Arm 校企协同育人

Arm SoC设计与新工科贯穿式课程体系建设

目录

1. ARM Processors

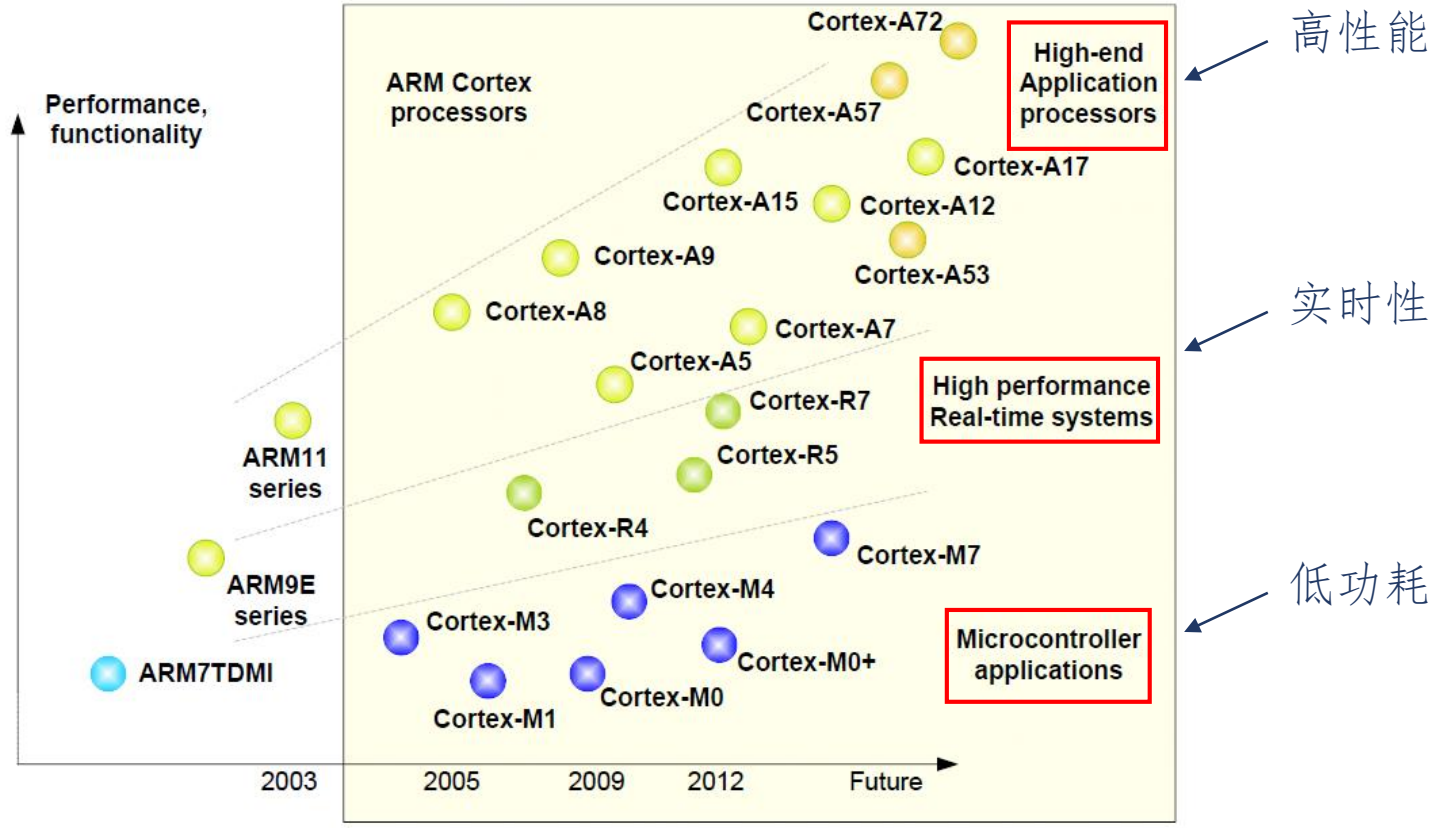


Figure 1.3
Overview of the ARM processor family.

《The Definitive Guide to ARM Cortex M0 and Cortex M0+ Processors》 Joseph Yiu

1. ARM Processors

- Cortex-A 处理器
- 处理高端操作系统(iOS, Android, Linux, Windows)
- 较长流水线，可以运行在相对较高的时钟频率下(超过1GHz)
- 具有高级OS的虚拟存储器寻址操作所需的存储器管理单元(MMU)
- 可选的增强Java支持和名为TrustZone的安全程序执行环境

- 缺点：无法对硬件事件作出快速反应(即实时性需求)

1. ARM Processors

- Cortex-R 处理器
- 实时、高性能，尤其适合数据分析
- 可以运行在较高时钟频率下(500MHz - 1GHz)
- 可以对硬件事件作出快速反应

- 缺点：设计复杂、功耗高

1. ARM Processors

- Cortex-M 处理器
- 较短流水线
- 功耗低
- 工作时钟频率较低
- 中断管理功能强大且易于使用(NVIC)

1. ARM Cortex M系列

- M0: 最小的ARM处理器，功耗低但能耗效率高
- M0+: 大小和M0类似，但系统级调试特性更多
- M1: 和M0类似，但具有专用于FPGA的存储器系统特性
- M3: 性能比M0高很多，哈佛总线架构
- M4: 支持M3所有特性，且可以添加浮点单元(FPU)
- M7: 高性能处理器，同时支持单精度和双精度浮点运算

- 实验选择ARM DesignStart计划开放的Cortex M0来搭建SoC

2. ARM DesignStart

- ARM DesignStart计划开放多个不同版本的处理器核，主要分为

1. Eval版：提供网表形式的verilog代码
2. FPGA版：提供针对FPGA优化的IP
3. Pro版：提供RTL级verilog代码
4. Physic版：提供物理实现的IP

其中Eval版和FPGA版可以免费申请

本次实验使用Eval版的Cortex M0一步步搭建出一个完整的SoC

2. ARM DesignStart

- 通过ARM DesignStart计划，我们可以做

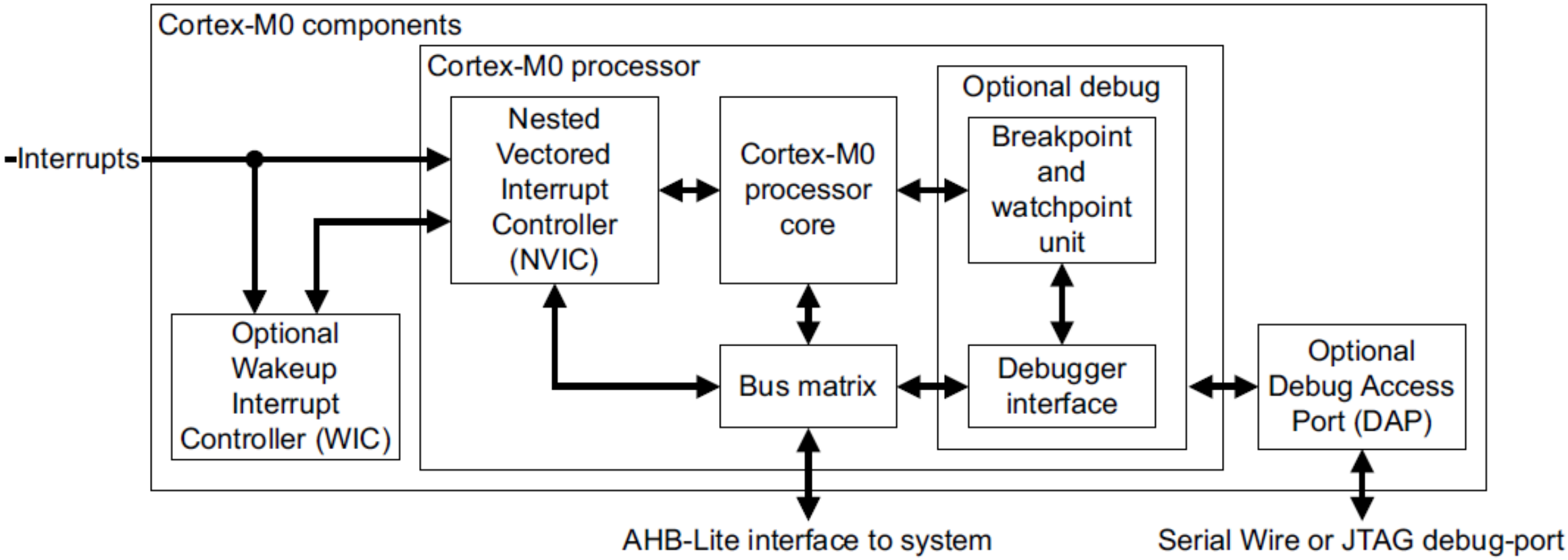
1. 基于Cortex M0搭建SoC (参考《Cortex-M0 Based SoC Building Quick Tutorial》, 赵天津编著)
2. 在SoC系统上进行软件开发

通过开展相关实验，可以很好地学习SoC设计及嵌入式软件编程

3. ARM Cortex M0

- 基于ARMv6-M架构(参考ARMv6-M Architecture Reference Manual)
- 三级流水线
- 基于Thumb指令集架构，使用Thumb ISA的一个子集
- 32位寻址空间
- 总线接口 AHB-Lite(参考AMBA 3 AHB Lite Protocol Specification)
- 具有嵌套向量中断控制器(NVIC)

3. ARM Cortex M0 Arch



CPU提供了中断向量端口、AHB-lite端口以及DAP端口

3. ARM Cortex M0 中断

- CPU中断处理过程：

1. CPU接收到中断信号（IRQ、NMI、Systick等等）
2. 将R0,R1,R2,R3,R12,LR,PC,xPSR寄存器入栈
3. 根据中断信号查找中断向量表（对应汇编启动代码中的__Vector段），跳转至中断处理函数
4. 中断处理函数执行完成后，利用链接寄存器返回，寄存器出栈，PC跳转

3. ARM Cortex M0 中断向量表

Exception number[a]	IRQ number[a]	Exception type	Priority	Vector address[b]	Activation
1	-	Reset	-3, the highest	0x00000004	Asynchronous
2	-14	NMI	-2	0x00000008	Asynchronous
3	-13	HardFault	-1	0x0000000C	Synchronous
4-10	-	Reserved	-	-	-
11	-5	SVCall	Configurable[e]	0x0000002C	Synchronous
12-13	-	Reserved	-	-	-
14	-2	PendSV	Configurable[e]	0x00000038	Asynchronous
15	-1	SysTick[c]	Configurable[e]	0x0000003C	Asynchronous
15	-	Reserved	-	-	-
16 and above[d]	0 and above	IRQ	Configurable[e]	0x00000040 and above[f]	Asynchronous

3. ARM Cortex M0 启动流程

1. 在复位使能时，CPU处于Reset异常状态；
2. 复位释放后，从地址0x00000000出加载栈顶地址，及汇编代码中__Vector的第一行则为栈顶地址；
3. 从地址0x00000004初加载复位处理函数的地址；
4. PC改变为0x00000004中的值，开始执行复位处理，同时CPU的工作状态从异常模式切换为线程模式，开始正常工作。

3. ARM Cortex M0 端口配置

- 在ARM DesignStart网址下载的Cortex-M0 DesignStart Eval文件资源中找到名为cortexm0ds_logic.v的文件，这便是处理器核的网表形式的Verilog代码。在实验开始前，我们需要对处理器核的时钟、复位、无用端口以及DAP的iobuf进行配置。
- 代码不可读，相关文档对核端口说明很少

- **根据DesignStart资料包中提供的参考设计理解各端口意义**

3. ARM Cortex M0 端口配置

- System Input端口

```
75          // System inputs
76          .FCLK           (clk),          //FREE running clock
77          .SCLK           (clk),          //system clock
78          .HCLK           (clk),          //AHB clock
79          .DCLK           (clk),          //Debug clock
80          .PORESETn       (RSTn),         //Power on reset
81          .HRESETn        (cpuresetn),    //AHB and System reset
82          .DBGRESETn      (RSTn),         //Debug Reset
83          .RSTBYPASS      (1'b0),         //Reset bypass
84          .SE             (1'b0),         // dummy scan enable port for synthesis
```

处理器核所需的各种时钟与复位信号，时钟与复位作为触发器的基本输入，设置错误将会直接导致整个系统的崩溃

3. ARM Cortex M0 端口配置

- Power management input/output端口

```
60   wire CDBGPWRUPREQ;
61   reg CDBGPWRUPACK;
62
63   always @(posedge clk or negedge RSTn)begin
64           if (~RSTn) CDBGPWRUPACK <= 1'b0;
65           else CDBGPWRUPACK <= CDBGPWRUPREQ;
66   end
86          // Power management inputs
87          .SLEEPHOLDREQn  (1'b1),          // Sleep extension request from PMU
88          .WICENREQ       (1'b0),          // WIC enable request from PMU
89          .CDBGPWRUPACK   (CDBGPWRUPACK),  // Debug Power Up ACK from PMU
90
91          // Power management outputs
92          .CDBGPWRUPREQ   (CDBGPWRUPREQ),
93          .SYSRESETREQ    (SYSRESETREQ),
```

处理器核功耗管理单元相关，将输出端口的CDBGPWRUPREQ信号经过同步后接在输入端口CDBGPWRUPACK处

3. ARM Cortex M0 端口配置

- System Bus端口

```
95          // System bus
96          .HADDR          (HADDR[31:0]),
97          .HTRANS         (HTRANS[1:0]),
98          .HSIZE          (HSIZE[2:0]),
99          .HBURST         (HBURST[2:0]),
100         .HPROT          (HPROT[3:0]),
101         .HMASTER        (HMASTER),
102         .HMASTLOCK      (HMASTLOCK),
103         .HWRITE         (HWRITE),
104         .HWDATA         (HWDATA[31:0]),
105         .HRDATA         (HRDATA[31:0]),
106         .HREADY         (HREADY),
107         .HRESP          (HRESP),
```

总线作为SoC的一个重要模块，负责几乎所有数据的传输，接下来将会利用此总线接口构建整个SoC

3. ARM Cortex M0 端口配置

- Interrupt端口

```
109         // Interrupts
110         .IRQ              (IRQ),              //Interrupt
111         .NMI              (1'b0),             //Watch dog interrupt
112         .IRQLATENCY       (8'h0),
113         .ECOREVNUM        (28'h0),
```

处理器核提供了NMI和IRQ两种中断，处理器在处理这两种中断的优先级不同，在实验三详细介绍

3. ARM Cortex M0 端口配置

- Systick端口

```
115         // Systick
116         .STCLKEN        (1'b0),
117         .STCALIB        (26'h0),
```

Systick是一个独立计数器，由于由于处理器核在工作时，指令与时间的相关性并不大，因此在处理一些依赖于时间的工作时利用指令实现计数往往是不可靠的，因此Systick作为独立计数模块，能够在完成预设计数后向处理器核提供中断，通过调用中断处理函数来及时处理相关指令。

3. ARM Cortex M0 端口配置

- Debug端口

```
12   wire SWDO;
13   wire SWDOEN;
14   wire SWDI;
15
16   assign SWDI = SWDIO;
17   assign SWDIO = (SWDOEN) ?  SWDO : 1'bz;
```

将SWDIO扩展为多个单向信号类型，对应核端口中的SWDI, SWDO, SWDOEN三个信号

```
119          // Debug - JTAG or Serial wire
120          // Inputs
121          .nTRST          (1'b1),
122          .SWDITMS        (SWDI),
123          .SWCLKTCK       (SWCLK),
124          .TDI            (1'b0),
125          // Outputs
126          .SWDO           (SWDO),
127          .SWDOEN         (SWDOEN),
128
129          .DBGRESTART     (1'b0),
130
131          // Event communication
132          .RXEV           (RXEV),         // Generate event when a DMA operation completed.
133          .EDBGRQ         (1'b0)          // multi-core synchronous halt request
```

使用CMSIS-DAP作为调试器，需要用到SW(Serial Wire)调试工具，数据通道(SWDIO)为inout类型

3. ARM Cortex M0 关键信号说明

信号名	描述
HXXXX(以H开头的信号)	总线相关，下一节将会详细介绍
vis_rX_o	通用寄存器，通过仿真观察这些寄存器能够知道相关汇编指令是否正常执行（ADD、MOV等）
vis_msp_o	栈指针
vis_r14_o	连接寄存器，程序返回出错时需要检查此寄存器
vis_pc_o	程序计数器，可以用来判断处理器是否在正常运行
vis_ipsr_o	中断状态寄存器，当发生错误的时候，PSR的值会改变，用于判断发生何种类型错误
LOCKUP	处理器处理NMI或者HardFault时又发生错误后，处理器将会被死锁

以上信号对于理解处理器的运行过程较为重要，在后续实验中需要重视它们随系统运行状态的变化

4. AMBA3 AHB-Lite

- AMBA3中的AHB(Advanced High-performance Bus)，高性能总线

1. 可以实现高性能的同步设计
2. 支持多个总线主设备(参考《Multi-layer AHB Overview》)
3. 提供高带宽操作

AHB-Lite是AHB的子集，简化了AHB总线的设计，只有一个主设备

4. AMBA3 AHB-Lite概述

- 对于AHB-Lite，包含数据总线、地址总线和额外的控制信号

1. 数据总线用于交换数据信息
2. 地址总线用于选择一个外设，或者一个外设中的某个寄存器
3. 控制信号用于同步和识别tradeoff

4. AMBA3 AHB-Lite信号说明

名称	来源	描述
HADDR[31:0]	Master	传输地址
HBURST[2:0]	Master	Burst类型
HSIZE[2:0]	Master	数据宽度 00：8bit Byte 01：16bit Halfword 10：32bit Word
HTRANS[1:0]	Master	传输类型 00：IDLE，无操作 01：BUSY 10：NONSEQ，主要的传输方式 11：SEQ
HWDATA[31:0]	Master	核发出的写数据
HWRITE	Master	读写选择（1：写，0：读）
HRDATA[31:0]	Slave	外设返回的读数据
HREADOUT	Slave	何时传输完成（通常为1）
HRESP	Slave	传输是否成功（通常为0）

4. AMBA3 AHB-Lite传输

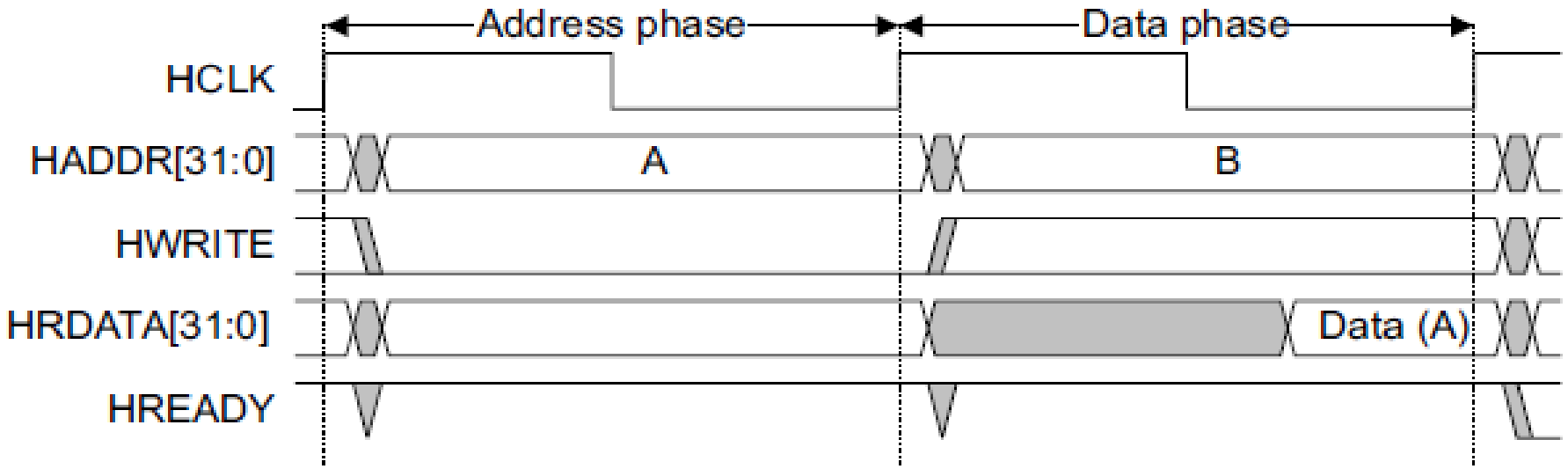
HTRANS[1:0]	Master	传输类型 00：IDLE，无操作 01：BUSY 10：NONSEQ，主要的传输方式 11：SEQ

```
1724    assign HPROT[1] = 1'b1;
1725    assign HBURST[2] = 1'b0;
1726    assign HBURST[1] = 1'b0;
1727    assign HBURST[0] = 1'b0;
1728    assign HMASTLOCK = 1'b0;
1729    assign HSIZE[2] = 1'b0;
1730    assign HTRANS[0] = 1'b0;
```

HTRANS[0]恒为0
M0只支持NONSEQ传输

4. AMBA3 AHB-Lite NONSEQ时序

- 基本读操作



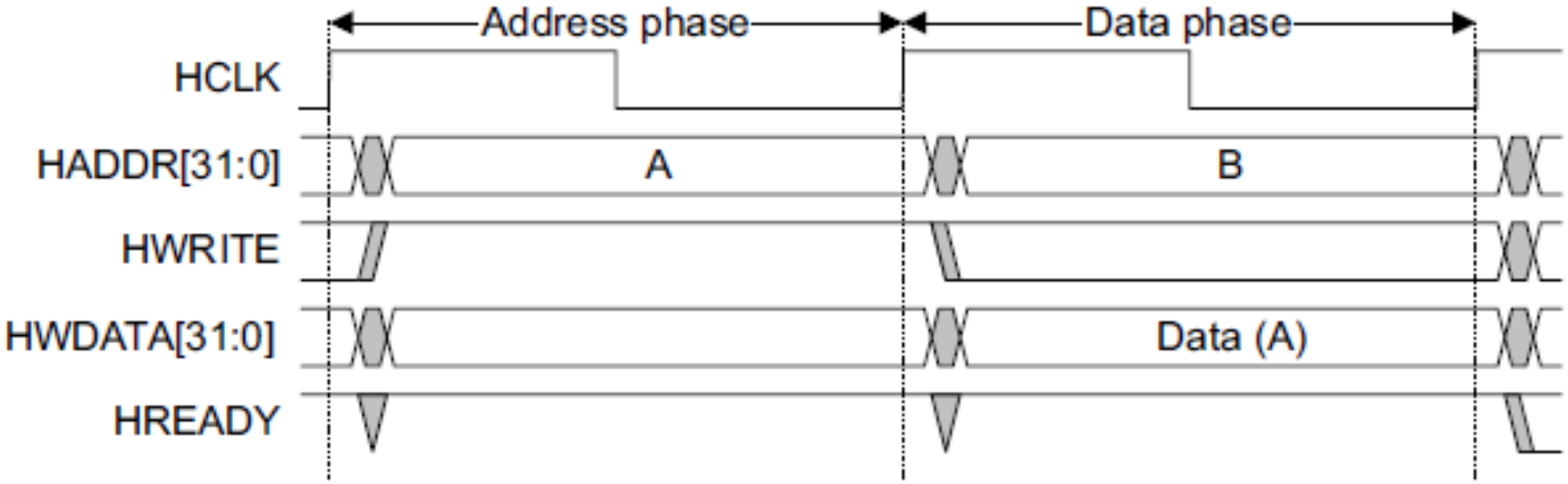
Master需要从外设读取数据时，总共需要经历两个阶段：Address phase & Data phase

Address phase：Master把地址输出在地址总线上，直到HREADY为1后进入Data phase

Data phase：Master会在HREADY为1时读取数据总线HRDATA上的数据，至此传输完成

4. AMBA3 AHB-Lite NONSEQ时序

- 基本写操作



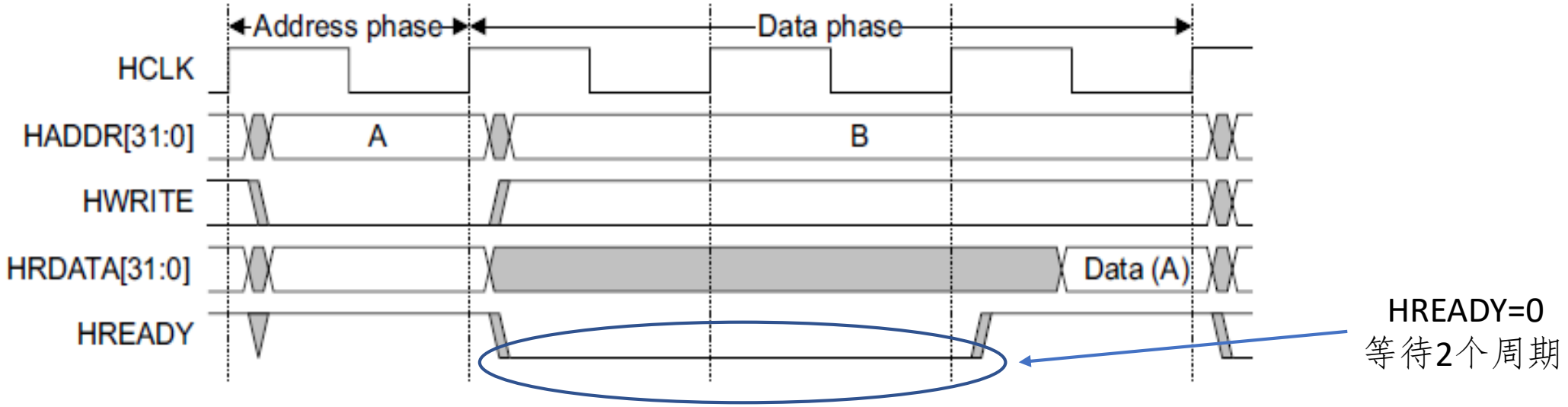
Master需要从外设读取数据时，总共需要经历两个阶段：Address phase & Data phase

Address phase：Master把地址输出在地址总线上，直到HREADY为1后进入Data phase

Data phase：Master把写数据放在数据总线HWDATA上，直到HREADY为1时传输完成

4. AMBA3 AHB-Lite NONSEQ时序

- 具有等待的读操作



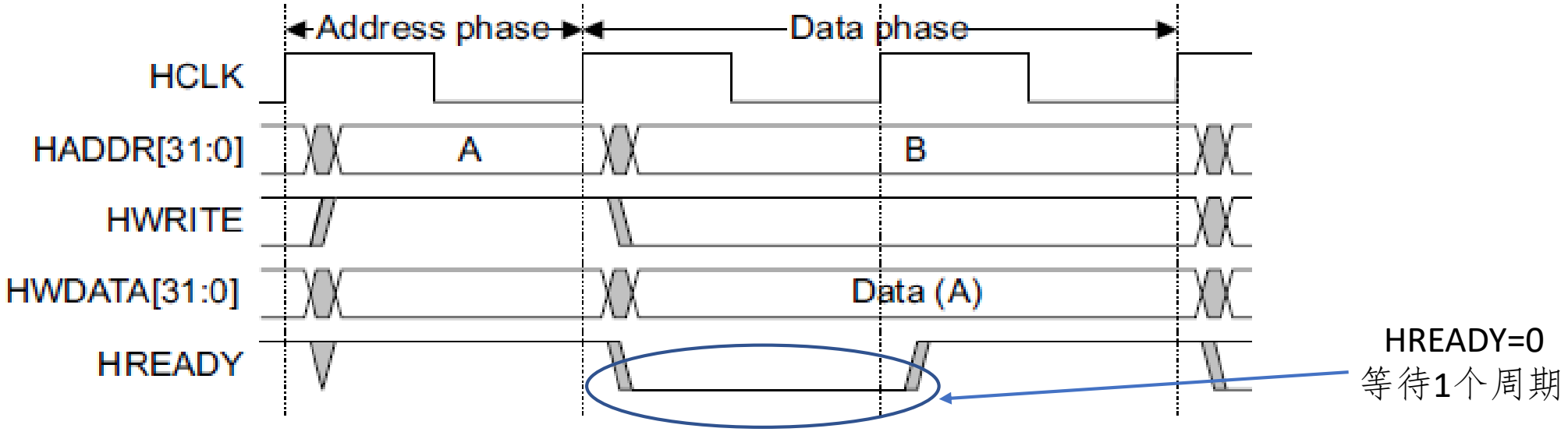
Master需要从外设读取数据时，总共需要经历两个阶段：Address phase & Data phase

Address phase：Master把地址输出在地址总线上，直到HREADY为1后进入Data phase

Data phase：Master会在HREADY为1时读取数据总线HRDATA上的数据，至此传输完成

4. AMBA3 AHB-Lite NONSEQ时序

- 具有等待的写操作



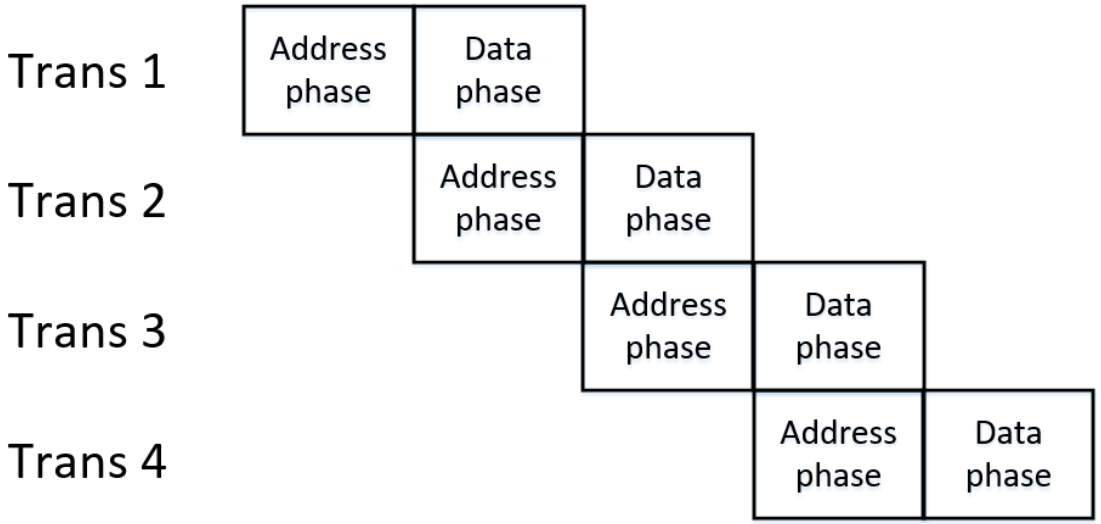
Master需要从外设读取数据时，总共需要经历两个阶段：Address phase & Data phase

Address phase：Master把地址输出在地址总线上，直到HREADY为1后进入Data phase

Data phase：Master把写数据放在数据总线HWDATA上，直到HREADY为1时传输完成

4. AMBA3 AHB-Lite 流水线传输

- 总线流水



在基本读写操作中，把A与B看作两次传输，A的data phase同时也作为B的address phase
实现上图所示的流水线传输

4. AMBA3 AHB-Lite Arch

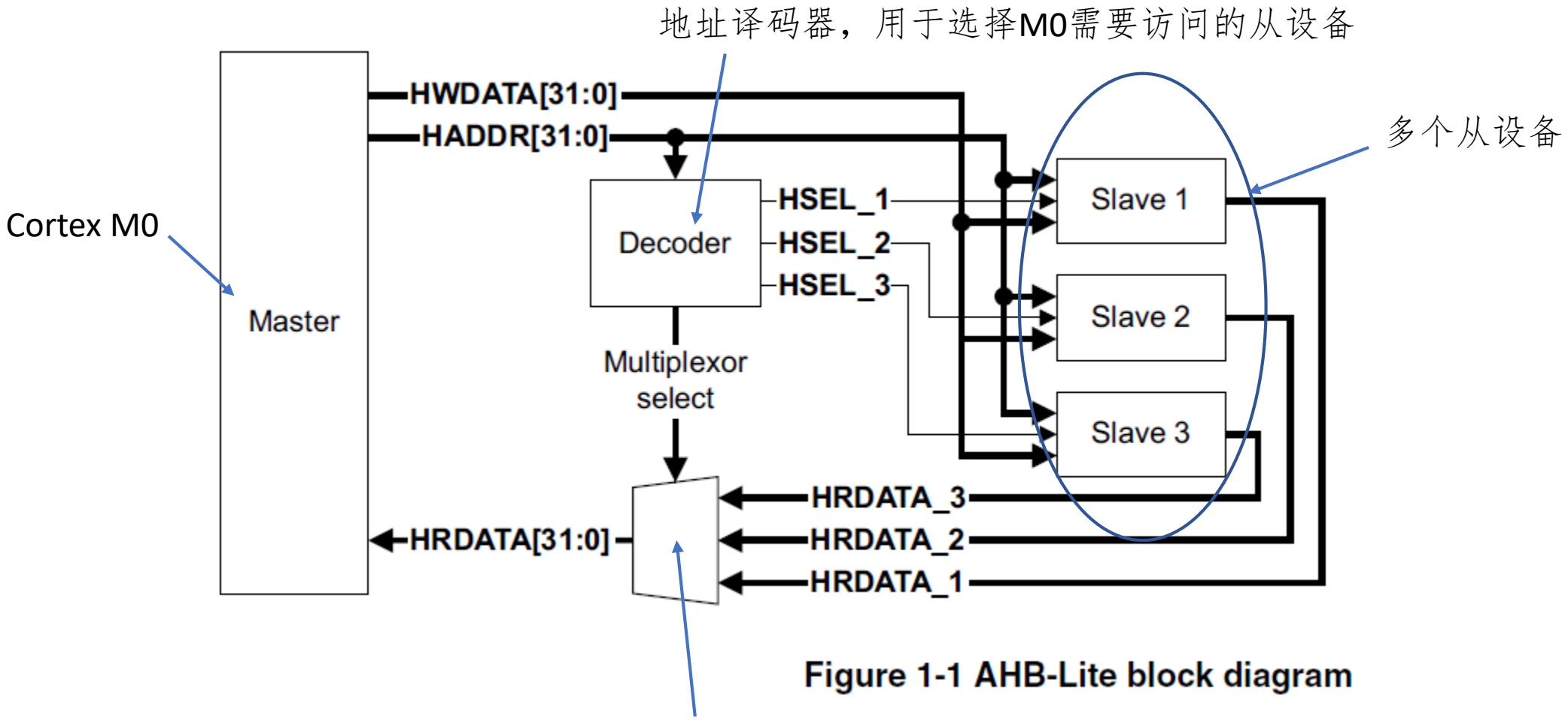
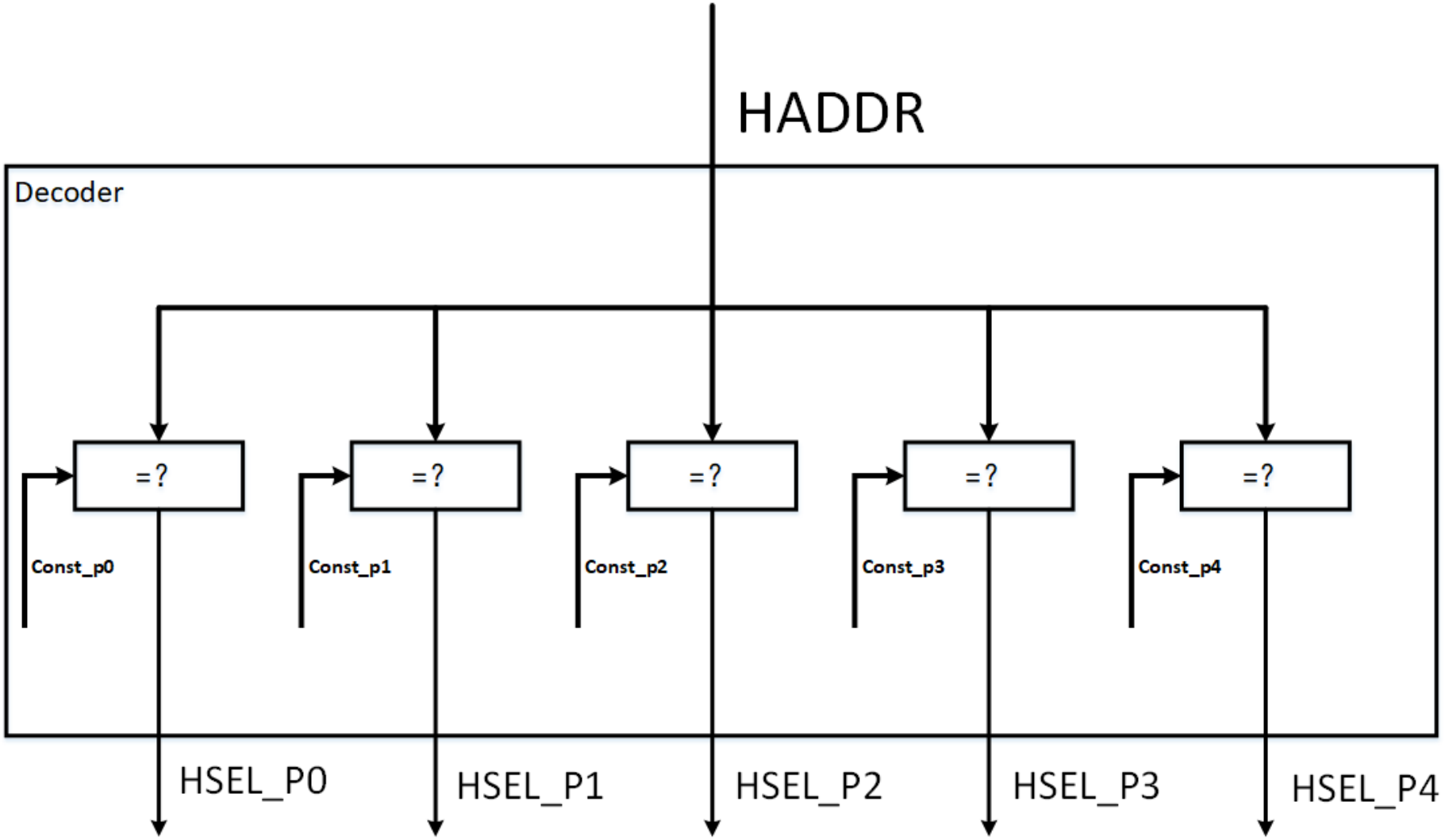
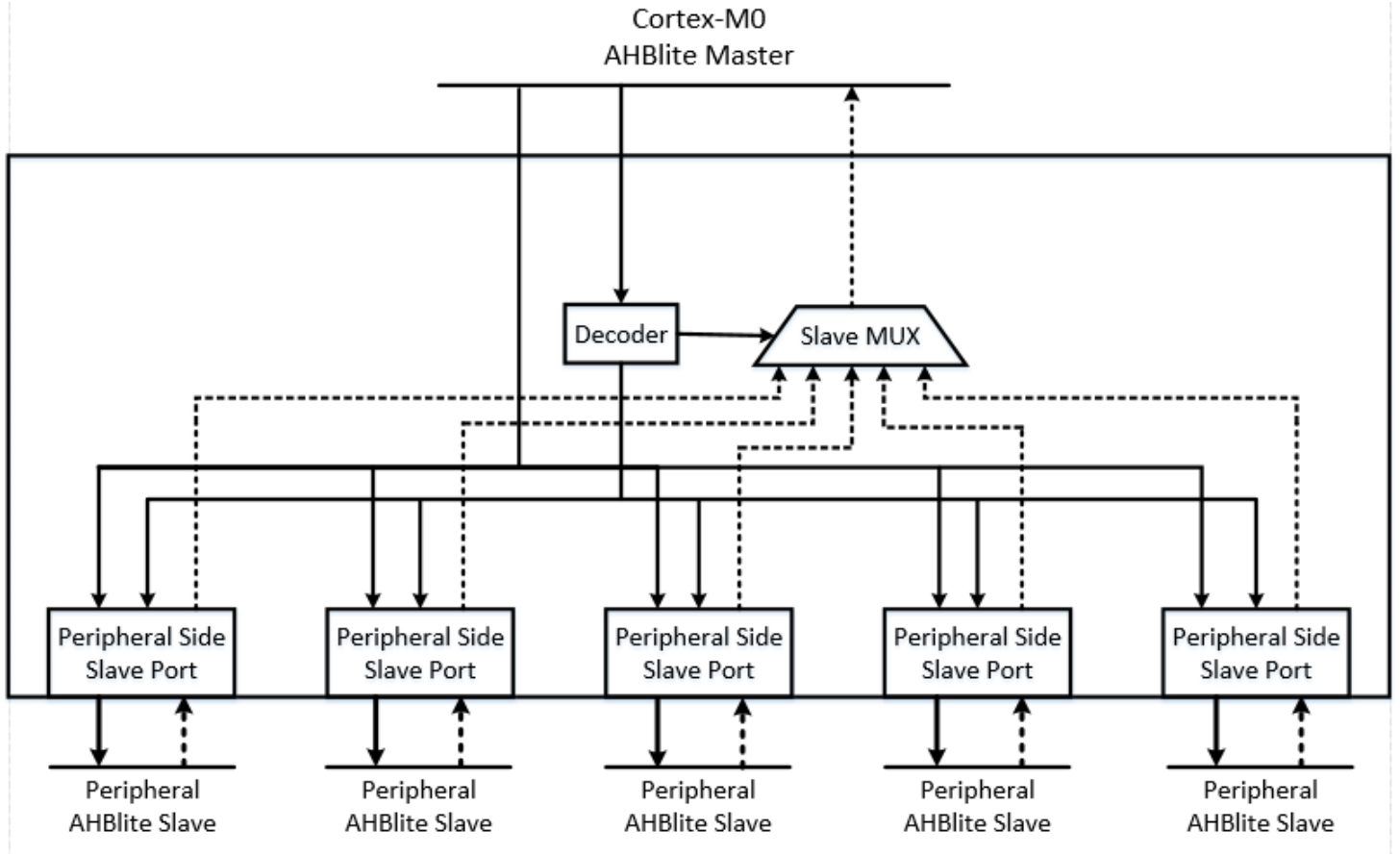


Figure 1-1 AHB-Lite block diagram

4. AMBA3 AHB-Lite 总线互连



4. AMBA3 AHB-Lite 总线互连

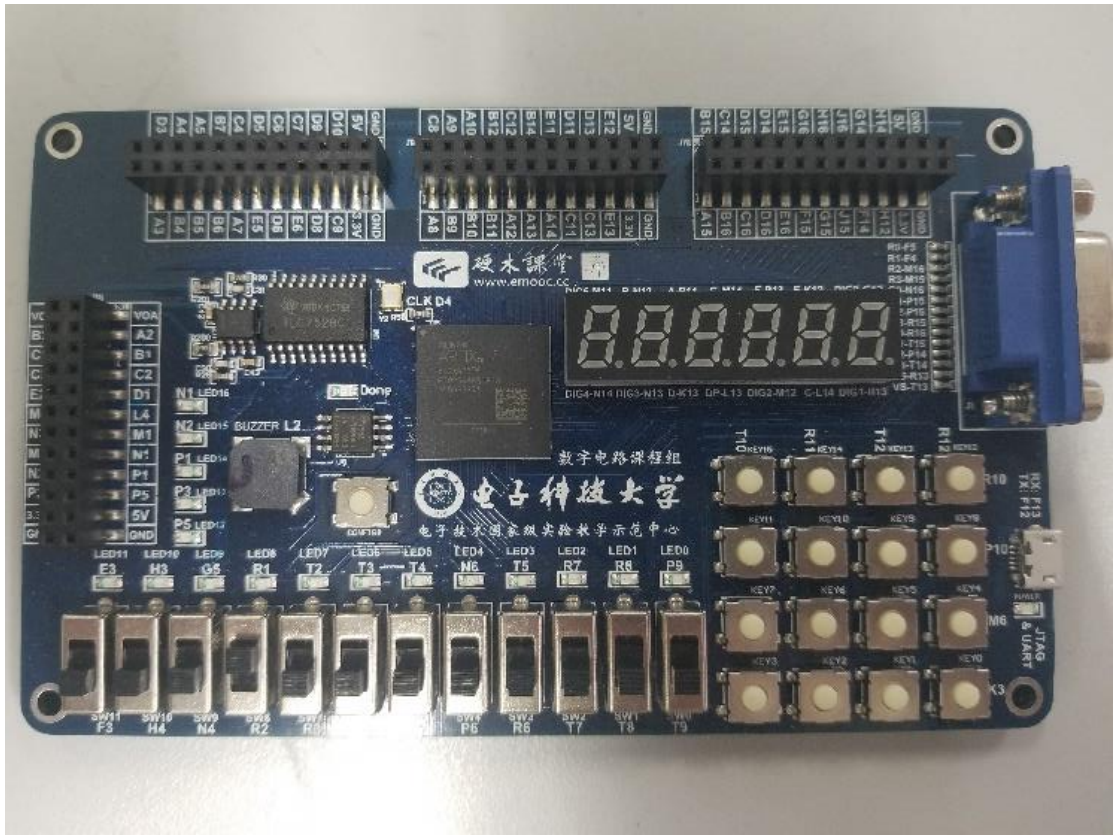


4. AMBA3 AHB-Lite 扩展外设

1. 扩展Decoder
2. 在Interconnect中增加Slave Port
3. 在Interconnect中将新增的Slave Port与扩展的Decoder HSEL信号以及Slave MUX连接
4. 顶层连接外设、外设接口、总线
5. C语言中编写外设接口结构体

5. SoC based on CM0

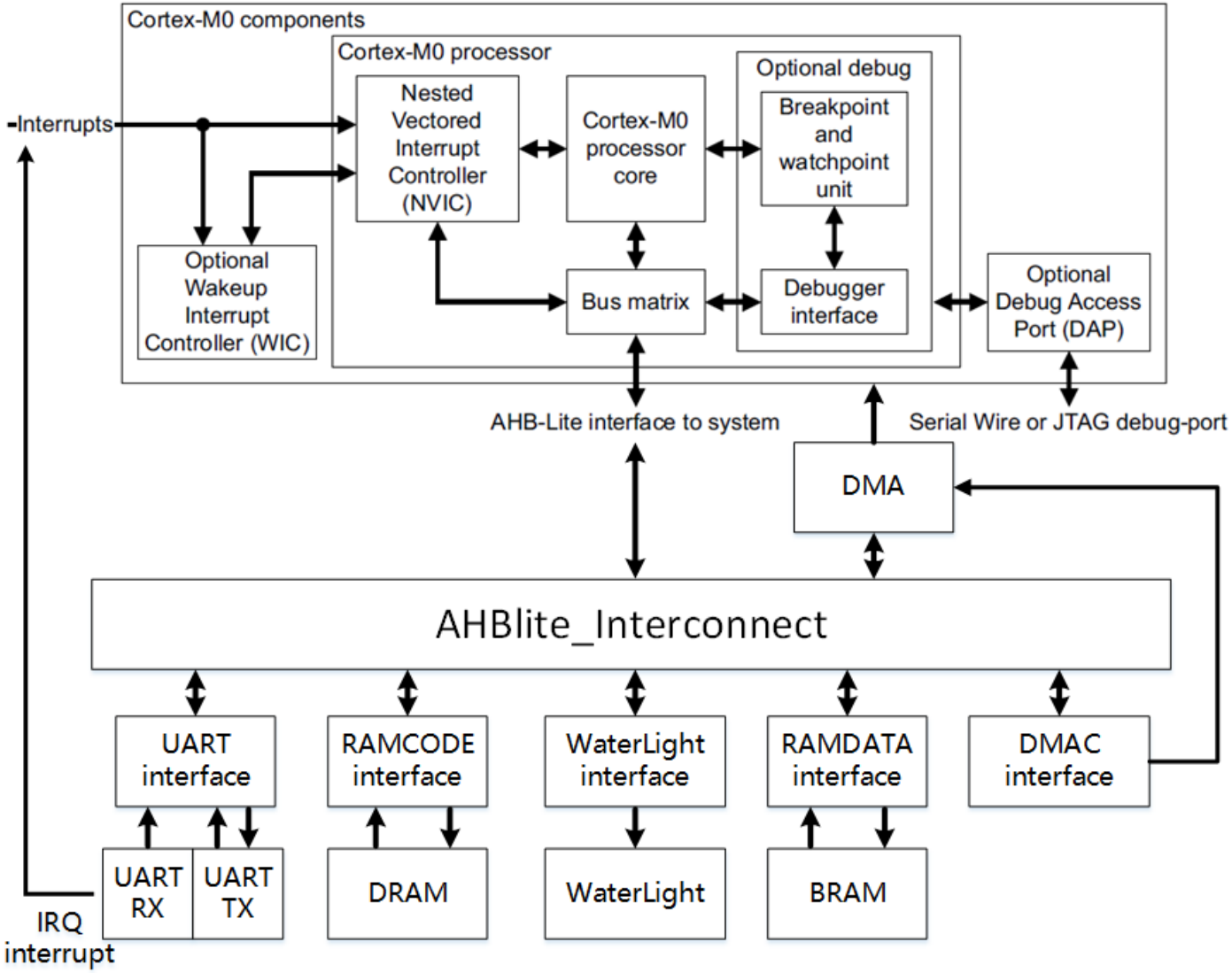
- 硬件平台



Xilinx Arty7 35t
硬木课堂

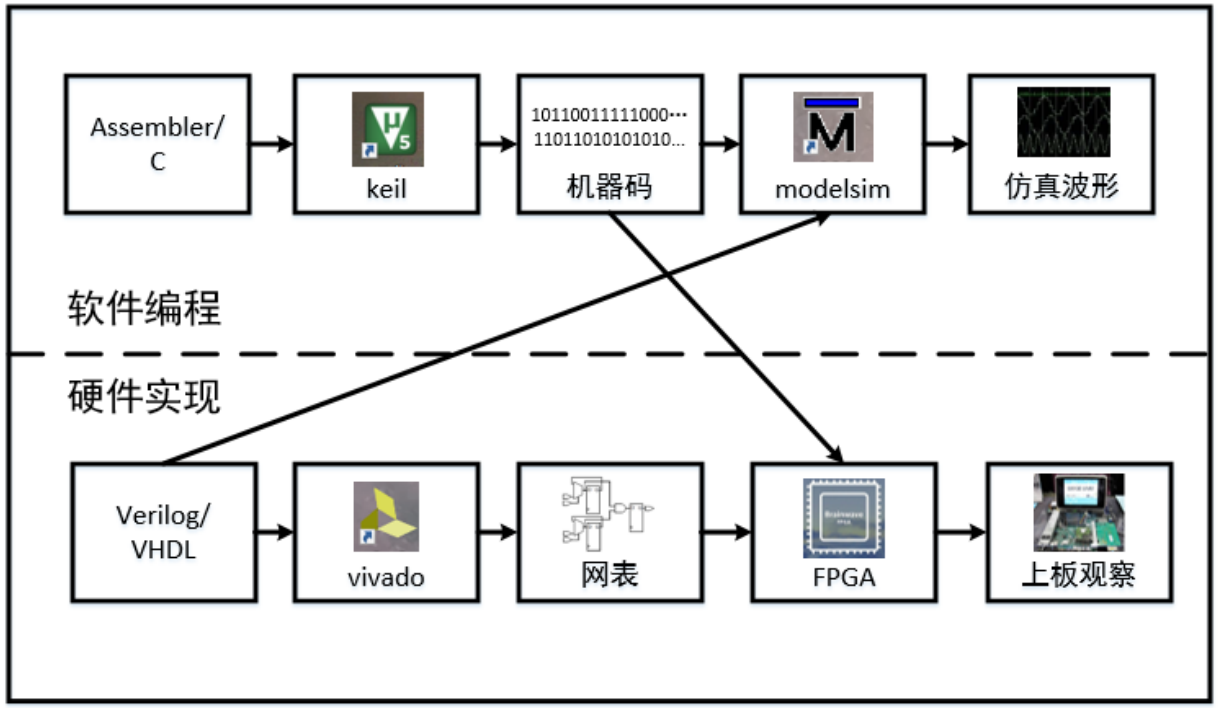
5. SoC based on CM0

- 总线互联



5. SoC based on CM0

- 设计流程



5. SoC based on CM0

- 请参考实验指导书完成实验一《总线、RAM、汇编与调试》
- 设计思路

1. 对Cortex M0核端口信号进行配置
2. 实现总线扩展模块
3. 设计RAM存储指令
4. 编写代码进行测试，观察SoC是否正常工作

由于大部分模块已经实现完成，只需要按实验指导书的提示修改部分代码即可

5. SoC based on CM0

- 添加外设的流程

1. 完成外设模块的实现
2. 修改总线地址译码器
3. 实现总线和外设之间的接口
4. 在顶层模块中实现总线互联

6. Some other experiments

- 请参考实验指导书完成实验二《数据存储器与流水灯》
- 设计思路

1. 设计数据存储器存储数据
2. 设计流水灯外设
3. 设计流水灯总线接口将流水灯连上SoC
4. 编写软件代码进行测试

由于大部分模块已经实现完成，只需要按实验指导书的提示修改部分代码即可

6. Some other experiments

- 请参考实验指导书完成实验三《中断与C语言编程》
- 设计思路

1. 设计UART外设
2. 设计UART总线接口将UART连上SoC
3. 将UART连上IRQ中断
4. 更新启动代码，完成C语言编程
5. 通过串口调试助手观察实验现象

由于大部分模块已经实现完成，只需要按实验指导书的提示修改部分代码即可

7. Waveform generator with virtual platform

- 通过一个简单的波形产生器熟悉虚拟平台的使用
- 设计思路

1. 在波形一个周期内取200个点将对应的值存在Waveform ROM里
2. 设计Waveform Generator控制Waveform ROM
3. 将Waveform Generator连在SoC上
4. 通过Cortex M0向Waveform Generator发送控制信号产生波形

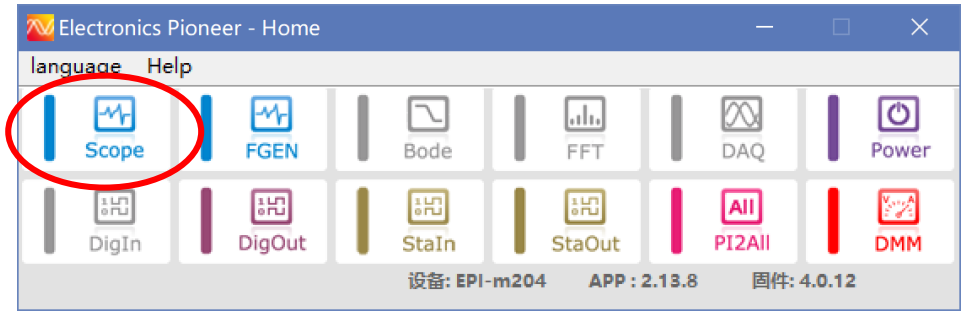
7. Waveform generator with virtual platform

1. Waveform Generator设计见代码
2. 在AHBLite_Decoder中添加对新外设地址的译码部分
3. 添加AHBLite_WaveformGenerator模块，实现总线到外设的接口
4. 在顶层模块CortexM0_SoC中例化新添加的模块
5. 修改各个RAM ROM里的路径(注意用’/’)
6. 编写汇编代码，通过CPU发使能信号控制波形产生
7. 添加新的约束管脚

7. Waveform generator with virtual platform

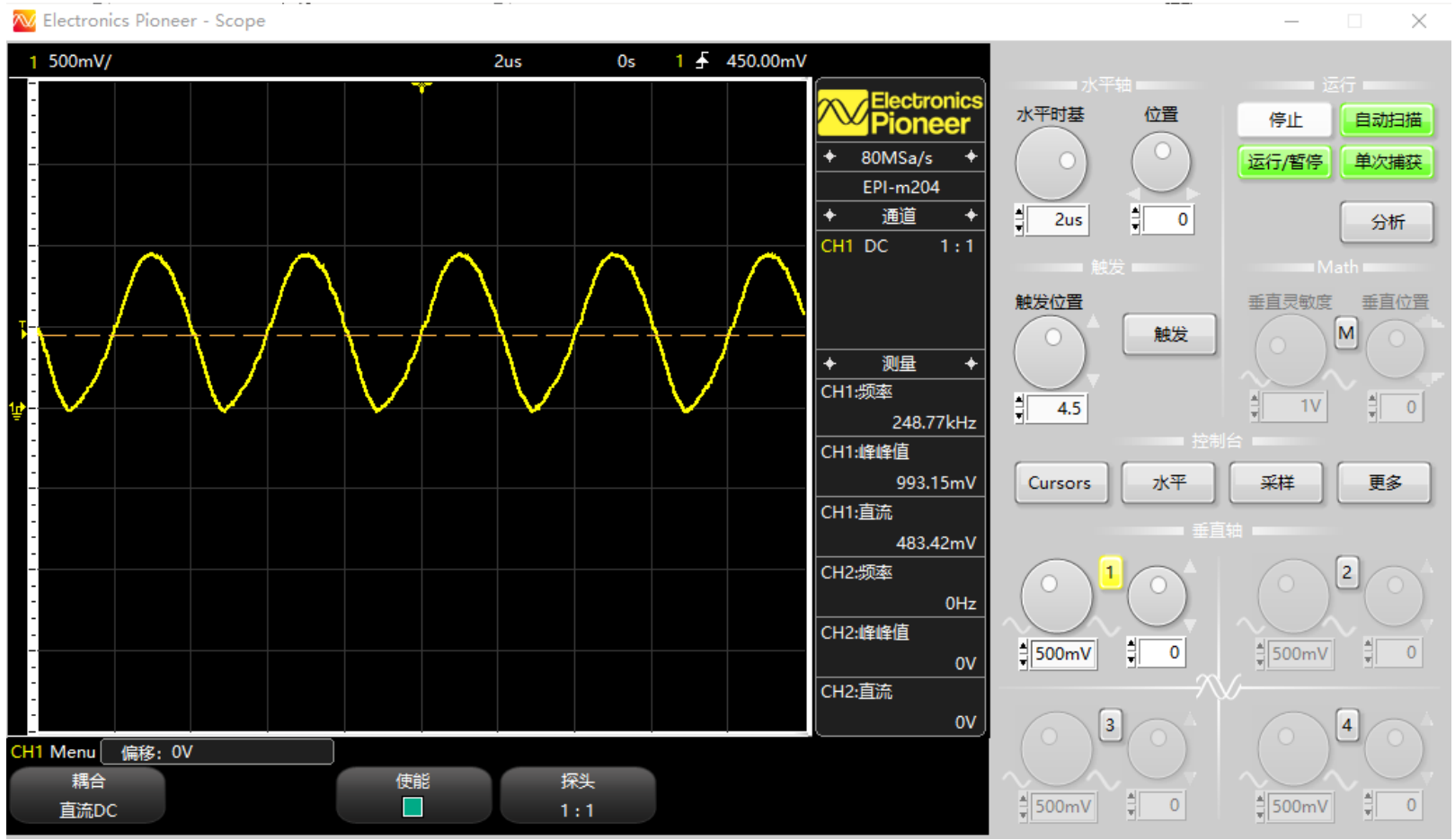
- 虚拟平台使用

1. 打开Electronics Pioneer
2. 选择Scope(示波器)
3. 将DAC输出管脚连接AIN1
4. 将FPGA Board的GND和虚拟平台的GND相连
5. 通过keil单步调试到波形产生器使能信号被赋值为1
6. 观察波形



7. Waveform generator with virtual platform

- Result



Thank you!