Zynq7000-AXI-GPIO 之 LED 点亮

一．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。

二．实验

1. 实验环境

Software

Windows 10 64bit
Vivado 2017.4
Xilinx SDK 2017.4

Hardware

Zybo-Z720

2. 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。

3. 预备知识

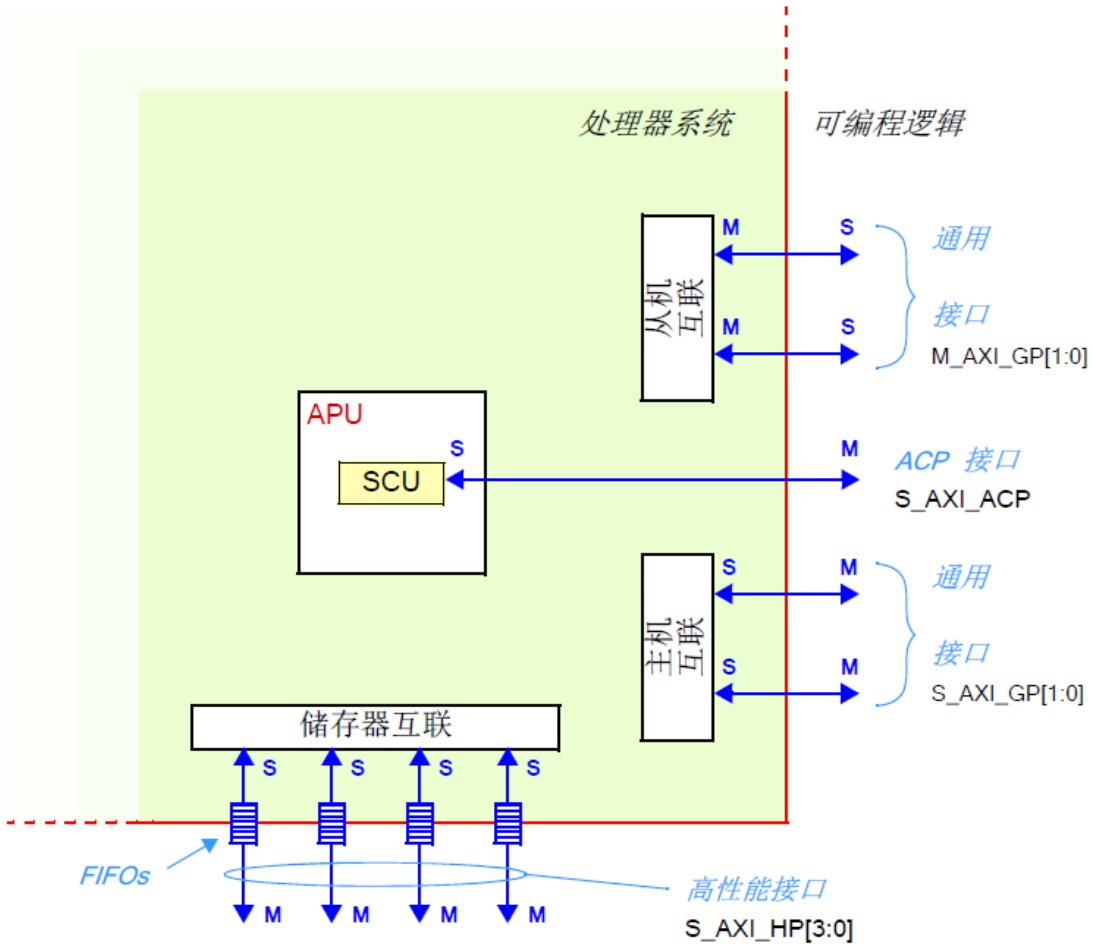
AXI 互联和接口

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。

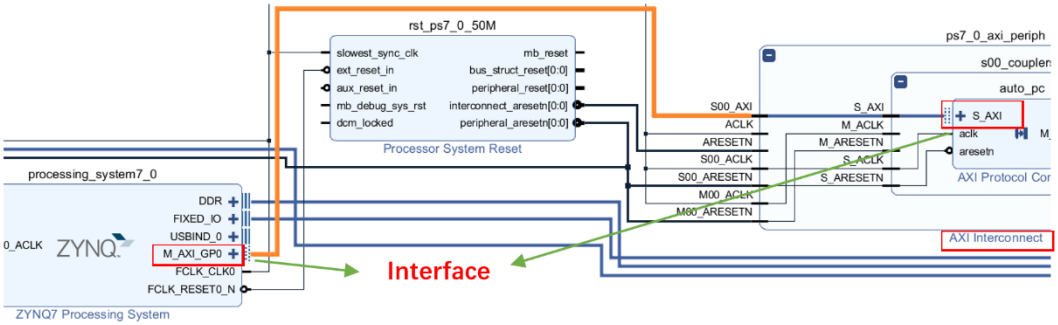
② 接口（Interface）：用于在系统内的主机和从机之间传递数据，地址和握手信号的点对点连接。

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。

接口名称	接口描述	主机	从机
M_AXI_GP0	通用 (AXI_GP)	PS	PL
M_AXI_GP1		PS	PL
S_AXI_GP0	通用 (AXI_GP)	PL	PS
S_AXI_GP1		PL	PS
S_AXI_ACP	加速器一致性端口 (ACP)，cache 一致性回话	PL	PS
S_AXI_HP0	带有读 / 写 FIFO 的高性能端口 (AXI_HP) 。（注意 AXI_HP 接口有时被称作 AXI Fifo 接口，或 AFI)。	PL	PS
S_AXI_HP1		PL	PS
S_AXI_HP2		PL	PS
S_AXI_HP3		PL	PS

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。

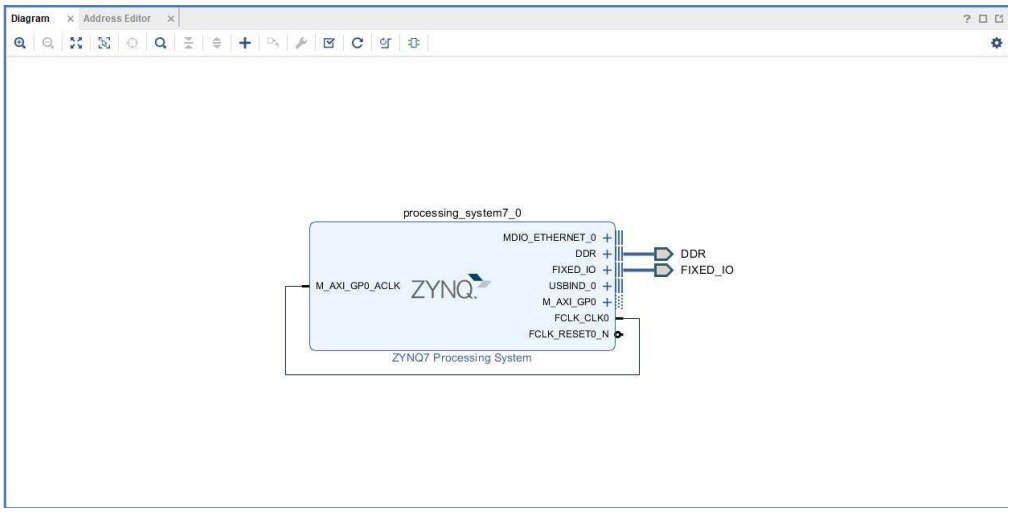
4. 实验步骤

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：

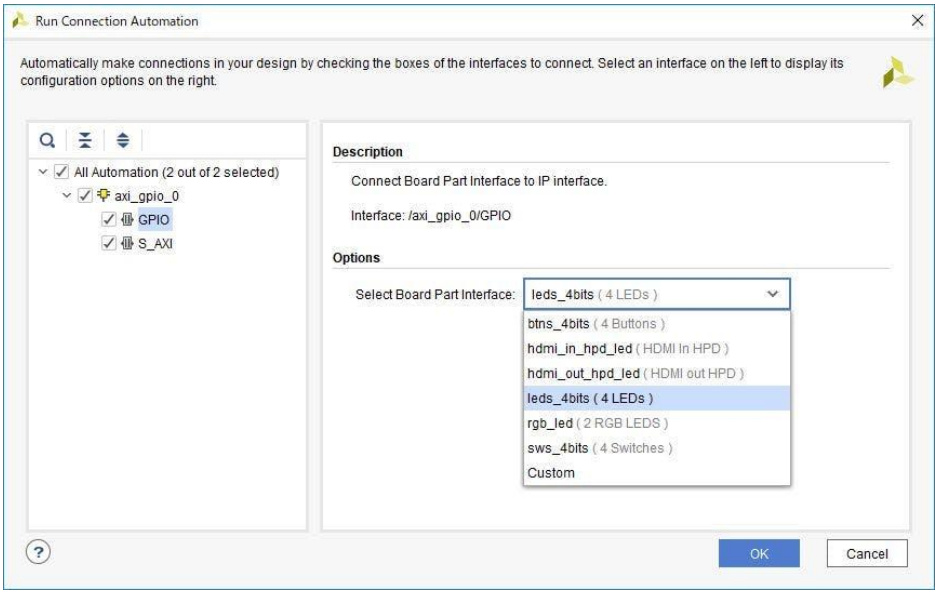


图表 3：Section3 Diagram

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：

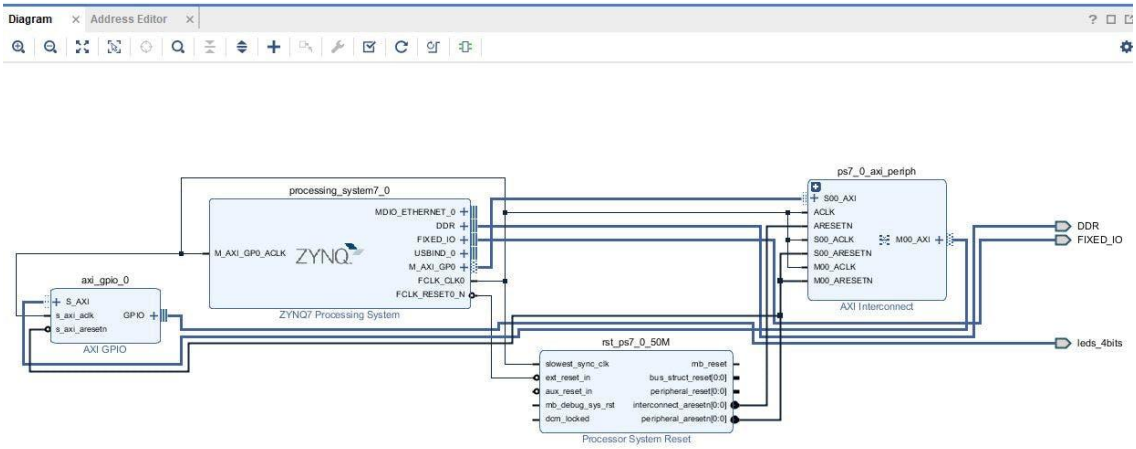
axi_gpio_0 （channel 0）

Select Board Part Interface（leds_4bits）：自动与 Zybo-7Z20 的四个 LED 灯相连接。



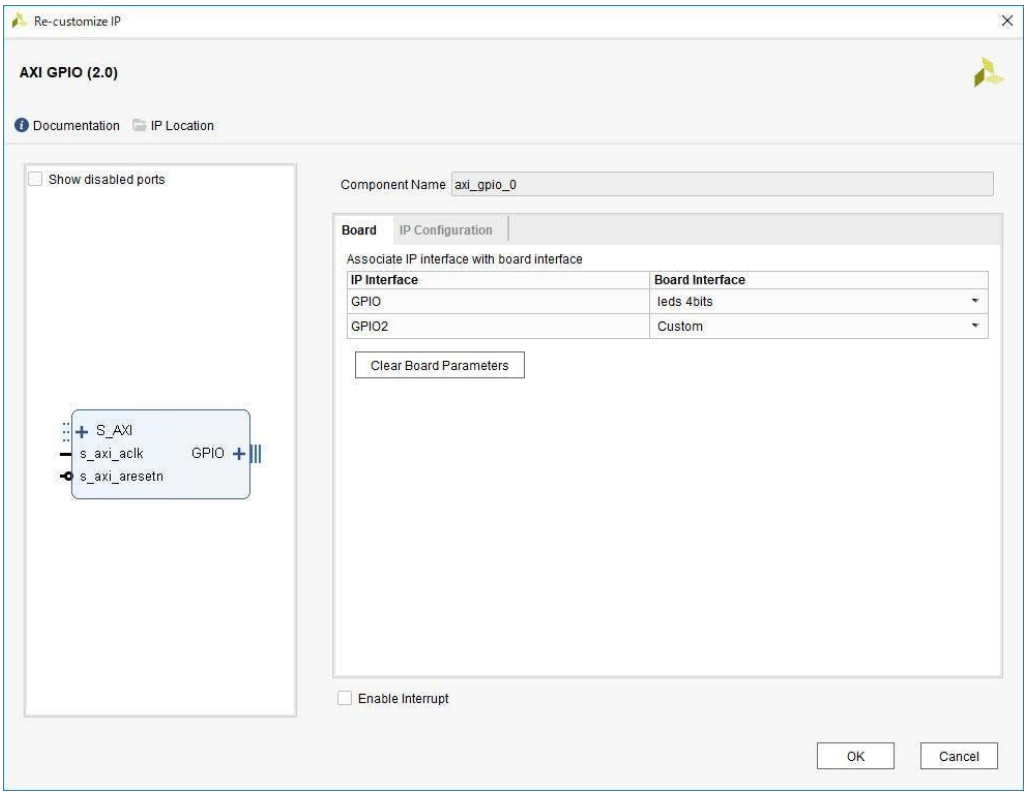
图表 4：AXI GPIO 设置

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。

13 Basic I/O

The Zybo Z7 board includes four slide switches, four push-buttons, four individual LEDs, and two tri-color LEDs connected to the Zynq PL, as shown in Figure 13.1 (the Zybo Z7-10 only has one tri-color LED). There are also two pushbuttons and one LED connected directly to the PS via MIO pins, also shown in Figure 13.1. The push-buttons and slide switches are connected to the Zynq via series resistors to prevent damage from inadvertent short circuits (a short circuit could occur if a pin assigned to a push-button or slide switch was inadvertently defined as an output). The push-buttons are "momentary" switches that normally generate a low output when they are at rest, and a high output only when they are pressed. Slide switches generate constant high or low inputs depending on their position.

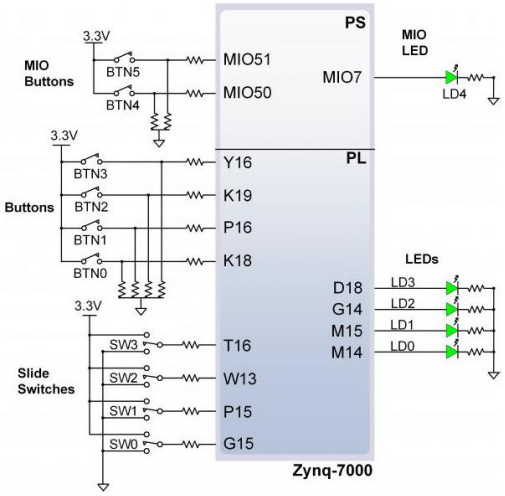


Figure 13.1. Zybo Z7 GPIO.

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！

```c
#include <stdio.h>
#include "platform.h"
#include "xil_printf.h"
#include "sleep.h"
#include "xgpio.h"

#define REG(address) *(volatile unsigned int*)(address)

#define GPIO_BASE (0x41200000) /* XPAR_AXI_GPIO_0_BASEADDR */
#define GPIO_DATA (GPIO_BASE + 0x0000)
#define GPIO_TRI (GPIO_BASE + 0x0004)


int main()
{
    init_platform();

    print("Hello World\n\r");

    /************* Register Control ****************/
    /*Set all of 4 pins (LEDS) as output*/
    REG(GPIO_TRI) = 0x00;

    while(1){
        /* Set all of 4 pins (LEDS) as High */
        REG(GPIO_DATA) = 0x0F;
        sleep(1);
        /* Set all of 4 pins (LEDs) as Low */
        REG(GPIO_DATA) = 0x00;
        sleep(1);
    }
    /************* Register Control ****************/
```

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。

```c
int main()
{
    init_platform();

    print("Hello World\n\r");

//    /************* Register Control ****************/
//    /*Set all of 4 pins (LEDS) as output*/
//    REG(GPIO_TRI) = 0x00;
//
//    while(1){
//      /* Set all of 4 pins (LEDS) as High */
//      REG(GPIO_DATA) = 0x0F;
//      sleep(1);
//      /* Set all of 4 pins (LEDs) as Low */
//      REG(GPIO_DATA) = 0x00;
//      sleep(1);
//    }
//    /************* Register Control ****************/

    /************* API Driver Control ****************/
    XGpio instXGpio;
    XGpio_Initialize(&instXGpio, XPAR_AXI_GPIO_0_DEVICE_ID);

    /* Set all of 4 pins(LEDS) as output */
    XGpio_SetDataDirection(&instXGpio,1,0);

    while(1){
        /*Set all of 4 pins(LEDs) as High*/
        XGpio_DiscreteWrite(&instXGpio,1,0x0F);
        sleep(1);
        /*Set all of 4 pins(LEDs) as Low*/
        XGpio_DiscreteWrite(&instXGpio,1,0x00);
        sleep(1);
    }
    /************* API Driver Control ****************/

    cleanup_platform();
    return 0;
}
```

5. 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。

三．参考文献

[1] Xilinx, The Zynq Book, 2014
[2] Xilinx, AXI GPIO v2.0 LogiCORE IP Product Guide
[3] Digilent, Zybo-Z720 TRM