Zynq7000-PS-GPIO 之 LED 点亮

一． 基础知识

开始本小节之前，希望你已经阅读了第 0，1，2 小节，了解了 Zynq7000 的大致架构，搭建好了完整的开发环境，掌握了 Vivado+SDK 开发套件的基本使用方法。接下来，我们开始一步步发掘 Zynq7000 的魅力吧!

二． 实验：通过 PS 的 GPIO 点亮 LED 灯

1. 实验环境

Software

Windows 10 64bit
Vivado 2017.4
Xilinx SDK 2017.4

Hardware

Zybo-Z7-20

2. 实验成果和目标

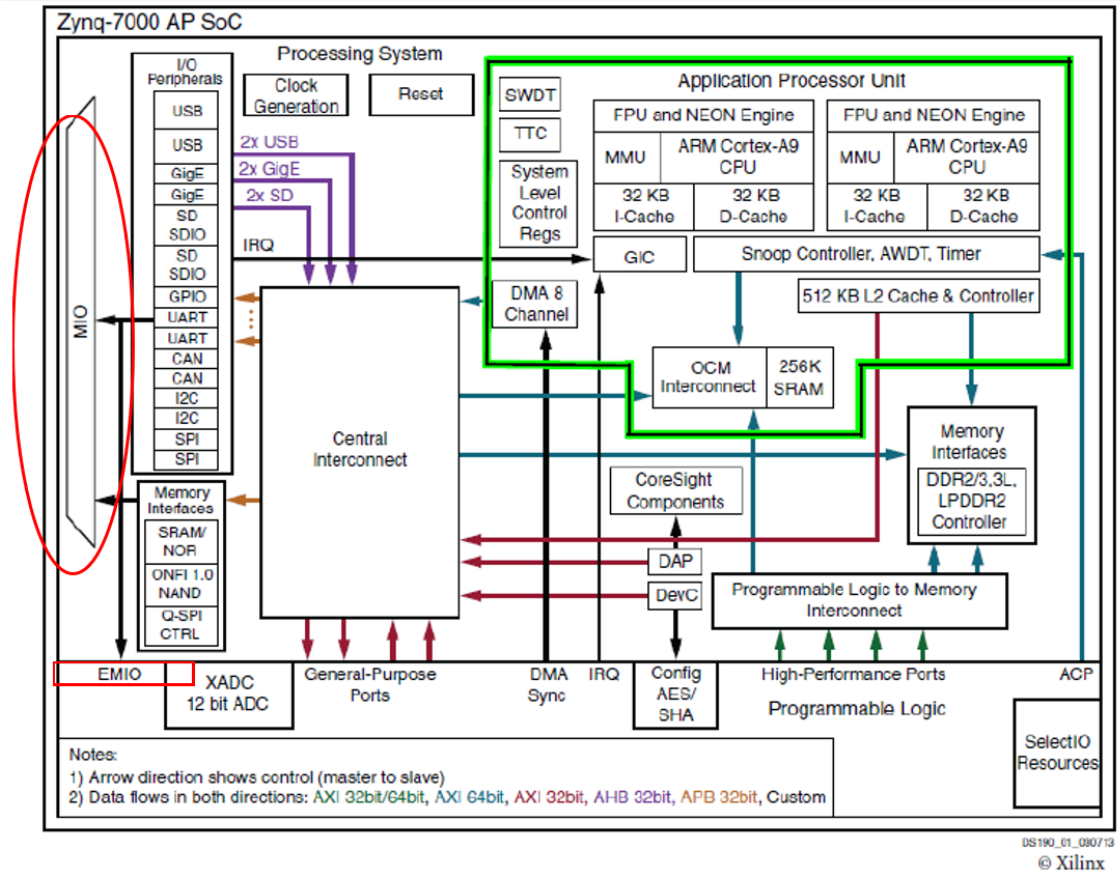
成果：分别用两种方式，直接访问寄存器和 API Driver，点亮 Zybo-Z7-20 的 LED4

目标：掌握了解 Zynq7000 和 Zybo-Z7-20 的技术手册使用方法，并了解 API 文档的查阅使用。

3. 预备知识

MIO & EMIO

Zynq 是由 PS 和 PL 两部分组成，PS 部分的 CPU 配有许多外围 I/O 接口，可以根据不同协议与外围设备通信，比如 UART,I2C,SPI,GPIO 等等。我们通过多路复用模块 MIO（Multiplexed I/O）来实现通信。注意，MIO 仅仅供 PS 部分使用。除此以外，当 PS 部分的 MIO 引脚不够用时，可以从 PL 部分“借用”一些引脚来供 PS 使用，我们称这些引脚为 EMIO（Extended Multiplexed I/O）。



图表 1：MIO 与 EMIO

4. 实验步骤

在本次实验中，我们主要采用两种方式对 PS 中 GPIO 接口进行控制。
方式 1：直接访问寄存器；方式 2：利用 Xilinx SDK Driver 中的 API

方式 1：直接访问寄存器

首先，我们需要查阅 Zynq-7000-TRM（Technology Reference Manual）技术手册，获取 PS 部 GPIO 接口的使用说明。打开 TRM 的 pdf 文件，Ctrl+F 输入“GPIO”，查找对应的章节如下。

在综述部分，我们发现 GPIO 有一系列功能供我们来开发使用，比如通过 MIO 模块引出 54 个独立可动态编程的引脚，来供我们任意配置 GPIO 为输入，输出，中断接口，采用各种通信协议（UART,I2C,SPI 等等）来与外围设备通信。或者借助 PL 部的 EMIO 模块，拓展更多的外围引脚。

XILINX

Chapter 14

General Purpose I/O (GPIO)

14.1 Introduction

The general purpose I/O (GPIO) peripheral provides software with observation and control of up to 54 device pins via the MIO module. It also provides access to 64 inputs from the Programmable Logic (PL) and 128 outputs to the PL through the EMIO interface. The GPIO is organized into four banks of registers that group related interface signals.

Each GPIO is independently and dynamically programmed as input, output, or interrupt sensing. Software can read all GPIO values within a bank using a single load instruction, or write data to one or more GPIOs (within a range of GPIOs) using a single store instruction. The GPIO control and status registers are memory mapped at base address `0xE000_A000`.

14.1.1 Features

Key features of the GPIO peripheral are summarized as follows:

- 54 GPIO signals for device pins (routed through the MIO multiplexer)
  - Outputs are 3-state capable
- 192 GPIO signals between the PS and PL via the EMIO interface
  - 64 Inputs, 128 outputs (64 true outputs and 64 output enables)
- The function of each GPIO can be dynamically programmed on an individual or group basis
- Enable, bit or bank data write, output enable and direction controls
- Programmable interrupts on individual GPIO basis
  - Status read of raw and masked interrupt
  - Selectable sensitivity: Level-sensitive (High or Low) or edge-sensitive (positive, negative, or both)

图表 2：GPIO 综述

原理分析

在官方手册中，给我们提供了一个如何配置 MIO-pin10 的例子，如下图所示：

14.3.2 GPIO Pin Configurations

Each individual GPIO pin can be configured as input/output. However, bank0 [8:7] pins must be configured as outputs. Refer to section 14.2.3 Bank0, Bits[8:7] are Outputs for further details.

Example: Configure MIO pin 10 as an output

1. **Set the direction as output:** Write `0x0000_0400` to the gpio.DIRM_0 register.
2. **Set the output enable**: Write `0x0000_0400` to the gpio.OEN_0 register.

Note: The output enable has significance only when the GPIO pin is configured as an output.

Example: Configure MIO pin 10 as an input

1. **Set the direction as input**: Write `0x0` to the gpio.DIRM_0 register. This sets gpio.DIRM_0[10] = 0.

图表 3：如何配置 GPIO（例 MIO-PIN10）

从例子中可以看出，若要将一个 MIO-PIN10 配置成为输出，第一步，将 0x0000_0400 写入 DIRM_0 寄存器，第二步，将 0x0000_0400 写入 OEN_0 寄存器。那么，我们好奇为什么把这个值写入对应的寄存器就可以将 MIO-PIN10 配置为输出 GPIO 呢?

第一步:16 进制 0x0000_0400, 低 16 位的值 0x0400 转化为 2 进制 0000_0100_0000_0000，不难发现，从低位到高位，第 10 位的值恰好为 1，对应 PIN10。

第二步：同理，对应 PIN10 位的值，置 1。

然后，我们查阅 TRM 附录 B 中对应的 GPIO（B.19）小节，如图 5 所示，GPIO 的基地址为 0xE000A000，主要相关偏移量有三个：

XGPIOPS_DATA_OFFSET：输出数据

XGPIOPS_DIRM_OFFSET：GPIO 的通信模式

XGOPOPS_OUTEN_OFFSET：输出使能信号

掌握了以上地址信息，我们便可以配置 GPIO，并通过其读取数据了。

14.3.2 GPIO Pin Configurations

Each individual GPIO pin can be configured as input/output. However, bank0 [8:7] pins must be configured as outputs. Refer to section 14.2.3 Bank0, Bits[8:7] are Outputs for further details.

Example: Configure MIO pin 10 as an output

1. **Set the direction as output:** Write 0x0000_0400 to the gpio.DIRM_0 register.

B.19 General Purpose I/O (gpio)

Module Name	General Purpose I/O (gpio)
Software Name	XGPIOPS
Base Address	0xE000A000 gpio
Description	General Purpose Input / Output
Vendor Info	

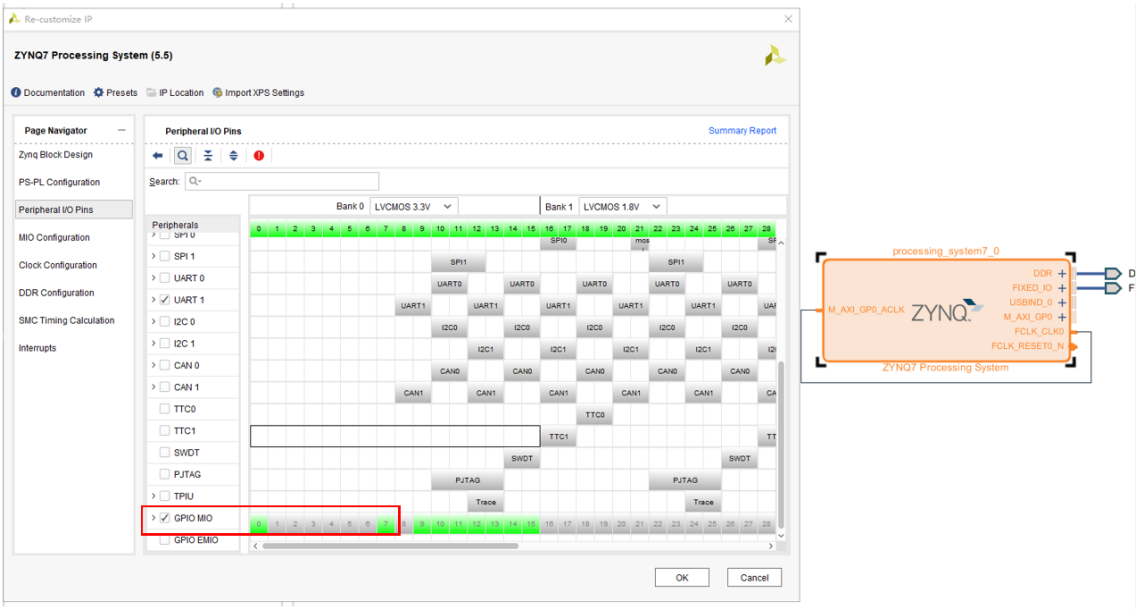
Register Summary

Register Name	Address	Width	Type	Reset Value	Description
XGPIOPS_DATA_LSW_OFFSET	0x00000000	32	mixed	x	Maskable Output Data (GPIO Bank0, MIO, Lower 16bits)
XGPIOPS_DATA_MSW_OFFSET	0x00000004	32	mixed	x	Maskable Output Data (GPIO Bank0, MIO, Upper 16bits)
MASK_DATA_1_LSW	0x00000008	32	mixed	x	Maskable Output Data (GPIO Bank1, MIO, Lower 16bits)
MASK_DATA_1_MSW	0x0000000C	22	mixed	x	Maskable Output Data (GPIO Bank1, MIO, Upper 6bits)
MASK_DATA_2_LSW	0x00000010	32	mixed	0x00000000	Maskable Output Data (GPIO Bank2, EMIO, Lower 16bits)
MASK_DATA_2_MSW	0x00000014	32	mixed	0x00000000	Maskable Output Data (GPIO Bank2, EMIO, Upper 16bits)
MASK_DATA_3_LSW	0x00000018	32	mixed	0x00000000	Maskable Output Data (GPIO Bank3, EMIO, Lower 16bits)
MASK_DATA_3_MSW	0x0000001C	32	mixed	0x00000000	Maskable Output Data (GPIO Bank3, EMIO, Upper 16bits)
XGPIOPS_DATA_OFFSET	0x00000040	32	rw	x	Output Data (GPIO Bank0, MIO)
DATA_1	0x00000044	22	rw	x	Output Data (GPIO Bank1, MIO)
DATA_2	0x00000048	32	rw	0x00000000	Output Data (GPIO Bank2, EMIO)
DATA_3	0x0000004C	32	rw	0x00000000	Output Data (GPIO Bank3, EMIO)
DATA_0_RO	0x00000060	32	ro	x	Input Data (GPIO Bank0, MIO)
DATA_1_RO	0x00000064	22	ro	x	Input Data (GPIO Bank1, MIO)
DATA_2_RO	0x00000068	32	ro	0x00000000	Input Data (GPIO Bank2, EMIO)
DATA_3_RO	0x0000006C	32	ro	0x00000000	Input Data (GPIO Bank3, EMIO)
XGPIOPS_DIRM_OFFSET	0x00000204	32	rw	0x00000000	Direction mode (GPIO Bank0, MIO)
XGPIOPS_OUTEN_OFFSET	0x00000208	32	rw	0x00000000	Output enable (GPIO Bank0, MIO)

图表 4：GPIO 寄存器详细配置

实际操作

STEP1：由于我们不需要更改硬件配置信息，本节继续沿用上一节的 HelloWord 工程。打开上一节的 HelloWord 工程，确认 Zybo-Z7-20 的外围引脚配置信息。MIO 模块的 0 号和 7 号引脚，分配为 GPIO 接口使用。



图表 5：Vivado 中查看 GPIO MIO 引脚配置

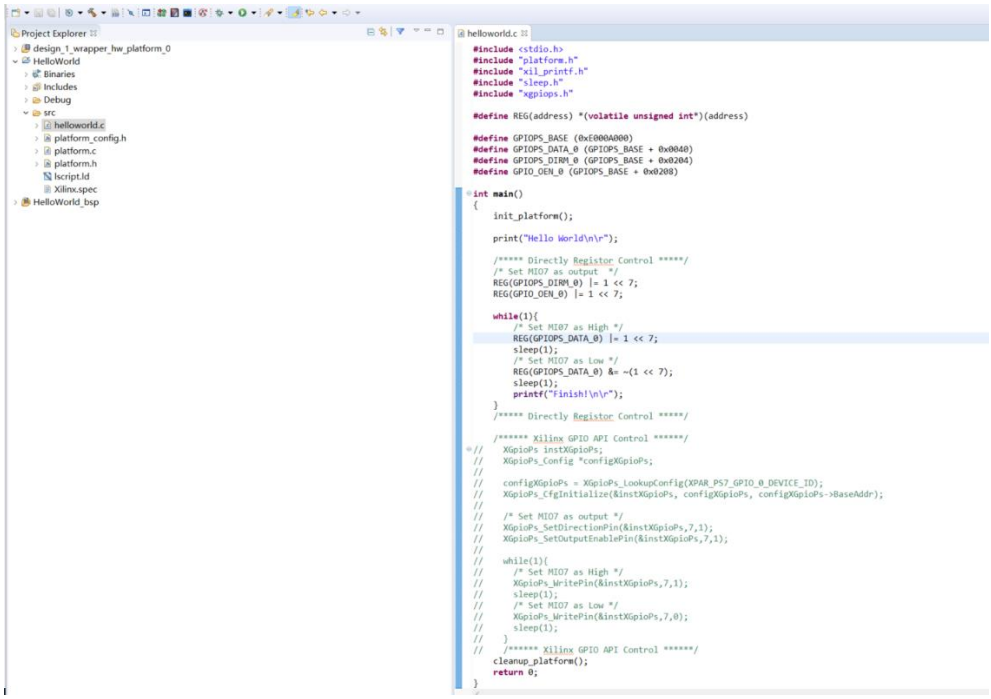
STEP2：查阅 Zybo-Z7-20 的 TRM，确认 Zynq7000 处理器的 MIO7 引脚在 Zybo-Z7-20 开发板中，与哪一个 LED 灯链接。不难发现，Zybo 开发板中，MIO7 引脚与板上的 LED4 对应。

Zybo Z7 Board Reference Manual　　DIGILENT

MIO 500 3.3 V	Peripherals		
Pin	Pmod	SPI Flash	GPIO
0	JF7		
1		CS	
2		DQ0	
3		DQ1	
4		DQ2	
5		DQ3	
6		SCLK	
7			LED4
8		SCLK FB	
9	JF8		
10	JF2		
11	JF3		
12	JF4		
13	JF1		
14	JF9		
15	JF10		

图表 6：Zybo-Z7-20 的 MIO 引脚分配

STEP3：确认好需要控制的电子器件以后，就要进行实际的代码编辑阶段了。因为 Zynq7000 的 PS 部分，由 2 个 ARM 处理构成，我们可以直接通过 C 语言进行控制。同上一节的 HelloWord 工程一样，直接在 Xilinx SDK 的 HelloWord.c 中进行代码更改即可。



图表 7：直接寄存器访问代码

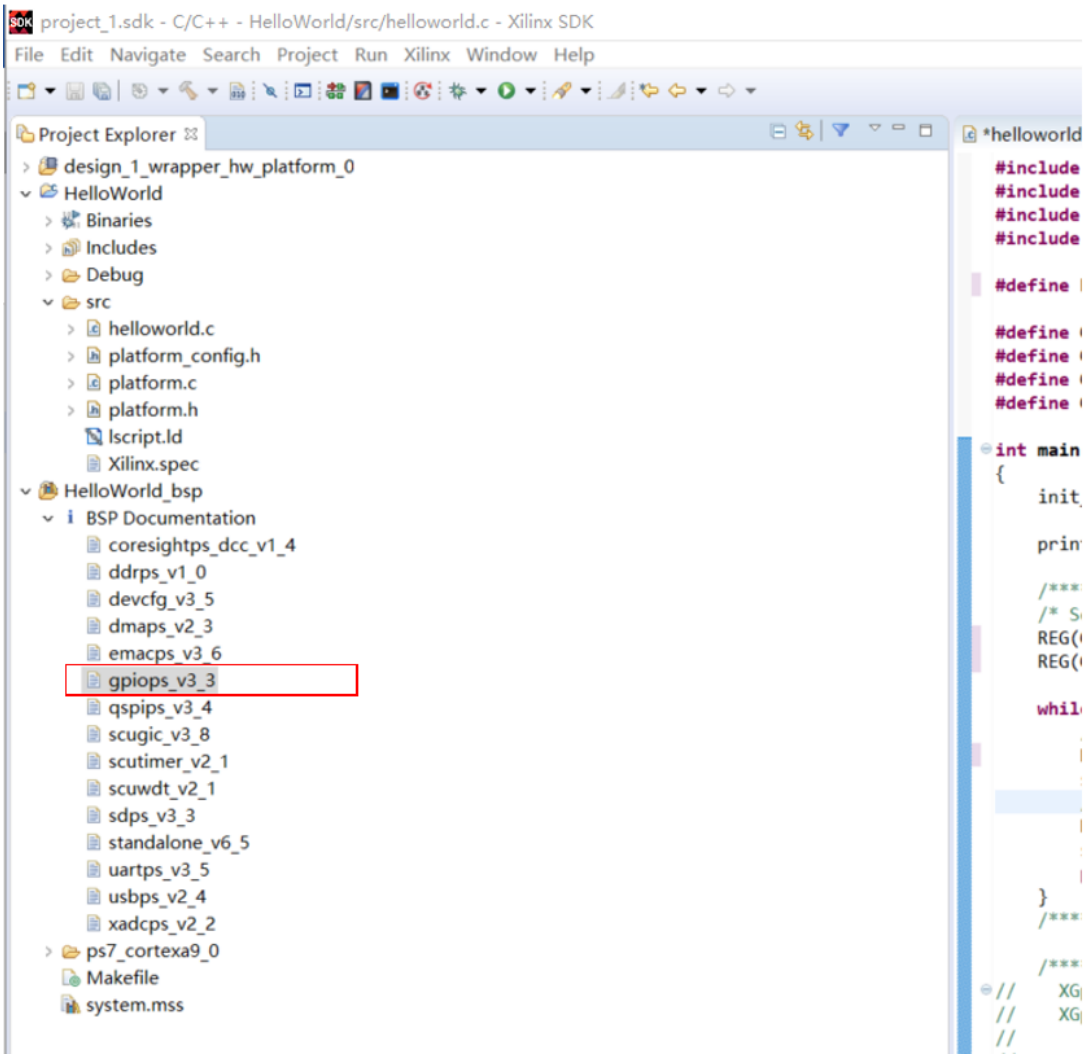
STEP4：完成实际代码编译后，同第二小节的步骤，Program FPGA Board -> Run as -> Launch on Hardware (System Debugger) -> LED4 间隔一秒进行闪烁。

方式 2：Xilinx SDK API

直接调用寄存器的方法可以很好的帮助我们看清本质，搞清楚事情的来龙去脉。我们会觉得费了九牛二虎之力仅仅点亮了一个 LED 灯，难免心中有些失落。不过，Xilinx SDK 中为我们提供了一套健全的 API，这大大的提高了我们的开发效率，实际工作中大规模项目的开发，离不开原厂 API 的使用。接下来，我们一起用新的方式去点亮 LED4 吧！

阅读 Xilinx SDK Driver API 的文档

在 Xilinx SDK 的 Project Explorer 中，此项目的 Board Support Package（BSP）文件中，可以找到 Xilinx 提供的 API 文档。本小节中，我们只涉及的 GPIO 的控制，查阅 gpiops_v3_3 文档，我们便可以知道所有的相关 API 信息。



图表 8：API 文档

读者可以根据自身需要参阅 API 文档，此小节点亮 LED4 的代码如下。按照同方式 1 的步骤，我们就可以顺利使 LED4 闪烁啦!

helloworld.c

```c
#include "platform.h"
#include "xil_printf.h"
#include "sleep.h"
#include "xgpiops.h"

#define REG(address) *(volatile unsigned int*)(address) //Register writing function

#define GPIOPS_BASE (0xE000A000)
#define GPIOPS_DATA_0 (GPIOPS_BASE + 0x0040)
#define GPIOPS_DIRM_0 (GPIOPS_BASE + 0x0204)
#define GPIO_OEN_0 (GPIOPS_BASE + 0x0208)

int main()
{
    init_platform();

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

//    /***** Directly Registor Control *****/
//    /* Set MIO7 as output  */
//    REG(GPIOPS_DIRM_0) |= 1 << 7; // Set MIO7 as output GPIO
//    REG(GPIO_OEN_0) |= 1 << 7; // Set MIO7 output enable
//
//    while(1){
//      /* Set MIO7 as High */
//      REG(GPIOPS_DATA_0) |= 1 << 7;
//      sleep(1);
//      /* Set MIO7 as Low */
//      REG(GPIOPS_DATA_0) &= ~(1 << 7);
//      sleep(1);
//      printf("Finish!\n\r");
//    }
//    /***** Directly Registor Control *****/

    /****** Xilinx GPIO API Control ******/
    XGpioPs instXGpioPs;
    XGpioPs_Config *configXGpioPs;

    configXGpioPs = XGpioPs_LookupConfig(XPAR_PS7_GPIO_0_DEVICE_ID);
    XGpioPs_CfgInitialize(&instXGpioPs, configXGpioPs, configXGpioPs->BaseAddr);

    /* Set MIO7 as output */
    XGpioPs_SetDirectionPin(&instXGpioPs,7,1);
    XGpioPs_SetOutputEnablePin(&instXGpioPs,7,1);

    while(1){
        /* Set MIO7 as High */
        XGpioPs_WritePin(&instXGpioPs,7,1);
        sleep(1);
        /* Set MIO7 as Low */
        XGpioPs_WritePin(&instXGpioPs,7,0);
        sleep(1);
    }
    /****** Xilinx GPIO API Control ******/
    cleanup_platform();
    return 0;
}
```

图表 9：API LED4 Control

5. 反思

参考图表 1 中 Zynq7000 的系统架构，除了 GPIO，与 PS 直接相连的外围 I/O 接口（Peripherals I/O）还有 I2C,SPI,UART,USB,CAN,Giga-Enthernet，既然我们已经掌握了 GPIO 的配置方法，那么同理，我们是不是同样可以举一反三的操作其它的 Peripherals I/O 了呢?

三． 参考文献

[1] Xilinx, Zynq7000 SoC Technical Reference Manual.
[2] Digilent, Zybo Z7 Board Reference Manual.
[3] Xilinx, The Zynq Book.
[4] iwatake2222, ZYBO (Zynq) 初心者ガイド (3) PS の GPIO で L チカ, https://qiita.com/iwatake2222/items/39d2ed00fa88a1204fce