

# 实验四：串行IO接口设计

专业班级：提高2201班  
姓名：王翎羽  
学号：U202213806

## 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 2018.1、Vivado SDK

## 实验任务

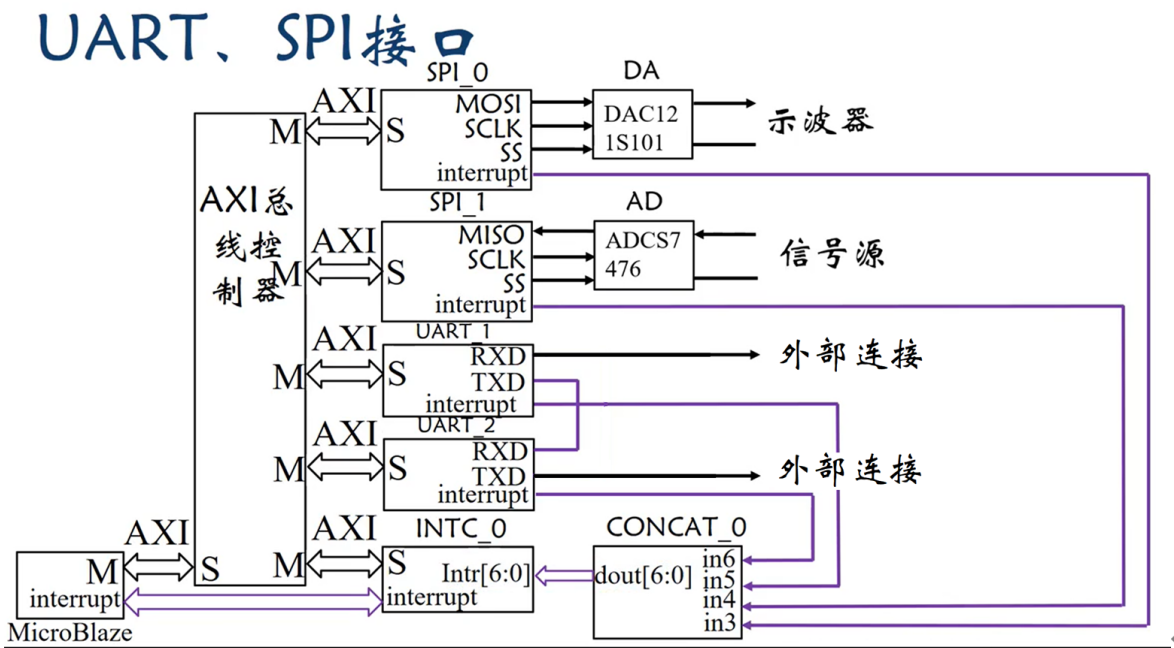
- 掌握串行AD接口设计

利用SPI IP 核，timer IP 核以及AD 模块，控制AD 模块对某模拟信号进行可变频率采样，采样频率由switch 控制。转换结果通过STDIO输出，采样频率最高为0.1s，最低为12.8s

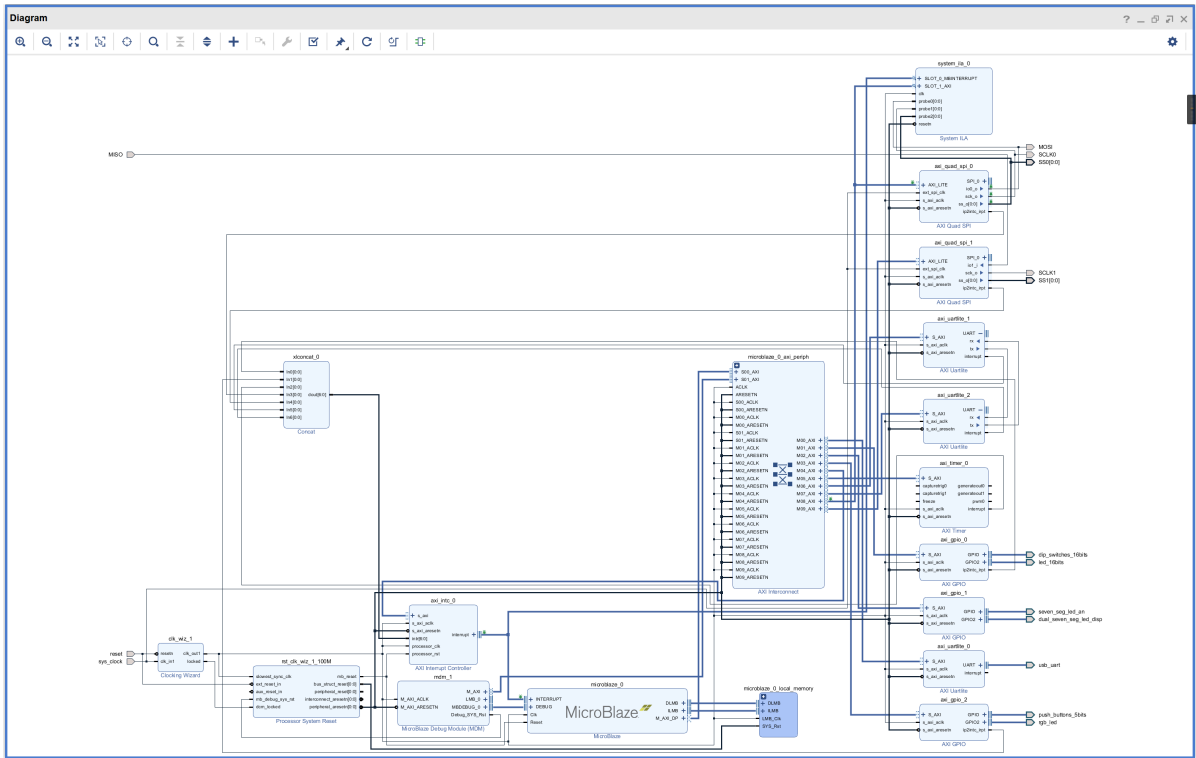
提示：switch 输入的数据，控制定时计数器的定时时间，定时计数器定时时间到，输入一个新AD 转换数据。

## 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：



## 实验源码

```
/*
 * Created on: 2024年5月22日
 * Author: Carl Wang
 */

#include "stdio.h"
#include <stdio.h>
#include "xil_io.h"
#include "xgpio_1.h"
#include "xtmrctr_1.h"
#include "xintc_1.h"
#include "xil_printf.h"
#include "xspi_1.h"

float timePeriod = 1.6 * 100000000 - 2;
int counter = 1;
uint16_t voltageLevel;

void toggleSwitchHandler();
void timeExpiredHandler();
void setupTimer();
void customInterruptServiceRoutine() __attribute__((interrupt_handler));

int main()
{
    // SPI configuration address
    Xil_Out32(XPAR_AXI_QUAD_SPI_1_BASEADDR + XSP_CR_OFFSET, XSP_CR_ENABLE_MASK |
XSP_CR_MASTER_MODE_MASK | XSP_CR_CLK_POLARITY_MASK);
    Xil_Out32(XPAR_AXI_QUAD_SPI_1_BASEADDR + XSP_SSR_OFFSET, 0xfffffffffe);
    Xil_Out32(XPAR_SPI_1_BASEADDR + XSP_SSR_OFFSET, 0x0);
    Xil_Out32(XPAR_SPI_1_BASEADDR + XSP_DTR_OFFSET, 0x0);
```

```

    setupTimer();

    // switch configuration
    Xil_Out32(XPAR_AXI_GPIO_0_BASEADDR + XGPIO_TRI_OFFSET, 0xffff);
    Xil_Out32(XPAR_AXI_GPIO_0_BASEADDR + XGPIO_IER_OFFSET, XGPIO_IR_CH1_MASK);
    Xil_Out32(XPAR_AXI_GPIO_0_BASEADDR + XGPIO_GIE_OFFSET,
XGPIO_GIE_GINTR_ENABLE_MASK);

    // interrupt controller configuration
    Xil_Out32(XPAR_AXI_INTC_0_BASEADDR + XIN_IMR_OFFSET, 0x20);
    Xil_Out32(XPAR_AXI_INTC_0_BASEADDR + XIN_IER_OFFSET,
XPAR_AXI_GPIO_0_IP2INTC_IRPT_MASK | XPAR_AXI_TIMER_0_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();

    return 0;
}

void toggleSwitchHandler()
{
    unsigned int switchState = 0;
    switchState = Xil_In16(XPAR_AXI_GPIO_0_BASEADDR + XGPIO_DATA_OFFSET);
    //set timePeriod:
    timePeriod = switchState;
    if (timePeriod == 0)
    {
        timePeriod = 1;
    }
    else
    {
        timePeriod = timePeriod / 10; // s
        timePeriod = timePeriod * 100000000 - 2;
        setupTimer();
    }

    // clear interrupt
    Xil_Out32(XPAR_AXI_GPIO_0_BASEADDR + XGPIO_ISR_OFFSET,
Xil_In32(XPAR_AXI_GPIO_0_BASEADDR + XGPIO_ISR_OFFSET));
}

void timeExpiredHandler()
{
    // Read and output voltage
    voltageLevel = Xil_In16(XPAR_AXI_QUAD_SPI_1_BASEADDR + XSP_DRR_OFFSET) %
0x8000;
    int calculatedVoltage;
    calculatedVoltage = voltageLevel * 3300 / 0xffff / 2;
    counter = -counter;

    xil_printf("The current voltage is %d mV\n", calculatedVoltage, counter);
    //
    Xil_Out32(XPAR_SPI_1_BASEADDR + XSP_DTR_OFFSET, 0x0);
}

```

```

        Xil_Out32(XPAR_AXI_TIMER_0_BASEADDR + XTC_TCSR_OFFSET,
Xil_In32(XPAR_AXI_TIMER_0_BASEADDR + XTC_TCSR_OFFSET)); // clear timer interrupt
    }

void setupTimer()
{
    // Timer initialization
    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); // write TCSR
    Xil_Out32(XPAR_AXI_TIMER_0_BASEADDR + XTC_TLR_OFFSET, timePeriod);
    Xil_Out32(XPAR_AXI_TIMER_0_BASEADDR + XTC_TCSR_OFFSET,
Xil_In32(XPAR_AXI_TIMER_0_BASEADDR + XTC_TCSR_OFFSET) | XTC_CSR_LOAD_MASK); //
Load count
    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);
    // Start timing with auto-reload, interrupt enabled, and counting down
}

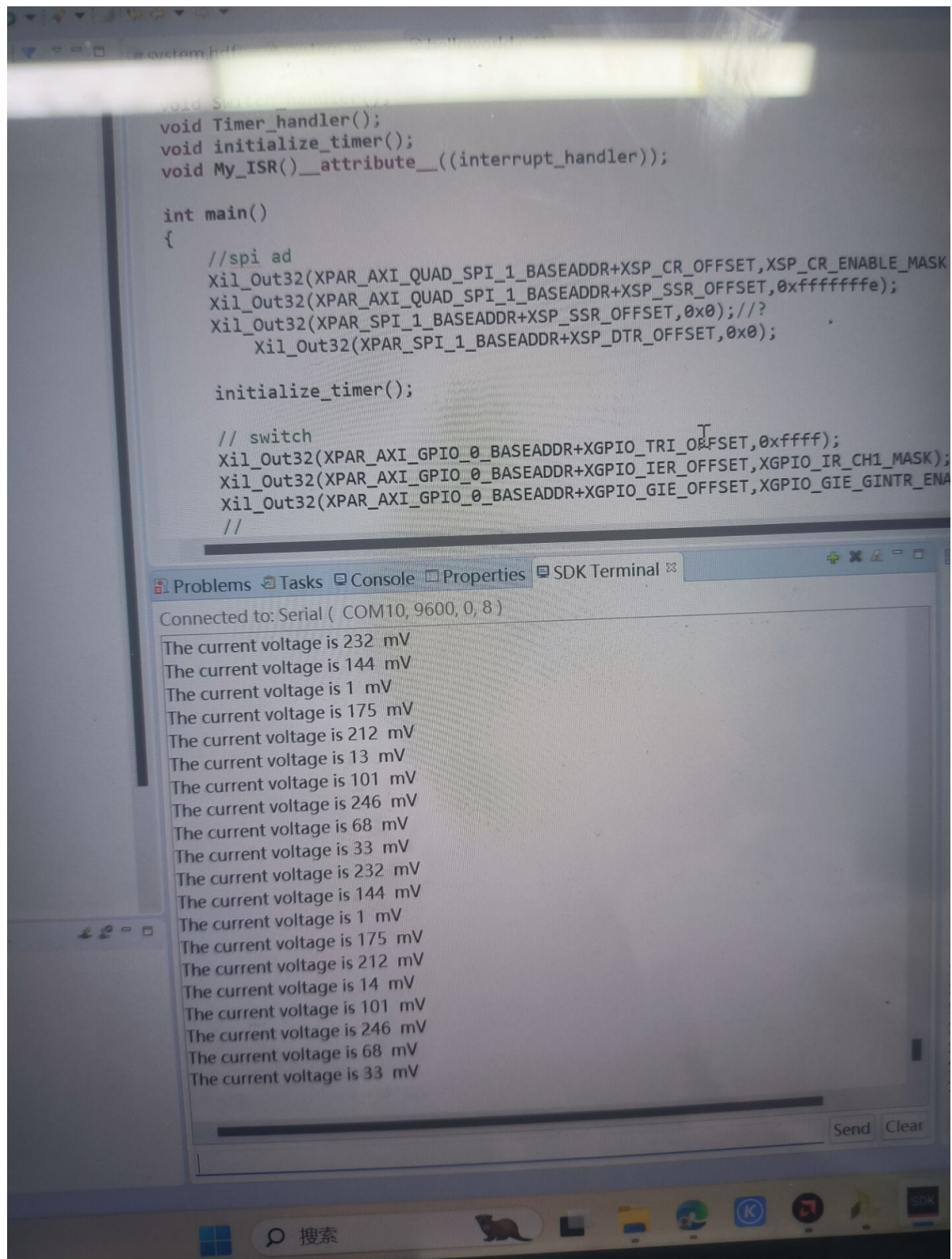
void customInterruptServiceRoutine()
{
    int interruptStatus;
    interruptStatus = Xil_In32(XPAR_AXI_INTC_0_BASEADDR + XIN_ISR_OFFSET);

    if ((interruptStatus & XPAR_AXI_TIMER_0_INTERRUPT_MASK) ==
XPAR_AXI_TIMER_0_INTERRUPT_MASK)
    {
        timeExpiredHandler();
    }

    if ((interruptStatus & XPAR_AXI_GPIO_0_IP2INTC_IRPT_MASK) ==
XPAR_AXI_GPIO_0_IP2INTC_IRPT_MASK)
    {
        toggleSwitchHandler();
    }
    Xil_Out32(XPAR_AXI_INTC_0_BASEADDR + XIN_IAR_OFFSET, interruptStatus);
}

```

## 实验结果



The image shows a screenshot of a development environment. The top part is a C code editor with the following code:

```
void Timer_handler();
void initialize_timer();
void My_ISR()__attribute__((interrupt_handler));

int main()
{
    //spi ad
    Xil_Out32(XPAR_AXI_QUAD_SPI_1_BASEADDR+XSP_CR_OFFSET,XSP_CR_ENABLE_MASK);
    Xil_Out32(XPAR_AXI_QUAD_SPI_1_BASEADDR+XSP_SSR_OFFSET,0xfffffffffe);
    Xil_Out32(XPAR_SPI_1_BASEADDR+XSP_SSR_OFFSET,0x0);//?
    Xil_Out32(XPAR_SPI_1_BASEADDR+XSP_DTR_OFFSET,0x0);

    initialize_timer();

    // switch
    Xil_Out32(XPAR_AXI_GPIO_0_BASEADDR+XGPIO_TRI_OFFSET,0xffff);
    Xil_Out32(XPAR_AXI_GPIO_0_BASEADDR+XGPIO_IER_OFFSET,XGPIO_IR_CH1_MASK);
    Xil_Out32(XPAR_AXI_GPIO_0_BASEADDR+XGPIO_GIE_OFFSET,XGPIO_GIE_GINTR_ENA);
    //
}
```

The bottom part is an SDK Terminal window showing the output of the program. It is connected to a serial port (COM10, 9600, 0, 8). The output consists of 16 lines, each reporting the current voltage in mV:

```
Connected to: Serial ( COM10, 9600, 0, 8 )
The current voltage is 232 mV
The current voltage is 144 mV
The current voltage is 1 mV
The current voltage is 175 mV
The current voltage is 212 mV
The current voltage is 13 mV
The current voltage is 101 mV
The current voltage is 246 mV
The current voltage is 68 mV
The current voltage is 33 mV
The current voltage is 232 mV
The current voltage is 144 mV
The current voltage is 1 mV
The current voltage is 175 mV
The current voltage is 212 mV
The current voltage is 14 mV
The current voltage is 101 mV
The current voltage is 246 mV
The current voltage is 68 mV
The current voltage is 33 mV
```

## 实验小结

理论课时我并没有很好的理解定时器及其代码，所以在第一周时做起来格外困难。此外，Vivado 2023.2版本的IDE并不支持串口输出，所以我回退到2018版本，才成功完成了任务。