器地址及功能函数
- 中间件访问层（MWAL）
  - Middleware Access Layer
  - 定义访问中间件的通用API



- 设备外设访问层（DPAL）
  - Device Peripheral Access Layer
  - 定义硬件寄存器的地址以及外设访问函数
  - 由芯片厂商提供
  - 芯片厂商需要对异常向量表进行扩展，以实现中断处理
  - 可引用CPAL层定义的地址和函数

STM32标准外设库（SPL）

- 固件函数包，由程序、数据结构和宏组成
  - 最早称为固件库
- 包括每个外设的驱动程序
  - 由一组函数组成，覆盖该外设所有功能
- 所有代码经过严格测试，易于理解和使用，并且配有完整的文档，方便二次开发和应用
  - 包含每个外设的驱动描述和应用实例
- 为开发者访问底层硬件提供中间API
  - 开发者无需深入掌握底层硬件细节即可应用每个外设
- 符合CMSIS标准；以标准ANSI-C书写；与MISRA C 2004兼容

STM32标准外设库实例

GPIO固件库函数

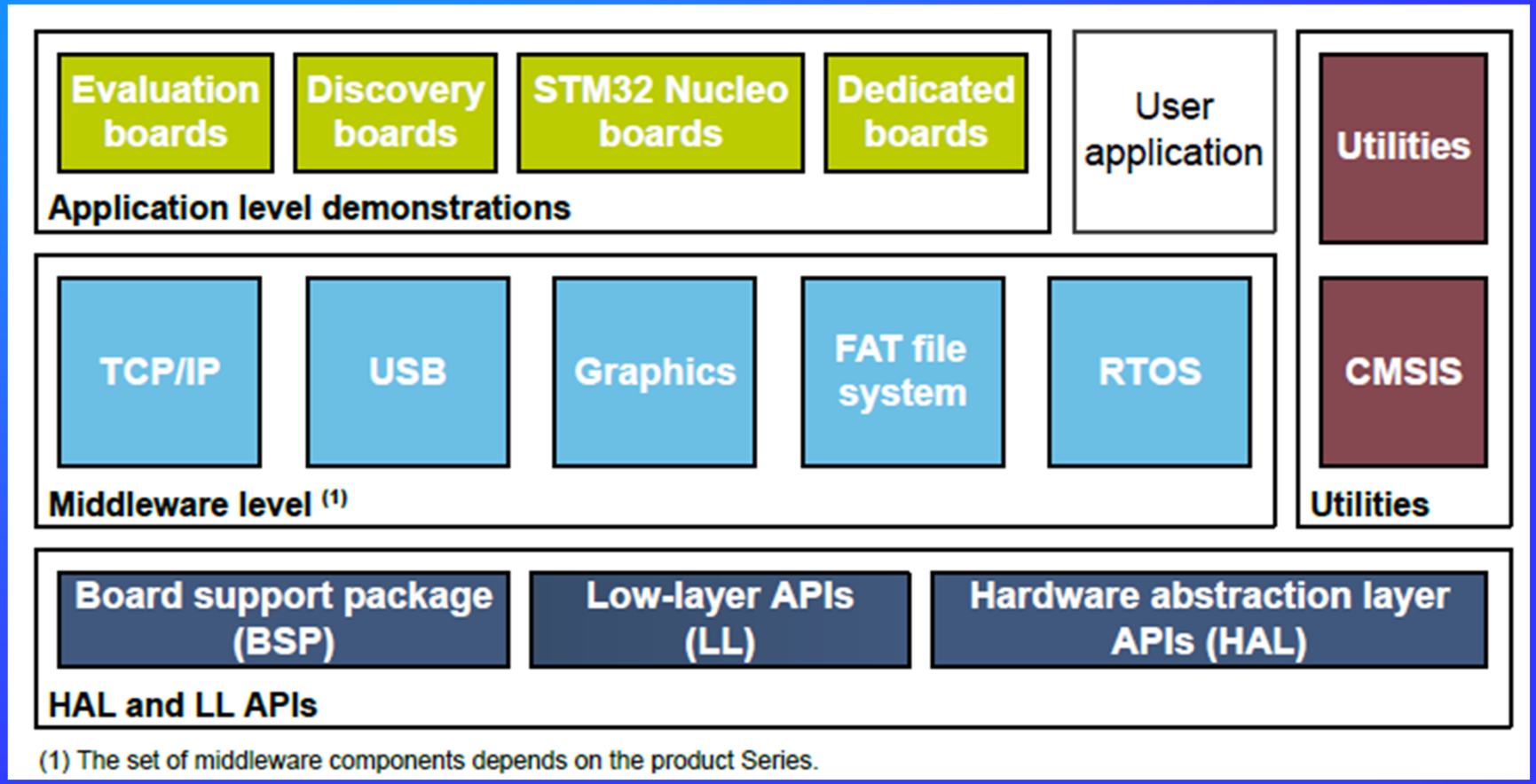
函数名	描述
GPIO_DeInit	将外设 GPIOx 寄存器重设为缺省值
GPIO_AFIODeInit	将复用功能（重映射事件控制和 EXTI 设置）重设为缺省值
GPIO_Init	根据 GPIO_InitStruct 中指定的参数初始化外设 GPIOx 寄存器
GPIO_StructInit	把 GPIO_InitStruct 中的每一个参数按缺省值填入
GPIO_ReadInputDataBit	读取指定端口管脚的输入
GPIO_ReadInputData	读取指定的 GPIO 端口输入
GPIO_ReadOutputDataBit	读取指定端口管脚的输出
GPIO_ReadOutputData	读取指定的 GPIO 端口输出
GPIO_SetBits	设置指定的数据端口位
GPIO_ResetBits	清除指定的数据端口位
GPIO_WriteBit	设置或者清除指定的数据端口位
GPIO_Write	向指定 GPIO 数据端口写入数据
GPIO_PinLockConfig	锁定 GPIO 管脚设置寄存器
GPIO_EventOutputConfig	选择 GPIO 管脚用作事件输出
GPIO_EventOutputCmd	使能或者失能事件输出
GPIO_PinRemapConfig	改变指定管脚的映射
GPIO_EXTILineConfig	选择 GPIO 管脚用作外部中断线路

STM32Cube

- 取代STM32标准外设库的解决方案
- 一个全面的软件平台，涵盖整个STM32产品组合
- 组成：
  - STM32CubeMX
    - 图形软件配置工具，允许使用图形向导生成C初始化代码
  - STM32Cube HAL（硬件抽象层）
    - STM32嵌入式软件层，确保在STM32产品组合中实现最大的可移植性
  - 底层（LL）API
    - 可移植，快速、轻巧，面向专家
    - 与HAL相比更接近于硬件，为HAL提供高性能、低占用空间的替代解决方案
    - 可以和HAL同时使用，但有限制
  - 中间件组件（RTOS、USB、FatFs、TCP/IP、Graphics等）

STM32Cube

STM32Cube MCU Package for STM32F4 Series with HAL, low-layer drivers and dedicated middleware



本章结束