一. 实验任务

采用 UART IP 核，实现 Nexys4 或 Nexys4 DDR 实验板 UART 接口之间的通信。要求当拨动开关时，将开关对应的值通过 UART1 发送到 UART2，同时利用 LED 灯指示 UART2 接收到的当前开关的值；当按下按键时，将按键对应的值通过 UART2 发送到 UART1，同时利用数码管指示 UART1 接收到的当前按下的按键位置码（C,U,d,L,r)。UART 波特率为 9600bps。

二. 实验目的

1. 掌握 UART 串行通信协议;
2. 掌握 UART 串行通信接口设计;
3. 掌握中断控制方式的串行 10 接口设计原理;
4. 掌握中断程序设计方法。

三. 实验环境

1. Windows 10 操作系统 ；
2. 嵌入式软件开发平台：Vivado 2018.1;
3. 硬件平台：并行 IO 接口设计实验任务中搭建的嵌入式系统;
3. 硬件平台开发板：Xilinx Nexys4。

四. 设计方案

1. **回顾 UART 串行通信协议：**

   UART(UniversalAsynchronousReceiver/Transmitter)是一种通用异步串行通信总线,可以实现全双工传输和接收,主要用于低速设备与计算机系统之间的串行通信。嵌入式 设计中,UART 用于主机与嵌入式设备通信,UART 作为嵌入式系统的标准输入/输出接口。 UART 点对点三线 通信连接电路 UART 接口采用 4 线接口,分别为 VCC、GND、RXD(接 收端)、TXD(发送端)。采用 UART 接口实现点对点通信时, 连接三线即可,连接电路如图 1 所示：

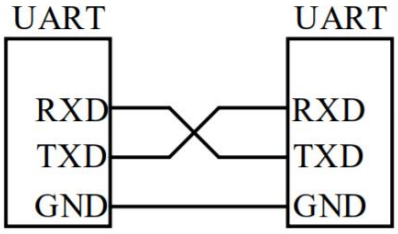


图 1 UART 拓扑结构

UART 规定了数据串行通信格式,数据通信以帧为单位，每一帧包含以下信息:①1 位起始位;②5~8 位数据位;③0~1 位奇或偶校验位;④1 位、1.5 位或 2 位停止位。数据位低位优先传送,信息位宽度由波特率(每秒传输的信息位数,以 bps 为单位)决定,UART 支持的波特率为 300、1200、2400、4800、9600、19.2k、38.4k、57.6k、115.2k…。波特率越高,通信距离越短。UART 总线空闲时,信号线维持高电平,若信号线出现下降沿表示通信的开始。

UART 一帧数据的通信格式如图 2 所示：

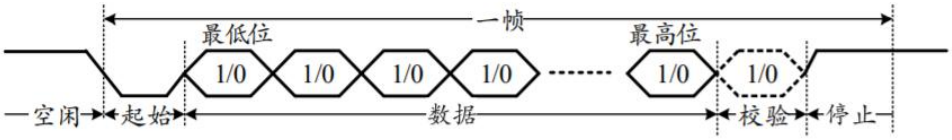


图 2 UART 一帧数据的通信格式

2. 硬件系统结构：

系统中用到 2 个 UART 通信接口，如图 3 所示：

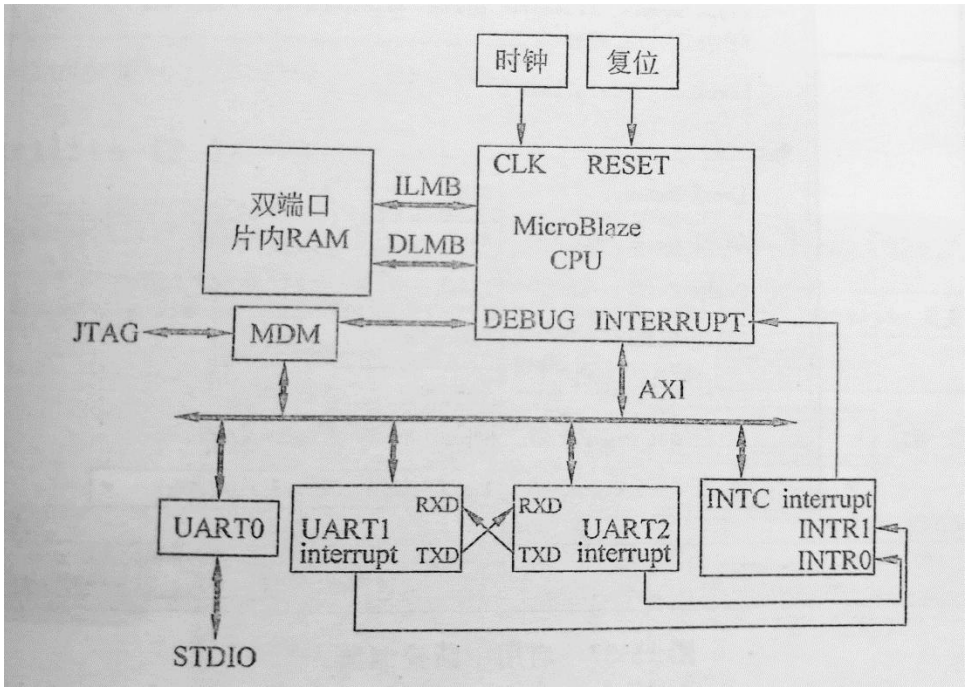


图 2 UART 通信系统电路原理图

3. 需求分析：

分析题目可知，主要需要实现 3 个任务：

① 开关与按键状态输入：

采用中断方式实现程序控制，开关状态输入以及按键状态输入都需要在按键的 GPIO 中断事务处理实现。要求使用 UART 通信，所以中断事务处理函数中只需把相关输入传输到对应 UART 接口即可。

② UART 数据处理：

采用中断方式实现程序控制，UART 在接收到数据后，调用中断事务处理函数，对收到的数据进行处理。

③ 数码管与 LED 显示：

在 UART 中断事务处理函数中，对处理完的信号直接进行对应数码管与 LED 灯的输出。

4. 模块结构：

由上面的分析可知，控制程序可以分为四个函数，他们的层级结构如图 3 所示：

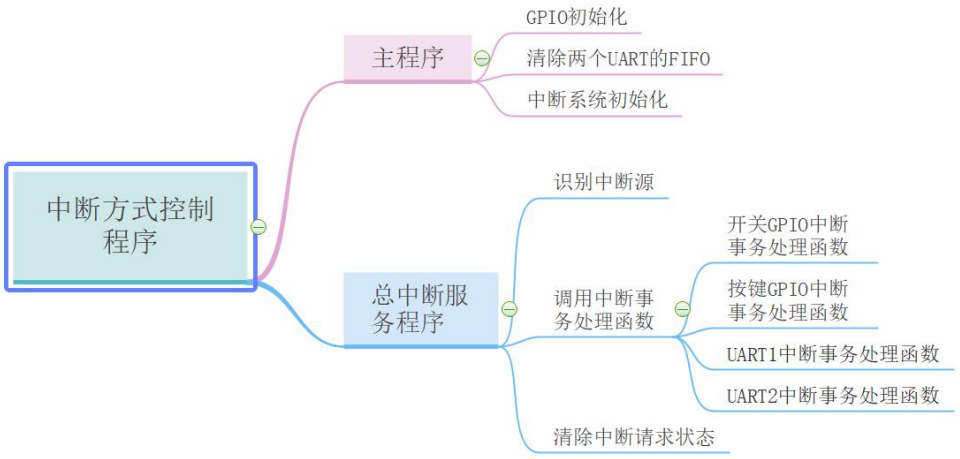


图 3 模块结构

5. 各函数功能：

①主程序：

主程序进行 GPIO 初始化与相关输入输出设置，并且清除 UART 的接受与发送 FIFO，然后对中断系统初始化并且使能微处理器开中断。

流程图如图 4 所示：

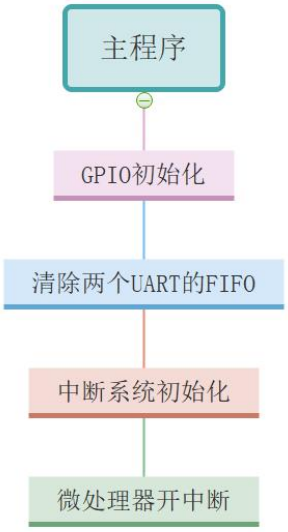


图 4 主程序

②总中断服务程序：

流程图如图 5 所示：

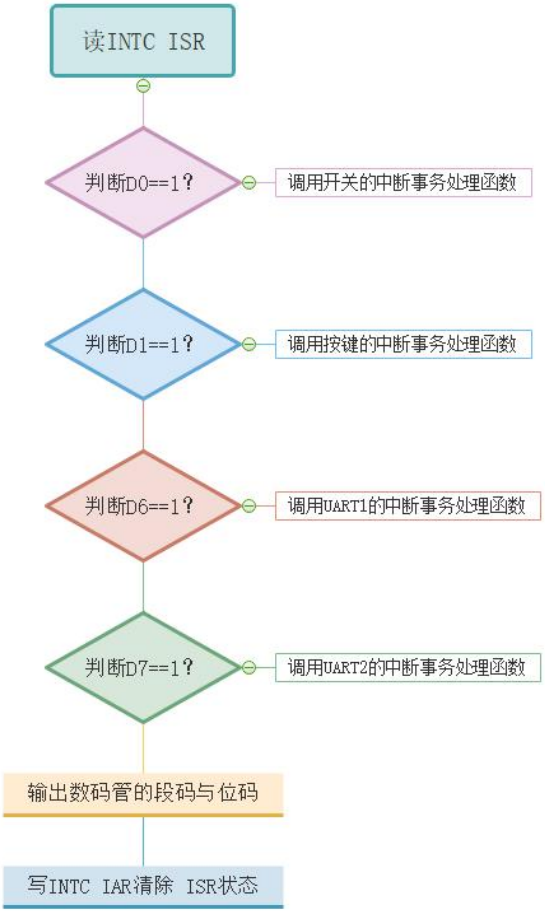


图 5 总断服务程序

③开关 GPIO 中断事务处理函数：
流程图如图 6 所示：

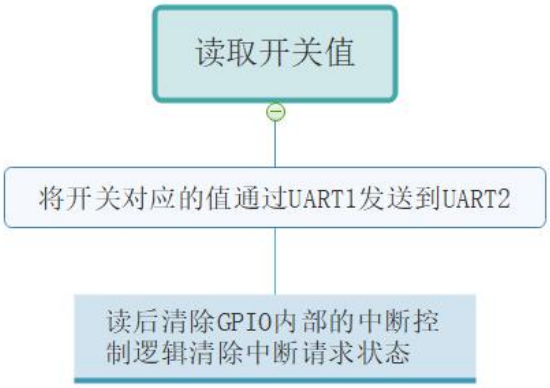


图 6 开关 GPIO 中断事务处理函数

④按键 GPIO 中断事务处理函数：
流程图如图 7 所示：

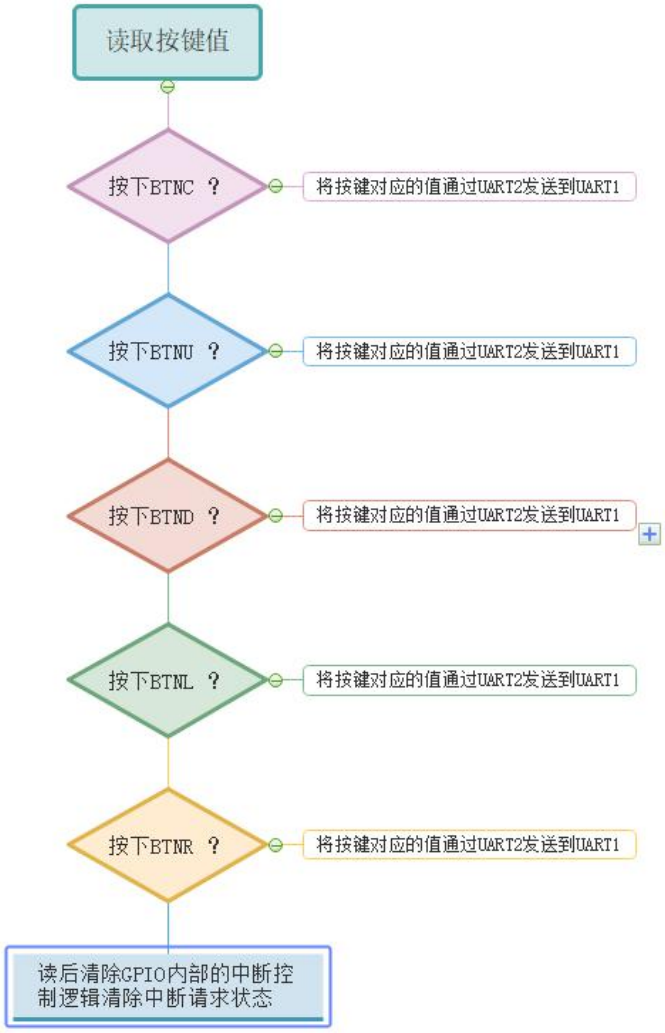


图 7 按键 GPIO 中断事务处理函数

⑤UART1 中断事务处理函数：
流程图如图 8 所示：

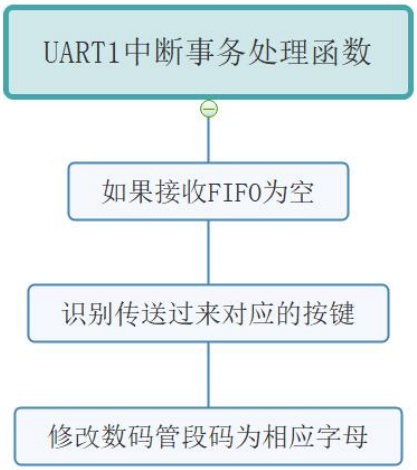


图 8 UART1 中断事务处理函数

⑥UART2 中断事务处理函数：
流程图如图 9 所示：

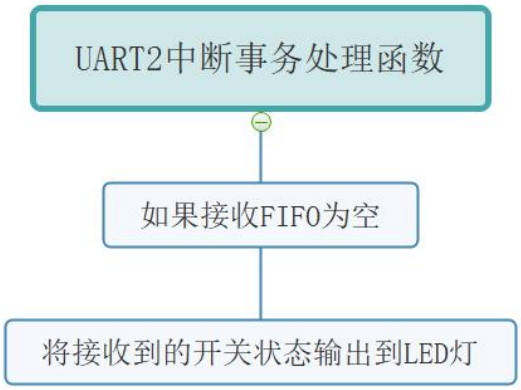


图 9 UART2 中断事务处理函数

五. 实现过程

1. **硬件平台搭建：**

在 Vivado 2018.1 中，使用 Xilinx Nexys4 开发板，搭建基于 MicroBlaze 软核的嵌入式系统硬件平台如下图 10 所示：

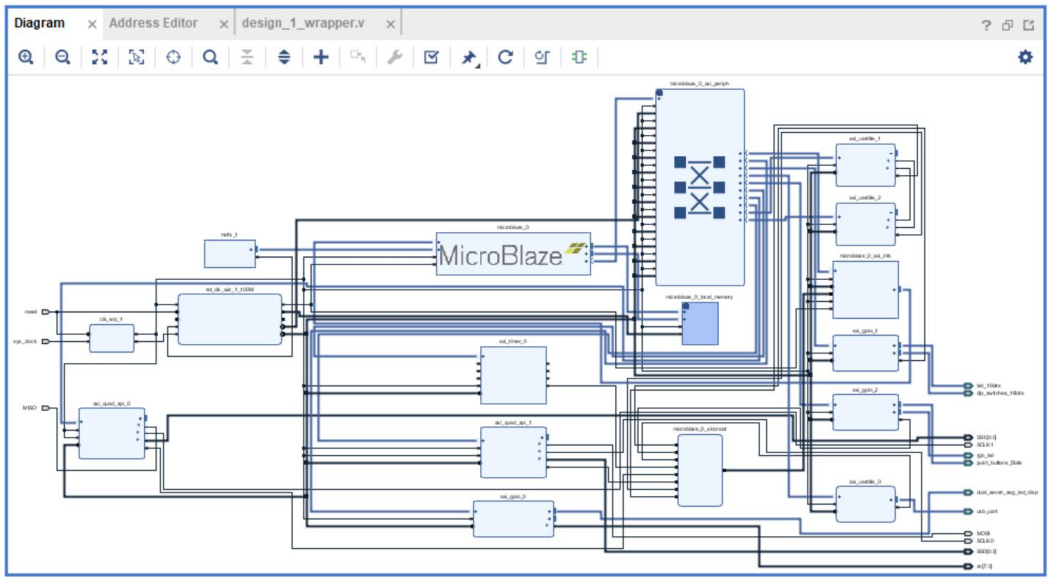


图 10 嵌入式系统硬件平台总框图

中断方式使用一个中断控制器：其中 GPIO_0 中断输出连接到 Intr0，GPIO_2 中断输出连接到 Intr1，Timer_0 中断输出连接到 Intr2；中断控制器的中断向量输出连接到 MicroBlaze 微处理器的中断输入总线上。

其中对常用并行 IO 外设 GPIO 接口进行了配置（如图 11）：

16 位开关和 16 位 LED 灯共用一个 GPIO IP 核（设为 GPIO_1），其中，开关使用 GPIO 通道，LED 灯使用 GPI2 通道；四位七段数码管的位码与段码共用另一个 GPIO IP 核（设为 GPIO_0），其中，位码使用 GPIO 通道，段码使用 GPIO2 通道；两位按键使用另一个 GPIO IP 核（设为 GPIO_2）的 GPIO 通道；延时 T0 与 T1 使用 Timer IP 核（设为 Timer_0）。

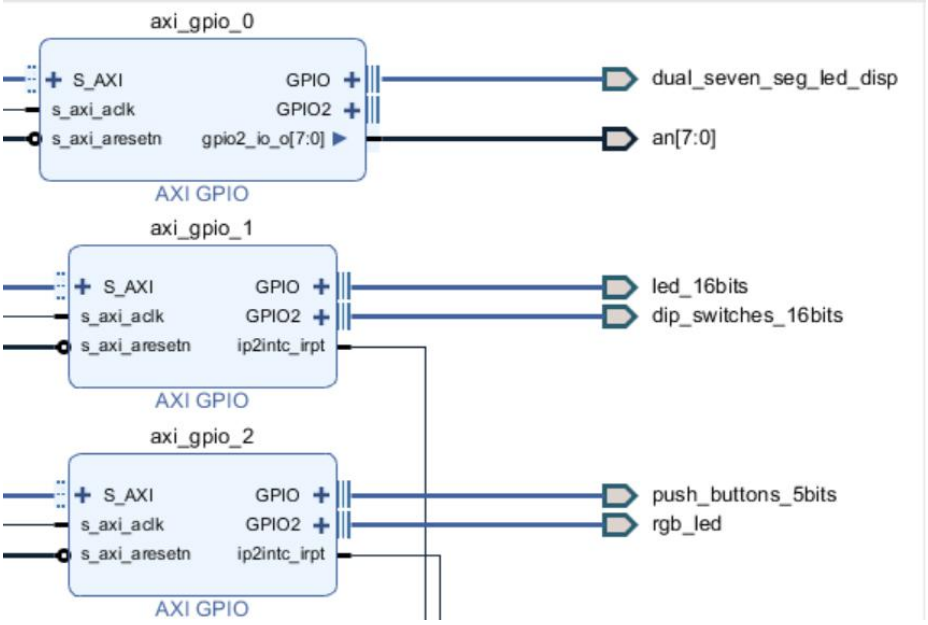


图 11 外设 GPIO 接口配置

对并行 IO 中断系统进行了配置（如图 12）：

中断方式使用一个中断控制器：其中 GPIO_1 中断输出连接到 Intr0;GPIO_2 中断输出连接到 Intr1；UART_0 中断输出连接到 Intr2；Timer_0 中断输出连接到 Intr3；SPI_0 中断输出连接到 Intr4； SPI_1 中断输出连接到 Intr5;；UART_1 中断输出连接到 Intr6；UART_2 中断输出连接到 Intr7.
中断控制器的中断向量输出连接到 MicroBlaze 微处理器的中断输入总线上。
如图所示：

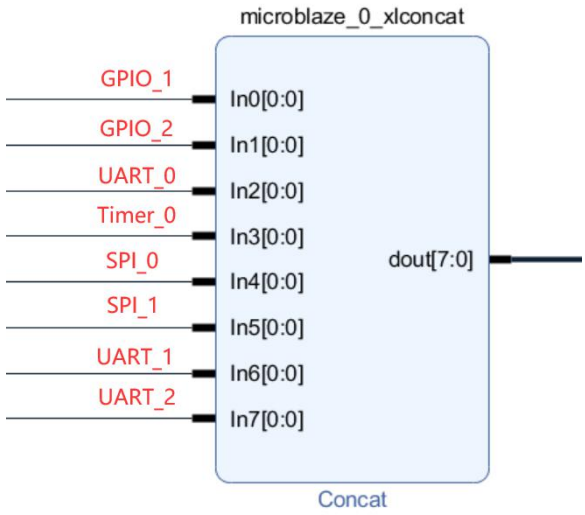


图 12 并行 IO 中断系统配置

对串行 IO 接口外设 UART、SPI 进行了配置（如图 13）：

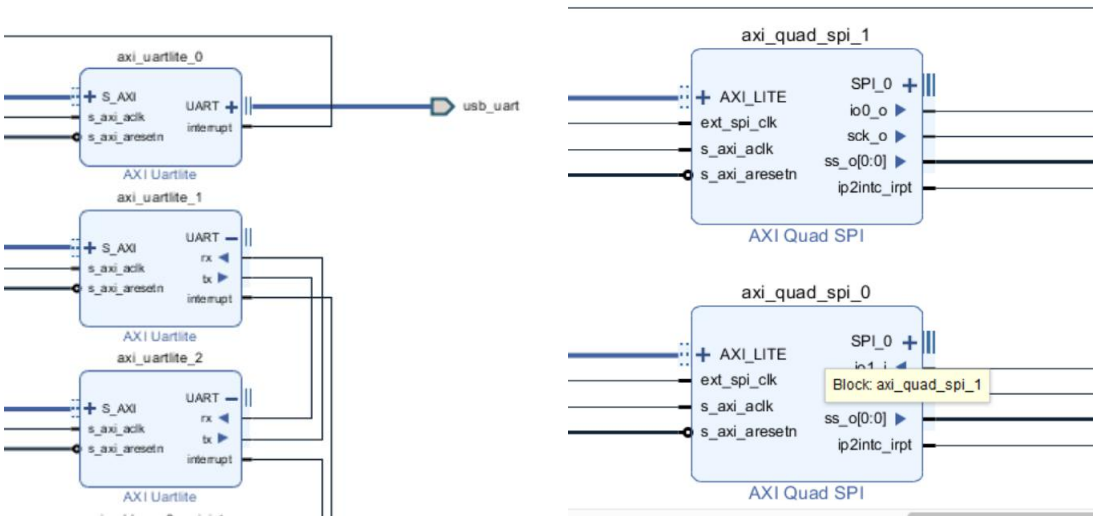
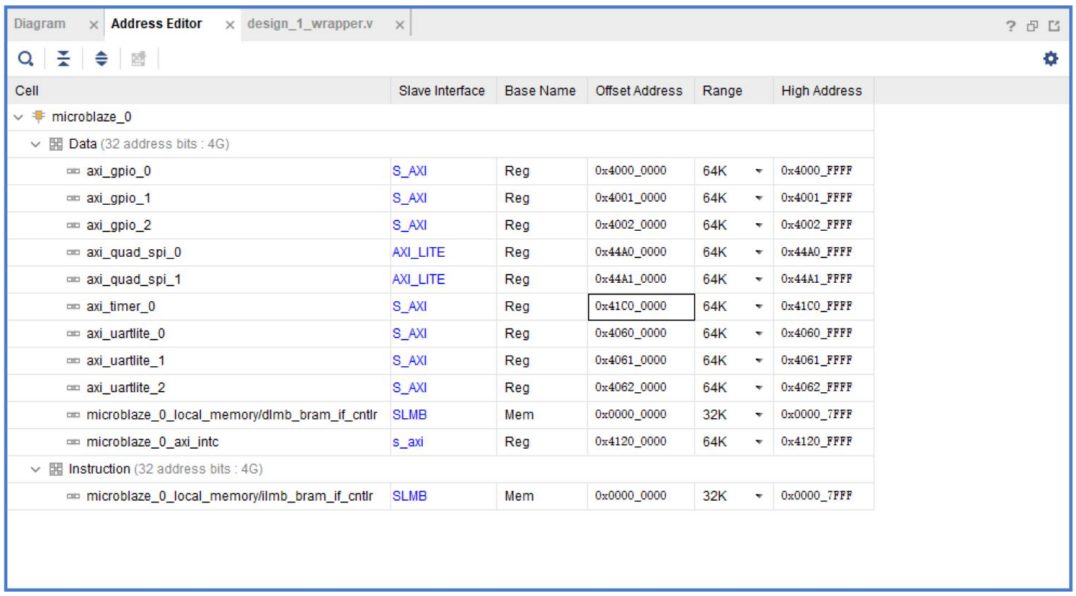


图 13 串行 IO 接口外设 UART、SPI 进行了配置

生成 HDL 封装，查看硬件平台存储空间布局如下（如图 14）：



Cell	Slave Interface	Base Name	Offset Address	Range	High Address
microblaze_0					
Data (32 address bits : 4G)					
axi_gpio_0	S_AXI	Reg	0x4000_0000	64K	0x4000_FFFF
axi_gpio_1	S_AXI	Reg	0x4001_0000	64K	0x4001_FFFF
axi_gpio_2	S_AXI	Reg	0x4002_0000	64K	0x4002_FFFF
axi_quad_spi_0	AXI_LITE	Reg	0x44A0_0000	64K	0x44A0_FFFF
axi_quad_spi_1	AXI_LITE	Reg	0x44A1_0000	64K	0x44A1_FFFF
axi_timer_0	S_AXI	Reg	0x41C0_0000	64K	0x41C0_FFFF
axi_uartlite_0	S_AXI	Reg	0x4060_0000	64K	0x4060_FFFF
axi_uartlite_1	S_AXI	Reg	0x4061_0000	64K	0x4061_FFFF
axi_uartlite_2	S_AXI	Reg	0x4062_0000	64K	0x4062_FFFF
microblaze_0_local_memory/dlmb_bram_if_cntlr	SLMB	Mem	0x0000_0000	32K	0x0000_7FFF
microblaze_0_axi_intc	s_axi	Reg	0x4120_0000	64K	0x4120_FFFF
Instruction (32 address bits : 4G)					
microblaze_0_local_memory/ilmb_bram_if_cntlr	SLMB	Mem	0x0000_0000	32K	0x0000_7FFF

图 14 按硬件平台存储空间布局

2. 各模块实现：

①主程序：

主程序进行 GPIO 初始化与相关输入输出设置，并且清除 UART 的接受与发送 FIFO，然后对中断系统初始化并且使能微处理器开中断。

代码如下：

```
int main()
{
    Xil_Out16(XPAR_AXI_GPIO_0_BASEADDR+XGPIO_TRI_OFFSET,0xffff);//开关配置
    Xil_Out32(XPAR_AXI_GPIO_1_BASEADDR+XGPIO_TRI_OFFSET,0x0);//数码管位码
    Xil_Out32(XPAR_AXI_GPIO_1_BASEADDR+XGPIO_TRI2_OFFSET,0x0);//数码管段码
    Xil_Out32(XPAR_AXI_GPIO_2_BASEADDR+XGPIO_TRI_OFFSET,0X1f);//按键配置
    Xil_Out16(XPAR_AXI_GPIO_0_BASEADDR + XGPIO_TRI2_OFFSET, 0x0);//LED 输出

    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);

    Xil_Out32(XPAR_AXI_GPIO_2_BASEADDR+XGPIO_IER_OFFSET,XGPIO_IR_CH1_MASK);//允许中断 button
    Xil_Out32(XPAR_AXI_GPIO_2_BASEADDR+XGPIO_GIE_OFFSET,XGPIO_GIE_GINTR_ENABLE_MASK);//
    Xil_Out32(XPAR_AXI_GPIO_0_BASEADDR+XGPIO_IER_OFFSET,XGPIO_IR_CH1_MASK);//允许中断 switch
    Xil_Out32(XPAR_AXI_GPIO_0_BASEADDR+XGPIO_GIE_OFFSET,XGPIO_GIE_GINTR_ENABLE_MASK);//
    Xil_Out32(XPAR_AXI_INTC_0_BASEADDR+XIN_IER_OFFSET,XPAR_AXI_GPIO_0_IP2INTC_IRPT_MASK|XPAR_AXI_GPIO_2_IP2INTC_IRPT_MASK|XPAR_AXI_UARTLITE_1_INTERRUPT_MASK|XPAR_AXI_UARTLITE_2_INTERRUPT_MASK);//对中断寄存器进行中断使能
    Xil_Out32(XPAR_AXI_INTC_0_BASEADDR+XIN_MER_OFFSET,XIN_INT_MASTER_ENABLE_MASK|XIN_INT_HARDWARE_ENABLE_MASK);
```

```
    microblaze_enable_interrupts();//允许处理器处理中断
}
```

②总中断服务程序：
实现代码如下：

```
void My_ISR()
{
    int status;
    status=Xil_In32(XPAR_AXI_INTC_0_BASEADDR+XIN_ISR_OFFSET);
    if((status&XPAR_AXI_GPIO_2_IP2INTC_IRPT_MASK)==XPAR_AXI_GPIO_2_IP2INTC_IRPT_MASK)
        ButtonHandler();
    if((status&XPAR_AXI_GPIO_0_IP2INTC_IRPT_MASK)==XPAR_AXI_GPIO_0_IP2INTC_IRPT_MASK)
        SwitchHandler();
    if((status&XPAR_AXI_UARTLITE_2_INTERRUPT_MASK)==XPAR_AXI_UARTLITE_2_INTERRUPT_MASK)
        Uar2Handler();
    if((status&XPAR_AXI_UARTLITE_1_INTERRUPT_MASK)==XPAR_AXI_UARTLITE_1_INTERRUPT_MASK)
        Uar1Handler();

        Xil_Out32(XPAR_AXI_GPIO_1_BASEADDR+XGPIO_DATA2_OFFSET,segcode);//缓冲区
        Xil_Out32(XPAR_AXI_GPIO_1_BASEADDR+XGPIO_DATA_OFFSET,pos);//位码
        Xil_Out16(XPAR_AXI_GPIO_0_BASEADDR+XGPIO_DATA2_OFFSET,sw);//LED

    Xil_Out32(XPAR_AXI_INTC_0_BASEADDR+XIN_IAR_OFFSET,status);//清除
}
```

③开关 GPIO 中断事务处理函数：
实现代码如下：

```
void SwitchHandler()
{
    Xil_Out32(XPAR_AXI_GPIO_0_BASEADDR+XGPIO_ISR_OFFSET,
                                        XGPIO_IR_CH1_MASK);
    Xil_Out16(XPAR_AXI_UARTLITE_1_BASEADDR+XUL_TX_FIFO_OFFSET,
            Xil_In16(XPAR_AXI_GPIO_0_BASEADDR+XGPIO_DATA_OFFSET))
;
}
```

④按键 GPIO 中断事务处理函数：
实现代码如下：

```
void ButtonHandler()
{
```

```
    Xil_Out32(XPAR_AXI_GPIO_2_BASEADDR+XGPIO_ISR_OFFSET,
                                    XGPIO_IR_CH1_MASK);
    int bu;
    bu = Xil_In8(XPAR_AXI_GPIO_2_BASEADDR+XGPIO_DATA_OFFSET)&0x1f;
    Xil_Out32(XPAR_AXI_UARTLITE_2_BASEADDR+XUL_TX_FIFO_OFFSET,bu);
}
```

⑤UART1 中断事务处理函数：
实现代码如下：

```
void Uar1Handler()
{
    int button;
    if((Xil_In32(XPAR_AXI_UARTLITE_1_BASEADDR+XUL_STATUS_REG_OFF
SET)&XUL_SR_RX_FIFO_VALID_DATA)==XUL_SR_RX_FIFO_VALID_DATA)
    {
        button=Xil_In32(XPAR_AXI_UARTLITE_1_BASEADDR+XUL_RX_FIFO
_OFFSET);

        switch(button){
                case 0x1:
                    segcode = 0xc1;
                    break;
                case 0x10:
                    segcode = 0xc6;
                    break;
                case 0x2:
                    segcode = 0xc7;
                    break;
                case 0x4:
                    segcode = 0xa1;
                    break;
                case 0x8:
                    segcode = 0xaf;
                    break;
                default:
                    break;
        }
    }
}
```

⑥UART2 中断事务处理函数：
实现代码如下：

```
void Uar2Handler()
{
    if((Xil_In32(XPAR_AXI_UARTLITE_2_BASEADDR+XUL_STATUS_REG_O
FFSET)&XUL_SR_RX_FIFO_VALID_DATA)==XUL_SR_RX_FIFO_VALID_DATA)
        {
            sw=Xil_In16(XPAR_AXI_UARTLITE_2_BASEADDR+XUL_RX_FI
FO_OFFSET);
        }
}
```

3. **完整代码：**

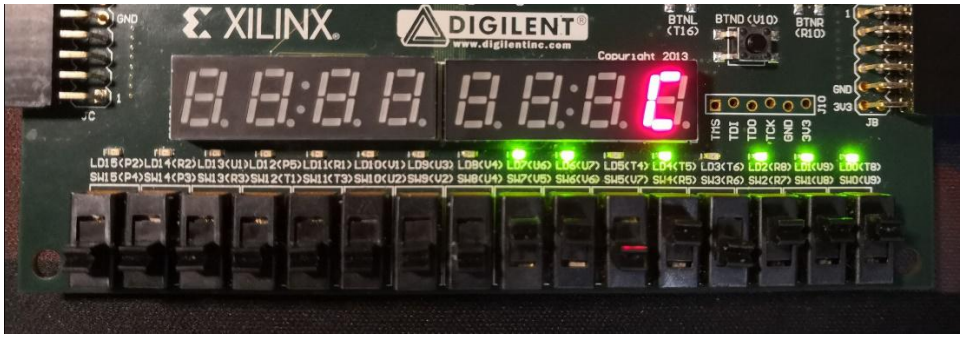
完整代码如下所示，上面展示过的相关函数已缩进：

```
#include "xil_io.h"
#include "stdio.h"
#include "xgpio_l.h"
#include "xintc_l.h"
#include "xparameters.h"
#include "xuartlite_l.h"

void SwitchHandler();
void ButtonHandler();
void Uar1Handler();
void Uar2Handler();
void My_ISR()__attribute__((interrupt_handler));
char segcode = 0xff;//缓冲区
short pos = 0xfffe;
short sw = 0x0000;

int main()…

void My_ISR()…


void SwitchHandler()…

void ButtonHandler()…

void Uar1Handler()…

void Uar2Handler()…
```

六. 实验结果

1. **要求当拨动开关时，将开关对应的值通过 UART1 发送到 UART2，同时利用 LED 灯指示 UART2 接收到的当前开关的值：**

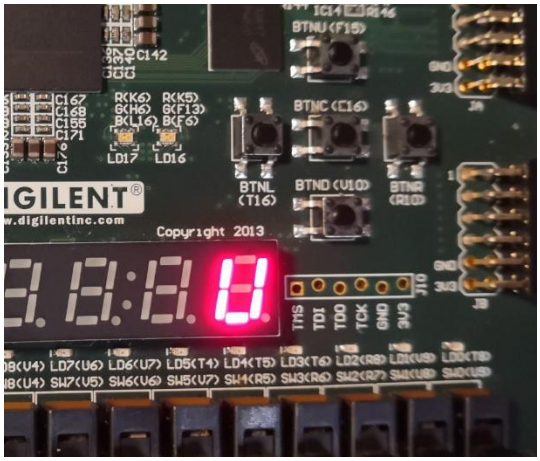
（1） 当开关拨至 0000000011010111：



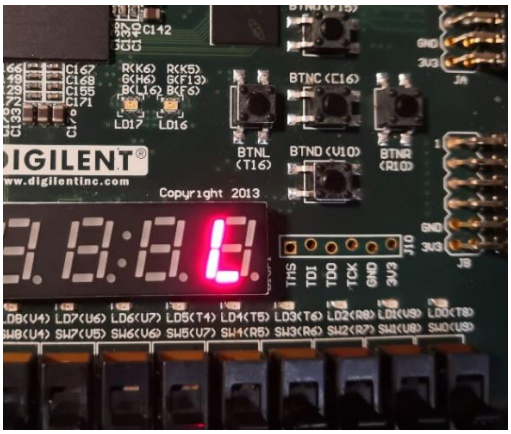
（2） 当开关拨至 00000000010110010：

2. **当按下按键时，将按键对应的值通过 UART2 发送到 UART1，同时利用数码管指示 UART1 接收到的当前按下的按键位置码（C,U,d,L,r）：**

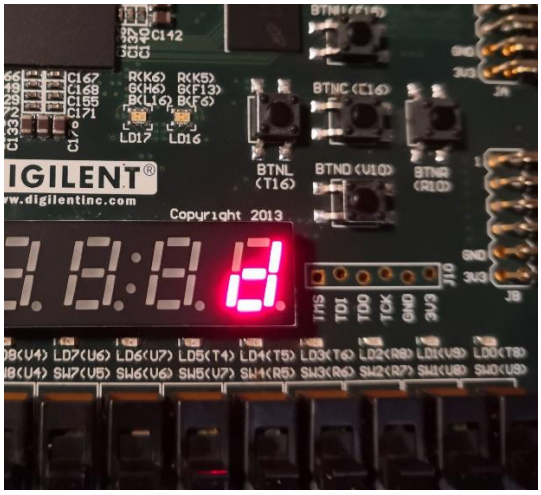
（1） 当按键按下 BTNU（如图）：



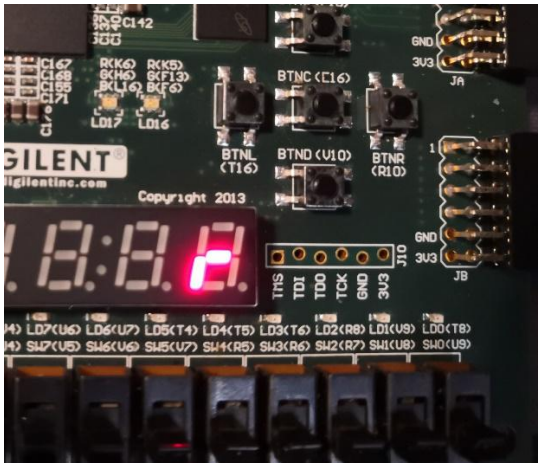
（2） 当按键按下 BTNL：



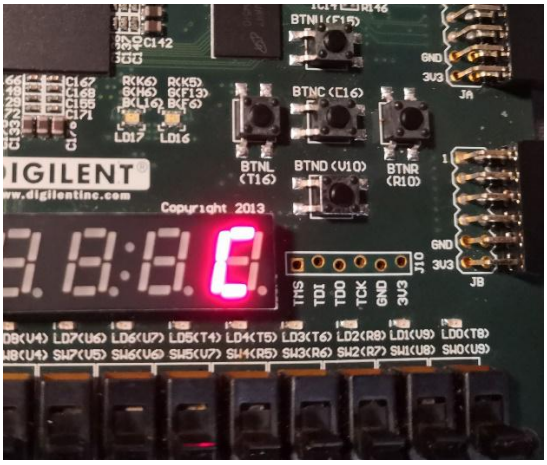
（3） 当按键按下 BTND：



（4） 当按键按下 BTNR：



（5） 当按键按下 BTNC：



七． 实验总结

本次实验是关于串行 IO 接口设计的计算机组成原理实验。关于串行 IO 接口设计，在之前的实验与上学期的大作业中并没有涉及到。其中，对于 UART 串行通信的工作方式、相关协议和具体使用就需要我们自主查阅书籍、研究实例代码来搞清楚，是对我们自主学习能力、综合实践能力和串行接口相关知识的一次综合考察。

通过本次实验，我对串行接口的特点有了更加实际和深入的了解；同时，与之前的并行 IO 接口的实验相比较，我对他们之间的异同和各自的优势与不同的应用范围与场景有了更加清晰的认识：随着计算机技术的快速发展,设备间通信速率要求越来越高。由于并行总线存在串扰, 不适合长距离传输,因此现代计算机系统外部总线大都为串行总线。所以了解计算机组成原理与接口技术时，作为重要的组成部分，我们了解串行接口的通信协议就显得非常重要。

同时，在使用 UART 通信时，我对串行通信协议的共同特征与底层逻辑有了实际的体会：为实现设备间通信,串行总线协议通常都可分为 3 层:①物理层描述总线或接口的机 械、电气特性;②链路层描述信息的传输格式;③传输层描述总线读写操作实现方式。

总之，通过本次实验，我完成了相关实验任务，也实现相关实验目的，初步了解和掌握 UART 串行通信协议和基本的 UART 串行通信接口设计方法；亲身实践了中断控制方式的串行 10 接口设计；进一步掌握中断程序设计方法。