基于 ARM 的 SoC 设计入门

我们跳过所有对 ARM 介绍性的描述，直接进入工程师们最关心的问题。

要设计一个基于 ARM 的 SoC，我们首先要了解一个基于 ARM 的 SoC 的结构。图 1 是一个典型的 SoC 的结构：

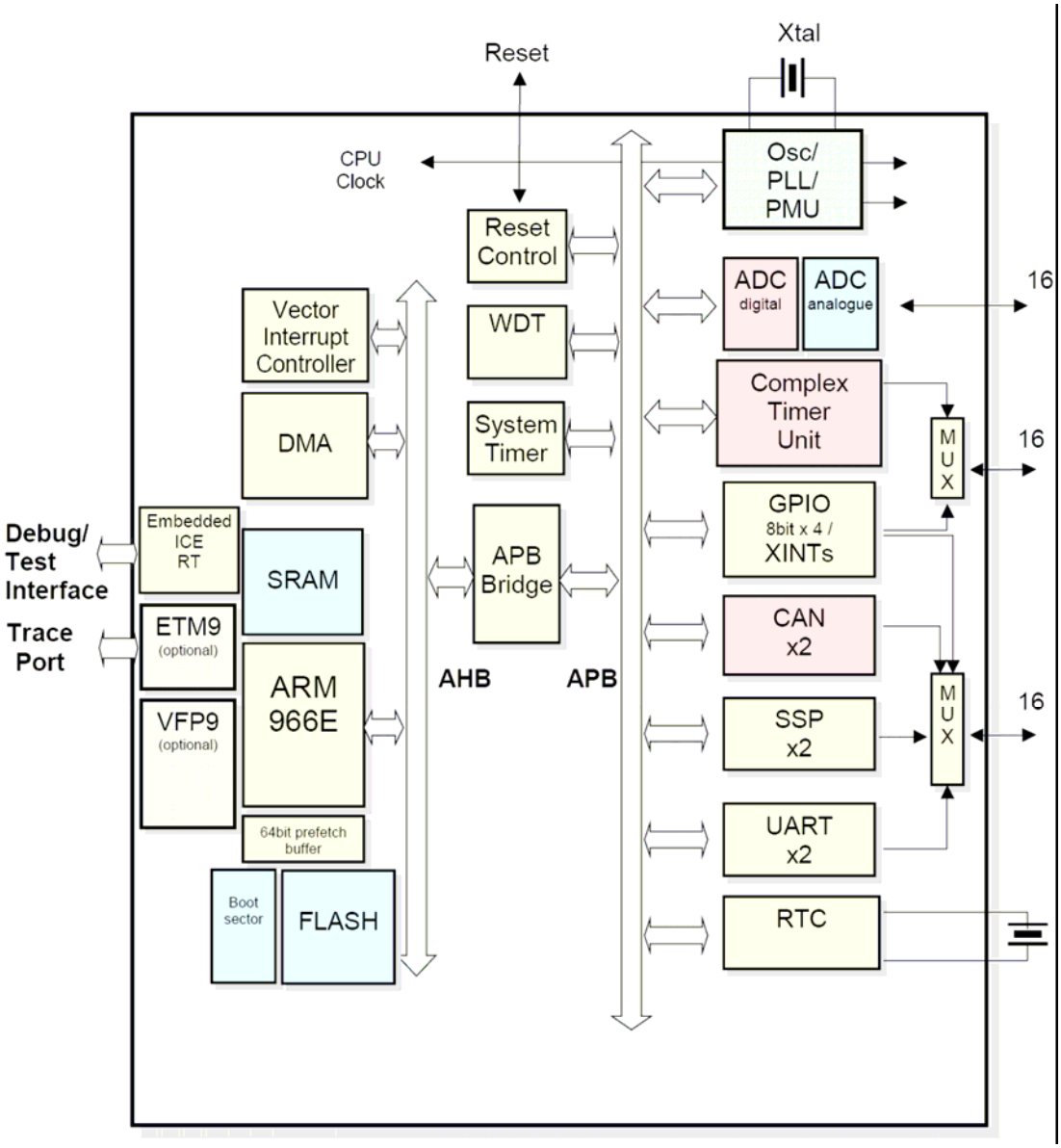

图 1

从图 1 我们可以了解这个的 SoC 的基本构成：

- ARM core：ARM966E
- AMBA 总线：AHB+APB
- 外设 IP(Peripheral IPs)：VIC(Vector Interrupt Controller), DMA, UART, RTC, SSP, WDT…
- Memory blocks：SRAM, FLASH…
- 模拟 IP：ADC, PLL…

如果公司已经决定要开始进行一个基于 ARM 的 SoC 的设计，我们将会面临一系列与这些基本构成相关的问题，在下面的篇幅中，我们尝试讨论这些问题。

1. 我们应该选择那种内核？

的确，ARM 为我们提供了非常多的选择，从下面的表-1 中我们可以看到各种不同 ARM 内核的不同特点：

ARM CPU CORES							
	Cache Size (Inst/Data)	Tightly Coupled Memory	Memory Mgt	Bus Interface	Thumb	DSP	Jazelle
APPLICATION CORES							
ARM1020E	32k/32k	-	MMU	2x AHB	Yes	Yes	No
ARM1022E	16k/16k	-	MMU	2x AHB	Yes	Yes	No
ARM1026EJ-S	Variable	Yes	MMU or MPU	2x AHB	Yes	Yes	Yes
ARM11 MPCore	Variable	-	MMU + cache coherency	1x or 2x AMBA AXI	Yes	Yes	Yes
ARM1136J(F)-S	Variable	Yes	MMU	5x AHB	Yes	Yes	Yes
ARM1176JZ(F)-S	Variable	Yes	MMU + TrustZone	4x AXI	Yes	Yes	Yes
ARM720T	8k unified	-	MMU	AHB	Yes	No	No
ARM920T	16k/16k	-	MMU	ASB	Yes	No	No
ARM922T	8k/8k	-	MMU	ASB	Yes	No	No
ARM926EJ-S	Variable	Yes	MMU	2x AHB	Yes	Yes	Yes
EMBEDDED CORES							
ARM Cortex-M3	-	-	MPU (optional)	-	Yes	No	No
ARM1026EJ-S	Variable	Yes	MMU or MPU	2x AHB	Yes	Yes	Yes
ARM1156T2(F)-S	Variable	Yes	MPU	3xAXI	Yes	Yes	No
ARM7EJ-S	-	-	-	Yes	Yes	Yes	Yes
ARM7TDMI	-	-	-	Yes**	Yes	No	No
ARM7TDMI-S	-	-	-	Yes	Yes	No	No
ARM946E-S	Variable	Yes	MPU	AHB	Yes	Yes	No
ARM966E-S	-	Yes	-	AHB	Yes	Yes	No
ARM968E-S		Yes	DMA	AHB-Lite	Yes	Yes	No
SECURE APPLICATIONS							
SecurCore SC100	-	-	MPU	-	Yes	No	No
SecurCore SC110	-	-	MPU	-	Yes	No	No
SecurCore SC200	-	-	MPU	-	Yes	Yes	Yes
SecurCore SC210	-	-	MPU	-	Yes	Yes	Yes

表 1

ARM 已经给出了基本的参考意见：

*如果您在开发嵌入式实时系统，例如汽车控制、工业控制或网络应用，则应该选择 Embedded core。

*如果您在开发以应用程序为主并要使用操作系统，例如 Linux, Palm OS, Symbian OS 或 Windows CE 等等，则应选择 Application core。

*如果您在开发象 Smart card，SIM 卡或者 POS 机一样的需要安全保密的系统，则需要选择 Secure Core。

举个例子，假如今天我们需要设计的是一个 VoIP 电话使用的 SoC，由于这个应用不需要使用到操作系统，所以我们可以考虑使用没有 MMU 的内核。另外由于网络协议盏对实时性的要求较高，所以我们可以考虑 ARM9 系列的内核。又由于 VoIP 有语音编解码方面的需求，所以需要有 DSP 功能扩展的内核，所以 ARM946E-S 或 ARM966E-S 应该是比较合适的选择。

当然，在实际工作中的问题要比这个例子要复杂的多，比如在上一个例子中，我们也可以选择ARM7TDMI内核加一个DSP的解决方案，由ARM来完成系统控制以及网络协议盏的处理，由单独的DSP来完成语音编解

码的功能。我们需要对比不同方案的面积，功耗和性能等方面的优缺点。同时我们还要考虑Cache size，TCM size，实际的内核工作频率等等相关问题，所以我们需要的一个能构快速建模的工具来帮助我们决定这些问题。现在的EDA工具为我们提供了这样的可能，例如Synopsys®的CCSS（CoCentric System Studio）以及Axys®公司的Maxsim®等工具都可以帮助我们实现快速建模，并在硬件还没有实现以前就可以提供一个软件的仿真平台，让我们在这个平台上进行软硬联仿，评估我们设想的硬件是否满足需求。

2.我们应该选择那种总线结构？

在提供内核给我们的同时，ARM 也提供了多种的总线结构。例如 ASB，AHB，AHB lite，AXI 等等，在定义使用何种总线的同时，我们还要评估到底怎样的总线频率才能满足我们的需求，而同时不会消耗过多的功耗和片上面积。这就是我们平时常说的 Architecture Exploration 的问题。

和上一个问题一样，这样的问题也需要我们使用快速建模的工具来帮我们作决定。通常，这些工具能为我们提供抽象级别很高的 TLM（Transaction
Level Models）模型来帮助我们建模，常用的 IP 在这些工具提供的库中都可以找到，例如各种 ARM core，AHB/APB BFM（Bus Function Model），DMAC 以及各种外设 IP。这些工具和 TLM 模型提供了比 RTL 仿真快 100～10000 倍的软硬联仿性能，并提供系统的分析功能，如果系统架构不能满足需要，那么瓶颈在系统的什么地方，是否是内核速度不够？总线频率太低？Cache 太小？还是中断响应开销太多？是否需要添加 DMA？等等，诸如此类的问题，我们多可以在工具的帮助下解决。

当然，机器不是万能的，不要指望工具能告诉你问题在哪里并告诉你怎么解决，工具能提供给你的只是一些统计的数据，而需要我们工程师去分析问题出在哪里并想出解决办法，所以熟悉 AMBA 体系结构和 ARM 内核是非常必要的。

3.如何选择外设 IP，使用现成的 IP 还是自己定制？

使用IP最大的优势是Time to Market，ARM提供了相当多的外设IP供我们选择。ARM提供的外设IP集（PrimeCell® IP）包括了常用的绝大部分外设，我们可以参考表 2：

Ancillary (APB) peripherals	
SD-Card interface	SD-Card Host interface
MMC interface	Multimedia card host interface
UART	Similar to 16C550, modem and flow control, up to 115K2 bits/s IrDA SIR
Synchronous serial interface	Supports Motorola SPI, TI SSI, Microwire
General purpose I/O	1x8 bit with individual interrupt control
Real time clock	32-bit counter, match reg, requires 1Hz clock
Keyboard/mouse interface	PS/2 compatible
DC-DC converter	Programmable output 1.8MHz, 900kHz, 225kHz, 96kHz
Audio codec interface	8-bit, 16 byte FIFO, programmable data rate
Advanced audio codec interface	Supporting the AC'97 multi-channel codec interface standard
Smartcard interface	Compliant with EMV standard and ISO 7816-3

Main system (AHB) peripherals	
DMA controller	Dual AHB Master, 8-channel DMA Controller Single AHB Master, 2-channel DMA Controller
Vector Interrupt controller	32 interrupts sources and 16 vector addresses
Static memory controller	SRAM, Flash and ROM
SDRAM controller	Four-port SDRAM memory controller supports AHB on all four ports
Color LCD controller	Color and mono with grayscale, supports TFT and STN, single/dual panel

表 2：

PrimeCell®的IP一直在增加，ARM会不定期在网站上公布最新可用的PrimeCell® IP，详情情参考：
http://www.arm.com/products/solutions/PrimeCellPeripherals.html

ARM 除了提供这些 IP 的 RTL 之外，还提供这些外设的驱动程序及测试程序。

使用现成 IP 方便的，但同时也带来灵活性的限制。举个例子，当我们需要一个 SPI 总线接口的时候，我们应该使用 ARM 的 SSP(Synchronous Serial Interface)这个 IP，但是这个 IP 为了提供能与 Motorola SPI，TI SSP，NS Microwire 都兼容的功能，牺牲了片上面积，导致 IP 复杂度增加了。如果我们的应用仅仅是需要和 Motorola SPI 的标准兼容，那我们又何必需要一个这样复杂的 IP 呢？

自己定制IP虽然得到了灵活性的优势，但是确需要设计工程师完成自己的一套验证，同时也要为这个IP开发驱动程序，工作增加了许多。我的建议是具体情况具体分析，在选择IP的时候也可以考虑第三方公司提供的基于AMBA总线标准的IP，比如Synopsys®的DesignWare® IP库中就有很多基于AMBA标准的IP可供选择，有时这些IP能够提供比ARM的IP更好的灵活性并同时使用更少的片上面积和功耗。比如同样是Memory Controller，DesignWare®的DW_memctl就比ARM的MPMC(Multi-port Memory Controller)更灵活，可以定制更多的参数。关于DesignWare® IP的详细资料，请参考：
http://www.synopsys.com/products/designware/designware.html

4.自己设计的连接在 AMBA 总线上的 IP 如何验证？

如果我们确实要自己设计连接在 AMBA 总线的 IP，那么熟悉 AMBA 的总线标准是必须的。但是设计往往不是问题，问题是如何验证我们的 IP 能符合所有 AMBA 标准定义的行为 呢？

作为一个 IP 的验证，我们常常会使用所谓的 Component-Based Verification，具体做法如图-2 所示：

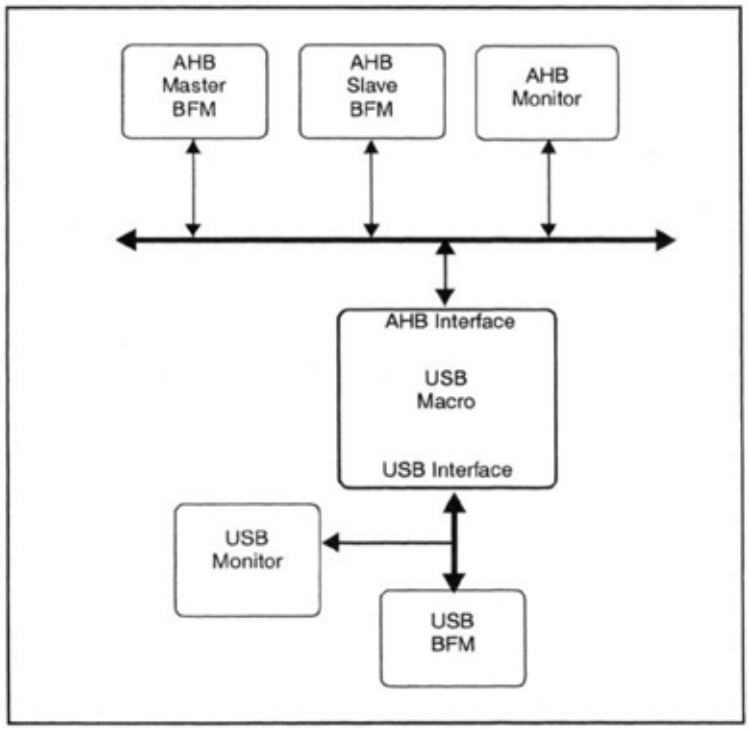

图 2

在图 2 中我们要验证的是一个USB的IP，这个USB模块直接连到AHB总线上，在我们的testbench中需要如图中所示的各种VIP（Verification IP），如图中所示的BFM，Bus Monitor，才能够模拟一个USB模块会在真实的总线结构中会遇到的全部情况。这些VIP可以由EDA vender提供，也可以由工程师自己编写，但通常这些VIP不会使用HDL语言编写，而是使用HVL(Hardware Verification Language)语言。例如：e®，Vera等语言，当然同时也要使用支持这些语言的工具，如Verisity®的Specman®。由于系统高速总线（通常是AHB或AXI）上的行为比较复杂，所以我建议这样的VIP不要由工程师自己开发，而尽量使用EDA vender提供经过了完善测试的VIP。对于低速外设总线（APB）或SPI，I2C，USB等总线，则自己开发BFM模型和Monitor是可行的。

5. 搭建好的平台如何验证？

我们选择了适合的 ARM core 和总线结构，挑选了合适的 IP，然后搭好了积木，TOP 完成了，问题也来了，TOP 该如何验证？

关于 SoC 的验证确实是个大题目，特别是在以 IP 为基础的 SoC 设计方法出现以后，在工程师的设计能力和验证能力中间出现了差距，也就是我们能在短时间内完成设计，却需要化数倍于设计的时间和人力来验证。最消耗时间的工作一般来说发生在软硬联仿(SW/HW Co-simulation/Co-verification)的阶段。

大家知道，抽象级越高的仿真越快，反之越慢，所以如果在我们的 TOP 文件中所有的模块都是 RTL 或 Gate level 的（包括 ARM core），那么仿真的速度是谁也无法接受的，所以现实一点的方法是使用 ARM core 的 DSM（Design Simulation Model）模型。具体方法如图-3 所示：

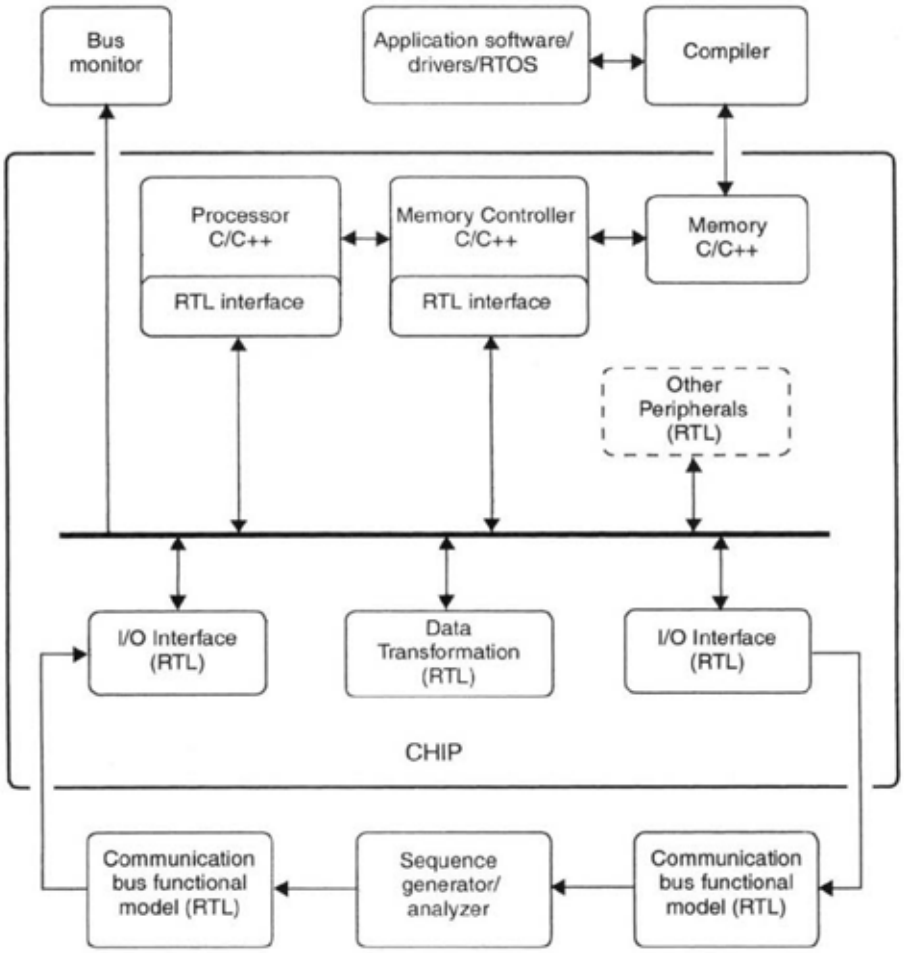

图 3

我们在图 3 中可以看到，对于内核(Processor)和 Memory Controller 我们都使用了使用高级语言编写的行为模型，并在这些行为模型和真正的 RTL 之间使用 PLI(Program Language Interface)语言编写的接口。当然，所有别的外设 IP 和总线结构都还是使用 RTL(因为我们就是要验证这些 RTL)。DSM 模型大多由 IP 提供商提供，如果使用的是 ARM core，当然就由 ARM 提供了。软件由 ARM 提供的开发工具(ADS，RealView)编译好，产生 bin 文件，然后储存的 Memory 的模型中。

这时我们已经可以开始仿真了，ARM 的 DSM 模型会在仿真的过程中产生一个 log.eis 文件，这个文件顺序记录了 ARM core 曾经执行过的所有指令，通过这个文件，我们就可以对软件进行 debug 了。

当然，使用这个文件对软件进行debug是很痛苦的，因为工程师不仅不能中断、跟踪、单步执行软件，更不能使用Semihosting功能进行文件操作和调试信息传递。如果可以使用AXD或Realview Debugger来对软件进行debug将给工程师带来极大的方便，所以EDA公司也推出了相关的产品，例如Mentor Graphics®公司的Seamless®以及我们前面提到的Axys®公司的Maxsim®等工具都能提供与AXD或者Realview Debugger协同仿真的接口。这样我们就可以象在目标板上调软件一样在仿真平台上调试软件了。

在这个抽象级别的仿真速度比纯RTL平台要快一些，大约能够做到 1～100 指令每秒的速度。在这样的平台上进行驱动程序和启动代码验证是可行的，但是如果要进行应用程序的全功能验证，特别是有操作系统的应用，这样的平台还是太慢了。比如在这样的平台上启动一个uClinux™，往往需要化数周的时间。显然，在这样的速度下验证应用程序还是不现实的。

解决这个问题我们有两个个选择：

(1) 使用硬件加速器

某些EDA vender会提供相关的解决方案，例如Cadence®公司的PALLADIUM® Accelerator/Emulator 和 Mentor Graphics®的VStationPRO® Emulation system。这些都是能够加速我们仿真的加速器，但是一般价格昂贵，所以对大多数的Design House来说，这个方法不但性价比不高，而且也没有必要。

(2)使用 FPGA 原型进行测试
这个方法对于大多数公司来说是比较现实的。这正是我们的下一个问题。

6. 如何完成 FPGA 原型验证？

完善的 FPGA 验证对芯片功能验证是非常必要的，同时正如我们在上一个问题中提到的，要完成完整的功能验证，没有 FPGA 原型的帮助是非常困难的。具体到基于 ARM 的 SoC，我们可以选择以下的一些方法：

(1) 由ARM公司提供的Integrator® prototyping board
ARM提供了一套名叫Integrator®开发套版，使工程师能够在这个套版上搭建和设计芯片尽量一致的验证平台。简单来说，ARM提供了Integrator® CT (Core Tile)来实现相应的ARM Core的功能和行为；使用Integrator® LT(Logic Tile)来实现我们芯片中除了ARM Core以外的所有数字逻辑（Integrator® LM上有个FPGA），使用Integrator® IM(Interface Module)连接模拟器件，再把这三个板作为子板统统插接到Integrator® AP（ASIC development Platform）。这样我们就像装电脑一样装出了一个SoC。在这个平台上，基本上所有的功能验证都可以做到，只要你对频率的要求不是太高（比如在你的应用中ARM core要跑在 100MHz），这个平台是可以完成实时测试的。

(2) 由第三方供应商提供的 FPGA 验证平台
例如ALDEC®公司的Riviera-IPT FPGA verification system。这个系统的硬件是一块PCI接口的板卡，这个板卡的核心的一个FPGA，我们的数字逻辑还是放在这个FPGA中。同时，在这个母板上可以插上不同的ARM core的Integrator® CM，这样就完成了数字部分的搭建了。这个系统同样能提供与ARM的方案差不多的性能，但是它比ARM的方案有更多一些的灵活性。

Riviera提供了一个能让ARM core， FPGA中的已综合逻辑和未综合的RTL三方协同仿真的功能。这个功能的好处的我们可以复用原来在工作站环境下仿真时使用的Testbench、激励和参考输出，并可以把RTL象搬积木一样一块一块的搬到FPGA中。也就是说，在开始时所有RTL和Testbench都可以在PC机上进行仿真（当然是使用Riviera提供的仿真器），这时仿真的速度是比较慢的；一旦工程师觉得哪一块的RTL已经OK了，那么他就可以将这一块RTL综合到FPGA中，随着越来越多的模块进入FPGA，仿真的速对会越来越快。最后，所有的数字逻辑都综合到了FPGA中。在RTL仿真和FPGA之间建立交互还有一个好处是在FPGA debug的时候给我们带来了很多方便。调试过FPGA的工程师常常有着痛苦的回忆，由于FPGA内部的信号不可见，FPGA的debug往往非常耗时，Riviera在提供RTL和FPGA联仿的同时，还可以提供观察FPGA内部信号的功能，类似Xilinx®的CHIPSCOPE。详细资料请参照网页：
http://www.aldec.com/products/riviera-ipt/pages/coverification/

(3) 自己开发 FPGA 原型板
当然，如果自己设计 FPGA 原型板，那么工程师就会拥有最大的灵活性，自己的开发板上可以放置任何需要的器件。选用的 FPGA 可以尽量贴近实际 SoC 的运行速度，如果有 Analog IP 对应的 Analog 器件，那么功能验证的覆盖率将会非常小，最大程度减少投片不成功的风险。这个方案的代价是设计和调试验证板的时间，有时这个时间还会超过芯片设计的时间，同时也需要工程师拥有设计高速 PCB 的相关知识。

ARM core 的测试样片是验证板的核心，这个测试样片实际上就是直接将内核拿去流片得到的（当然还要加上必要的 PLL）。通常，ARM 授权的 Foundry 会提供这样的测试样片，需要注意的是这个测试样片是否能达到应用所要求的速度，如果不能，那么实时测试将不可能实现。

另外一个需要注意的问题是FPGA的容量。在做AMBA总线结构FPGA综合的时候工程师会发现以AMBA总线为基础的RTL对FPGA资源的消耗非常惊人，有时一个 150K Gate Count的数字逻辑会无法综合到一个 150 万门的FPGA中（如Xilinx®的XC2V1500），所以在验证板的规划初期一定要选择一个留有余量的FPGA（或几个FPGA组成阵列）。

作者：蒋燕波
意法办导体深圳设计中心