Zynq7000-AXI-GPIO之LED点亮

# 基础知识

在第3小节中，我们使用PS专用的GPIO，通过MIO模块点亮了Zybo-Z720板上的LED4。第3小节的实验完全使用PS（ARM）处理，并没有用到PL（FPGA）部分。在本小节的学习中，我们将以这样的流程：PS -> AXI GPIO -> PL -> LED，来初步了解AXI GPIO，并感受Zybo-Z720的PS与PL相结合的独特魅力。

# 实验

## 实验环境

### Software

Windows 10 64bit

Vivado 2017.4

Xilinx SDK 2017.4

### Hardware

Zybo-Z720

## 成果与目标

成果：分别采用两种方式，直接访问寄存器和API Driver, 点亮Zybo-Z720的LED

目标：通过本次实验，初步了解PS-PL的桥梁 — AXI总线。重点理解互联（Interconnect）和接口（Interface）这两个术语。除此以外，熟练TRM的查阅，以及API Driver的使用方法。

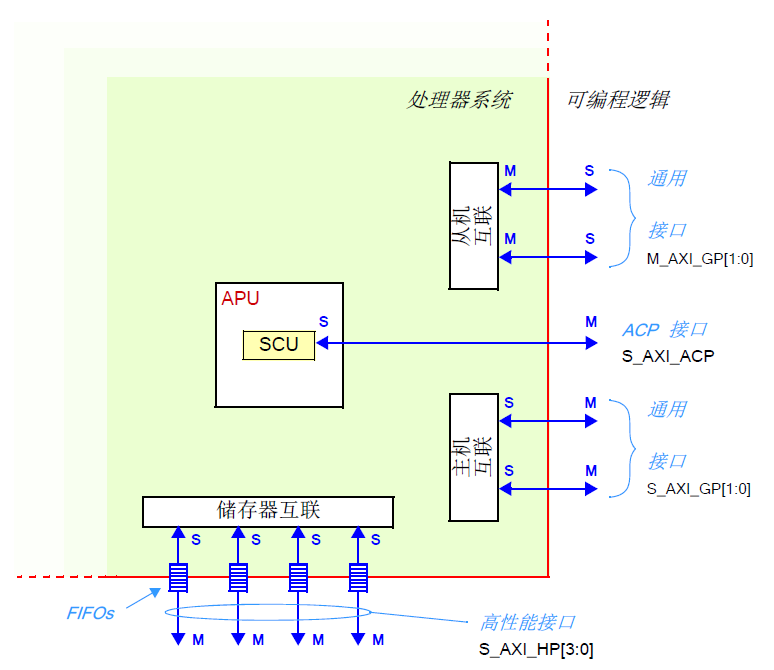
## 预备知识

### AXI互联和接口

在PS和PL之间的主要链接是通过一组9个AXI接口[1]，每个接口由多个通道组成。这些形成了PS内部的互联以及PL的链接，如图1所示，这里有必要定义两个重要的术语：

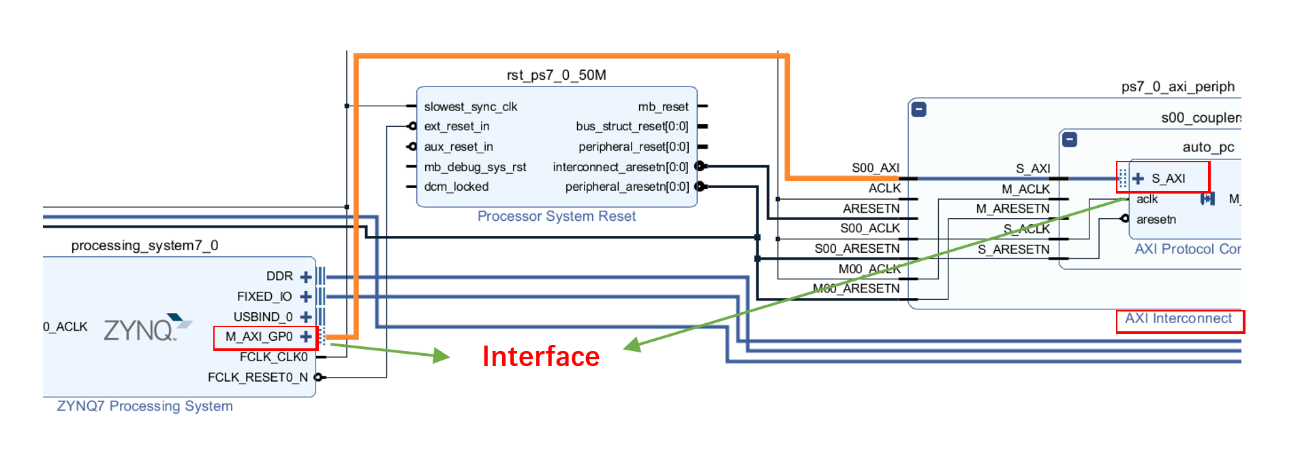
* 1. 互联（Interconnect）: 互联实际上是一个“开关”，管理并直接传递所连接的AXI接口之间的通信。在PS内有几个互联，其中有些还直接连接到PL，而另一些是只用于内部连接的。这些互联之间的连接也是用AXI接口所构成的。
  2. 接口（Interface）: 用于在系统内的主机和从机之间传递数据，地址和握手信号的点对点连接。

从图上可以注意到所有的接口都明确地连接到PS内的AXI互联，唯一例外的是ACP接口，它直接连接到APU里面的一致性控制单元（SCU）。



图表 1：连接PS和PL的AXI互联和接口的架构

在本次实验中，我们构建的block diagram中可以看出，Vivado的自动配线功能将ZYNQ Processing System 通过 M\_AXI\_GP0接口与AXI Interconnect的S00\_AXI接口相连。



图表 2：实例：Interface & Interconnect

下表给出了图1 中的箭头所表示的接口的总结。它给出了每个接口的简述，标出了主机和从机（按照惯例，主机是控制总线并发起会话的，而从机是做响应的）。注意接口命名的规范（在表2.2 的第一列）是表示了PS 的角色的，也就是说，第一个字母“M” 表示PS 是主机，而第一个字母“S” 表示PS 是从机。



进一步解释这些不同类型的PS-PL AXI接口的作用：

·通用AXI\_GP（General Purpose AXI）: 一条32位数据总线，适合PL和PS之间的中低速通信。接口是透传的不带缓冲。总共有四个通用接口：两个PS坐主机，另两个PS做从机。

·加速器一致性端口（Accelerator Coherency Port）: 在PL和APU内的SCU之间的单个异步连接，总线宽度为64位。这个端口用来实现APU cache和PL单元之间的一致性。PL做主机。

·高性能端口（High Performance Ports）: 四个高性能AXI接口，带有FIFO缓冲来提供“批量”读写操作，并支持PL和PS中的存储器单元的高速率通信。数据宽度是32或64位，在所有的四个接口中PL都是做主机的。

每条总线都是由一组信号组成，这些总线上的会话是根据所定义的总线标准，也就是AXI4来发生的，下面会介绍这个标准。关于AXI总线的会话的深入解释超出了目前讨论的范围，我们将会在之后的章节进行讨论。

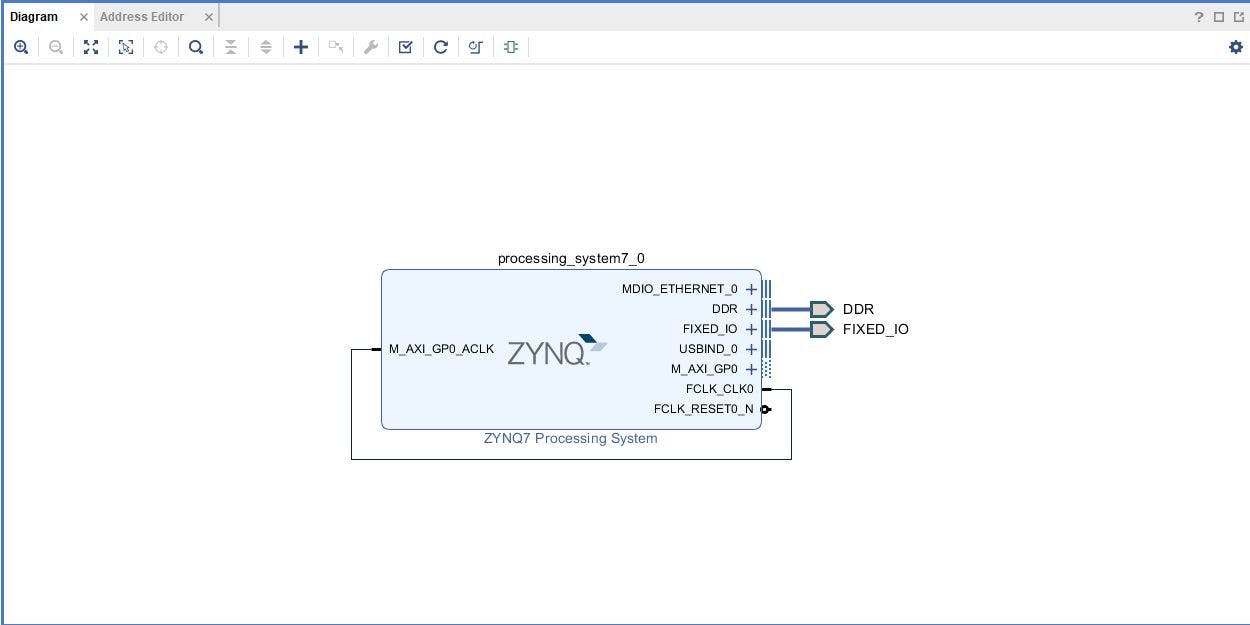
## 实验步骤

### 配置硬件

首先，我们需要在第3小节的基础上，扩展我们的硬件配置，加入AXI GPIO IP核，以及通过Vivado自动完成IP核之间的连接。

#### 添加IP核

本次实验继续沿用Helloword工程，在第3小节中，我们的Block Design的Diagram中只有一个ZYNQ7 Processing模块。如下图3所示：

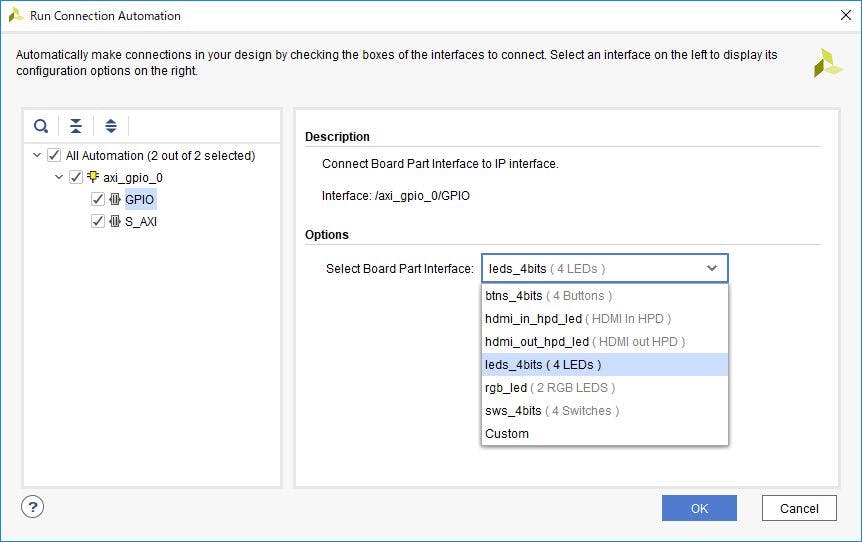


图表 3：Section3 Diagram

在此基础之上，我们添加一个AXI GPIO，这是由Xilix提供的IP核，Add IP Core -> AXI GPIO双击添加即可。接下来，Run Connection Automation的设定如下图4所示。在这里我们选择了：

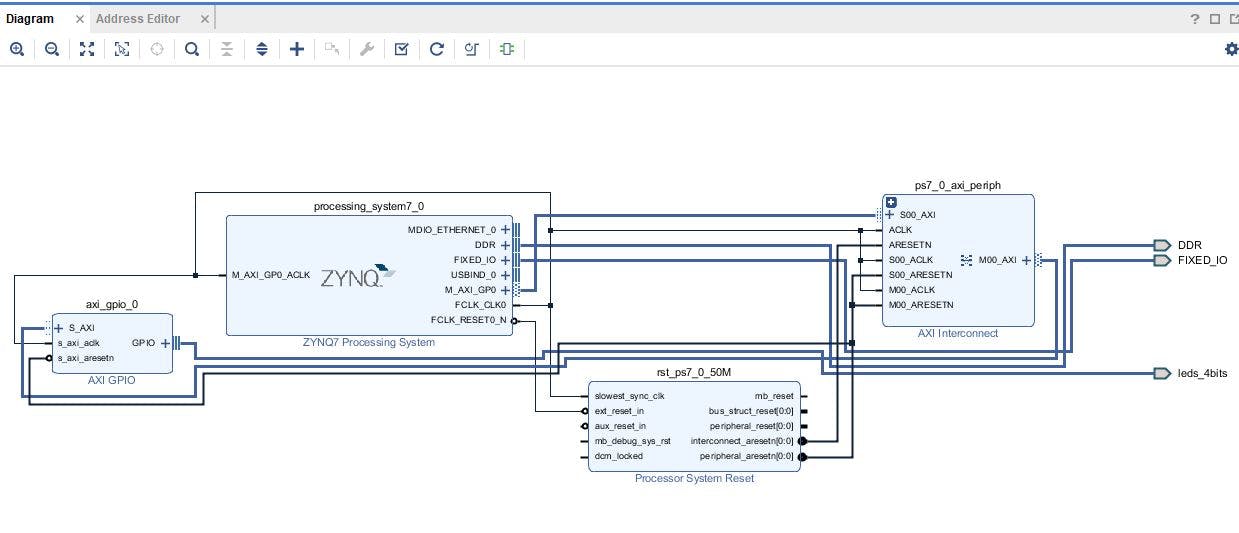
axi\_gpio\_0 （channel 0）

Select Board Part Interface（leds\_4bits）: 自动与Zybo-7Z20的四个LED灯相连接。



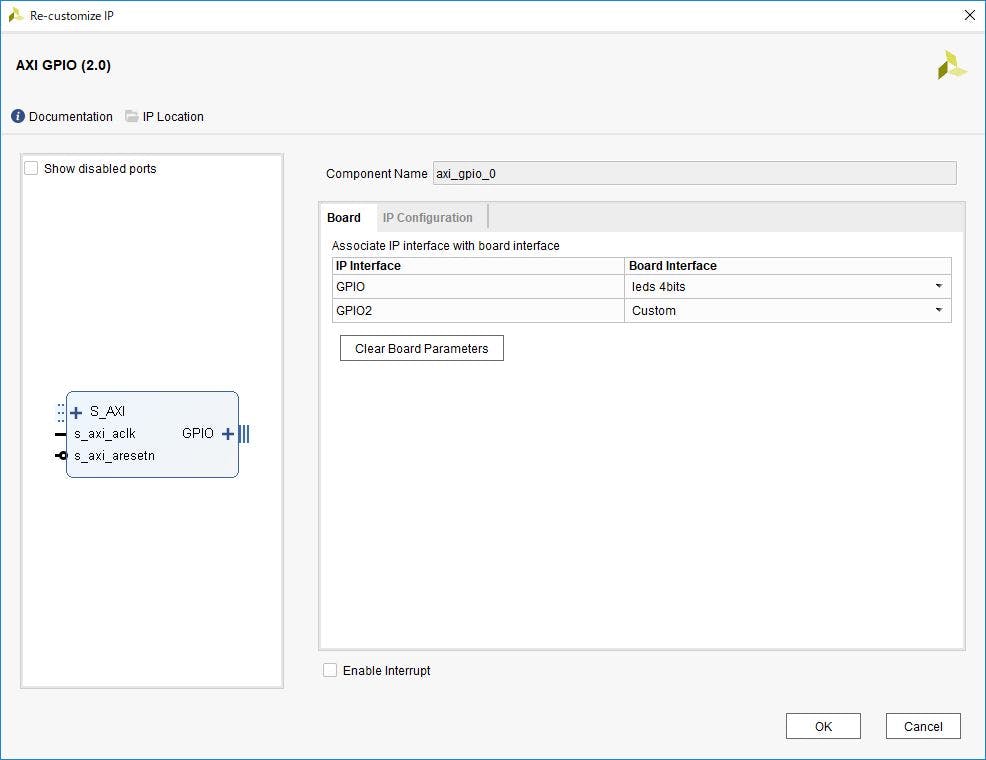
图表 4：AXI GPIO 设置

单击OK，Vivado自动配置完成后，发现Diagram中新增了3个IP Block。其中，Processor System Reset顾名思义，就是控制始终重置的某块，AXI Interconnect就是我们在预备知识模块中提及的“互联”模块，它负责在各个AXI Interface（接口）之间相互协调运作。



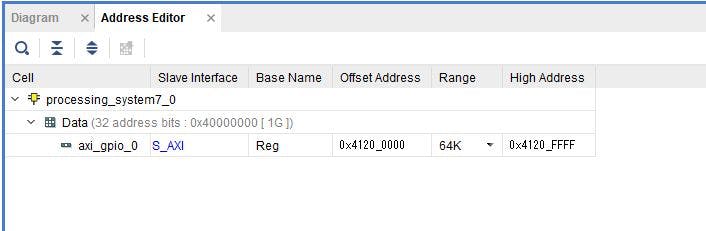
图表 5：AXI GPIO配置完成后的Diagram

最后，我们双击AXI GPIO模块，确认配置完成后的详细信息。可以发现，GPIO(Channel 0)与4个LED相连，GPIO2（Channel 1）处于常规设定状态。



图表 6：配置完成后AXI GPIO的详细信息

除此以外，在Address Editor面板中，可以查看个IP核对应寄存器的地址，这次我们创建的AXI GPIO（axi\_gpio\_0）的地址为0x41200000，我们查阅Xilinx AXI GPIO IP核的技术文档[2]，也可以获取更多信息。同第3小节一样，我们要用到这个地址信息，来编程我们的C语言程序。



图表 7：Address Editor

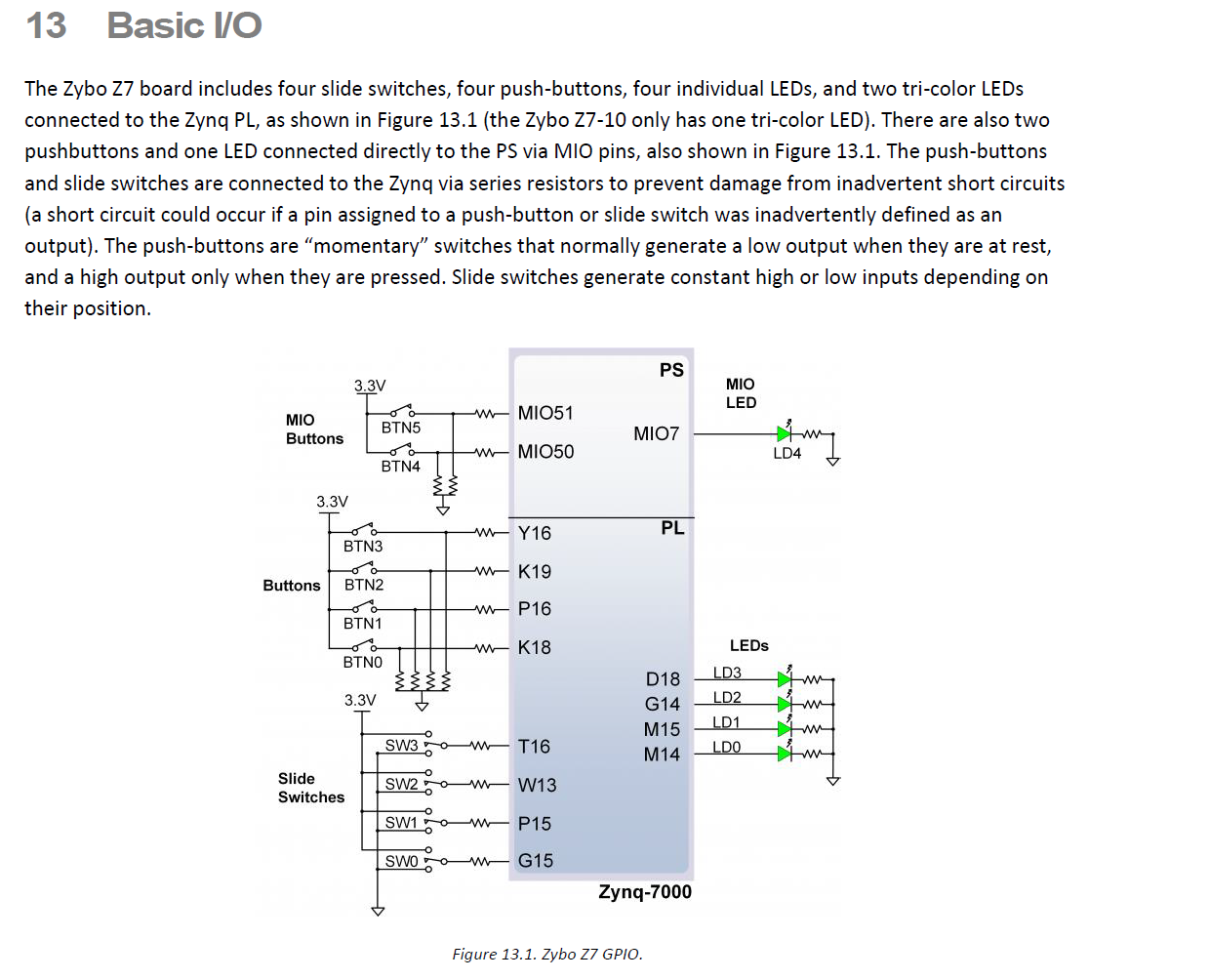
#### 生成bit和hdf文件

确认完所有硬件信息后，我们按部就班进行以下步骤：

* Source -> design\_1 右击 -> Generate Output Products -> Create HDL Wrapper
* Flow Navigator -> Generate Bitstream
* File -> Export -> Export Hardware (include bitstream)

### SDK开发

在第3小节中，通过只能由PS访问的MIO，点亮了LD4。本次实验，我们要经过AXI GPIO访问PL控制的LD0~LD3。通过查阅Zybo-Z720的TRM[3]，我们可以确定其引脚信息。

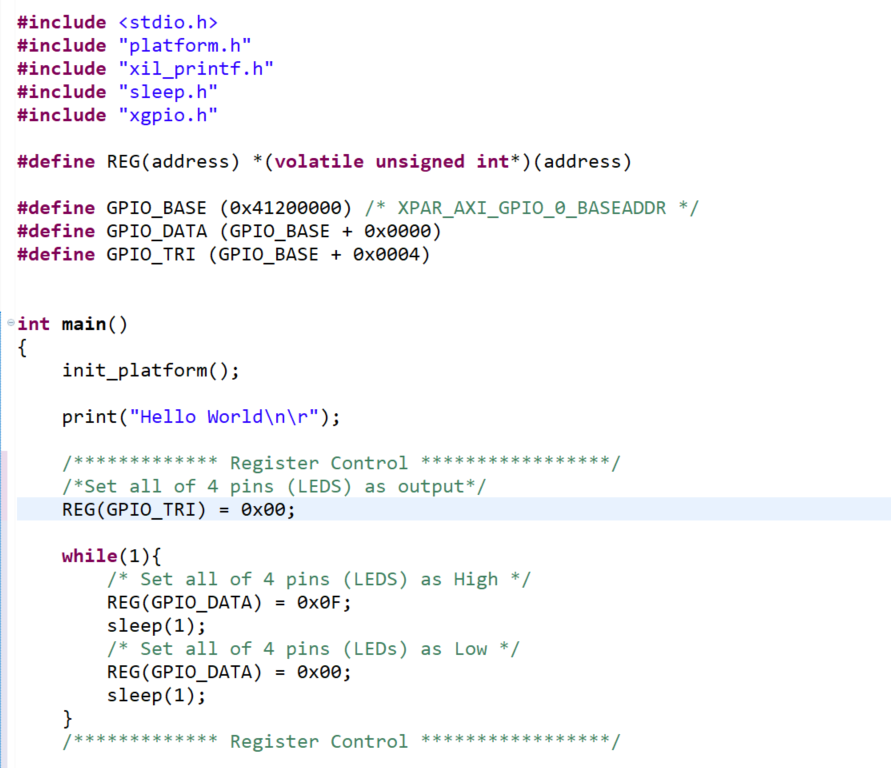


图表 8：Zybo-Z720 Basic I/O

因为我们的硬件配置全部由Vivado自动生成，我们不需要加入单独的引脚约束文件（.xdc）。所以，我们现在只需要将注意力放在C语言控制代码的编写上即可。

#### 直接寄存器访问

由于我们在图7中已经确定了axi\_gpio\_0的地址信息，采用同第3小节一样的思路，我们便可以轻易的控制LED灯啦！



#### API Driver

同理，Xilinx 为我们提供了API来方便我们的开发，相关文档的查阅已经在第3小节中详细介绍过，这里就不再过多说明了，直接上代码。



## 反思

通过本节实验，我们初步了解了AXI GPIO，通过其建立了PS与PL之间的通信，并成功点亮了PL部的LED灯。在预备知识环节我们了解到，PL与PS之间主要由9组（4组通用，4组高速，1组直接访问cache）AXI接口连接，本次实验我们仅仅用了最简单的通用AXI GPIO，对于不用的应用开发，我们可以根据需要来采用不同的AXI接口。但这并不容易，需要更进一步的学习和挖掘。

# 参考文献

[1] Xilinx, The Zynq Book, 2014

[2] Xilinx, AXI GPIO v2.0 LogiCORE IP Product Guide

[3] Digilent, Zybo-Z720 TRM