实验四：串行IO接口设计

专业班级：**通信2101班** 姓名：　　**罗畅** 学号：　　**U202113940**

实验名称

串行IO接口设计

实验目的

- 掌握GPIO IP核的工作原理和使用方法
- 掌握中断控制方式的IO接口设计原理
- 掌握中断程序设计方法
- 掌握IO接口程序控制方法

实验仪器

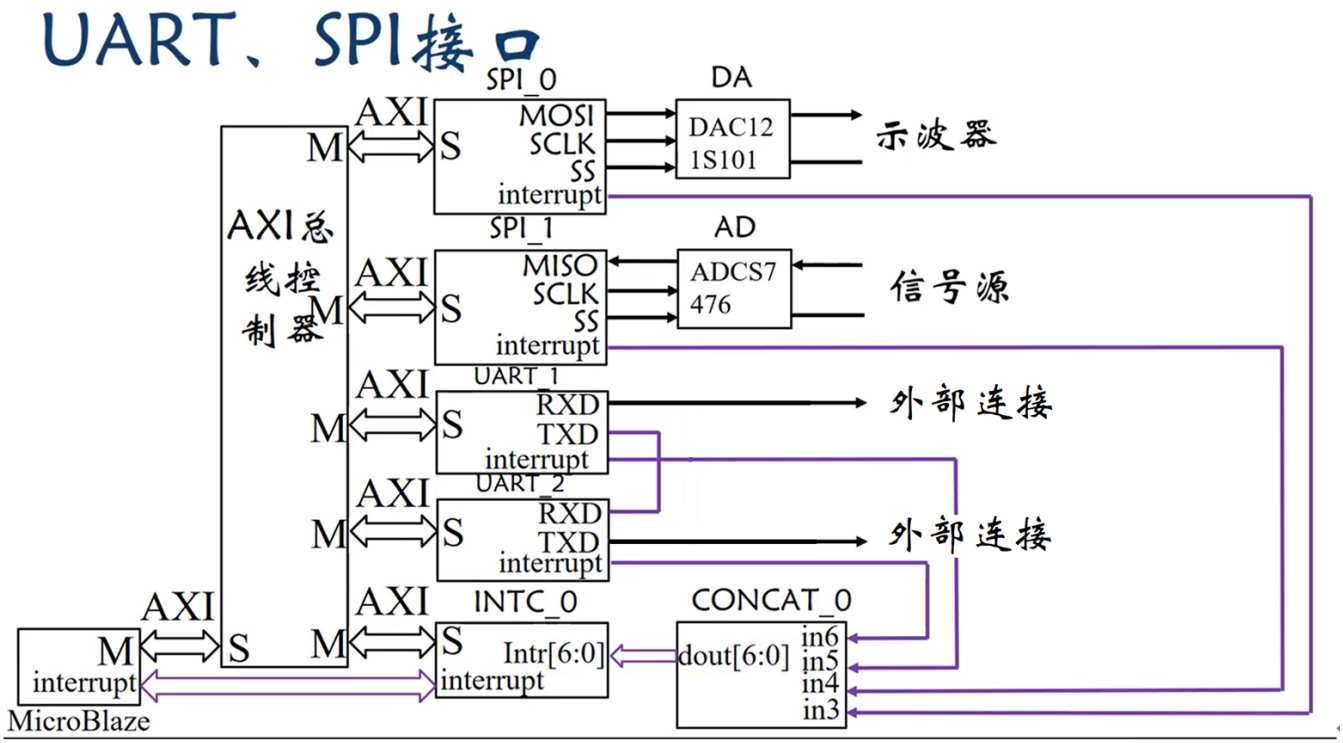
Vivado 2018.3*、*Vivado SDK*、*Visual Studio Code-Insiders

实验任务

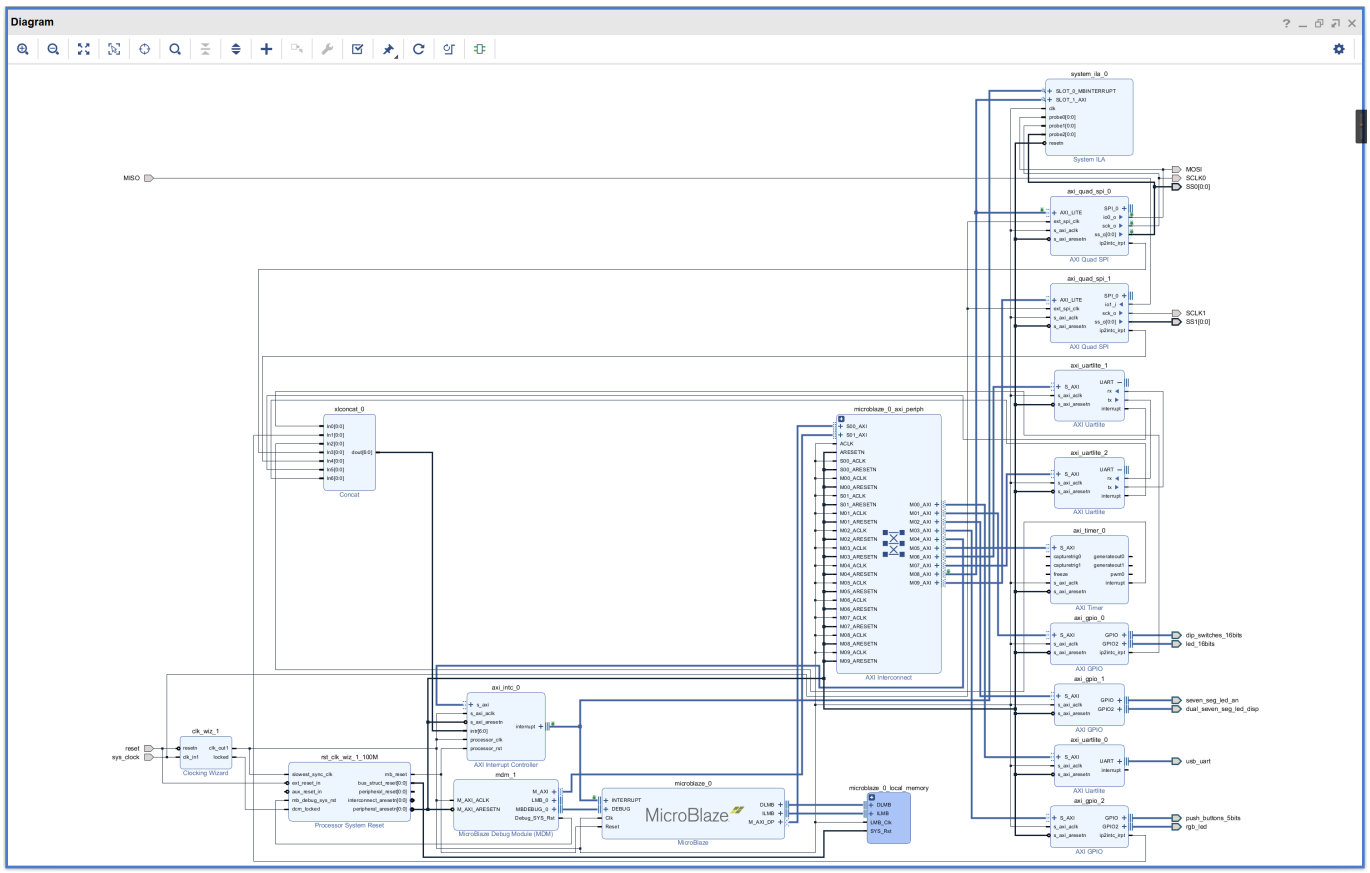
- 理解UART串行通信协议以及接口设计
- 理解SPI串行通信协议
- 掌握UART串行接口设计
- 掌握SPI串行接口设计
- 掌握串行DA接口设计
- 掌握串行AD接口设计

实验原理

硬件电路框图



根据硬件电框图搭建的硬件平台整体框图如下：



软件流程图


串行IO接口实验
UART2产生中断
发送开关的值
发送低5~0位，在后两位添加00，如XXXXXX00
发送低10~6位，在后两位添加01，如0XXXXX01
发送低15~11位，在后两位添加10，如0XXXXX10
发送按键的值
在后两位添加11，如0XXXXX11
UART1产生中断
接收并判断后两位
00
右移两位后赋给LED低5~0位
01
右移两位后赋给LED低10~6位
10
右移两位后赋给LED低15~11位
11
判断为按键

实验源码

```c
/*
 * SeriesIOControl.c
 *
 *  Created on: 2023.4.26
 *      Author: Luo Chang
 */

#include "xil_io.h"
#include "stdio.h"
#include "xgpio_l.h"
#include "xintc_l.h"
#include "xtmrctr_l.h"
#include <xuartlite_l.h>

#define RESET_VALUE 100000

void UART_SEND();
void UART_RECV();
void Seg_TimerCounterHandler();
void BtnHandler();
void SwitchHandler();
void My_ISR() __attribute__((interrupt_handler));
int i = 0;

unsigned short current_btn, last_btn, real_btn;
char segcode[6] = {0xc6, 0xc1, 0xc7, 0x88, 0xa1, 0xff};
short code[8] = {5, 5, 5, 5, 5, 5, 5, 5};
int k = 0;
short pos = 0xff7f;

int main() {

    Xil_Out32(XPAR_AXI_GPIO_0_BASEADDR + XGPIO_TRI2_OFFSET, 0x0);  // 设置LED为输出
    Xil_Out32(XPAR_AXI_GPIO_0_BASEADDR + XGPIO_TRI_OFFSET, 0xffff);  // 设置Switch为输
入

    Xil_Out32(XPAR_AXI_GPIO_1_BASEADDR + XGPIO_TRI_OFFSET, 0x0);
    Xil_Out32(XPAR_AXI_GPIO_1_BASEADDR + XGPIO_TRI2_OFFSET, 0x0);
    Xil_Out32(XPAR_AXI_GPIO_2_BASEADDR + XGPIO_TRI_OFFSET, 0x1f);

    Xil_Out32(XPAR_AXI_TIMER_0_BASEADDR + XTC_TCSR_OFFSET,
              Xil_In32(XPAR_AXI_TIMER_0_BASEADDR + XTC_TCSR_OFFSET) &
~XTC_CSR_ENABLE_TMR_MASK);  // 写TSCR，停止计数器
    Xil_Out32(XPAR_AXI_TIMER_0_BASEADDR + XTC_TLR_OFFSET, RESET_VALUE);  // 写TLR，预制
计数值
    Xil_Out32(XPAR_AXI_TIMER_0_BASEADDR + XTC_TCSR_OFFSET,
              Xil_In32(XPAR_AXI_TIMER_0_BASEADDR + XTC_TCSR_OFFSET) |
XTC_CSR_LOAD_MASK);  // 启动定时器的加载操作
    Xil_Out32(XPAR_AXI_TIMER_0_BASEADDR + XTC_TCSR_OFFSET,
              (Xil_In32(XPAR_AXI_TIMER_0_BASEADDR + XTC_TCSR_OFFSET) &
```

```
~XTC_CSR_LOAD_MASK) | XTC_CSR_ENABLE_TMR_MASK | XTC_CSR_AUTO_RELOAD_MASK |
XTC_CSR_ENABLE_INT_MASK | XTC_CSR_DOWN_COUNT_MASK);
    // 开始计数时运行，自动获取，允许中断，减计数

    // 使能中断，清除RX，TX
    Xil_Out32(XPAR_AXI_UARTLITE_1_BASEADDR + XUL_CONTROL_REG_OFFSET,
XUL_CR_ENABLE_INTR | XUL_CR_FIFO_RX_RESET | XUL_CR_FIFO_TX_RESET);
    Xil_Out32(XPAR_AXI_UARTLITE_2_BASEADDR + XUL_CONTROL_REG_OFFSET,
XUL_CR_ENABLE_INTR | XUL_CR_FIFO_RX_RESET | XUL_CR_FIFO_TX_RESET);
    // 对中断器进行中断使能
    Xil_Out32(XPAR_AXI_INTC_0_BASEADDR + XIN_IER_OFFSET,
XPAR_AXI_TIMER_0_INTERRUPT_MASK | XPAR_AXI_UARTLITE_1_INTERRUPT_MASK |
XPAR_AXI_UARTLITE_2_INTERRUPT_MASK);

    Xil_Out32(XPAR_AXI_INTC_0_BASEADDR + XIN_MER_OFFSET, XIN_INT_MASTER_ENABLE_MASK |
XIN_INT_HARDWARE_ENABLE_MASK);
    microblaze_enable_interrupts();  //允许处理器中断

    UART_SEND();
    return 0;
}

void My_ISR() {
    int status;
    status = Xil_In32(XPAR_AXI_INTC_0_BASEADDR + XIN_ISR_OFFSET);
    if ((status & XPAR_AXI_TIMER_0_INTERRUPT_MASK) == XPAR_AXI_TIMER_0_INTERRUPT_MASK)
{
            Seg_TimerCounterHandler();
    }
    if ((status & XPAR_AXI_UARTLITE_2_INTERRUPT_MASK) ==
XPAR_AXI_UARTLITE_2_INTERRUPT_MASK) {
        UART_SEND();
    } else if ((status & XPAR_AXI_UARTLITE_1_INTERRUPT_MASK) ==
XPAR_AXI_UARTLITE_1_INTERRUPT_MASK) {
        UART_RECV();
    }
    Xil_Out32(XPAR_AXI_INTC_0_BASEADDR + XIN_IAR_OFFSET, status);
}

void UART_SEND() {
    Xil_Out32(XPAR_AXI_UARTLITE_2_BASEADDR + XUL_TX_FIFO_OFFSET,
(Xil_In32(XPAR_AXI_GPIO_0_BASEADDR + XGPIO_DATA_OFFSET) & 0x3f) << 2);  // 发送开关低5-
0位
    Xil_Out32(XPAR_AXI_UARTLITE_2_BASEADDR + XUL_TX_FIFO_OFFSET,
((Xil_In32(XPAR_AXI_GPIO_0_BASEADDR + XGPIO_DATA_OFFSET) >> 4) & 0x7c) | 0x1);  // 发
送开关低10-6位
    Xil_Out32(XPAR_AXI_UARTLITE_2_BASEADDR + XUL_TX_FIFO_OFFSET,
((Xil_In32(XPAR_AXI_GPIO_0_BASEADDR + XGPIO_DATA_OFFSET) >> 9) & 0x7c) | 0x2);  // 发
送开关低15-11位
    Xil_Out32(XPAR_AXI_UARTLITE_2_BASEADDR + XUL_TX_FIFO_OFFSET,
(Xil_In32(XPAR_AXI_GPIO_2_BASEADDR + XGPIO_DATA_OFFSET) << 2) | 0x3);  // 发送按键
```

```
}

void UART_RECV() {
    static u32 sw = 0;
    static u32 btn = 0;
    if ((Xil_In32(XPAR_AXI_UARTLITE_1_BASEADDR + XUL_STATUS_REG_OFFSET) &
XUL_SR_RX_FIFO_VALID_DATA) == XUL_SR_RX_FIFO_VALID_DATA) {
        u32 tmp = Xil_In32(XPAR_AXI_UARTLITE_1_BASEADDR + XUL_RX_FIFO_OFFSET);
        switch(tmp & 0x3) {//发送过来的数据低两位和0x3做和运算，作为判断位
        case 0: {  // 判断为开关5-0位
            sw &= 0xffc0;
            sw |= tmp >> 2;
            Xil_Out32(XPAR_AXI_GPIO_0_BASEADDR + XGPIO_DATA2_OFFSET, sw);
            break;
        }
        case 0x1: {  // 判断为开关10-6位
            sw &= 0xf83f;
            sw |= (tmp & 0x7c) << 4;
            Xil_Out32(XPAR_AXI_GPIO_0_BASEADDR + XGPIO_DATA2_OFFSET, sw);
            break;
        }
        case 0x2 : {  // 判断为开关15-11位
            sw &= 0x7ff;
            sw |= (tmp & 0x7c) << 9;
            Xil_Out32(XPAR_AXI_GPIO_0_BASEADDR + XGPIO_DATA2_OFFSET, sw);
            break;
        }
        case 0x3: {  // 判断为按键
            if (btn == (tmp & 0x7c) >> 2) {
                    return;
                }
            btn = (tmp & 0x7c) >> 2;
            switch(btn) {
            case 0: {
                break;
            }
            case 0x01: {
                code[k] = 0;
                k = (k + 1) % 8;
                break;
            }
            case 0x02: {
                code[k] = 1;
                k = (k + 1) % 8;
                break;
            }
            case 0x04: {
                code[k] = 2;
                k = (k + 1) % 8;
                break;
            }
```

```
            case 0x08: {
                code[k] = 3;
                k = (k + 1) % 8;
                break;
            }
            case 0x10: {
                code[k] = 4;
                k = (k + 1) % 8;
                break;
            }
            }
        }
        }
    }
    // 接收&处理完数据后，打开 UART1 接收中断、重置接收 FIFO 和发送 FIFO端口，这样可以保证在下一次接收数据时不会存在任何未知的数据
    Xil_Out32(XPAR_AXI_UARTLITE_1_BASEADDR + XUL_CONTROL_REG_OFFSET, XUL_CR_ENABLE_INTR | XUL_CR_FIFO_RX_RESET | XUL_CR_FIFO_TX_RESET);
}

void Seg_TimerCounterHandler() {
    Xil_Out32(XPAR_AXI_GPIO_1_BASEADDR + XGPIO_DATA2_OFFSET, segcode[code[(i + k) % 8]]);
    Xil_Out32(XPAR_AXI_GPIO_1_BASEADDR + XGPIO_DATA_OFFSET, pos);
    pos = pos >> 1;
    i++;
    if (i == 8) {
        i = 0;
        pos = 0xff7f;
    }
    // 重置 AXI Timer IP 核的计数器寄存器的值，从而在下一个计时周期开始之前将计时器的值重置为 0
    Xil_Out32(XPAR_AXI_TIMER_0_BASEADDR + XTC_TCSR_OFFSET, Xil_In32(XPAR_AXI_TIMER_0_BASEADDR + XTC_TCSR_OFFSET));
}
```

实验结果

首先断开UART2的TX和UART的RT之间的连线，拨动开关和按下按键，LED灯和数码管都没变化。连接上连接UART2的TX和UART的RT之间的连线后，再次拨动开关，对应的LED灯被点亮，按下开关后数码管亮起了最近按下的开关对应的字符，所以实验结果满足要求。

实验小结

本次实验在已有硬件平台的基础上只用编写新的C语言代码，实验还算比较顺利，一次课就验收通过了，***Vivado SDK*** 并不算很好用，还得是 ***Visual Studio Code*** 编写代码起来比较顺手