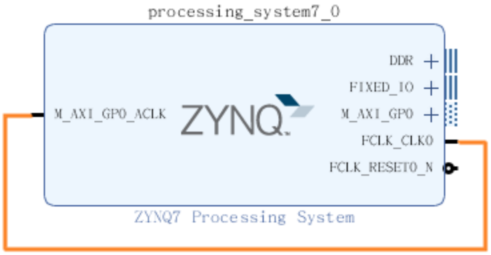
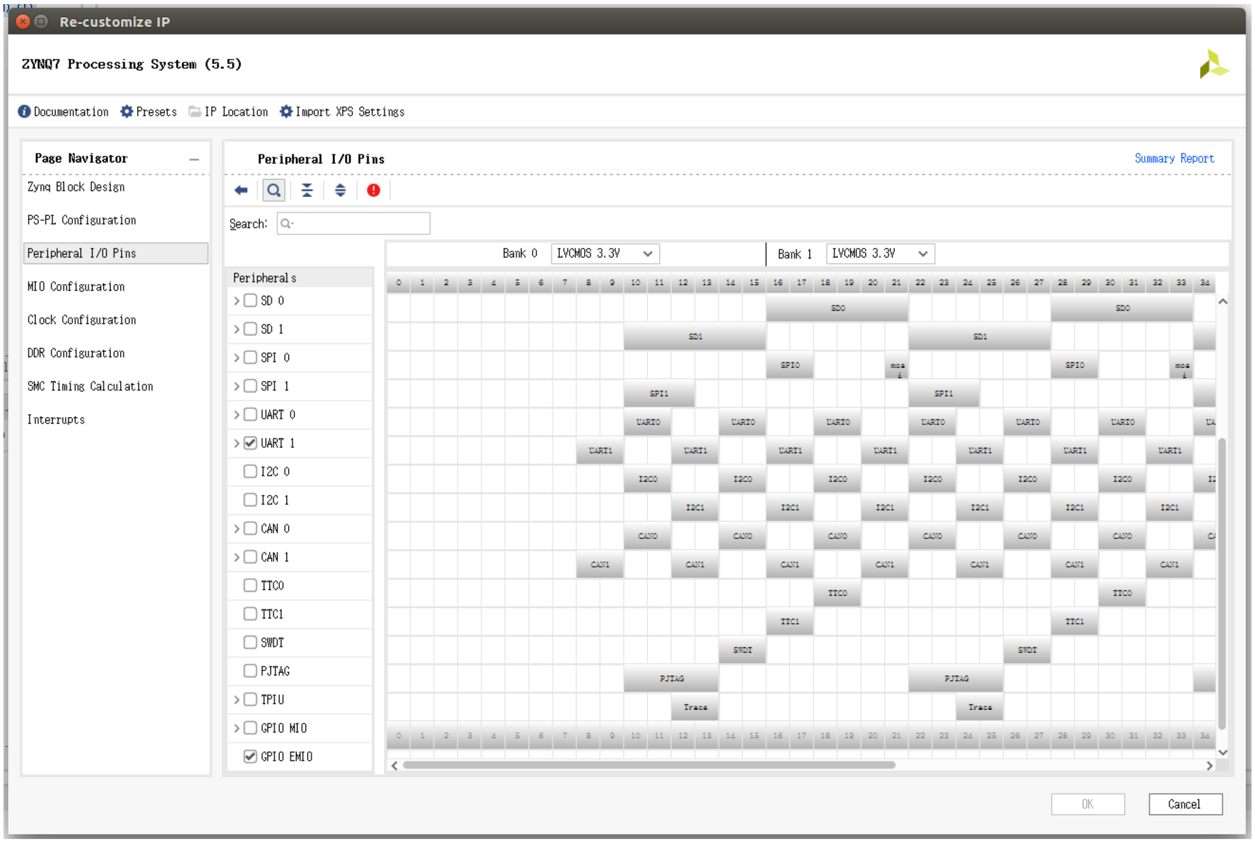
[L19]从SDK到Linux驱动 - GPIO

这里需要用到UG585和具体原理图,如果是MIO那比较容易,但是MIO很多都绑定了具体的驱动,而我们板子上接的是EMIO,比如T12接的LED就是PL一侧的,不过用起来区别也不大,创建bd图后先把两个时钟连一起.



使能EMIO外设,当然我额外使能了UART,当然DDR不要忘记.这些都是很常规的事情了,以后就不多说了.



Re-customize IP

ZYNQ7 Processing System (5.5)

Documentation | Presets | IP Location | Import XPS Settings

Page Navigator
- Zynq Block Design
- PS-PL Configuration
- Peripheral I/O Pins
- MIO Configuration
- Clock Configuration
- DDR Configuration
- SMC Timing Calculation
- Interrupts

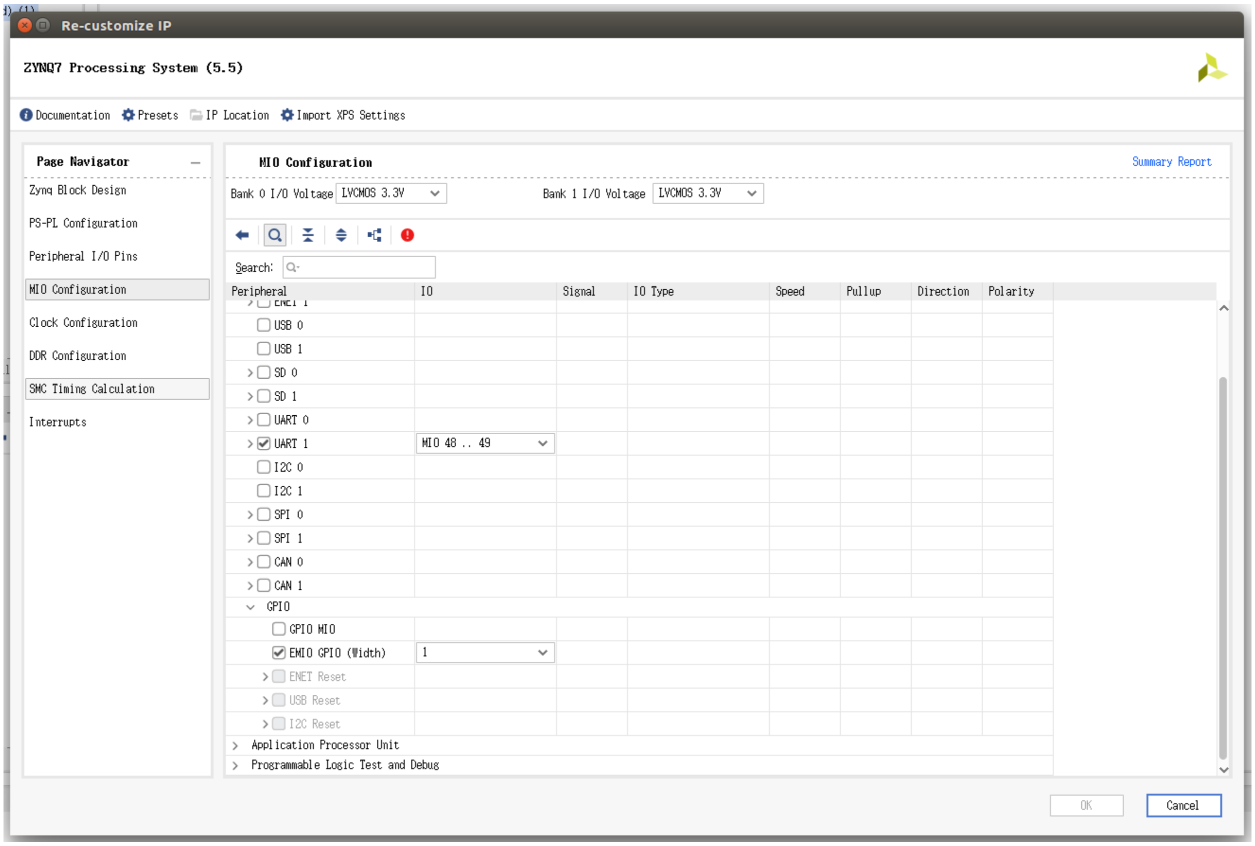
Peripheral I/O Pins | Summary Report

Search:

Bank 0 LVCMOS 3.3V | Bank 1 LVCMOS 3.3V

Peripherals
- □ SD 0
- □ SD 1
- □ SPI 0
- □ SPI 1
- □ UART 0
- ☑ UART 1
- □ I2C 0
- □ I2C 1
- □ CAN 0
- □ CAN 1
- □ TTC0
- □ TTC1
- □ SWDT
- □ PJTAG
- □ TPIU
- □ GPIO MIO
- ☑ GPIO EMIO

只绑定一个,EMIO宽度1就可以,在ZYNQ 7010中,EMIO有2条32bit的总线,具体看UG585,我们这里只用1b.(另外说一个,就算不用EMIO,自建AXI也是可以驱动的.)



Re-customize IP

ZYNQ7 Processing System (5.5)

Documentation Presets IP Location Import XPS Settings

Page Navigator

- Zynq Block Design
- PS-PL Configuration
- Peripheral I/O Pins
- MIO Configuration
- Clock Configuration
- DDR Configuration
- SMC Timing Calculation
- Interrupts

MIO Configuration Summary Report

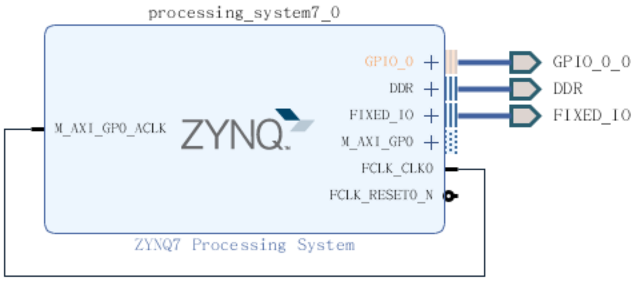
Bank 0 I/O Voltage: LVCMOS 3.3V Bank 1 I/O Voltage: LVCMOS 3.3V

Search:

Peripheral	IO	Signal	IO Type	Speed	Pullup	Direction	Polarity
□ ENET 1							
□ USB 0							
□ USB 1							
□ SD 0							
□ SD 1							
□ UART 0							
☑ UART 1	MIO 48 .. 49						
□ I2C 0							
□ I2C 1							
□ SPI 0							
□ SPI 1							
□ CAN 0							
□ CAN 1							
GPIO							
□ GPIO MIO							
☑ EMIO GPIO (Width)	1						
□ ENET Reset							
□ USB Reset							
□ I2C Reset							
Application Processor Unit							
Programmable Logic Test and Debug							

OK Cancel

之后自动连接并导出他,然后一步一步配置IO生成bitstream,最后导出SDK,应该熟悉到不能再熟悉了.



具体的很多参考代码可以看mss文件上显示的,我这里写成这样.请先自行摸索并实现.

```
/******************************************************************************
*
* Copyright (C) 2009 - 2014 Xilinx, Inc.  All rights reserved.
*
* Permission is hereby granted, free of charge, to any person obtaining a copy
* of this software and associated documentation files (the "Software"), to deal
* in the Software without restriction, including without limitation the rights
* to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
* copies of the Software, and to permit persons to whom the Software is
* furnished to do so, subject to the following conditions:
*
* The above copyright notice and this permission notice shall be included in
* all copies or substantial portions of the Software.
*
* Use of the Software is limited solely to applications:
* (a) running on a Xilinx device, or
* (b) that interact with a Xilinx device through a bus or interconnect.
*
* THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
* IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
* FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL
* XILINX  BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
* WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF
* OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
* SOFTWARE.
*
* Except as contained in this notice, the name of the Xilinx shall not be used
* in advertising or otherwise to promote the sale, use or other dealings in
* this Software without prior written authorization from Xilinx.
*
******************************************************************************/

/*
 * helloworld.c: simple test application
 *
 * This application configures UART 16550 to baud rate 9600.
 * PS7 UART (Zynq) is not initialized by this application, since
 * bootrom/bsp configures it to baud rate 115200
 *
 * ------------------------------------------------
 * | UART TYPE   BAUD RATE                        |
 * ------------------------------------------------
 *   uartns550   9600
 *   uartlite    Configurable only in HW design
 *   ps7_uart    115200 (configured by bootrom/bsp)
 */

#include <stdio.h>
#include "platform.h"
#include "xstatus.h"
#include "xgpiops.h"
#include "xil_printf.h"

#define GPIO_DEVICE_ID XPAR_XGPIOPS_0_DEVICE_ID

#define EMIO_INPUT 0
#define EMIO_OUTPUT 1

#define EMIO_OUTPUT_DIS 0
#define EMIO_OUTPUT_EN 1

#define EMIO_GPIO_LOW 0
#define EMIO_GPIO_HIGH 1

int main()
{
  XGpioPs Gpio;
  XGpioPs_Config *ConfigPtr = NULL;

    init_platform();

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

    ConfigPtr = XGpioPs_LookupConfig(GPIO_DEVICE_ID);
    if(!ConfigPtr){
      print("GPIO Lookup Failed!\n\r");
      return XST_FAILURE;
    }

    if(XGpioPs_CfgInitialize(&Gpio,ConfigPtr,ConfigPtr->BaseAddr) != XST_SUCCESS){
      print("GPIO Cfg Failed!\n\r");
      return XST_FAILURE;
    }

    XGpioPs_SetDirectionPin(&Gpio,54,EMIO_OUTPUT);
    XGpioPs_SetOutputEnablePin(&Gpio,54,EMIO_OUTPUT_EN);

    for(;;){
      XGpioPs_WritePin(&Gpio,54,EMIO_GPIO_LOW);
      usleep(1000 * 1000);
      XGpioPs_WritePin(&Gpio,54,EMIO_GPIO_HIGH);
      usleep(1000 * 1000);
    }
    cleanup_platform();
    return 0;
}
```

具体来说,要想操作GPIO,分为以下步骤.

1. XGpioPs_CfgInitialize → 开启GPIO时钟,屏蔽全部GPIO中断.
2. XGpioPs_SetDirectionPin → 设置为输出
3. XGpioPs_SetOutputEnablePin → 设置输出使能
4. XGpioPs_WritePin → 设置实际电平

写成伪代码就是这样.

```
#define SLCR_BASE_ADDR 0xF8000000
#define GPIO_BASE_ADDR 0xE000A000

#define APER_CLK_CTRL (SLCR_BASE_ADDR + 0x0000012C)

#define GPIO_DATA_2   (GPIO_BASE_ADDR + 0x00000048)
#define GPIO_DIRM_2   (GPIO_BASE_ADDR + 0x00000284)
#define GPIO_OUTEN_2  (GPIO_BASE_ADDR + 0x00000288)
#define GPIO_INTDIS_2 (GPIO_BASE_ADDR + 0x00000294)

#define GPIO_CLK_EN 0x40000000

#define EMIO_PIN 0x00000001

#define EMIO_INPUT 0
#define EMIO_OUTPUT 1

#define EMIO_OUTPUT_DIS 0
#define EMIO_OUTPUT_EN 1

#define EMIO_GPIO_LOW 0
#define EMIO_GPIO_HIGH 1

// 初始化
APER_CLK_CTRL |= GPIO_CLK_EN;
GPIO_INTDIS_2 |= EMIO_PIN;
GPIO_DIRM_2   |= EMIO_PIN;
GPIO_OUTEN_2  |= EMIO_PIN;

// 点亮LED
GPIO_DATA_2   |= EMIO_PIN;

// 熄灭LED
GPIO_DATA_2   &= ~EMIO_PIN;
```

对于Linux来说,任何设备都是一个字符,因此,我可以按照这个想法做一个简单的字符设备驱动,实际和直接操作寄存器没什么区别.

```
#include <linux/types.h>
#include <linux/kernel.h>
#include <linux/delay.h>
#include <linux/ide.h>
#include <linux/init.h>
#include <linux/module.h>
#include <linux/errno.h>
#include <linux/gpio.h>
#include <asm/mach/map.h>
#include <asm/uaccess.h>
#include <asm/io.h>
#include <linux/cdev.h>

#define KERNEL_LED_DEVIE_CNT 1
#define KERNEL_LED_NAME "kernel_led"

#define APER_CLK_CTRL 0xF800012C

#define GPIO_DATA_2   0xE000A048
#define GPIO_DIRM_2   0xE000A284
#define GPIO_OUTEN_2  0xE000A288
```

```
#define GPIO_INTDIS_2 0xE000A294

#define GPIO_CLK_EN (0x1U << 22)

#define EMIO_PIN 0x00000001

#define EMIO_INPUT 0
#define EMIO_OUTPUT 1

#define EMIO_OUTPUT_DIS 0
#define EMIO_OUTPUT_EN 1

#define EMIO_GPIO_LOW 0
#define EMIO_GPIO_HIGH 1

static void __iomem *DATA;
static void __iomem *DIRM;
static void __iomem *OUTEN;
static void __iomem *INTDIS;
static void __iomem *APER;

struct kernel_led_dev
{
    dev_t devid;
    struct cdev cdev;
    struct class *class;
    struct device *device;
    int major;
    int minor;
};

static struct kernel_led_dev dev;

/* 设备打开时候会被调用 */
static int led_open(struct inode *inode,struct file *filp){
    filp->private_data = &dev;

    return 0;
}

/* 设备读取时候会被调用 */
static ssize_t led_read(struct file *filp,char __user *buf,size_t cnt,loff_t *offset){
    int ret;
    char kbuf[1];

    if (cnt != 1){
        return -EFAULT;
    }

    kbuf[0] = readl(DATA) & EMIO_PIN;
    ret = copy_to_user(buf,kbuf,cnt);
    if(ret){
        /* 复制失败了 */
        return -EFAULT;
    }

    return 0;
}

/* 设备写入时候会被调用 */
static ssize_t led_write(struct file *filp,const char __user *buf,size_t cnt,loff_t *offset){
    int ret;
    int val;
    char kbuf[1];

    if (cnt != 1){
        return -EFAULT;
    }

    ret = copy_from_user(kbuf,buf,cnt);
    if(ret){
        /* 复制失败了 */
        return -EFAULT;
    }

    val = readl(DATA);
    if(kbuf[0] == 0){
        val &= ~EMIO_PIN;
    }else{
        val |= EMIO_PIN;
    }
    writel(val,DATA);

    return 0;
}

/* 设备释放时候会被调用 */
static int led_release(struct inode *inode,struct file *filp){
    return 0;
}

static struct file_operations fops =
{
    .owner = THIS_MODULE,
    .open = led_open,
    .read = led_read,
    .write = led_write,
    .release = led_release,
};

static int __init led_init(void){
    int val;
    int ret;

    /* 寄存器映射 */
    DATA = ioremap(GPIO_DATA_2,4);
    DIRM = ioremap(GPIO_DIRM_2,4);
    OUTEN = ioremap(GPIO_OUTEN_2,4);
    INTDIS = ioremap(GPIO_INTDIS_2,4);
    APER = ioremap(APER_CLK_CTRL,4);

    /* 初始化 */
    val = readl(APER);
    val |= GPIO_CLK_EN;
    writel(val,APER);

    val = readl(INTDIS);
    val |= EMIO_PIN;
    writel(val,INTDIS);

    val = readl(DIRM);
    val |= EMIO_PIN;
    writel(val,DIRM);

    val = readl(OUTEN);
    val |= EMIO_PIN;
    writel(val,OUTEN);

    val = readl(DATA);
    val |= EMIO_PIN;
    writel(val,DATA);

    printk("APER reg 0x%08x\n",readl(APER));
    printk("INTDIS reg 0x%08x\n",readl(INTDIS));
    printk("DIRM reg 0x%08x\n",readl(DIRM));
    printk("OUTEN reg 0x%08x\n",readl(OUTEN));
    printk("DATA reg 0x%08x\n",readl(DATA));

    /* 申请一个设备号 */
    ret = alloc_chrdev_region(&dev.devid,0,KERNEL_LED_DEVIE_CNT,KERNEL_LED_NAME);
    if(ret){
        goto alloc_fail;
    }
    dev.major = MAJOR(dev.devid);
    dev.minor = MINOR(dev.devid);

    dev.cdev.owner = THIS_MODULE;
    cdev_init(&dev.cdev,&fops);

    ret = cdev_add(&dev.cdev,dev.devid,KERNEL_LED_DEVIE_CNT);
    if(ret){
        goto add_fail;
    }

    dev.class = class_create(THIS_MODULE,KERNEL_LED_NAME);
    if(IS_ERR(dev.class)){
        ret = PTR_ERR(dev.class);
        goto class_fail;
    }

    dev.device = device_create(dev.class,NULL,dev.devid,NULL,KERNEL_LED_NAME);
    if(IS_ERR(dev.device)){
        ret = PTR_ERR(dev.class);
        goto dev_fail;
    }

    return 0;

dev_fail:
    class_destroy(dev.class);

class_fail:
    cdev_del(&dev.cdev);

add_fail:
    unregister_chrdev_region(dev.devid,KERNEL_LED_DEVIE_CNT);

alloc_fail:
    /* 寄存器取消映射 */
    iounmap(DATA);
    iounmap(DIRM);
    iounmap(OUTEN);
    iounmap(INTDIS);
    iounmap(APER);
    return ret;
}

static void __exit led_exit(void){
    device_destroy(dev.class,dev.devid);

    class_destroy(dev.class);

    cdev_del(&dev.cdev);

    unregister_chrdev_region(dev.devid,KERNEL_LED_DEVIE_CNT);

    iounmap(DATA);
    iounmap(DIRM);
    iounmap(OUTEN);
    iounmap(INTDIS);
    iounmap(APER);
}

module_init(led_init);
module_exit(led_exit);

MODULE_AUTHOR("Taterli <admin@taterli.com>");
MODULE_DESCRIPTION("Led GPIO");
MODULE_LICENSE("GPL");
```

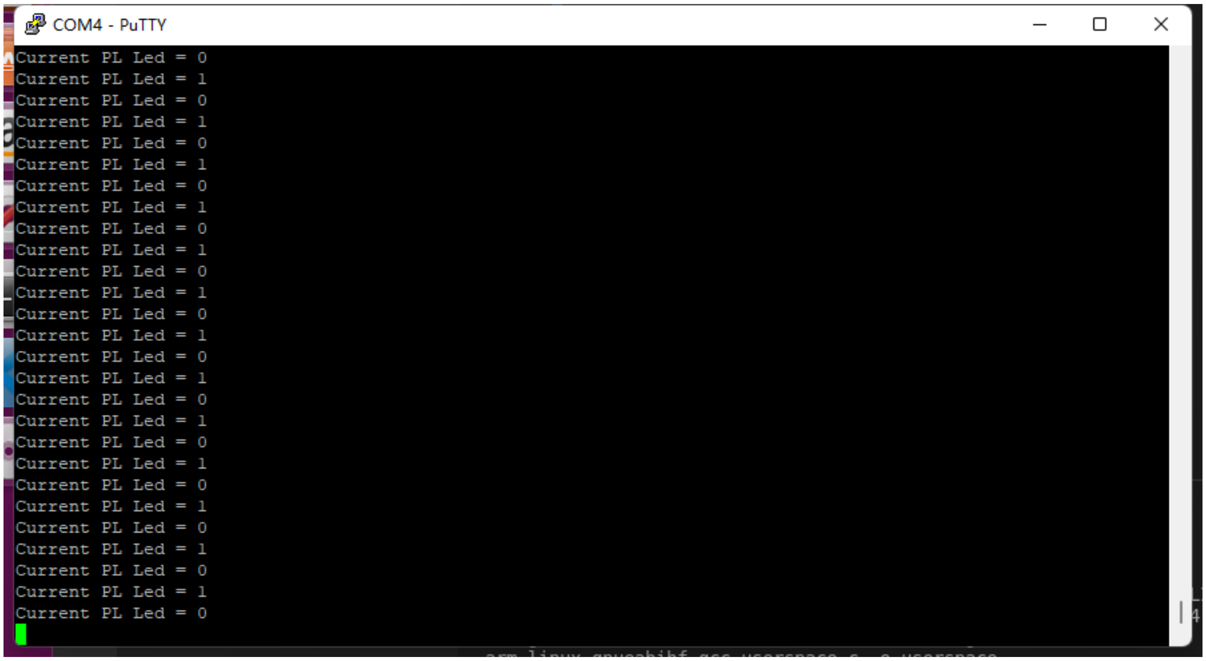
然后编译出ko模块,最后还要重制BOOT.bin文件,然后再写一个用户端的测试程序.

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdlib.h>
#include <string.h>

int main(int argc,char **argv){
    int fd,ret;
    char buf[1];

    fd = open("/dev/kernel_led",O_RDWR);
    if(fd < 0){
        return -1;
    }

    for(;;){
        buf[0] = 0;
        ret = write(fd,buf,1);
        if(ret < 0){
            return -2;
        }

        ret = read(fd,buf,1);
        if(ret < 0){
            return -3;
        }
        printf("Current PL Led = %d \n\r",buf[0]);

        usleep(1000 * 1000);

        buf[0] = 1;
        ret = write(fd,buf,1);
        if(ret < 0){
            return -2;
        }

        ret = read(fd,buf,1);
        if(ret < 0){
            return -3;
        }
        printf("Current PL Led = %d \n\r",buf[0]);

        usleep(1000 * 1000);
    }
}
```

编译Makefile参考:

```
KERN_DIR:=/home/taterli/PYNQ/sdbuild/build/TATERLI-Z7/petalinux_project/build/tmp/work/plnx_zynq7-xilinx-linux-gnueabi/linux-xlnx/4.14-xili

obj-m:=kernel_led.o

all:
  make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- -C $(KERN_DIR) M=`pwd` modules
  arm-linux-gnueabihf-gcc userspace.c -o userspace

clean:
  make -C $(KERN_DIR) M=`pwd` clean
  rm userspace
```

执行起来和想的一样,LED闪烁且能读取回来.

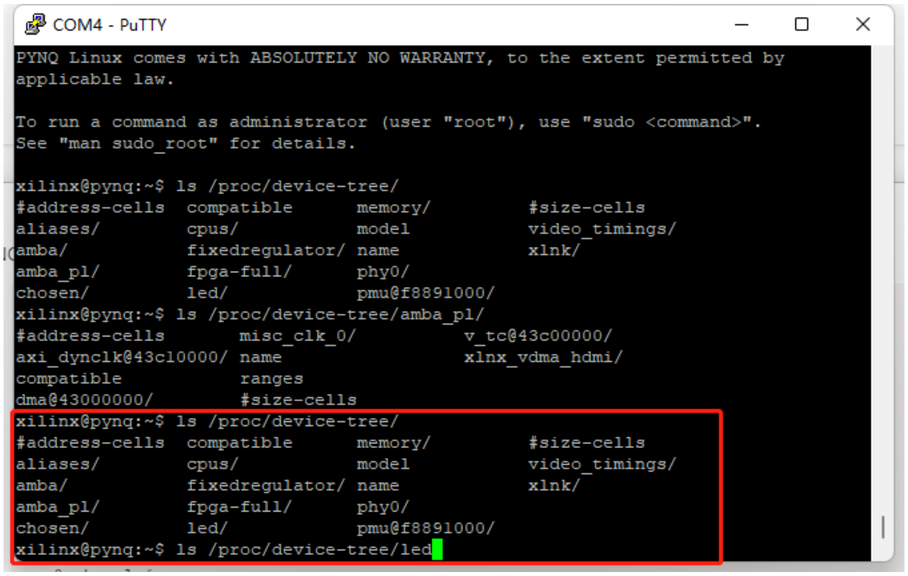


```
Current PL Led = 0
Current PL Led = 1
Current PL Led = 0
Current PL Led = 1
Current PL Led = 0
Current PL Led = 1
Current PL Led = 0
Current PL Led = 1
Current PL Led = 0
Current PL Led = 1
Current PL Led = 0
Current PL Led = 1
Current PL Led = 0
Current PL Led = 1
Current PL Led = 0
Current PL Led = 1
Current PL Led = 0
Current PL Led = 1
Current PL Led = 0
Current PL Led = 1
Current PL Led = 0
Current PL Led = 1
Current PL Led = 0
Current PL Led = 1
Current PL Led = 0
Current PL Led = 1
Current PL Led = 0
```

虽然现在可以操作了,但是其实非常Old School,现在不都主流设备树吗,所以必须迁移,所以首先要给设备树设置一个节点,现在把下面的设备树描述加入system-user.dtsi文件根节点中并重新构建/烧录镜像,下面给出的是完整示例,根据你的实际情况调整.

```
/include/ "system-conf.dtsi"

#define GPIO_ACTIVE_HIGH 0
#define GPIO_ACTIVE_LOW  1

/ {
    model = "Navigator Development Board";
    compatible = "zynq7010,zynq-7020","xlnx,zynq-7000";


    usb_phy0: phy0 {
        compatible = "ulpi-phy";
        #phy-cells = <0>;
        reg = <0xe0002000 0x1000>;
        view-port = <0x170>;
        drv-vbus;
    };

    video_timings {
            timing_4x3_480x272: timing0 {
                clock-frequency = <9000000>;
                hactive = <480>;
                vactive = <272>;

                hback-porch = <40>;
                hsync-len = <20>;
                hfront-porch = <5>;
                vback-porch = <8>;
                vsync-len = <3>;
                vfront-porch = <8>;

                hsync-active = <0>;
                vsync-active = <0>;
                de-active = <1>;
                pixelclk-active = <0>;
            };

            timing_1920x1080: timing1 {
                    clock-frequency = <148500000>;
                    hactive = <1920>;
                    vactive = <1080>;

                    hback-porch = <148>;
                    hsync-len = <44>;
                    hfront-porch = <88>;
                    vback-porch = <36>;
                    vsync-len = <5>;
                    vfront-porch = <4>;

                    hsync-active = <0>;
                    vsync-active = <0>;
                    de-active = <1>;
                    pixelclk-active = <1>;
            };
    };

    led {
        compatible = "taterli,led";
        status = "okay";
        default-state = "on";

        reg = <0xF800012C 0x4
            0xE000A048 0x4
            0xE000A284 0x4
            0xE000A288 0x4
            0xE000A294 0x4
            >;
    };
};

&usb0{
  dr_mode = "host";
        usb-phy = <&usb_phy0>;
      };

&axi_dynclk_0 {
    compatible = "digilent,axi-dynclk";
    clocks = <&clkc 15>;
    #clock-cells = <0>;
};

&v_tc_0 {
    compatible = "xlnx,v-tc-5.01.a";
};

&amba_pl {
    xlnx_vdma_hdmi {
        compatible = "xilinx,vdmafb";
        status = "okay";

        xlnx,vtc = <&v_tc_0>;
        clocks = <&axi_dynclk_0>;
        clock-names = "hdmi_pclk";
        dmas = <&axi_vdma_0 0>;
        dma-names = "hdmi_vdma";

        is-hdmi = <0x1>;

        display-timings = <&timing_1920x1080>;
        xlnx,pixel-format = "bgr888";
    };
};
```

启动后就能看到根下的led节点了.



```
PYNQ Linux comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

xilinx@pynq:~$ ls /proc/device-tree/
#address-cells  compatible      memory/         #size-cells
aliases/        cpus/           model           video_timings/
amba/           fixedregulator/ name            xlnk/
amba_pl/        fpga-full/      phy0/
chosen/         led/            pmu@f8891000/
xilinx@pynq:~$ ls /proc/device-tree/amba_pl/
#address-cells       misc_clk_0/          v_tc@43c00000/
axi_dynclk@43c10000/ name                 xlnx_vdma_hdmi/
compatible           ranges
dma@43000000/        #size-cells
xilinx@pynq:~$ ls /proc/device-tree/
#address-cells  compatible      memory/         #size-cells
aliases/        cpus/           model           video_timings/
amba/           fixedregulator/ name            xlnk/
amba_pl/        fpga-full/      phy0/
chosen/         led/            pmu@f8891000/
xilinx@pynq:~$ ls /proc/device-tree/led
```

然后代码也能轻松改成dts获取寄存器的方式,主要修改是引入头文件,获取dts参数过程,dts方式专用的MMU映射.(驱动中未实现status和default-state,希望大家实现并实验后再继续后续的过程!)

```
#include <linux/types.h>
#include <linux/kernel.h>
#include <linux/delay.h>
#include <linux/ide.h>
#include <linux/init.h>
#include <linux/module.h>
#include <linux/errno.h>
#include <linux/gpio.h>
#include <asm/mach/map.h>
#include <asm/uaccess.h>
#include <asm/io.h>
#include <linux/cdev.h>
#include <linux/of.h> /* dts操作相关 */
#include <linux/of_address.h> /* dts地址相关 */

#define KERNEL_LED_DEVIE_CNT 1
#define KERNEL_LED_NAME "kernel_led"

#define GPIO_CLK_EN (0x1U << 22)
```

```
#define EMIO_PIN 0x00000001

#define EMIO_INPUT 0
#define EMIO_OUTPUT 1

#define EMIO_OUTPUT_DIS 0
#define EMIO_OUTPUT_EN 1

#define EMIO_GPIO_LOW 0
#define EMIO_GPIO_HIGH 1

static void __iomem *DATA;
static void __iomem *DIRM;
static void __iomem *OUTEN;
static void __iomem *INTDIS;
static void __iomem *APER;

struct kernel_led_dev
{
    dev_t devid;
    struct cdev cdev;
    struct class *class;
    struct device *device;
    int major;
    int minor;
    struct device_node *nd; /* 设备节点 */
};

static struct kernel_led_dev dev;

/* 设备打开时候会被调用 */
static int led_open(struct inode *inode,struct file *filp){
    filp->private_data = &dev;

    return 0;
}

/* 设备读取时候会被调用 */
static ssize_t led_read(struct file *filp,char __user *buf,size_t cnt,loff_t *offset){
    int ret;
    char kbuf[1];

    if (cnt != 1){
        return -EFAULT;
    }

    kbuf[0] = readl(DATA) & EMIO_PIN;
    ret = copy_to_user(buf,kbuf,cnt);
    if(ret){
        /* 复制失败了 */
        return -EFAULT;
    }

    return 0;
}

/* 设备写入时候会被调用 */
static ssize_t led_write(struct file *filp,const char __user *buf,size_t cnt,loff_t *offset){
    int ret;
    int val;
    char kbuf[1];

    if (cnt != 1){
        return -EFAULT;
    }

    ret = copy_from_user(kbuf,buf,cnt);
    if(ret){
        /* 复制失败了 */
        return -EFAULT;
    }

    val = readl(DATA);
    if(kbuf[0] == 0){
        val &= ~EMIO_PIN;
    }else{
        val |= EMIO_PIN;
    }

    writel(val,DATA);
    return 0;
}

/* 设备释放时候会被调用 */
static int led_release(struct inode *inode,struct file *filp){
    return 0;
}

static struct file_operations fops =
{
    .owner = THIS_MODULE,
    .open = led_open,
    .read = led_read,
    .write = led_write,
    .release = led_release,
};

static int __init led_init(void){
    const char *str;
    int val;
    int ret;

    /* 新增的从dts获取数据的过程 */
    dev.nd = of_find_node_by_path("/led");
    if(dev.nd == NULL){
        return -EINVAL;
    }

    ret = of_property_read_string(dev.nd,"compatible",&str);
    if(ret < 0){
        return -EINVAL;
    }

    if(strcmp(str,"taterli,led")){
        return -EINVAL;
    }

    /* 寄存器映射 */
    APER = of_iomap(dev.nd,0);
    DATA = of_iomap(dev.nd,1);
    DIRM = of_iomap(dev.nd,2);
    OUTEN = of_iomap(dev.nd,3);
    INTDIS = of_iomap(dev.nd,4);

    /* 初始化 */
    val = readl(APER);
    val |= GPIO_CLK_EN;
    writel(val,APER);

    val = readl(INTDIS);
    val |= EMIO_PIN;
    writel(val,INTDIS);

    val = readl(DIRM);
    val |= EMIO_PIN;
    writel(val,DIRM);

    val = readl(OUTEN);
    val |= EMIO_PIN;
    writel(val,OUTEN);

    val = readl(DATA);
    val |= EMIO_PIN;
    writel(val,DATA);

    printk("APER reg 0x%08x\n",readl(APER));
    printk("INTDIS reg 0x%08x\n",readl(INTDIS));
    printk("DIRM reg 0x%08x\n",readl(DIRM));
    printk("OUTEN reg 0x%08x\n",readl(OUTEN));
    printk("DATA reg 0x%08x\n",readl(DATA));

    /* 申请一个设备号 */
    ret = alloc_chrdev_region(&dev.devid,0,KERNEL_LED_DEVIE_CNT,KERNEL_LED_NAME);
    if(ret){
        goto alloc_fail;
    }
    dev.major = MAJOR(dev.devid);
    dev.minor = MINOR(dev.devid);

    dev.cdev.owner = THIS_MODULE;
    cdev_init(&dev.cdev,&fops);

    ret = cdev_add(&dev.cdev,dev.devid,KERNEL_LED_DEVIE_CNT);
    if(ret){
        goto add_fail;
    }

    dev.class = class_create(THIS_MODULE,KERNEL_LED_NAME);
    if(IS_ERR(dev.class)){
        ret = PTR_ERR(dev.class);
        goto class_fail;
    }

    dev.device = device_create(dev.class,NULL,dev.devid,NULL,KERNEL_LED_NAME);
    if(IS_ERR(dev.device)){
        ret = PTR_ERR(dev.class);
        goto dev_fail;
    }

    return 0;

dev_fail:
    class_destroy(dev.class);

class_fail:
    cdev_del(&dev.cdev);

add_fail:
    unregister_chrdev_region(dev.devid,KERNEL_LED_DEVIE_CNT);

alloc_fail:
    /* 寄存器取消映射 */
    iounmap(DATA);
    iounmap(DIRM);
    iounmap(OUTEN);
    iounmap(INTDIS);
    iounmap(APER);
    return ret;
}

static void __exit led_exit(void){
    device_destroy(dev.class,dev.devid);

    class_destroy(dev.class);

    cdev_del(&dev.cdev);

    unregister_chrdev_region(dev.devid,KERNEL_LED_DEVIE_CNT);

    iounmap(DATA);
    iounmap(DIRM);
    iounmap(OUTEN);
    iounmap(INTDIS);
    iounmap(APER);
}

module_init(led_init);
module_exit(led_exit);

MODULE_AUTHOR("Taterli <admin@taterli.com>");
MODULE_DESCRIPTION("Led GPIO");
MODULE_LICENSE("GPL");
```

但是这样还是有问题,毕竟我们真的不想什么外设都这样查手册挨个戳寄存器,那要请出GPIO子系统,在讨论他之前,先试试把dts换成EMIO IO0,这里大家可以思考下为什么是IO54,另外Zynq是不用设置pinctl(IO特性复用等等)的,因为他在fsbl做了,也就是我们的bd图上做了.

```
led {
    compatible = "taterli,led";
    status = "okay";
    default-state = "on";

    led-gpio = <&gpio0 54 GPIO_ACTIVE_HIGH>;
}
```

驱动也简单很多,不需要再看寄存器了.

```
#include <linux/types.h>
#include <linux/kernel.h>
#include <linux/delay.h>
#include <linux/ide.h>
#include <linux/init.h>
#include <linux/module.h>
#include <linux/errno.h>
#include <linux/gpio.h>
#include <asm/mach/map.h>
#include <asm/uaccess.h>
#include <asm/io.h>
#include <linux/cdev.h>
#include <linux/of.h> /* dts操作相关 */
#include <linux/of_address.h> /* dts地址相关 */
#include <linux/of_gpio.h> /* gpio子系统相关 */

#define KERNEL_LED_DEVIE_CNT 1
#define KERNEL_LED_NAME "kernel_led"

#define GPIO_CLK_EN (0x1U << 22)

#define EMIO_PIN 0x00000001

#define EMIO_INPUT 0
#define EMIO_OUTPUT 1

#define EMIO_OUTPUT_DIS 0
#define EMIO_OUTPUT_EN 1

#define EMIO_GPIO_LOW 0
#define EMIO_GPIO_HIGH 1

struct kernel_led_dev
{
    dev_t devid;
    struct cdev cdev;
    struct class *class;
    struct device *device;
    int major;
    int minor;
    struct device_node *nd; /* 设备节点 */
    int gpio; /* gpio编号 */
};

static struct kernel_led_dev dev;

/* 设备打开时候会被调用 */
static int led_open(struct inode *inode,struct file *filp){
    filp->private_data = &dev;

    return 0;
}

/* 设备读取时候会被调用 */
static ssize_t led_read(struct file *filp,char __user *buf,size_t cnt,loff_t *offset){
    int ret;
    char kbuf[1];

    if (cnt != 1){
        return -EFAULT;
    }

    ret = gpio_get_value(dev.gpio);
    if(ret < 0){
        return ret;
    }

    /* 不是高就是低! */
    kbuf[0] = ret;
    ret = copy_to_user(buf,kbuf,cnt);
    if(ret){
        /* 复制失败了 */
        return -EFAULT;
    }

    return 0;
}
```

```
/* 设备写入时候会被调用 */
static ssize_t led_write(struct file *filp,const char __user *buf,size_t cnt,loff_t *offset){
    int ret;
    char kbuf[1];

    if (cnt != 1){
        return -EFAULT;
    }

    ret = copy_from_user(kbuf,buf,cnt);
    if(ret){
        /* 复制失败了 */
        return -EFAULT;
    }

    gpio_set_value(dev.gpio,kbuf[0]?1:0);

    return 0;
}

/* 设备释放时候会被调用 */
static int led_release(struct inode *inode,struct file *filp){
    return 0;
}

static struct file_operations fops =
{
    .owner = THIS_MODULE,
    .open = led_open,
    .read = led_read,
    .write = led_write,
    .release = led_release,
};

static int __init led_init(void){
    const char *str;
    int ret;

    /* 新增的从dts获取数据的过程 */
    dev.nd = of_find_node_by_path("/led");
    if(dev.nd == NULL){
        return -EINVAL;
    }

    ret = of_property_read_string(dev.nd,"status",&str);
    if(ret < 0){
        return -EINVAL;
    }

    if(strcmp(str,"okay")){
        return -EINVAL;
    }

    ret = of_property_read_string(dev.nd,"compatible",&str);
    if(ret < 0){
        return -EINVAL;
    }

    if(strcmp(str,"taterli,led")){
        return -EINVAL;
    }

    /* IO当然也可以是一个数组 */
    dev.gpio = of_get_named_gpio(dev.nd,"led-gpio",0);
    if(!gpio_is_valid(dev.gpio)){
        /* IO是独占资源,因此可能申请失败! */
        return -EINVAL;
    }

    /* 申请IO并给一个名字 */
    ret = gpio_request(dev.gpio,"taterli-kernel-led");
    if(ret < 0){
        /* 除了返回EINVAL,也可以返回上一层传递的错误. */
        return ret;
    }

    ret = of_property_read_string(dev.nd,"default-state",&str);
    if(ret < 0){
        return -EINVAL;
    }
```

测试起来效果是一样的,到目前位置,我们已经完成了基本的GPIO驱动.

```
    if(!strcmp(str,"on")){
         /* 设置输出和默认电平 */
        gpio_direction_output(dev.gpio,1);
    }else if(!strcmp(str,"off")){
        gpio_direction_output(dev.gpio,0);
    }else{
        return -EINVAL;
    }

    /* 不需要寄存器映射了,因为有子系统! */

    /* 申请一个设备号 */
    ret = alloc_chrdev_region(&dev.devid,0,KERNEL_LED_DEVIE_CNT,KERNEL_LED_NAME);
    if(ret){
        goto alloc_fail;
    }
    dev.major = MAJOR(dev.devid);
    dev.minor = MINOR(dev.devid);

    dev.cdev.owner = THIS_MODULE;
    cdev_init(&dev.cdev,&fops);

    ret = cdev_add(&dev.cdev,dev.devid,KERNEL_LED_DEVIE_CNT);
    if(ret){
        goto add_fail;
    }

    dev.class = class_create(THIS_MODULE,KERNEL_LED_NAME);
    if(IS_ERR(dev.class)){
        ret = PTR_ERR(dev.class);
        goto class_fail;
    }

    dev.device = device_create(dev.class,NULL,dev.devid,NULL,KERNEL_LED_NAME);
    if(IS_ERR(dev.device)){
        ret = PTR_ERR(dev.class);
        goto dev_fail;
    }

    return 0;

dev_fail:
    class_destroy(dev.class);

class_fail:
    cdev_del(&dev.cdev);

add_fail:
    unregister_chrdev_region(dev.devid,KERNEL_LED_DEVIE_CNT);

alloc_fail:
    /* 这里就清爽很多了,释放IO就行. */
    gpio_free(dev.gpio);
    return ret;
}

static void __exit led_exit(void){
    device_destroy(dev.class,dev.devid);

    class_destroy(dev.class);

    cdev_del(&dev.cdev);

    unregister_chrdev_region(dev.devid,KERNEL_LED_DEVIE_CNT);

    gpio_free(dev.gpio);
}

module_init(led_init);
module_exit(led_exit);

MODULE_AUTHOR("Taterli <admin@taterli.com>");
MODULE_DESCRIPTION("Led GPIO");
MODULE_LICENSE("GPL");
```

测试起来效果是一样的,到目前位置,我们已经完成了基本的GPIO驱动.