

面向二十一世纪的嵌入式系统设计技术



第八讲： 嵌入式项目开发过程

Embedded System Project Management

主讲教员：徐欣



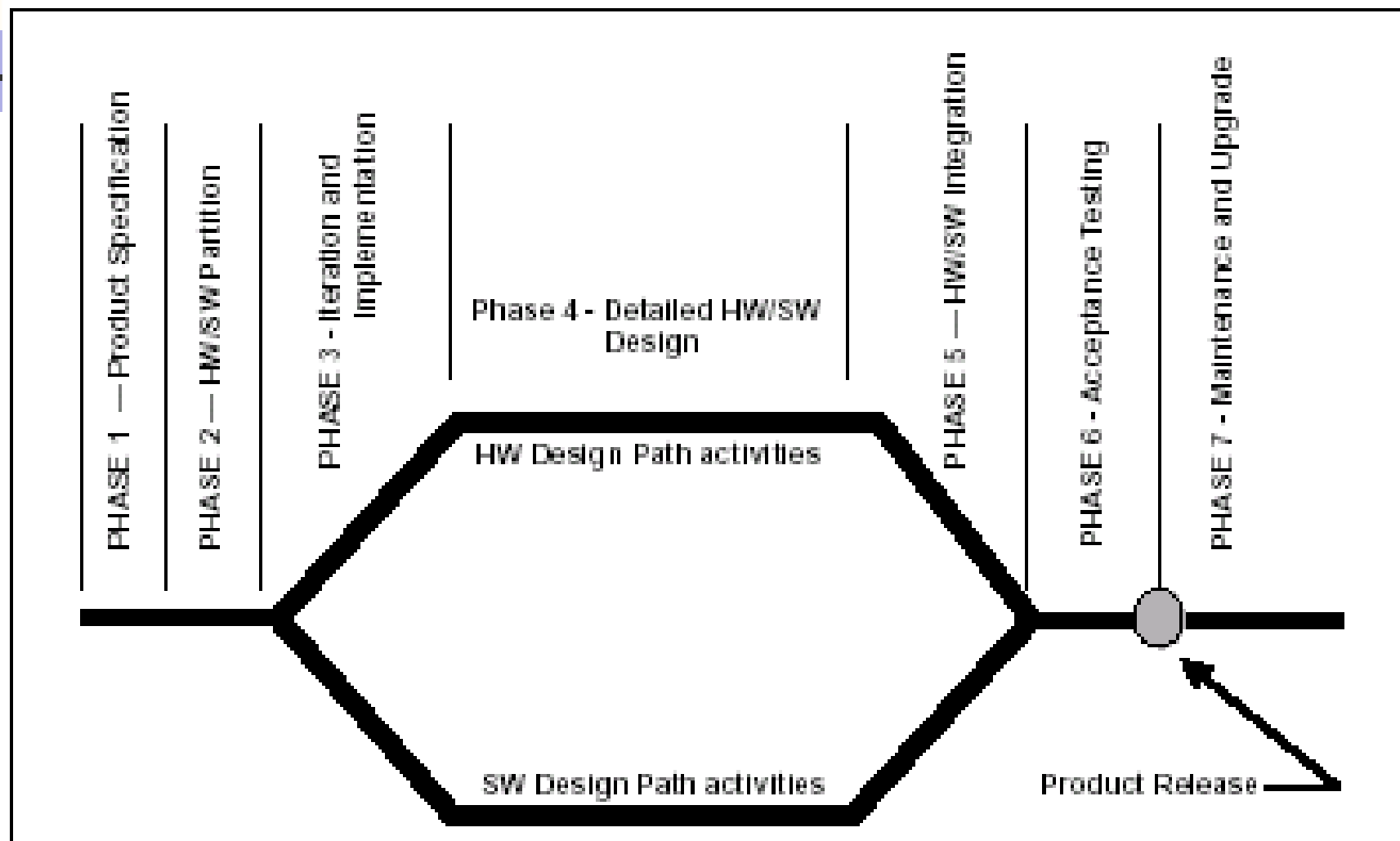
国防科大电子科学与工程学院
嵌入式系统开放研究小组



主要内容

- 嵌入式设计生命周期
- 选择过程
- 划分决策
- 详细的硬件与软件设计
 - 嵌入式硬件开发过程
 - 嵌入式软件开发过程
 - 软硬件协同设计过程
- 开发、调试环境与工具

嵌入式项目设计的各个阶段（图）



A phase representation of the embedded design life cycle.

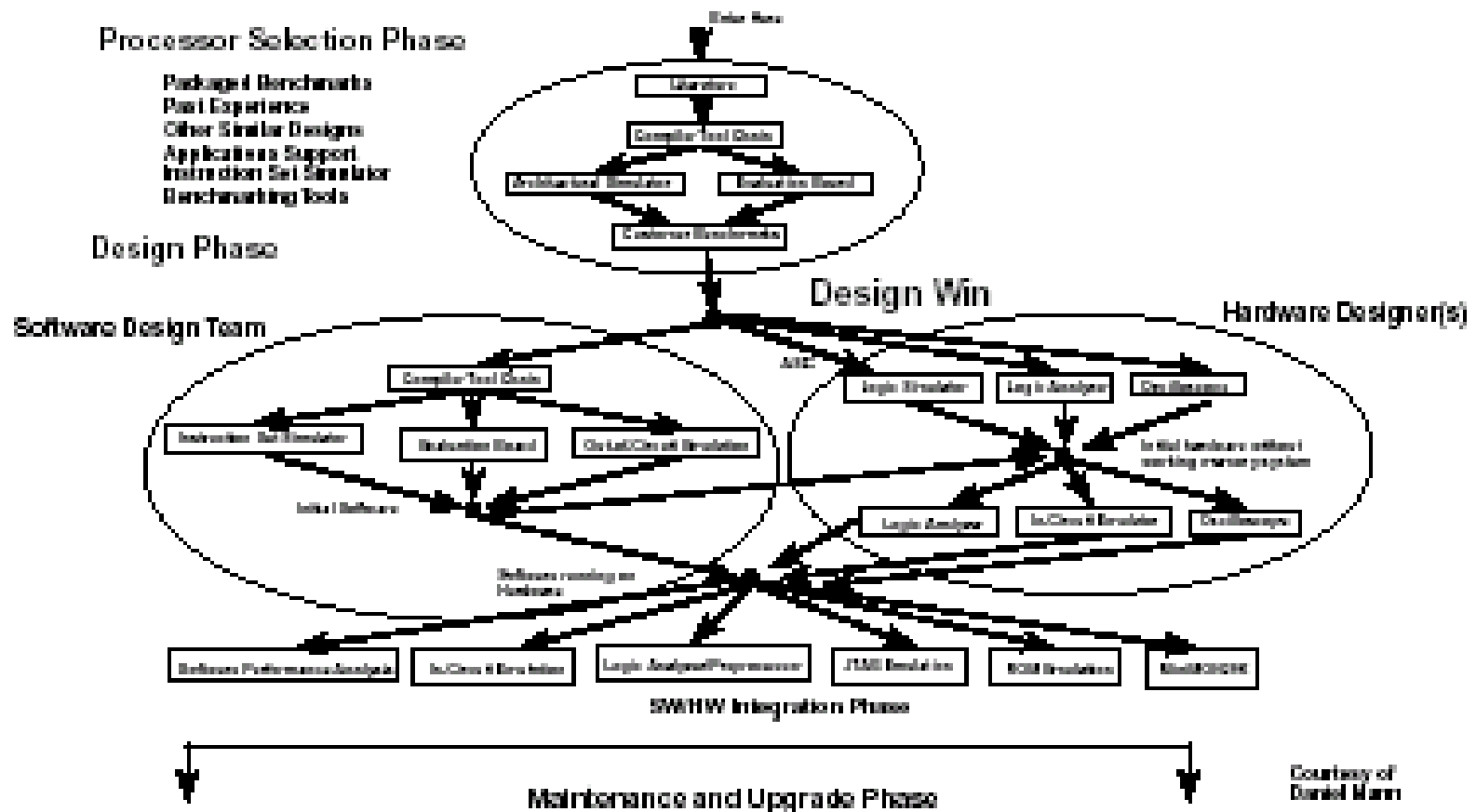


嵌入式项目设计的七个具体阶段

- 产品定义
- 软件与硬件的划分
- 迭代与实现
- 详细的硬件与软件设计
- 硬件与软件集成
- 产品测试与发布
- 持续维护与升级

嵌入式项目开发过程中使用的工具

- 参见PDF文档中的Figure 1.2





嵌入式项目设计生命周期（一）

产品定义

- 工程师 - - 追求卓越的功能和性能
 - 浪费时间和资源
- 决策层 - - 早期一般不允许工程师接触客户
 - 损失了一些有用的建议和观点
- →理想的客户研究访问
 - 首席：市场营销；第二成员：记录与提问
 - 其他技术人员：参与探讨并形成产品蓝图
- 列出必做适宜清单，找到设计产品的共同蓝图



嵌入式项目设计生命周期（二）

硬件与软件的划分

- 观点：软硬件是可以互相替换的
 - 如：浮点运算与浮点处理器（FPU）等
- 两种不同的划分策略
 - 优化处理器能力和软件
 - 通过ASIC设计找到解决途径
- 划分中需要考虑的许多需求
 - 价格低、性能领先、市场竞争、知识产权等
- CPU的选择将影响划分决策和开发工具选择

嵌入式项目设计生命周期（三）

迭代与实现

- 迭代与实现阶段的主要特点：
 - 主要障碍可能还是在软硬件的详细划分上
 - 设计约束被深刻理解和建模
 - 保留软硬件划分之间的余地
- 软硬件设计人员之间的迭代
 - 结构体系模拟器：Simulator
 - 评估板或开发板：Evaluation Board
- 目的：减小设计阶段后期风险

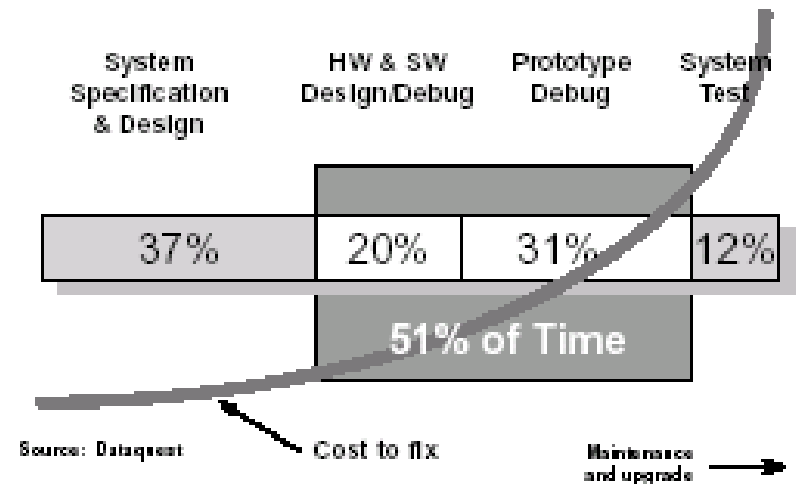
嵌入式项目设计生命周期（四）

详细的硬件与软件设计 - - 文档管理

- 这里不详细讨论软硬件设计问题
 - 大部分同学在其他课程中学到的C/C++/JAVA编程技术、数字设计和微处理器知识使他们有足够的机会解决设计中遇到的问题

- 文档管理与质量控制

- 设计复用和可视化
- 减小设计修改成本
- 有助于测试和质量控制





嵌入式项目设计生命周期（五）

硬件与软件集成

- Not a easy Problem
 - Big Endian/Little Endian引发的问题
 - 调试过程及实时系统调试方法带来的一些问题等
- 嵌入式系统设计中软硬件集成的颠峰状态
 - 由第一个硬件原型、应用软件、驱动代码、操作系统设计出完美的系统
 - 没有致命错误
 - 没有飞线
 - 不用重新设计ASIC或FPGA
 - 没有太多的软件设计修改



嵌入式项目设计生命周期（六）

产品测试与发布

- 嵌入式产品测试具有特殊的意义
 - 人们或许可以容忍PC偶然死机，但是核电站报警系统？！导弹控制系统？！
 - PC外围硬件 - - Is there any problem with you?
- 测试的目的
 - 不仅是确信软件不会在关键时刻崩溃
 - 还必须查明是否在运行时能接近最优性能，尤其是用高级语言编写或多个开发人员编写的程序
- 每个微小的错误都可能是致命的
 - 如轻微内存泄漏，长时间运行才能发现的问题等



嵌入式项目设计生命周期（七）

产品维护和升级

- 产品维护的模式
 - 维护/支持小组！ = 设计小组
 - 维护 = 详细文档 + 经验技巧 + 上一代产品
- 产品升级的巨大代价
 - 理解原设计人员的思路与代码
 - 反向逆推并改进原始设计小组的工作
 - 需要非凡的技艺或强大的反向设计工具
 - 否则，不如开始新的设计，这是原供应商和生产上所不愿意看到的



主要内容

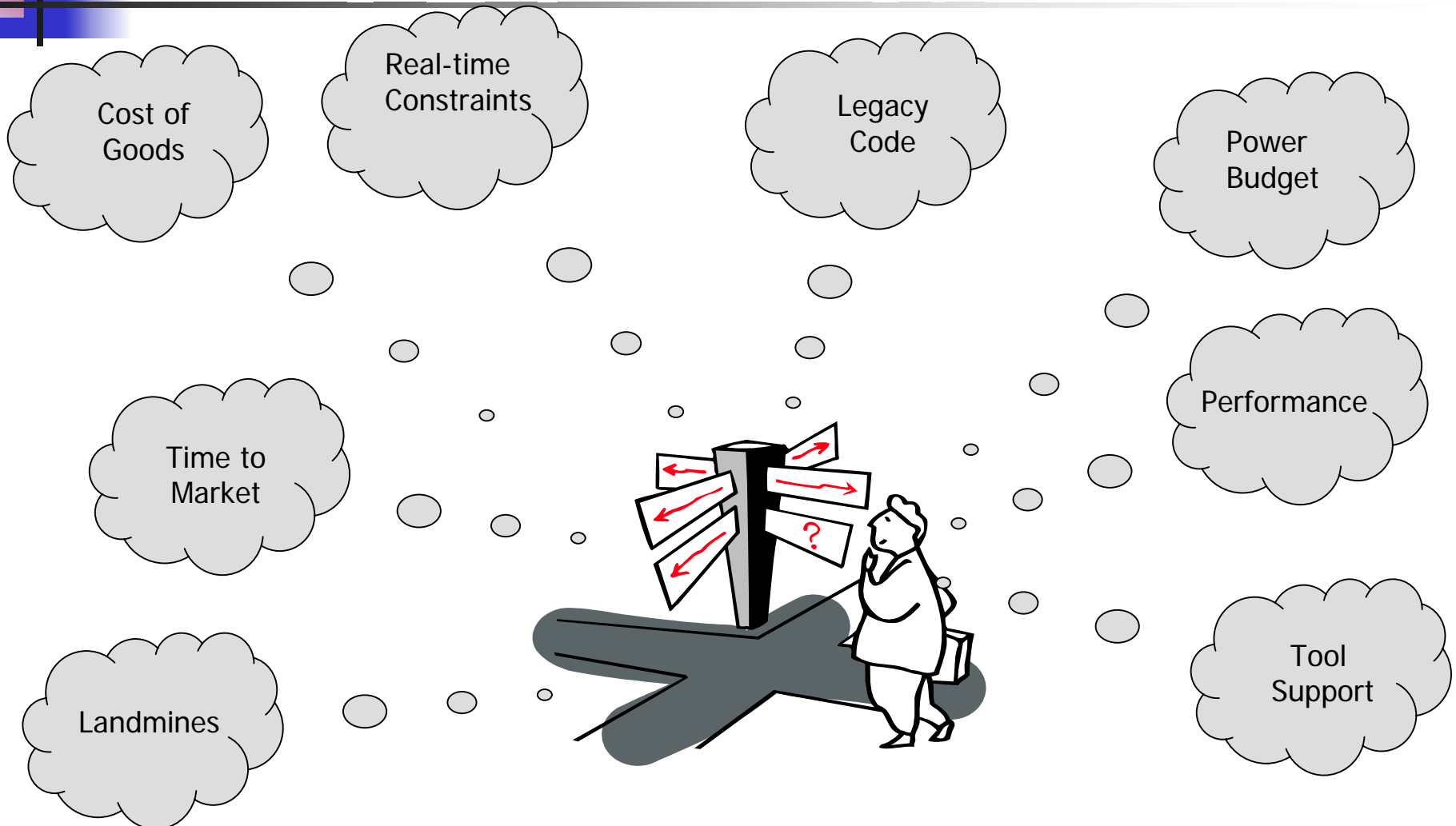
- 嵌入式设计生命周期
- 选择过程
- 划分决策
- 详细的硬件与软件设计
 - 嵌入式硬件开发过程
 - 嵌入式软件开发过程
 - 软硬件协同设计过程
- 开发、调试环境与工具



选择过程 - - 处理器平台

- 选择处理器是一个复杂的工作，它不仅是一个简单的“优化”问题，必须通过四道关键测试：
 - 是否便于实现
 - 是否能够提供足够的性能
 - 是否有合适的操作系统支持
 - 是否有大量合适的开发工具（和设计资源）支持
- 其他因素可能会影响这种选择
 - 上市时间、企业对特定开发商的偏好或承诺等

How do we choose microprocessor ?





Clock Speed

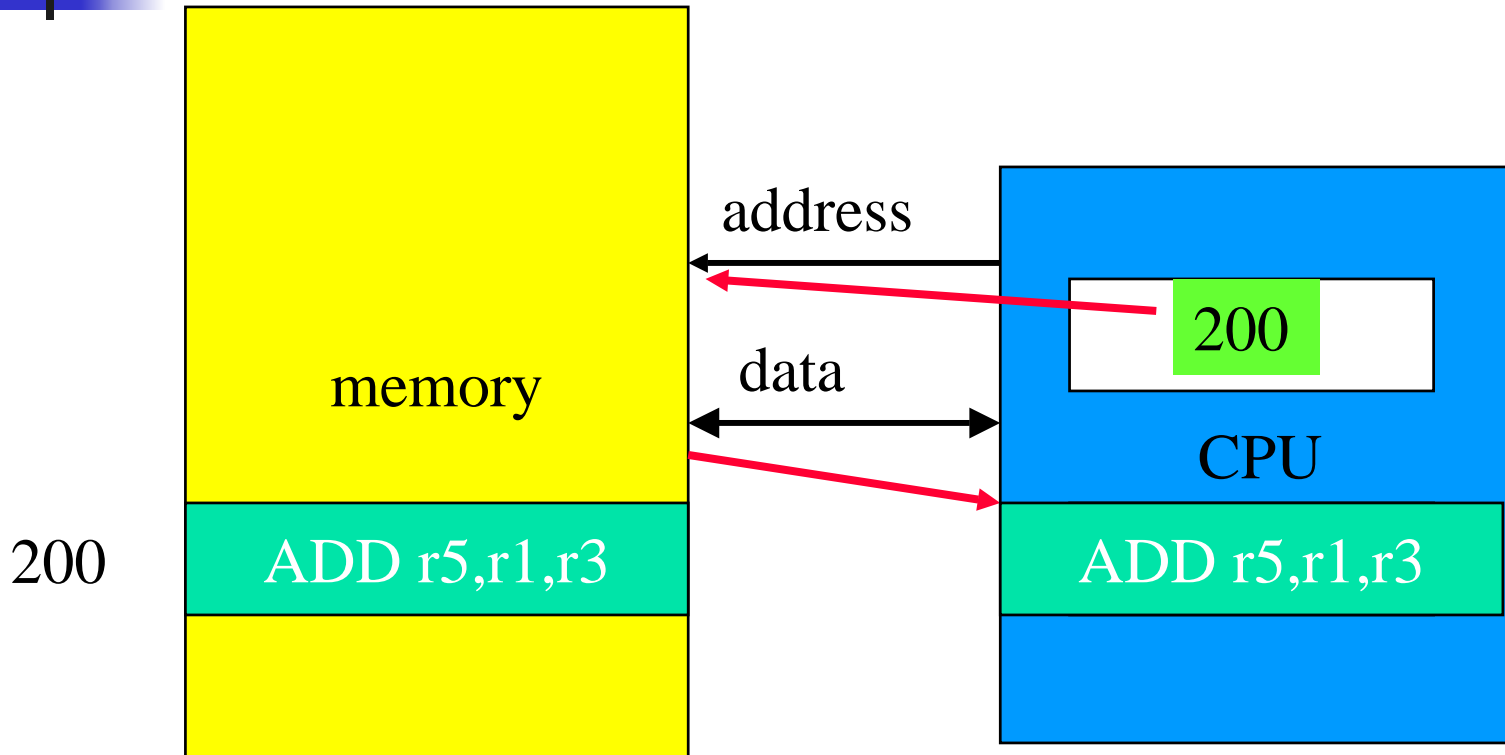
- Brute force method of improving performance
 - Bottleneck could be in software design or compiler !
- Faster isn't always better
- Performance ~ Clock speed
- Trade-off:
 - As clock speed ↑ energy ↑
 - Memory costs increase
 - Other peripheral devices will cost more



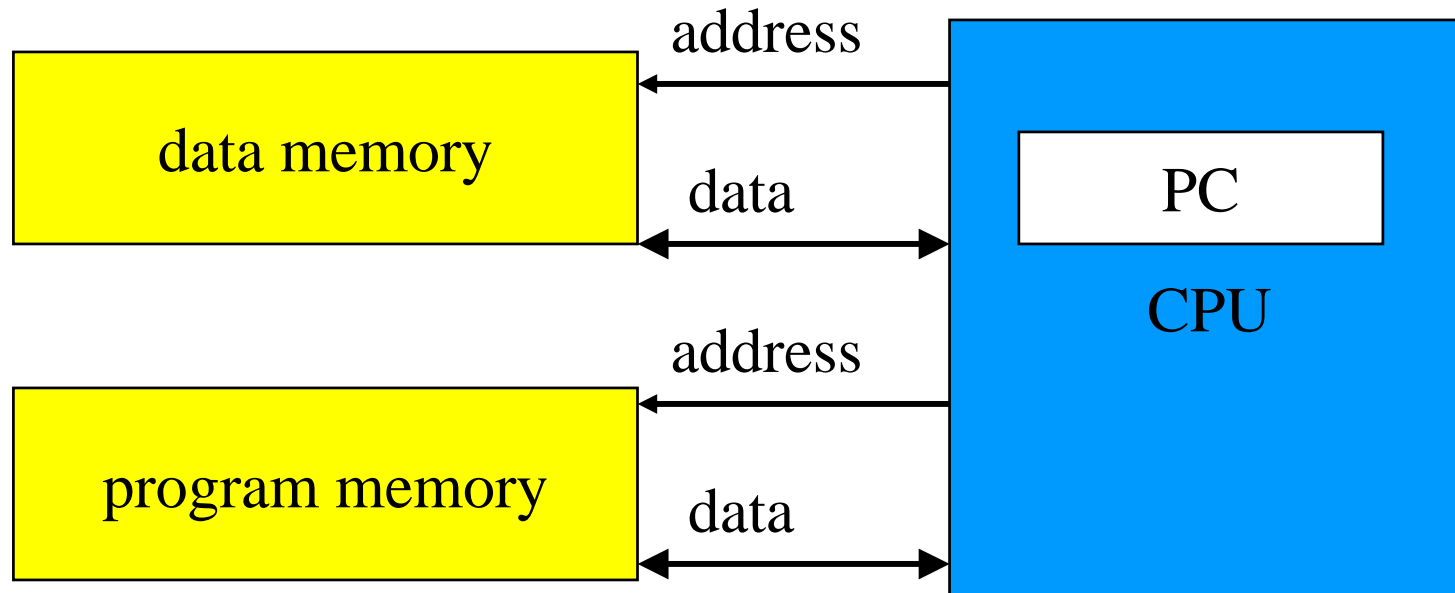
Evaluating processor performance

- **Clock speed**: but instructions per cycle may differ
- **Instructions/sec**: but work per instruction may differ
- **Dhrystone**: Synthetic benchmark, developed in 1984
- SPEC: realistic benchmarks, but oriented to desktops
- EEMBC – EDN Embedded Benchmark Consortium, www.eembc.org
 - Suites of benchmarks: automotive, consumer electronics, networking, office automation, telecommunications

von Neumann Architecture



Harvard architecture





von Neumann vs. Harvard

- Harvard can't use self-modifying code.
- Harvard allows two simultaneous memory fetches.
- Most DSP use Harvard architecture for streaming data:
 - greater memory bandwidth;
 - more predictable bandwidth.



ARM vs. SHARC

- ARM7 is von Neumann architecture
 - We will concentrate on ARM7
- ARM9 is Harvard architecture
- SHARC is modified Harvard architecture.
 - On chip memory (> 1Gbit) evenly split between **program memory (PM)** and **data memory (DM)**
 - Program memory can be used to store some data.
 - Allows data to be fetched from both memory in parallel



uP Performance

- Width of data path
 - performance $\sim (\text{Width of Data Path})^2$
 - The most general categorization of processor performance
 - Typical data bus widths: 4, 8, 16, 32, 64, 128 bits wide
 - Wider data busses -> greater data processing capability
 - Data bus width trade-off, the wider data path:
 - Is more complex to design
 - Takes up more room on PC boards
 - Generates greater amounts of energy
 - Requires more costly memory designs
 - Is not compatible with existing hardware



More on data path width

- Data path width generally determines functionality
 - 4,8 bits - Appliances, modems, simple applications
 - 16 bits - Industrial controllers, automotive
 - 32 bits - Telecomm, laser printers, high-performance apps
 - 64 bits - PC's, UNIX workstations, games
 - 128, 256 bits (VLIW) - Next generation
- Internal and external data paths may differ in size
 - Narrower memory is more economical
 - MC68000: 32-bit internal/16-bit external
 - MC68008: 32-bit internal/8-bit external
 - 80C188: 16-bit internal/8-bit external
- Remember: An 8-bit processor can do almost everything a 64-bit processor can do, it will just take longer to accomplish



Processor Micro-architecture

- On-chip instruction/data cache, how big?
- Pipelines
- Superscalar/VLIW
- Trade-off -> high performance costs money, power
- Address bus design
 - Address bus width: 16 - 36 bits
 - Multiplexed, synchronous, asynchronous
- Processor type: CISC, RISC, DSP
 - What is the nature of the algorithm to implement?
 - Control rich: CISC
 - Data rich: RISC
 - Data transforms and mathematical processing: DSP



More on address bus width

- The amount of externally accessible memory is defined as the **Address Space** of the processor
 - Can vary from 1KB for simple microcontrollers to over 60 GB in high performance processors
 - Size of the address space doesn't mean that you have that much memory, it only means that the capabilities exist to directly access it
 - Processors with smaller address spaces can still manipulate larger memory arrays with techniques such as **Paging**
 - Special memory or I/O location used to swap in and out memory pages
 - Example: An 8-bit Z80 processor with a 16-bit address bus (64K) can address a 1Mbyte address space by swapping between one of 16, 64Kbyte, memory pages



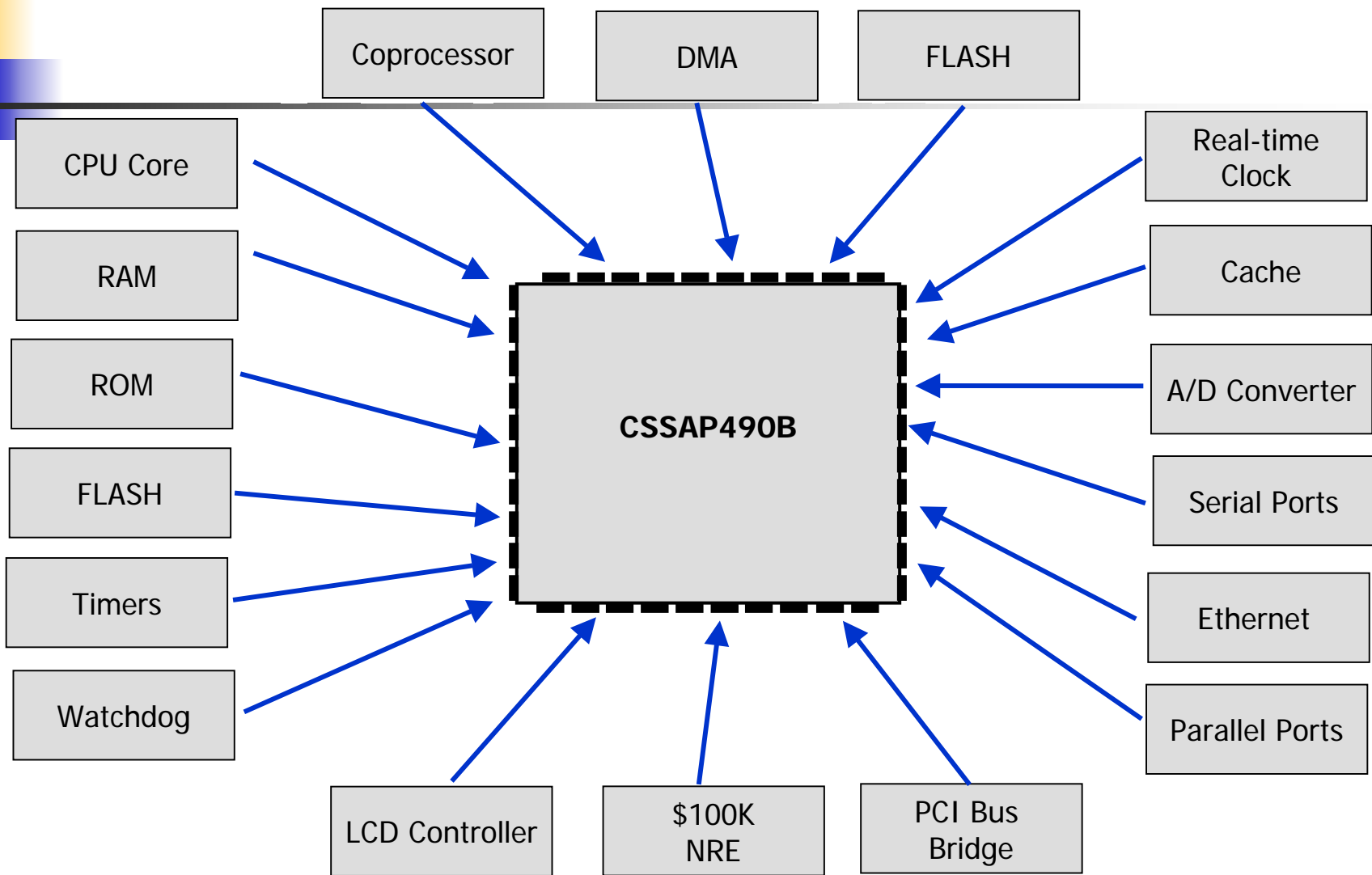
Single or Multiple processors

- Combine CISC, RISC and DSP in a single design
 - Tight coupling or loose coupling
 - Architecture
 - Code design, compiler capabilities
 - Debug tool availability
 - System simulation tools



Integration of functions

- Microprocessor or microcontroller?
- Review:
 - A microprocessor contains the basic CPU functionality, and more
 - A microcontroller combines the CPU core with peripheral devices
- The microprocessor is usually the leading edge of performance
 - Lowest level of integration
 - Highest cost
- Higher levels of integration imply
 - Lower system costs
 - Greater reliability
 - Less power
 - Faster
 - Higher processor cost
- As uP matures the core moves into the uC families





主要内容

- 嵌入式设计生命周期
- 选择过程
- 划分决策
- 详细的硬件与软件设计
 - 嵌入式硬件开发过程
 - 嵌入式软件开发过程
 - 软硬件协同设计过程
- 开发、调试环境与工具



划分决策

- 软件与硬件的双重性
- 软件与硬件的分离：基于开发成本和性能的决策
- 新的硬件描述语言：HDL - > Handel-C
- 协同设计过程



Hardware/Software Partitioning

- Definition

- The process of deciding, for each subsystem, whether the required functionality is more advantageously implemented in hardware or software

- Goal

- To achieve a partition that will give us the required performance within the overall system requirements (in size, weight, power, cost, etc.)
- This is a multivariate optimization problem that when automated, is an NP-hard problem



HW/SW Partitioning Issues

- Partitioning into hardware and software affects overall system cost and performance
- Hardware implementation
 - Provides higher performance via hardware speeds and parallel execution of operations
 - Incurs additional expense of fabricating ASICs
- Software implementation
 - May run on high-performance processors at low cost (due to high-volume production)
 - Incurs high cost of developing and maintaining (complex) software



Partitioning Approaches

- Start with all functionality in software and move portions into hardware which are time-critical and can not be allocated to software (*software-oriented partitioning*)
- Start with all functionality in hardware and move portions into software implementation (*hardware-oriented partitioning*)



主要内容

- 嵌入式设计生命周期
- 选择过程
- 划分决策
- 详细的硬件与软件设计
 - 嵌入式硬件开发过程
 - 嵌入式软件开发过程
 - 软硬件协同设计过程
- 开发、调试环境与工具



软硬件设计过程中的文档管理

- 需求分析文档（产品定义阶段）
- 总体方案设计（选择过程和软硬件划分）
- 概要设计文档（软硬件初步设计）
- 详细设计文档（软硬件详细设计）
- 测试需求文档（模块测试及联调准备）
- 系统测试报告（测试小组）
- 使用说明文档/源程序注释



总体方案设计

- 项目概述（来自需求分析文档）
- 功能与指标描述（来自需求分析文档）
- 系统外部接口描述
- 系统软硬件设计框架（选择过程和划分决策）
- 软硬件模块化设计概要
 - 功能、接口
- 时间与进度安排（甘特图）
- 产品成本估算
- 研制经费需求

甘特图

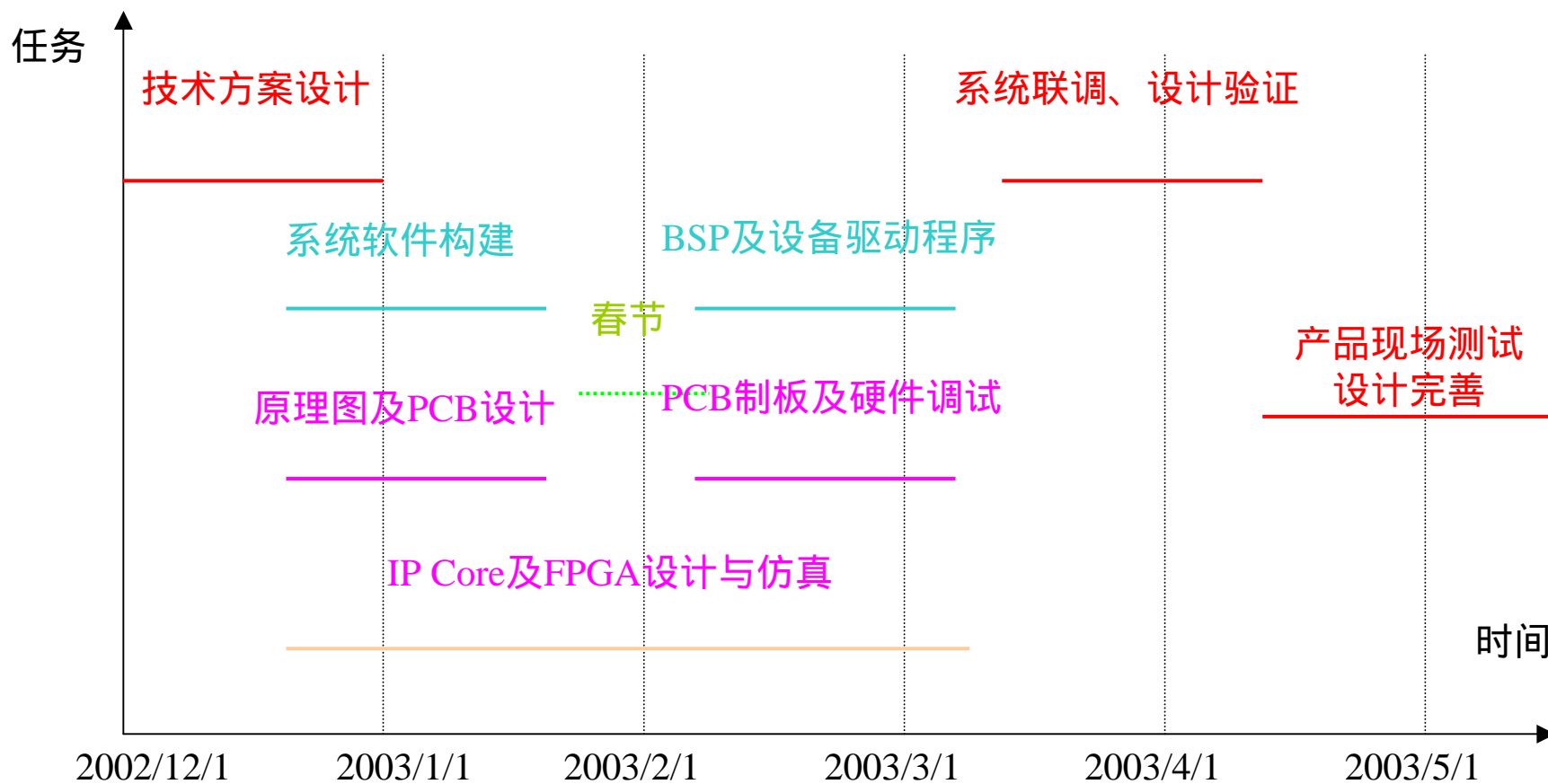


图4 项目研制进度与计划安排

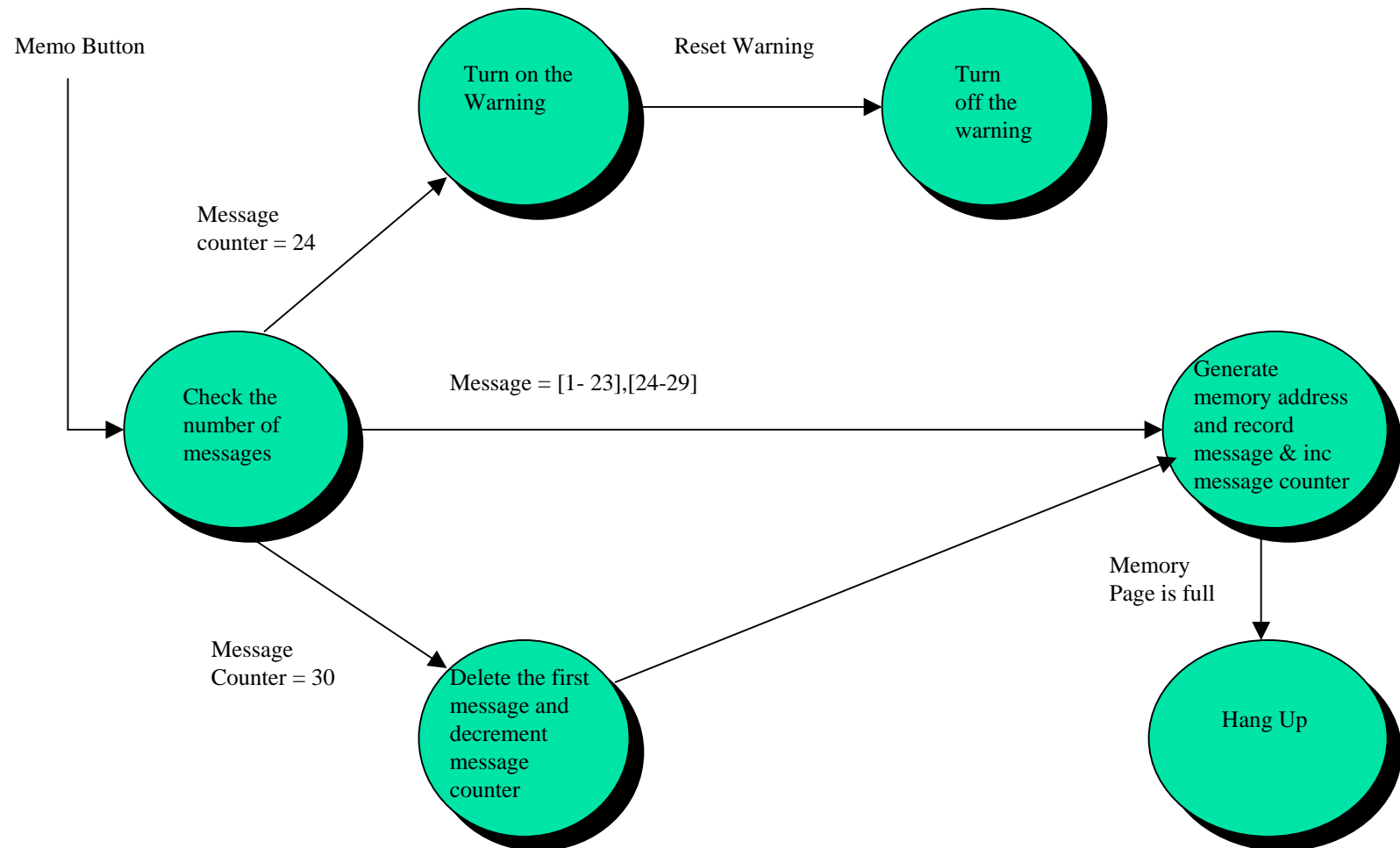


模块化设计

- 功能描述
- 接口描述：硬件接口与软件参数
- 设计流图
 - 自然语言
 - 流程框图
 - 原理框图
 - 状态流图

UML：统一建模描述语言

状态流图 - - 软硬件统一描述方式



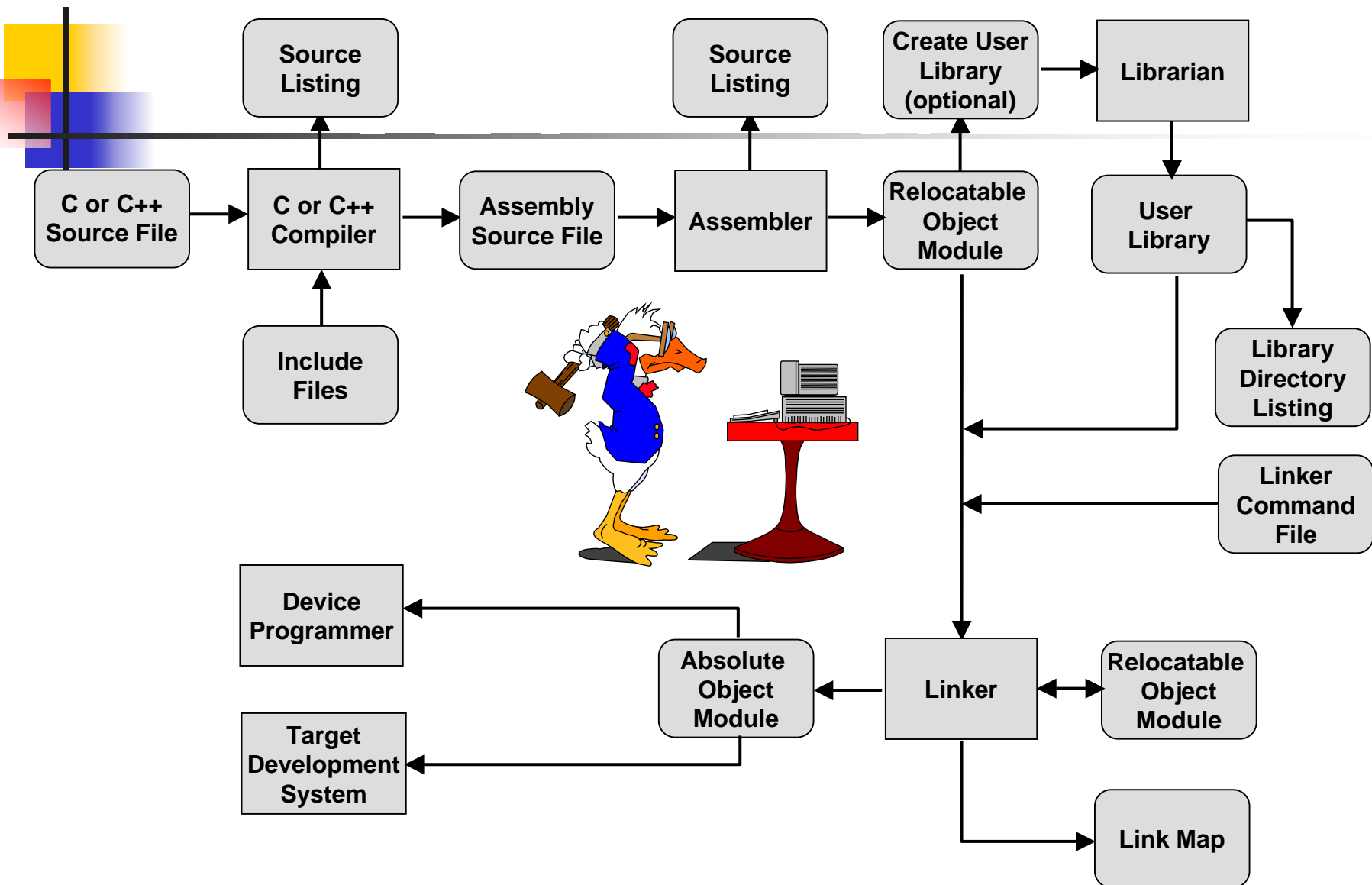
State Diagram of Recording a Message



主要内容

- 嵌入式设计生命周期
- 选择过程
- 划分决策
- 详细的硬件与软件设计
 - 嵌入式硬件开发过程
 - 嵌入式软件开发过程
 - 软硬件协同设计过程
- 开发、调试环境与工具

嵌入式软件开发环境





Debugging Toolset

- Instruction Set Simulator (ISS)
- Debug Monitor
- ROM Emulator
- Logic Analyzer
- In-Circuit Emulator (ICE)
- Joint Test Action Group (JTAG)



Debugging embedded systems

- Challenges:

- target system may be hard to observe;
- target may be hard to control;
- may be hard to generate realistic inputs;
- setup sequence may be complex.



Instruction Level Simulator

- Host based software that simulates the functionality and instruction set of the target processor
- Two types
 - Functionality-accurate
 - Implements only instruction set
 - Cycle-accurate
 - Maintain cycle-by-cycle accuracy of processor, including cache, pipeline, and memory behavior
- Useful in early stage of the project
- Disadvantage:
 - No simulation of peripherals
 - Cannot model interaction with I/O devices



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



Debug monitor

- A monitor program residing on the target ROM provides basic debugger functions
 - Read register x
 - Modify register y
 - Read n bytes of memory starting at address
 - Modify data at address
 - Run to breakpoint
 - Single step
 - Load code
- Debugger should have minimal footprint in memory.
- User program must be careful not to destroy debugger program

Debug monitor (Contd.)

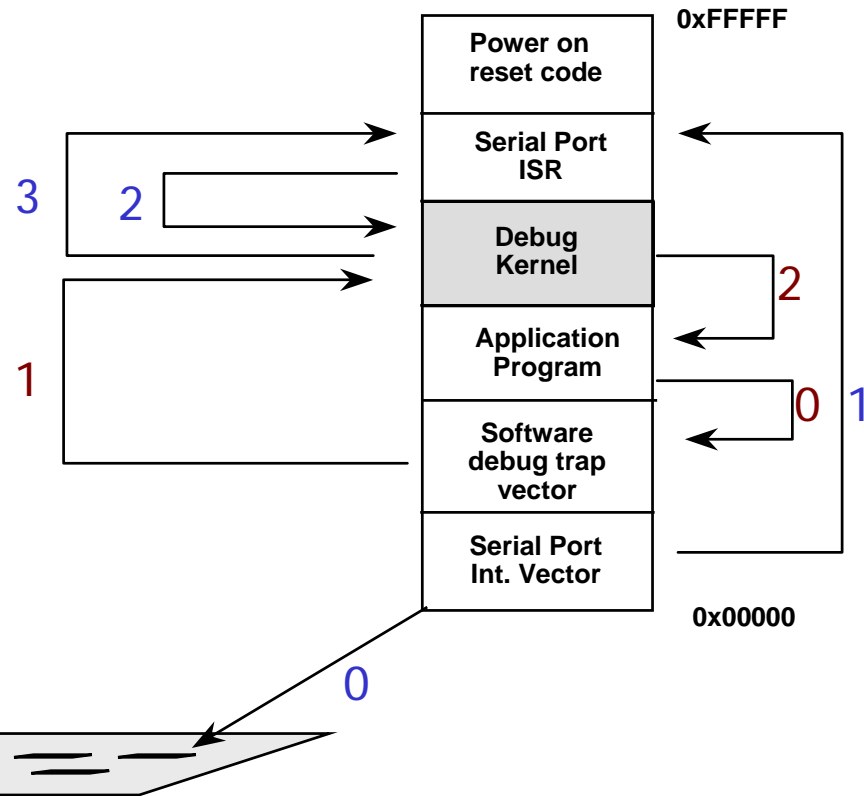
HOST-BASED DEBUGGER PROGRAM

- Knowledge of source files
- Knowledge of object files
 - > Symbol Table
 - > Cross reference files

SYSTEM ROM CODE PARTITION



SERIAL COMM LINK





Debug monitors: Advantages

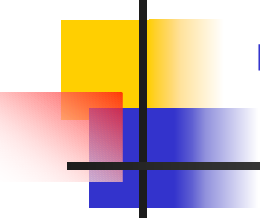
- Low cost: \$0 to <\$1K
- Provides most services software designer needs
- Simple serial link is all that is required
- Good choice for code development when hardware is stable
- Can easily be integrated into a design team environment



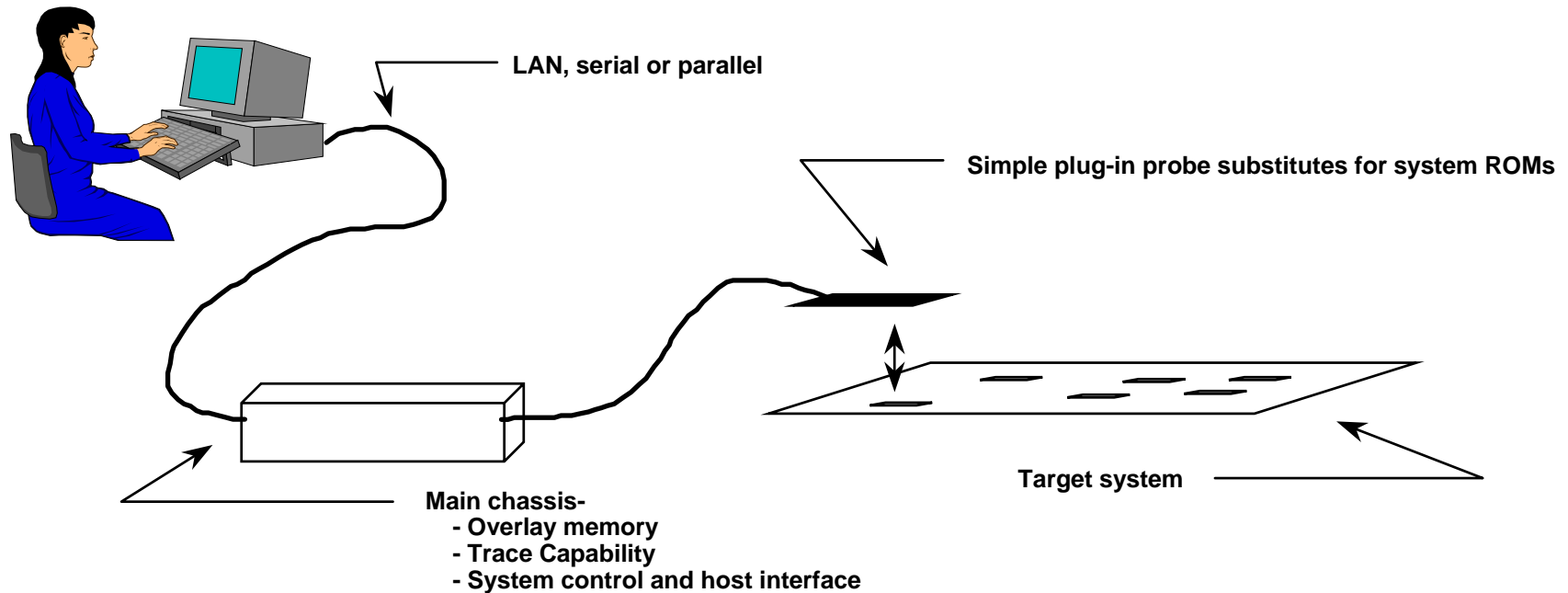
Debug Monitor: Disadvantages

- **Depends upon a stable memory subsystem in target**
 - **Not suitable for initial hw/sw integration**
- **Not “real time”**
 - **System performance may differ with a debugger present**
 - **Debugger can co-exist with real time interrupts through careful assignment of interrupt priority levels**
- **Difficulty in running out of ROM-based memory**
 - **Can’t single step or insert breakpoints**
- **Requires that the target has additional services**
 - **For many target systems this is an unacceptable cost**
- **Debugger may not always have control of the system**
 - **Depends upon code being “well behaved”**

ROM Emulator

- 
- Devices that plug into a ROM socket on target system, but contain RAM rather than ROM
 - Components:
 - Cabling device to match mechanical footprint of target system ROM device
 - Fast RAM
 - Local control processor
 - Communication port to host
 - Additional features, such as trace memory
 - Provides a simple and fast way to download ROM-based code into a target system via the ROM socket
 - Enables communications to host debugger for target systems that lack a serial port

ROM Emulator (Contd.)





ROM Emulator: Advantages

- Very cost-effective (\$1K - \$5K)
- Compatible with different memory configurations
- High-speed download to target of large blocks of code
- Can trace ROM code activity in real time
- Can be integrated with other hw/sw integration tools
- Can set breakpoints in “ROM”
 - Normally cannot do this in most target systems



ROM Emulator: Disadvantages

- Requires target system memory to be in stable condition
- Only works if code is contained in standard ROM
 - Special ASIC
 - Microcontrollers with onboard ROM (e.g. 8751)
- Real time trace is possible only if program executes directly out of ROM memory
 - Many targets transfer code to RAM for performance reasons



Hardware-assisted debug

- Real time debugging places special requirements on tools designed to control and observe the behavior of embedded systems
- Software debug monitors are ineffective when the processor-to-memory interface is not functional
 - Can be destroyed by code running wild
 - Transitory real time events cannot be observed
 - Debug monitor can perturb the target system and may interfere or prevent its operation
- Specialized tools have evolved to address the needs of real time system debugging

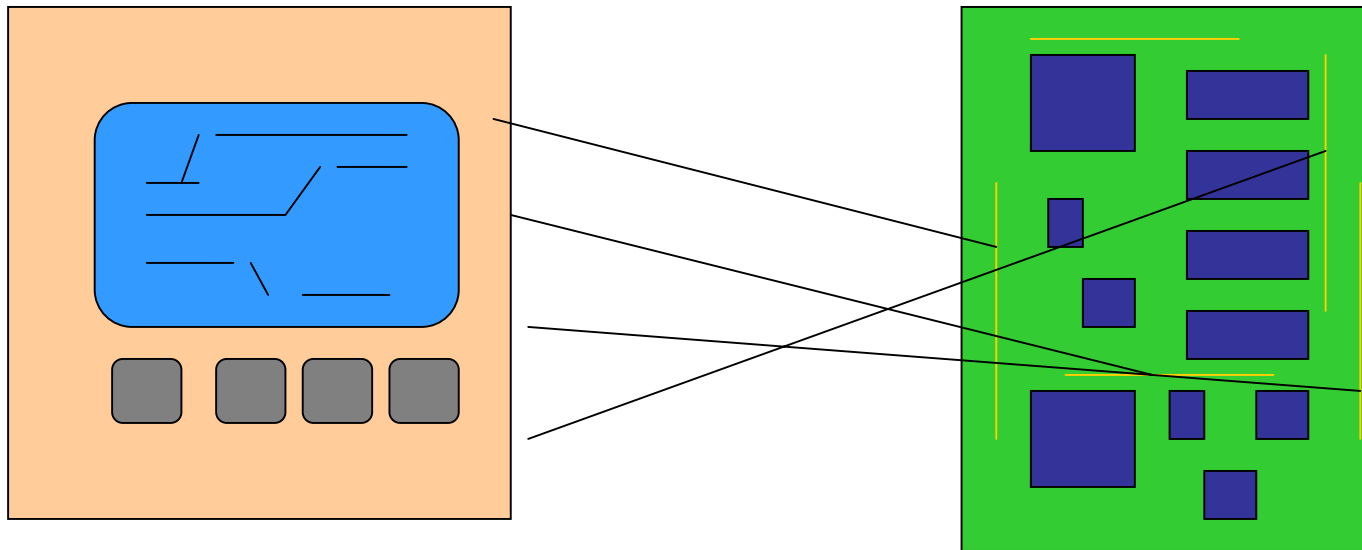


Oscilloscope

- Used to examine any electrical signal, analog or digital, on any piece of hardware
 - Example: voltage on a pin
- Restricted to about four inputs
- No triggering logic

Logic analyzer

- A logic analyzer is an array of low-grade oscilloscopes:



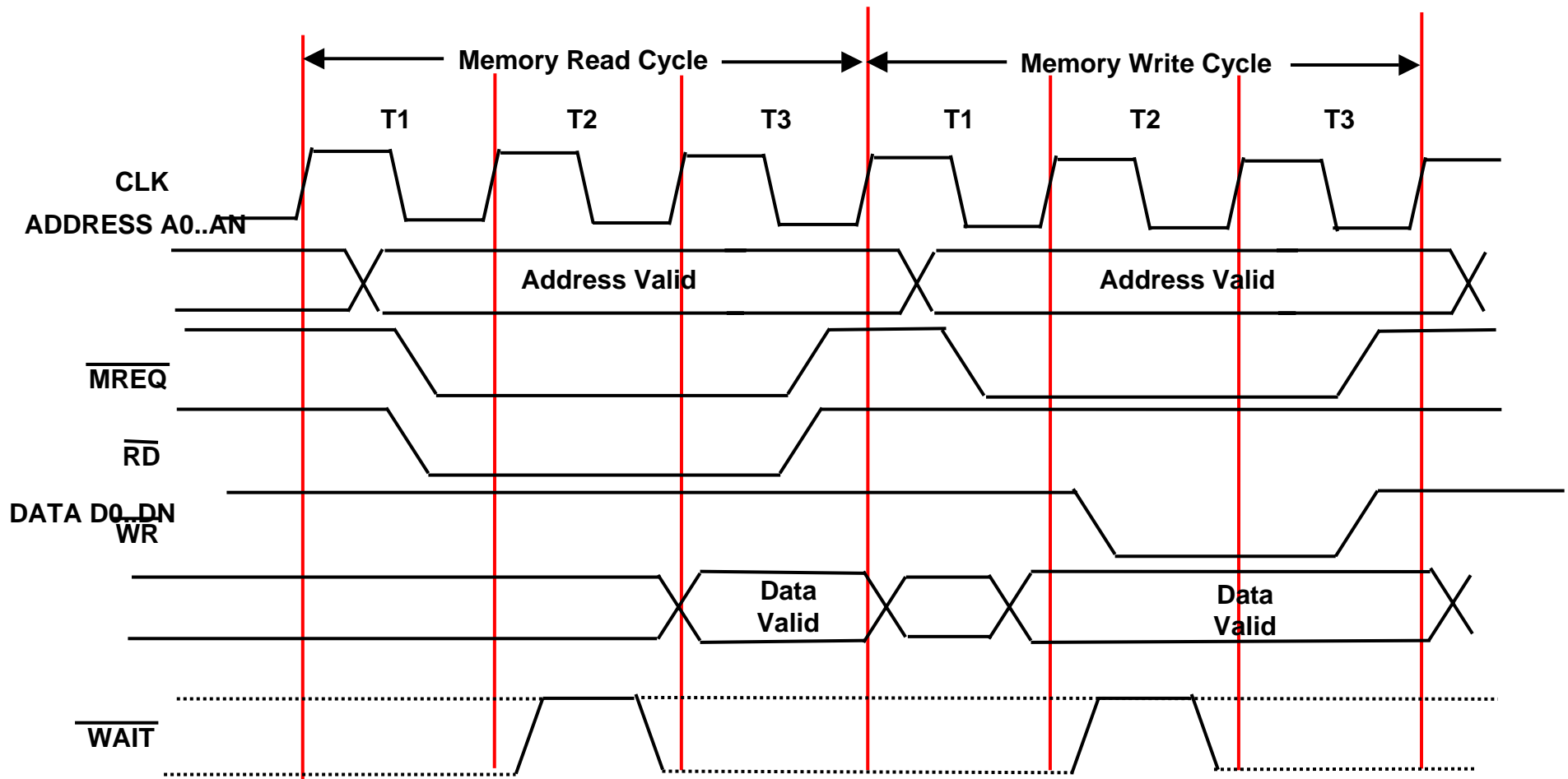


Logic Analyzer (Contd.)

- Checks whether electrical signal is at 1 or 0
- Multiple inputs
- Can display data in two modes:
 - State: Data capture synchronous to processor clk
 - Timing: Data capture synchronous to LA clk
- Triggering logic:
 - Display the values of input signals 1 through 4 only when inputs 5 and 6 are both zero

Example: Tracking bus activity

- Record state of the processor busses each clock cycle
- Post-processing software reduces trace to instructions





Logic Analyzer: Advantages

- Most prevalent tool used for digital system design
- Very powerful measurement & triggering capabilities
- Generic, easily customized with per-processor module
- Totally configurable to any current microprocessor
 - > 200 channels, > 1GHz data rates
- Can observe entire digital system at the same time
- Can be used in conjunction with **instrumented code** for real time measurements in cached processors



Logic Analyzer: Disadvantages

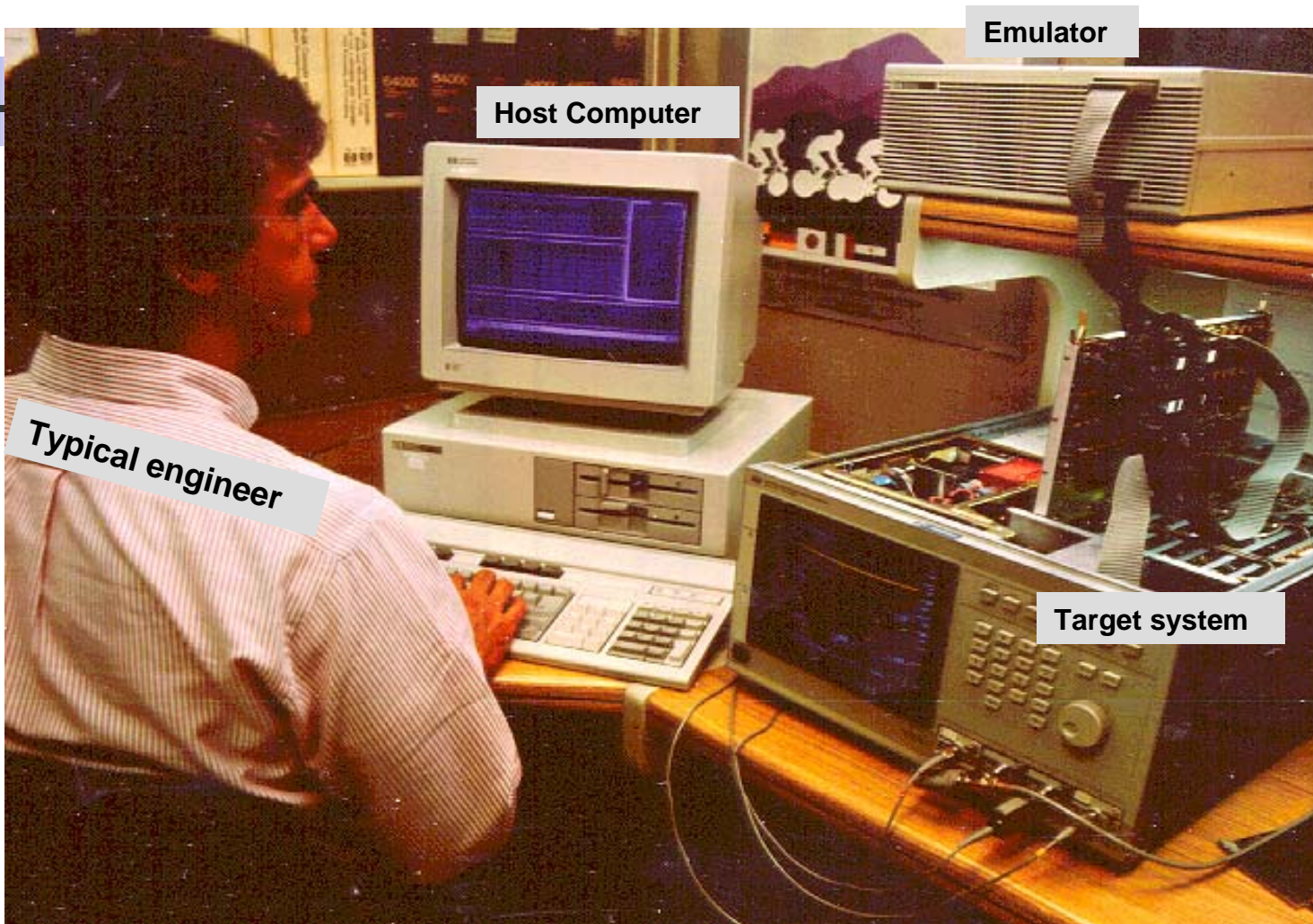
- Can be very expensive
- Strictly passive, cannot provide control of processor
- Complex
- Not well-integrated into software design environment
- Must combine with other tools for complete solution
- Cached processors cannot be observed
- Code must be instrumented if cache is not visible



In-circuit emulators (ICE)

- An ICE provides an integration of the most important functions required to debug embedded systems
 - Microprocessor run Control (Debug monitor)
 - Memory substitution (ROM Emulator)
 - Real Time Trace (Logic Analyzer)
- A microprocessor in-circuit emulator is a specially-instrumented microprocessor.

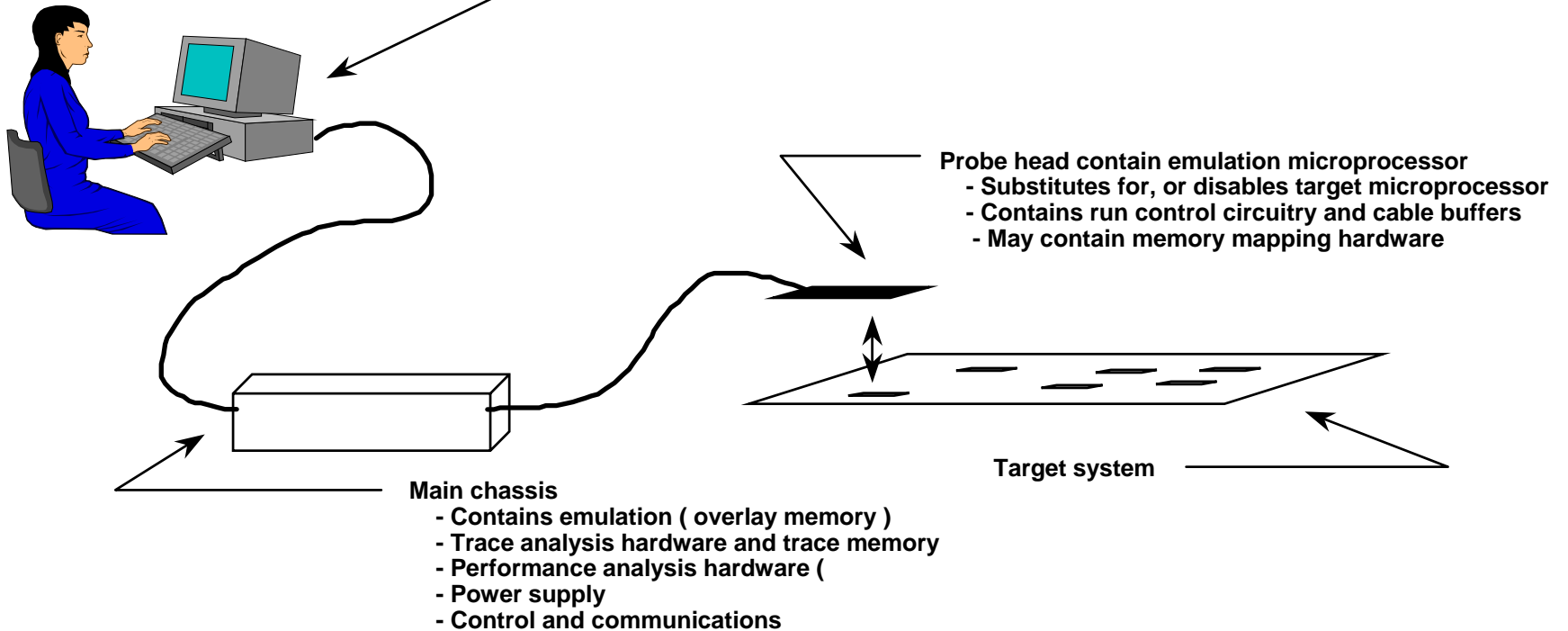
A typical engineer with emulator



ICE (Contd.)

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





ICE (Contd.)

- Run control
 - Similar to the functionality of a software debug kernel
 - Peek/poke memory
 - Modify registers
 - Single step
 - Set breakpoints
 - Disassemble memory code
 - Download code



ICE (contd.)

■ Overlay memory

- Similar to functionality of ROM emulator
- Provides overlay (substitution) memory for target system memory
- Entire memory space can be “mapped” to provide memory regions in anywhere in the address space of target processor
- Emulation memory interspersed with target system memory
- Emulation memory can be assigned special attributes
 - ROM (no writes allowed)
 - Guarded (break on write)
 - Code space or data space
 - Shared memory (simulated I/O)



ICE Advantages

- Premier tool for embedded system hw/sw integration
- Provides all the functionality needed for connectivity, observation and control of an embedded system
- Control of the microprocessor is guaranteed, independent of the state of target system hardware
- Overlay memory substitutes for the target system ROM to permit trace of code execution
- Traces program flow
 - Filter out extraneous bus activity (pre-fetches)
 - Special circuitry can display cache activity
- Tight integration through single user interface on host
- Can provide very unique and valuable measurements



ICE: Disadvantages

- **Not always most cost-effective solution**
 - **Can be extremely expensive (~ \$20K)**
 - **Costly to equip a team of designers**
 - **New emulator often required for new processor or upgrade**
- **Viewed as a complex and fragile instrument**
- **Availability usually lags early silicon**
- **Cannot easily track microcontroller variants**
- **Connectivity to surface-mounted processors is very difficult, and the problem is getting worse**
- **Viewing cached program activity can be very difficult (or, at worst) impossible**
- **Not the tool of choice for code design at the upper levels of abstraction**

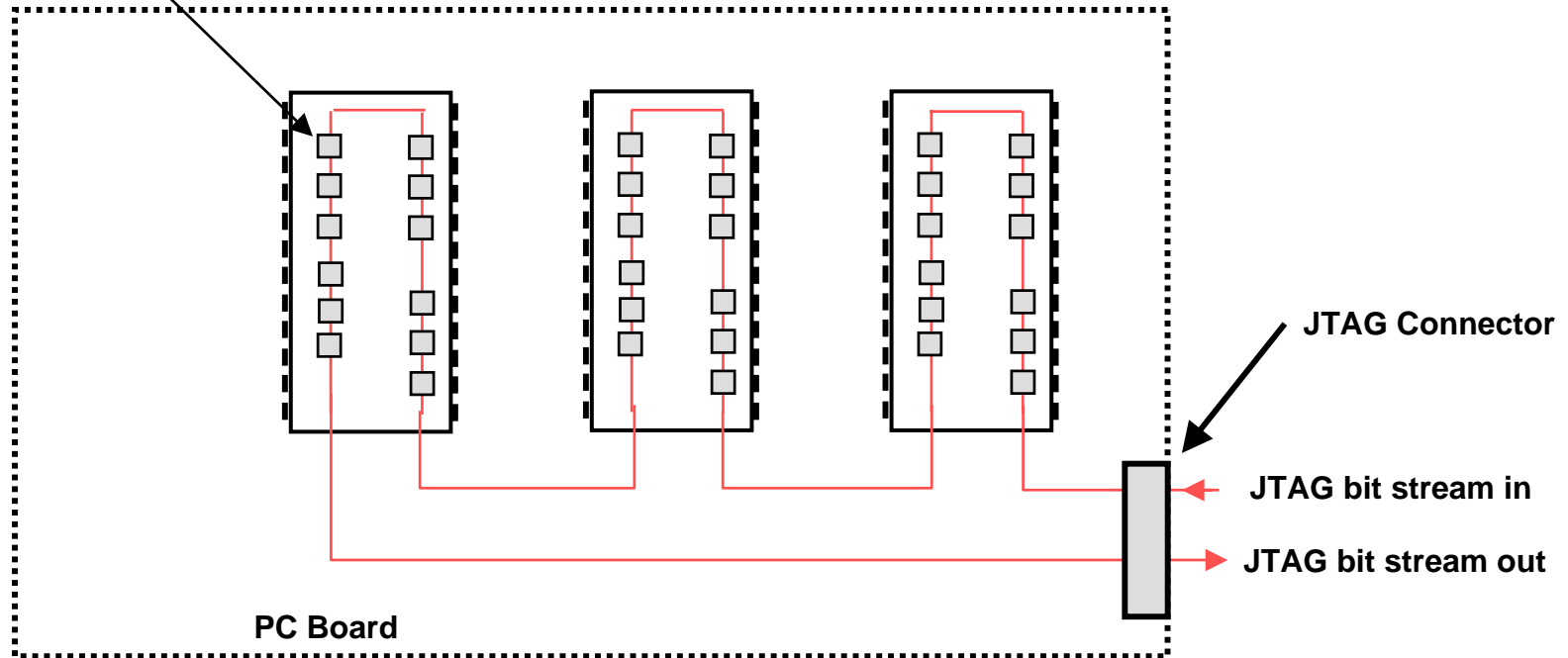


Joint Test Action Group (JTAG)

- Originally developed as a board test methodology to replace “bed of nails” testers
 - PC boards tested on complex machines with dense array of point contacts (bed of nails) to connect to every node on the board
 - Node is a shared interconnection between board components
- Link outputs of all digital devices on PC board with individual bits of one long shift register
- Can observe whether any circuit node is stuck high, low, shorted or disconnected

JTAG Loop

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



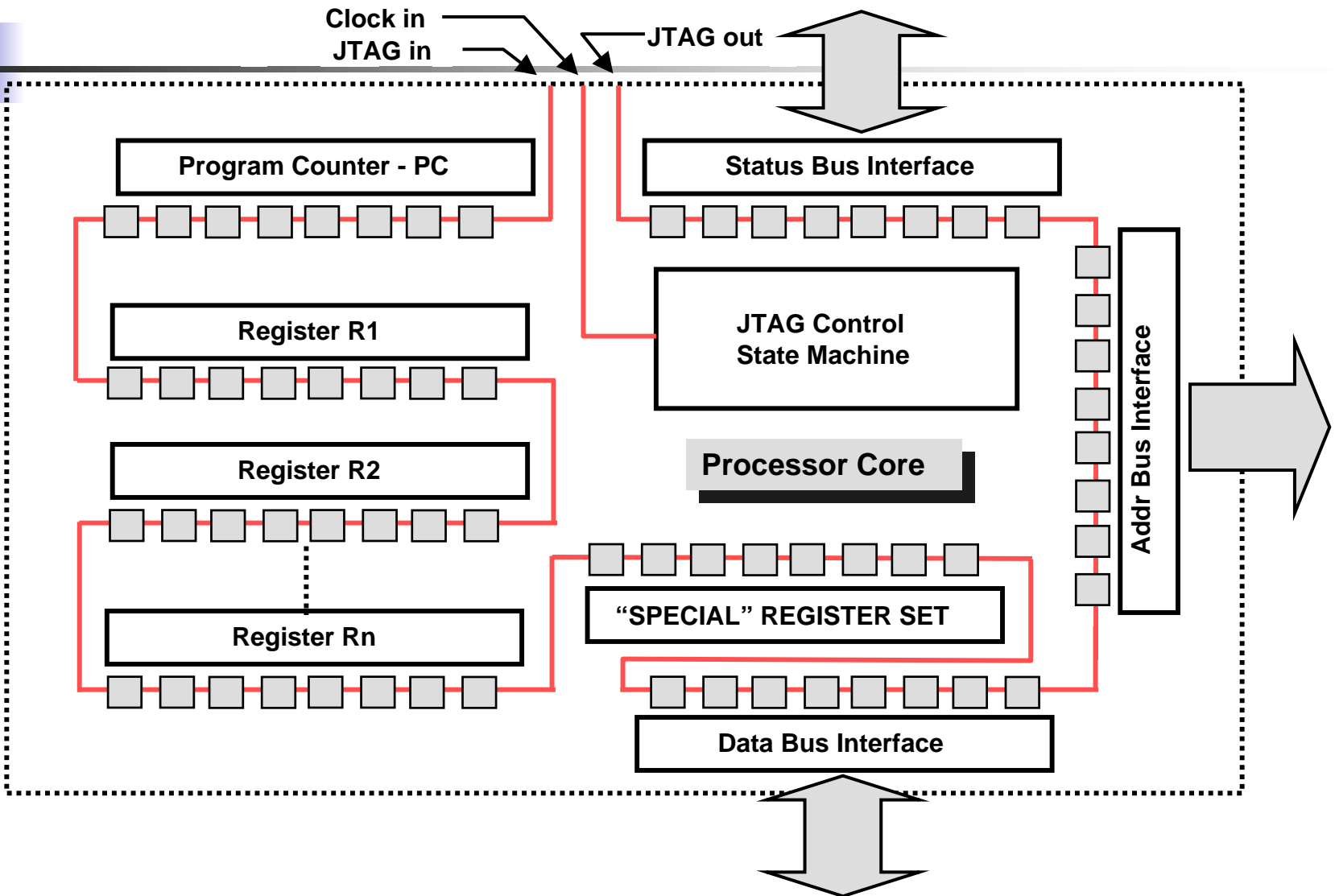
Bit stream forms one long shift-register



JTAG can be a debug protocol

- Link all the internal registers and other important circuit blocks in a JTAG loop
- Provide capability to drive output as well as sniff it
- Add internal debug registers accessible via JTAG
- Can provide a low-cost standard debug interface, independent of chip variations
- Great for silicon vendors and end users
 - One low-cost (<\$100 tool) supports any JTAG-compatible device

JTAG-based debug kernel





More on JTAG

- De-facto standard for on-chip debug
- Very cost effective
 - Can be re-programmed to support multiple processor families
 - Typically priced under \$1000
- Debug capability is defined by features of debug core
- Can be slow
 - Bus cycles must be constructed by laboriously shifting every bit
 - Loops can be extremely long (~10,000 bits)
- Method of choice for processors in ASIC cores
- Lends itself to RISC processors because core logic tends to be simpler than CISC
- Design-intensive as JTAG loops must be hand crafted

- 
-
- 有时间的话
 - 再看一个Project例子
 - 请参照例子准备Project交流课件