目录

ZYNQ\_UltraScale+MPSoc

[SDSoc简介 2](#__RefHeading___Toc877_1484980969)

[Linux下安装petalinux注意事项： 3](#__RefHeading___Toc879_1484980969)

[黑金7020开发板ZYNQ学习入门01：7020框架图 4](#__RefHeading___Toc881_1484980969)

[GIC：general interrupt controller 4](#__RefHeading___Toc883_1484980969)

[AXI 接口 5](#__RefHeading___Toc885_1484980969)

[AXI Interconnect 7](#__RefHeading___Toc760_285850914)

[使用VIVADO创建一个工程 7](#__RefHeading___Toc887_1484980969)

**SDSoc简介**

专业术语:

AXI: Advanced eXtensible Interface

PL: process logic

PS: process system

SDSoc设计项目是建立在一个“平台”概念上的，平台包括了软件平台和硬件平台。平台是利用Vivado、SDK、和OS工具创建的。

硬件平台(HPFM)，定义了如处理系统（PS）、I/O子系统、存储接口等，这些工作都基于一个明确的端口接口（AXI、AXI-S、时钟、复位、中断）。

软件平台(SPFM)，定义了OS、设备驱动、启动和加载程序(boot loaders)、文件系统、库等。

**Linux下安装petalinux注意事项：**

https://blog.csdn.net/xiang\_shao344/article/details/81984033

1. 查看支持的linux的发行版和版本号。

2. 安装依赖库。

3. 安装完依赖库后，xilinx要求安装必须不是超级用户。(其实很多的软件安装，如果是自己使用，尽量都不要使用超级用户super。)

4. 需要使用bash

#sudo dpkg-reconfigure dash （在出现的界面选择“否”）

**安装：**

1. 修改petalinux的执行权限，chmod

2. 选择安装在哪里，如～/petalinux201802

3. 进入目录，将安装文件拷贝到此。

4. 运行安装文件。

**环境变量配置：**

修改 ~/.bashrc 这个文件

在最后加上 $ source ~/petalinux/settings.sh (填写为自己的petalinux安装路径，如果配置后打开终端，出现Permission denied 则可以将source 命令改写为sudo source).

配置环境变量时会出现警告WARNING: No tftp server found。

**解决方法：**

开启tftp

sudo apt-get install tftpd-hpa //tftpd-hpa是服务器端

sudo vim /etc/default/tftpd-hpa

将文本中的ADDRESS地址项修改为自己本机的inet地址(用ifconfig查看)，保存退出。

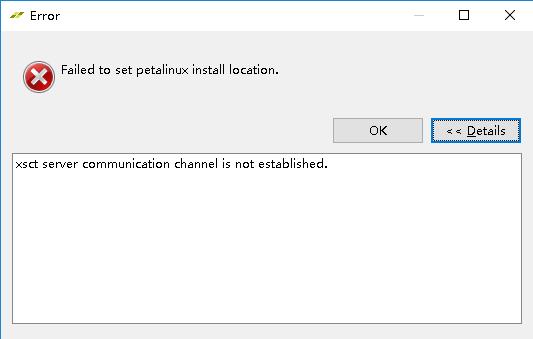
启动tftp：

sudo service tftpd-hpa restart即可

**注意**：

重新安装的时候，需要删除log文件。

**运行SDx IDE时提示：**

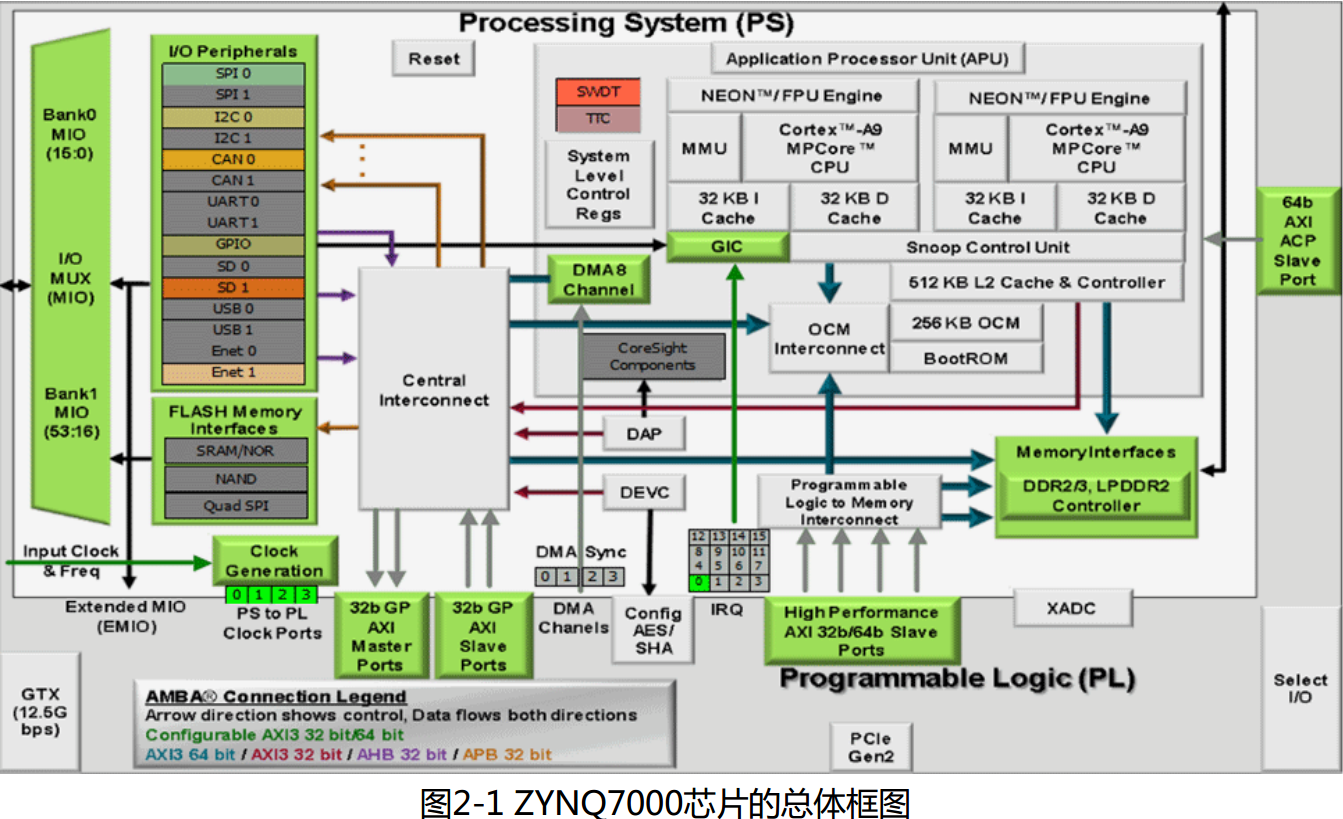


**xilinx学习中心**

xilinx官网的EmbeddedDesignHubs - Software Development Kit（SDK）

<https://china.xilinx.com/support/documentation-navigation/design-hubs/dh0015-sdk-hub.html>

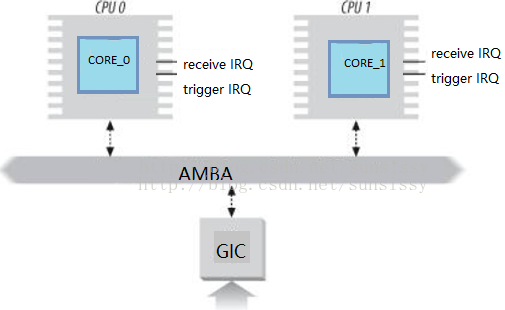
**黑金7020开发板ZYNQ学习入门01：7020框架图**



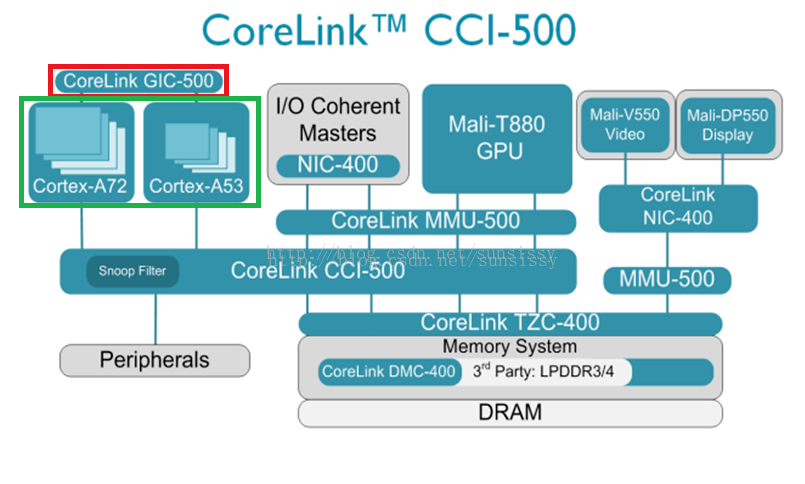
**GIC：general interrupt controller**

总结为，接受硬件中断信号，并进行简单处理，通过一定的设置策略，分给对应的CPU进行处理。

（https://blog.csdn.net/sunsissy/article/details/73791470）



如下图是ARM比较新的一个架构：



其中红色框中的就是GIC，CoreLink CCI-500是AMBA总线。

这是ARM比较新的架构图，其中CORELINK CCI-500是片上互联总线，也就是AMBA，在这总线上面挂了不同的设备。比如NIC-400设备，这可以理解为network interface设备，在这个设备上使用者可以再连接其它需要的器件。那么cpu，也就是cortex-72(大核)和cortex-53(小核)也是挂在总线上。上接GIC-500，也就是我们说的中断控制器，这个图其实是逻辑上的，实际中，GIC-500也是有线连接到CCI-500上，通过CCI-500和cpu连接交互，ARM的Distributor应该是为表示cpu和GIC之间的关系，才将其逻辑图表示成大家看到的。详细的可以参考ARM的官方网站介绍。

**AXI 接口**

1、 AXI（Advanced eXtensible Interface）协议主要描述了主设备（Master）和从设备（Slave）之间的数据传输方式，主设备和从设备之间通过握手信号建立连接。当主设备的数据准备好时，会发出和维持VALID信号，表示数据有效；当从设备准备好接收数据时，会发出READY信号。数据只有在这两个信号都有效时才开始传输。

2、 AXI协议（又称AXI4.0），包括3种接口标准：**AXI4**、**AXI-Stream**、**AXI-lite**。

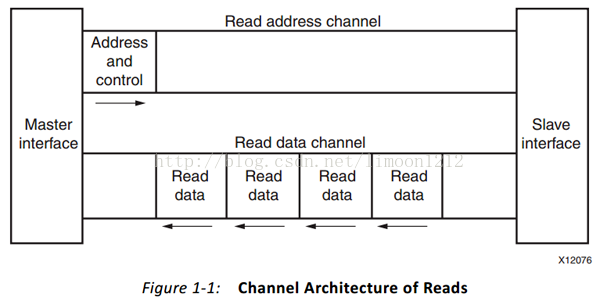
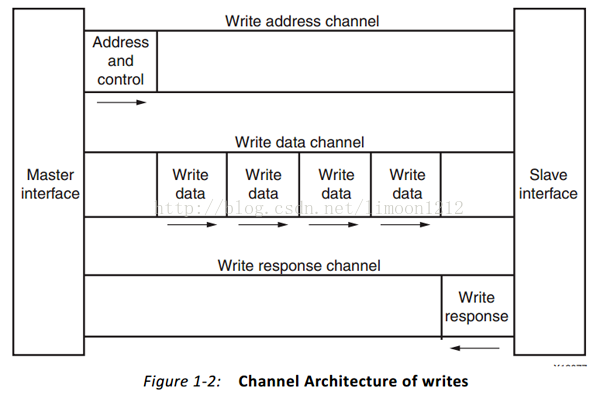
**AXI4**：适用于要求数据高速传输的场合。

**AXI-Stream**：如FIFO，数据传输不需要地址，而是主从设备间直接进行数据的读写，主要用于高速数据传输的场合，如视频、高速AD等。

**AXI-lite**：AXI4-Lite接口是AXI4接口的子集，专用于和元件内的控制寄存器进行通信。AXI4用于单个数据传输，主要用于访问一些低速外设。

3、 AXI接口具有5个独立通道：WriteAddress通道、Write Data通道、Write Response通道、Read Address通道、Read Address通道、Read Data通道。

4、 读/写通道并行地进行数据交互，明显提高了数据吞吐量，对写数据，从设备会返回确认信号，这样可以保证写数据通道的安全，读/写模型分别如图1-1、图1-2。

读模型：主设备发送读地址占用信号给从设备→从设备将数据写入主设备，实现读操作。

写模型：主设备发送写地址占用信号给从设备→主设备将数据写入从设备→从设备回复确认收到信号，实现写操作。

5、 AXI协议严格来讲是一个点对点的主/从接口协议，当多个外设需要互相交互数据时，我们需要加入一个AXI Interconnect模块，也就是AXI互联矩阵，AXI Interconnect的作用是将一个或多个AXI主设备连接到一个或多个AXI 从设备。

6、 AXI Interconnect IP核最多支持16个主设备和16个从设备，如果需要更多的接口可以在设计中加入多个IP核。

7、ZYNQ中的AXI接口包含三个类型，共9个，主要用于PS与PL的互联。

（1）**AXI\_HP**接口（PL模块作为主设备）

包括4个，主要用于PL访问PS上的存储器。每个接口都有两个FIFO缓冲器，一个是读缓冲，一个是写缓冲。

【实例：设计视频处理时，高清的图像可由FPGA直接完成采集、预处理，然后通过AXI\_HP接口将数据高速传输至DDR中，供APU（加速处理器）完成进一步的图像处理】

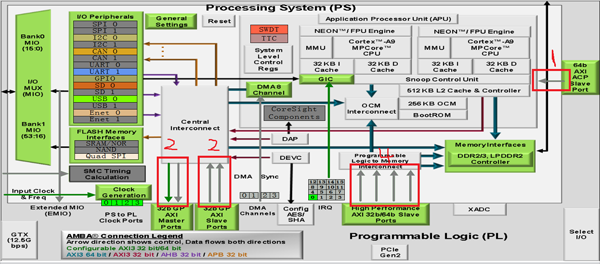
（2）**AXI\_ACP**接口（PS端是从设备端）

只有1个，又叫加速器一致性端口，适合做专用指令加速器模块接口。PL端可直接从PS部分的Cache中拿到CPU的计算结果，同时也可以第一时间将逻辑加速运算的结果送至Cache中，延时很小。

（3）**AXI\_GP**接口（PS端是从设备端）

通用AXI接口，总共有4个。可用于控制电机运转，获取传感器信号等逻辑模块的连接接口。

可以参考上面的架构图。



**https://blog.csdn.net/weixin\_41967965/article/details/82118259**

**AXI**全称Advanced eXtensible Interface，是Xilinx从6系列的FPGA开始引入的一个接口协议，主要描述了主设备和从设备之间的数据传输方式。在ZYNQ中继续使用，版本是AXI4，所以我们经常会看到AXI4.0，ZYNQ内部设备都有AXI接口。其实AXI就是ARM公司提出的AMBA（Advanced Microcontroller Bus Architecture）的一个部分，是一种高性能、高带宽、低延迟的片内总线，也用来替代以前的AHB和APB总线。第一个版本的AXI（AXI3）包含在2003年发布的AMBA3.0中，AXI的第二个版本AXI（AXI4）包含在2010年发布的AMBA 4.0之中。

AXI协议具有如下特点：

. 总线的地址/控制和数据通道是分离的；

. 支持不对齐的数据传输；

. 在突发数据传输中只需要首地址；

. 同时具有分离读/写数据通道；

. 支持显著传输访问和乱序访问；

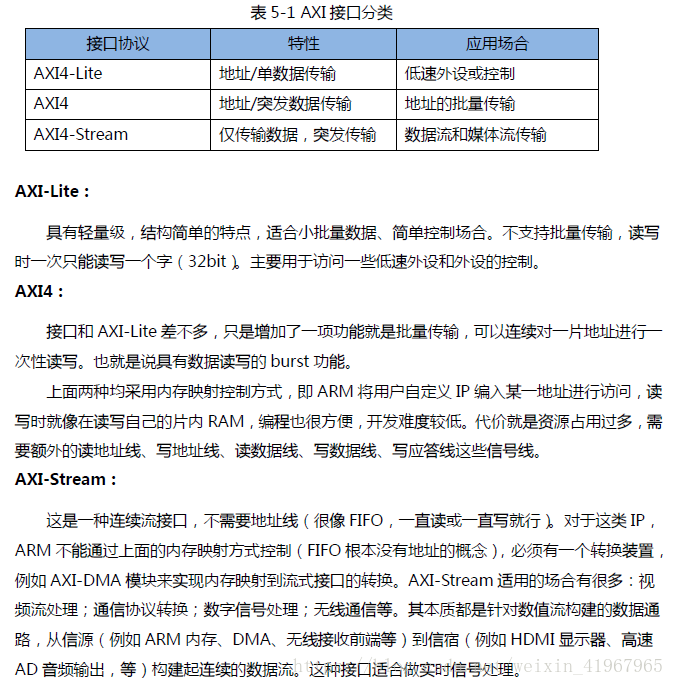
. 更加容易进行时序收敛

AXI4包含三种接口：

. AXI4——For high-performance memory-mapped requirements.

. AXI4-Lite——For simple, low-throughput memory-mapped communication (for example, to and from control and status registers).

. AXI4-Stream——For high-speed streaming data.



从上面的描述可以看出，AXI4协议相当于原来的AHB协议，提供高速的系统内部互连通道，可以支持burst模式，主要用于处理器访问存储等需要高速数据的场合；AXI4-Lite为外设童工单个数据传输，相当于原来的APB协议，用于访问一些低速外设；AXI4-Stream接口就像FIFO一样，数据传输的时候不需要地址，而是主从设备直接连续读写数据，主要用于如视频、高速AD、PCIe、DMA接口等需要高速数据传输的场合，跟Xilinx原来的Local Link协议类似。

**AXI Interconnect**

AXI协议严格的讲是一个点对点的主从接口协议，当多个外设需要互相交互数据时，我们需要加入一个AXI Interconnect模块，也就是AXI互联矩阵，作用是提供将一个或多个AXI主设备连接到一个或多个AXI从设备的一种交换机制（有点类似于交换机里面的交换矩阵）。Xilinx为我们提供了实现这种互联矩阵的IP核axi\_interconnect\_1，在前面的例子中，我们在XPS中可以看到。这个IP核最多可以支持16个主设备、16个从设备，如果需要更多的接口，可以多加入几个IP核。关于AXI Interconnect更多的知识，可参考Xilinx官方文档DS768。

AXI4和AXI4-Lite接口包含5个不同的通道：

. Read Address Channel

. Write Address Channel

. Read Data Channel

. Write Data Channel

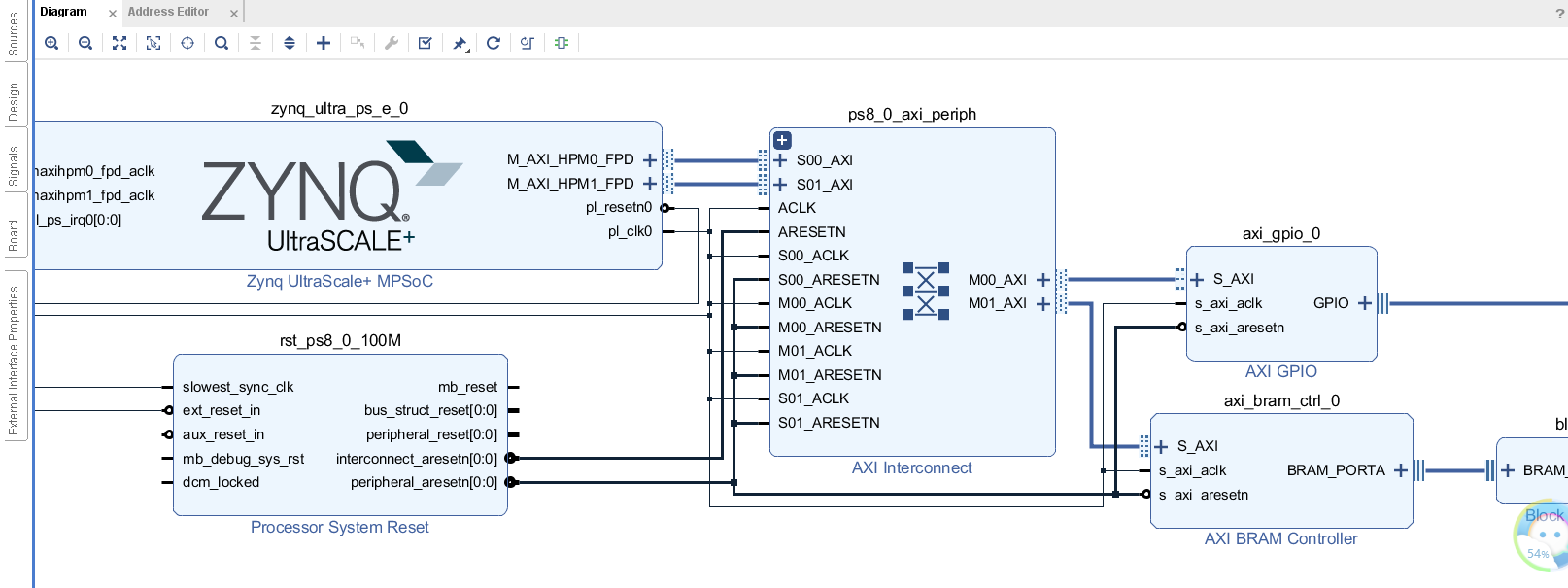
. Write Response Channel

其中每个通道都是一个独立的AXI握手协议。下面两个图分别显示了读和写的模型：

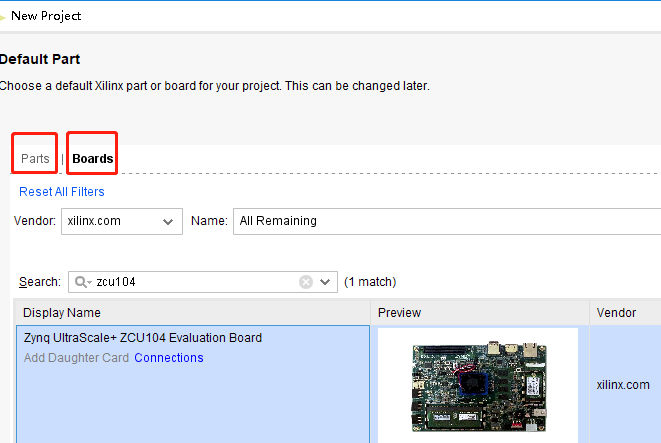
接上面链接的图。

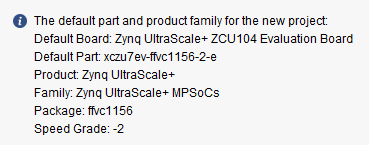
使用vivado打开ZCU104 evb一个basic示例project：

下图中间的就是AXI Interconnect。



**使用VIVADO创建一个工程**



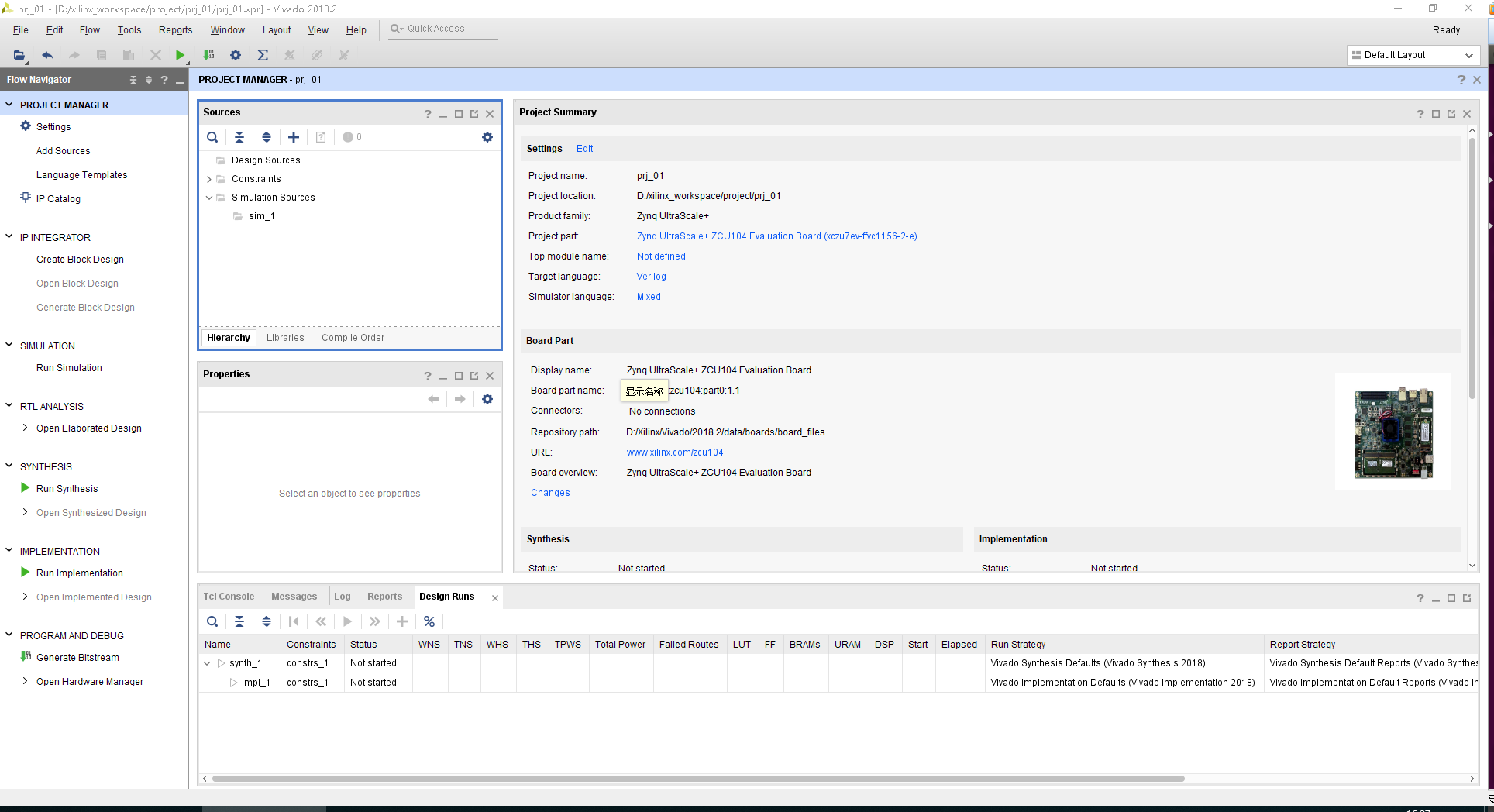


可以看到zcu104上的期间型号：xczu7ev-ffvc1156-2-e

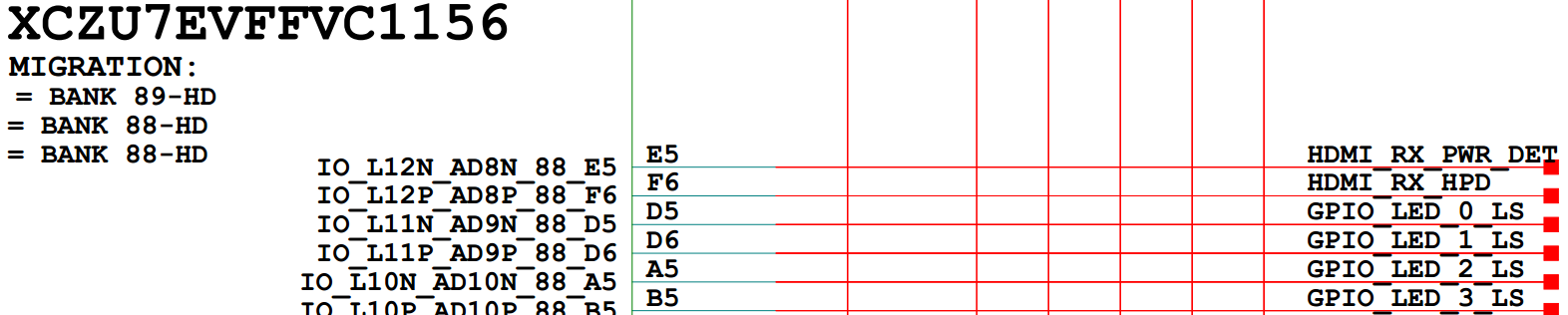
封装：ffvc1156

速度等级：-2

最后点击完成，进入project的设计：



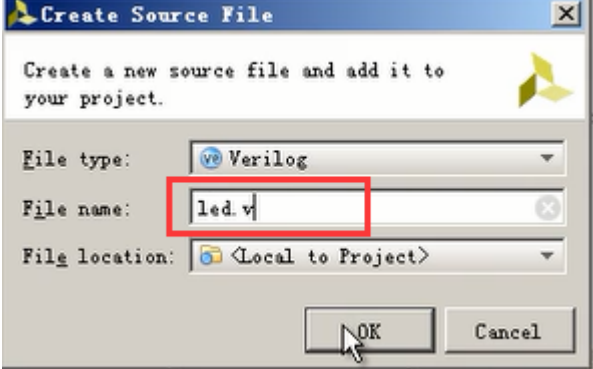
ZCU104开发的led引脚对应关系：



**创建source文件：**

然后可以创建source file：led.v（verilog程序文件）

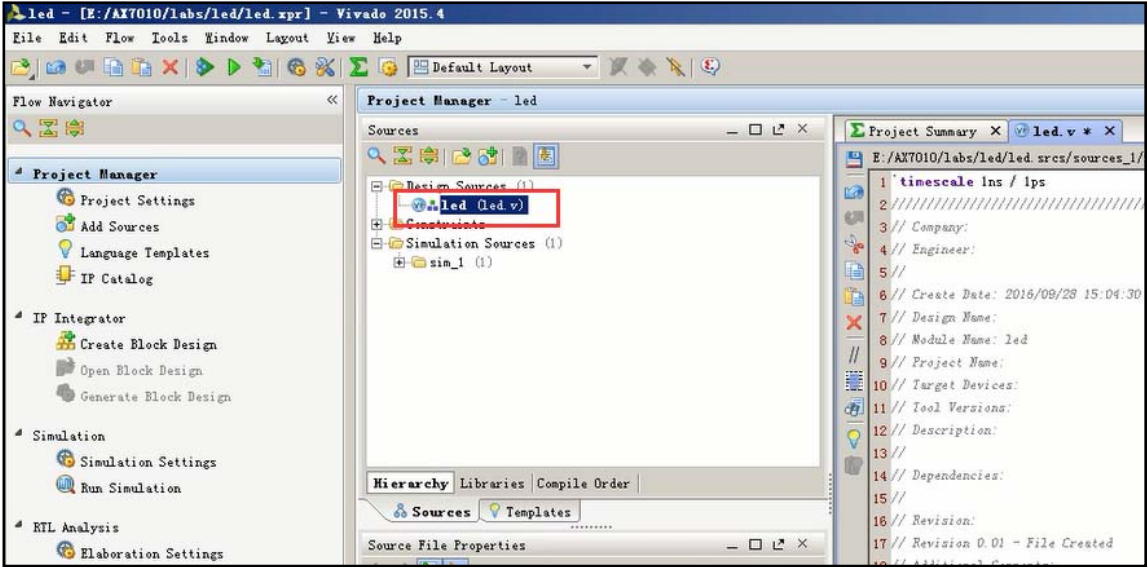




向导会提示您定义 I/O 的端口，这里我们可以不定义，后面自己在程序中编写就可以。

这时在 Project Manager 界面下的 Design Sources 里已经有了一个 led.v 文件, 并且自动成

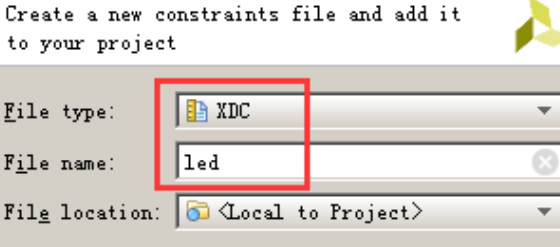
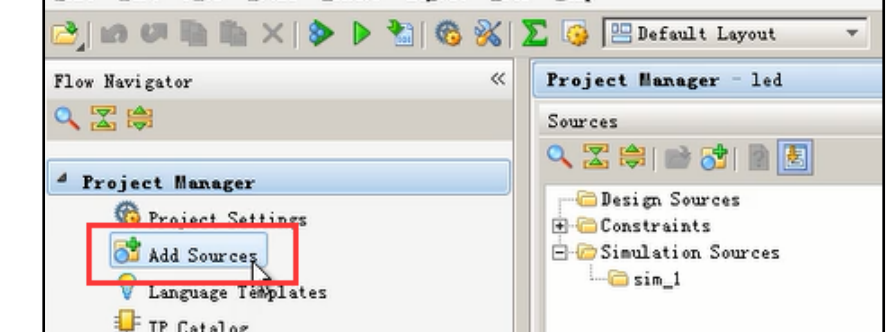
为项目的 Top 模块了。

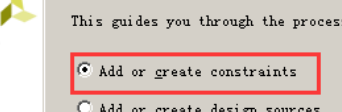


可以编写verilog程序文件。

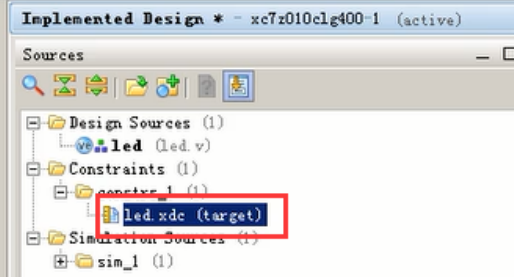
**7.4 添加 XDC 管脚约束文件**

约束文件，完成管脚的约束、时钟的约束、以及组的约束等。这里我们要对led.v程序中的输出端口分配导FPGA的真实管脚上。这需要准备一个FPGA的引脚绑定文件.xdc并添加道工程中。

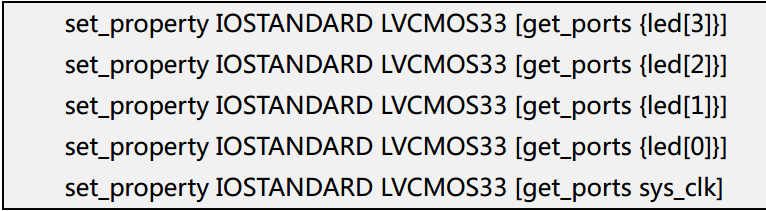


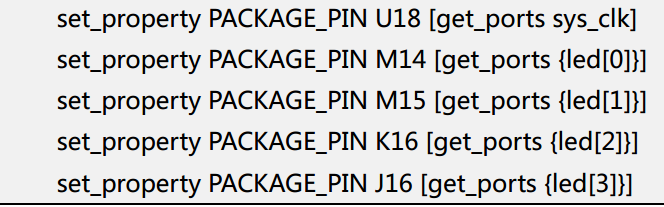


最后，点击Finish后，可能在design source中生成一个led.xdc文件。



双击打开这个 led.xdc 文件，在这个文件里添加以下的引脚定义。





**约束文件的基本语法：**

下面来介绍一下最基本的 XDC 编写的语法，普通 IO 口只需约束引脚号和电压，管脚约束

如下：

**set\_property PACKAGE\_PIN "引脚编号" [get\_ports “端口名称” ]**

电平信号的约束如下：

**set\_property IOSTANDARD "电压" [get\_ports “端口名称” ]**

这里需要注意文字的大小写，端口名称是数组的话用{ }刮起来**{led[0]}**，端口名称必须和源代码中

的名字一致，且端口名字不能和关键字一样。

完成后选择菜单 File->Save all files 保存所有文件。

**7.5 综合生成网表**

点击 Run Synthesis，即可开始**综合并生成网表文件**：

**点击** Implementation，实现布局布线。

**点击**，Generate Bitastream可以生成bit流文件。

在Project Summary中可以看到资源的使用：

**LUT：**查找表

**FF** ： flip flop寄存器

**IO** ：管脚

**BUFG**：时钟Buffer

**BRAM：**

**URAM：**

**DSP：**

7.6下载和调试

可以把bit文件烧写道ZYNQ芯片中，看以下实际的运行效果。点击 Open Hardware Manager

**黑金zynq的开发流程：**

ZYNQ 的开发也是先硬件后软件的方法。具体流程如下：

(1). 在 Vivado 上新建工程，增加一个嵌入式的源文件。

(2). 在 Vivado 里添加和配置 PS 和 PL 部分基本的外设，或需要添加自定义的外设。

(3). 在 Vivado 里生成顶层 HDL 文件，并添加约束文件。再编译生成比特流文件

system.bit。

(4). 导出到 SDK 软件开发环境，在 SDK 环境里可以编写一些调试软件验证硬件和软

件，结合比特流文件单独调试 ZYNQ 系统。

(5). 在 SDK 里生成 FSBL 文件。

(6). 在 VMware 虚拟机里生成 u-boot.elf bootloader 镜像。

(7). 在 SDK 里通过 FSBL 文件, 比特流文件 system.bit 和 U-boot.elf 文件生成一个 boot.bin

文件。

(8). 在 VMware 里生成 Ubuntu 的内核镜像文件 Zimage 和 Ubuntu 的文件系统 ramdisk。

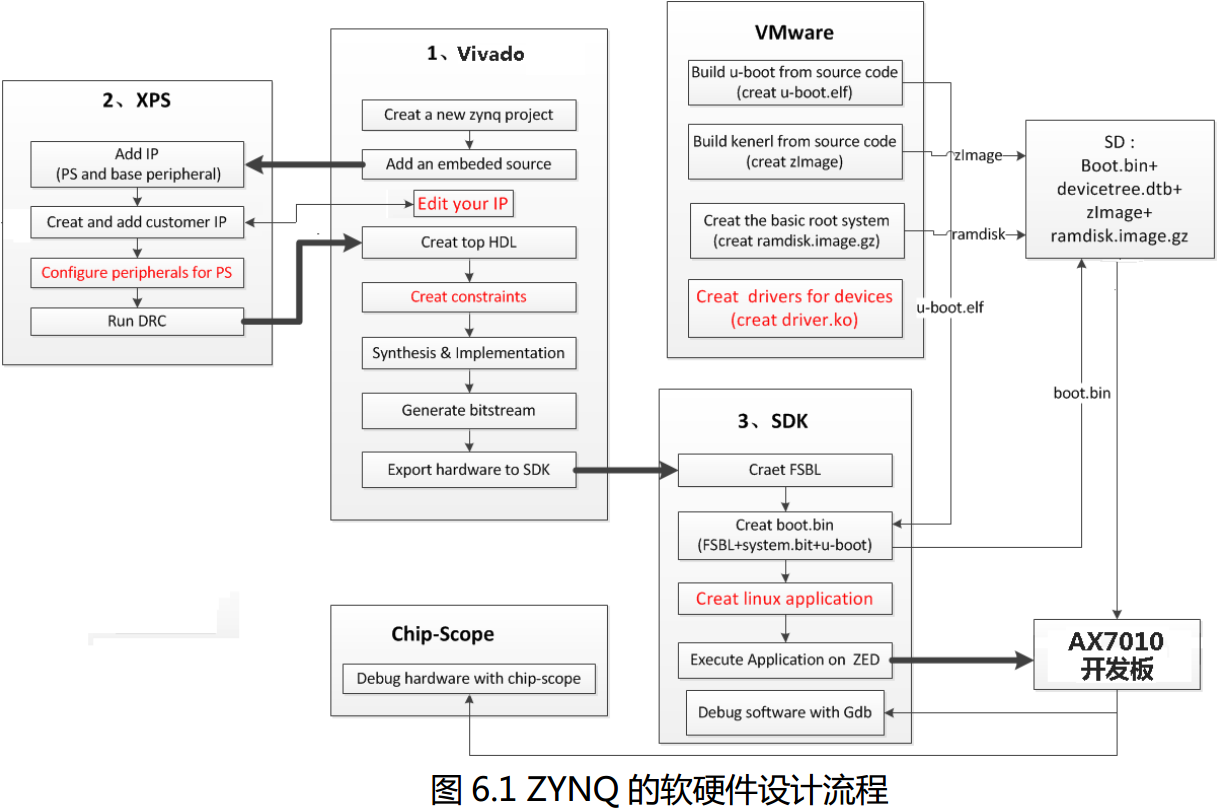
另外还需要要对 FPGA 自定义的 IP 编写驱动。

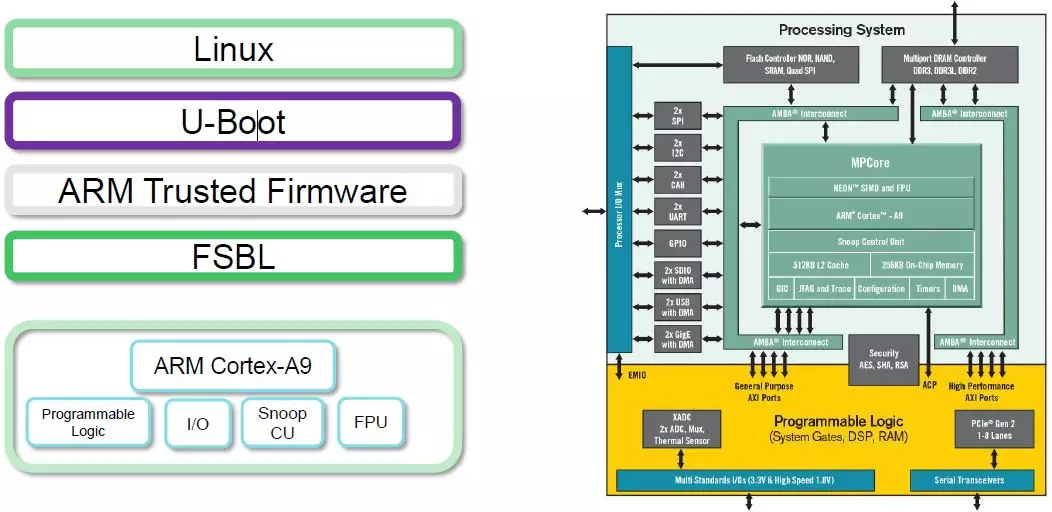
(9). 把 boot.bin，zimage 和 ramdisk 三个文件放入到 SD 卡的 FAT 分区中，启动开发板电

源，Ubuntu 操作系统会从 SD 卡里启动。

在设计和调试过程中，我们也会使用 chipscope 工具观察信号或使用 GDB, GDBserver

来调试 Linux 下的程序。 ZYNQ 软硬件设计的流程图如图：





Petalinux 相应的命令功能：

