串行 IO 接口程序设计

通信 2002 班　涂增基　U202013990

一、实验任务

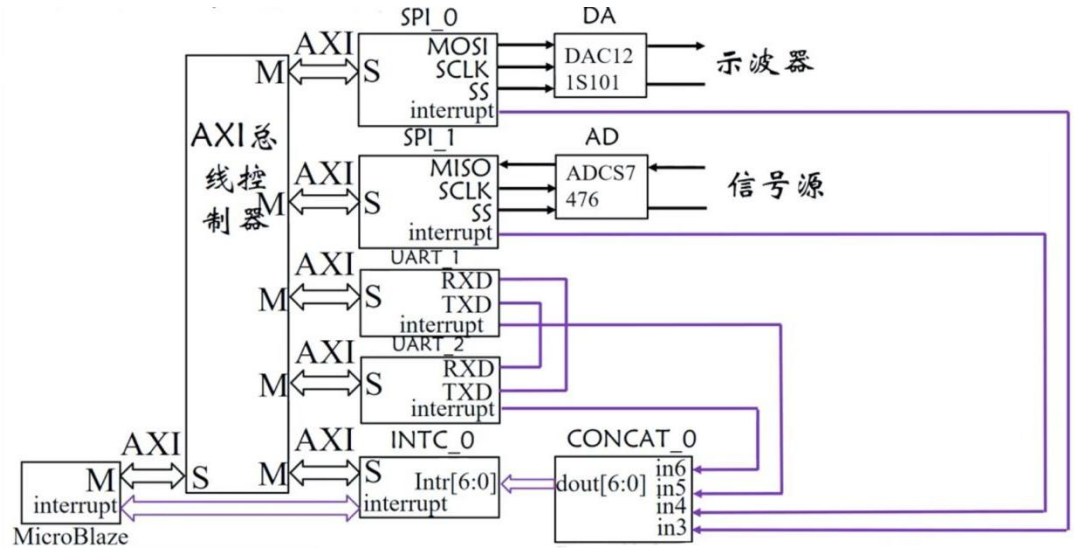
利用 SPI IP 核，timer IP 核、GPIO IP 核以及 DA 模块，控制 DA 模块输出周期可变锯齿波，且锯齿波周期由 switch 控

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，输出一个新数据到 DA 转换器。

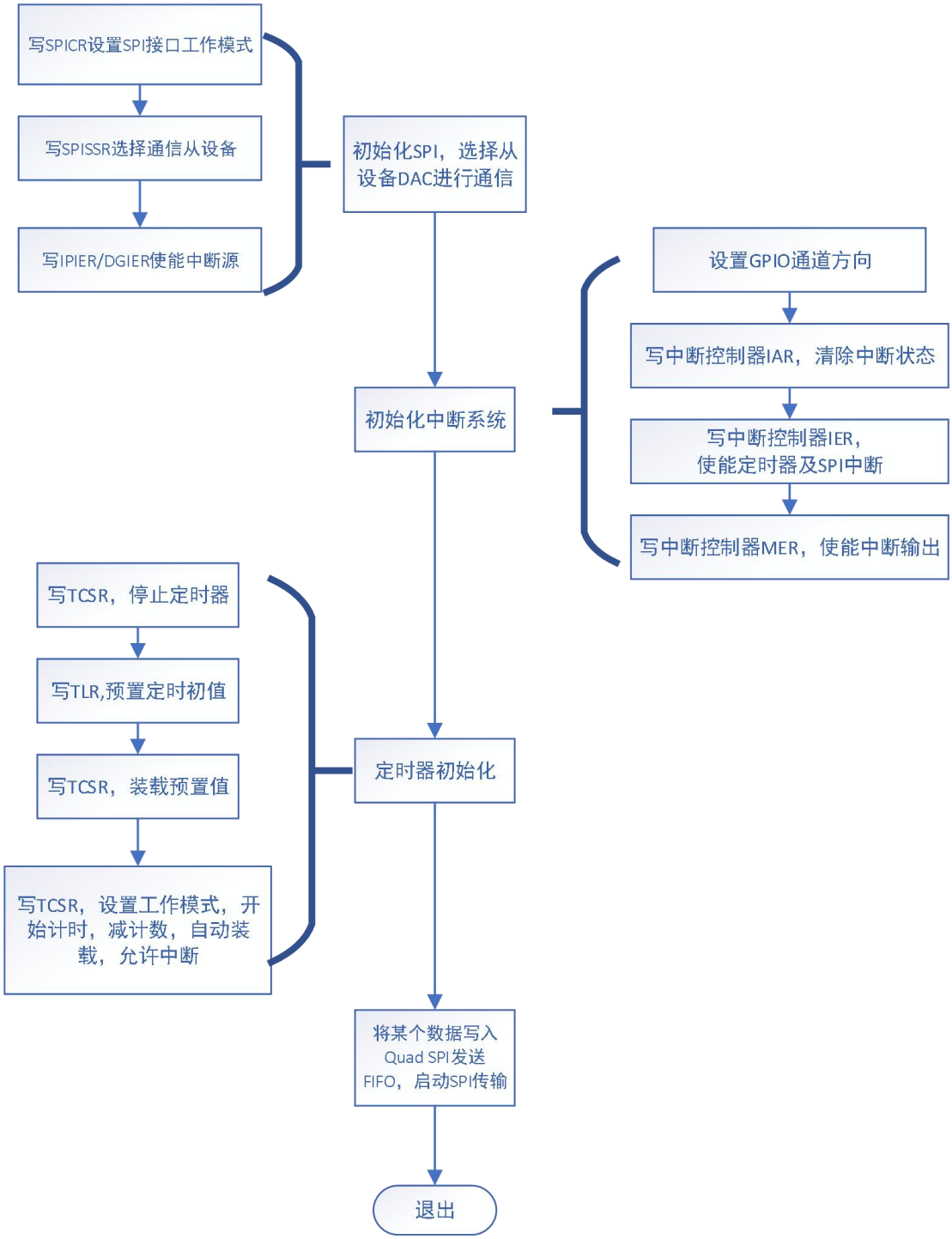
二、实验目的

- 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

三、硬件电路框图



四、软件流程图


写SPICR设置SPI接口工作模式
写SPISSR选择通信从设备
写IPIER/DGIER使能中断源
初始化SPI，选择从设备DAC进行通信
设置GPIO通道方向
写中断控制器IAR，清除中断状态
写中断控制器IER，使能定时器及SPI中断
写中断控制器MER，使能中断输出
初始化中断系统
写TCSR，停止定时器
写TLR,预置定时初值
写TCSR，装载预置值
写TCSR，设置工作模式，开始计时，减计数，自动装载，允许中断
定时器初始化
将某个数据写入Quad SPI发送FIFO，启动SPI传输
退出

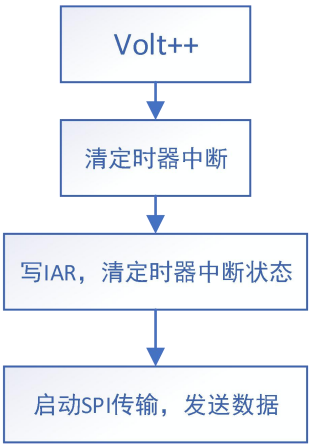


图3 定时器中断服务程序流程

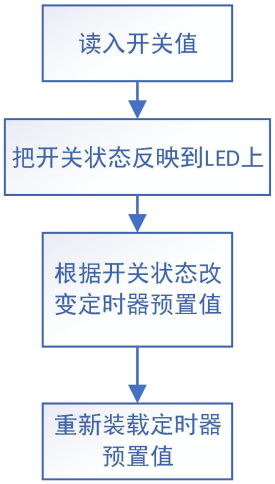


图4 开关中断服务程序流程

五、软件源代码

```
#include "xil_io.h"
#include "xil_exception.h"
#include "xintc_l.h"
#include "xspi_l.h"
#include "xtmrctr_l.h"
#include "xgpio_l.h"
#include "xparameters.h"

int RESET_VALUE = 100000000/0xfff-2;
void My_ISR() __attribute__((interrupt_handler));
u16 volt=0;

void switchHandler();     //开关中断
void timerHandler();      //按键中断

int main()
{

      RESET_VALUE = 100000000/0xfff-2;
      //设定 SPI 接口的通信模式，设定 SPI 为主设备，CPOL=1,CPHA-0,时钟相位 180°，自动方式，高位优先传送
      Xil_Out32(XPAR_AXI_QUAD_SPI_0_BASEADDR+XSP_CR_OFFSET,XSP_CR_ENABLE_MASK|XSP_CR_MASTER
_MODE_MASK|XSP_CR_CLK_POLARITY_MASK);
    //设定 SSR 寄存器
      Xil_Out32(XPAR_AXI_QUAD_SPI_0_BASEADDR+XSP_SSR_OFFSET,0xfffffffe);
      //开放 SPI 发送寄存器空中断
      Xil_Out32(XPAR_AXI_QUAD_SPI_0_BASEADDR+XSP_IIER_OFFSET,XSP_INTR_TX_EMPTY_MASK);//中断源为
```

```
SPI 接口发送完数字信号则产生中断
    Xil_Out32(XPAR_AXI_QUAD_SPI_0_BASEADDR+XSP_DGIER_OFFSET,XSP_GINTR_ENABLE_MASK);  //开启
SPI 接口的中断输出

    //GPIO 中断使能
    Xil_Out32(XPAR_AXI_GPIO_0_BASEADDR+XGPIO_TRI_OFFSET,0xffff);//开关 switch 设置为输入
    Xil_Out16(XPAR_AXI_GPIO_0_BASEADDR+XGPIO_TRI2_OFFSET,0X0);        //LED 设置为输出
    Xil_Out32(XPAR_AXI_GPIO_0_BASEADDR+XGPIO_IER_OFFSET,XGPIO_IR_CH1_MASK);//GPIO_0 中断使能
    Xil_Out32(XPAR_AXI_GPIO_0_BASEADDR+XGPIO_GIE_OFFSET,XGPIO_GIE_GINTR_ENABLE_MASK);//GPIO_
0 全局中断使能

    //定时器初始化
        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 TCSR，停止定时器
        Xil_Out32(XPAR_AXI_TIMER_0_BASEADDR+XTC_TLR_OFFSET,RESET_VALUE);//写 TLR，预置计数初
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);
//装载计数初值
        Xil_Out32(XPAR_AXI_TIMER_0_BASEADDR+XTC_TCSR_OFFSET

                ,(Xil_In32(XPAR_AXI_TIMER_0_BASEADDR+XTC_TCSR_OFFSET)&~XTC_CSR_LOAD_MASK)
\

|XTC_CSR_ENABLE_TMR_MASK|XTC_CSR_AUTO_RELOAD_MASK|XTC_CSR_ENABLE_INT_MASK|XTC_CSR_DOWN
_COUNT_MASK);//开始计时 自主获取允许中断减计数 */



    //中断控制器 intr0 中断源使能
      Xil_Out32(XPAR_INTC_0_BASEADDR+XIN_IER_OFFSET,
            XPAR_AXI_TIMER_0_INTERRUPT_MASK|
            XPAR_AXI_QUAD_SPI_1_IP2INTC_IRPT_MASK|
            XPAR_AXI_GPIO_2_IP2INTC_IRPT_MASK);      //开放定时器 T0 及 SPI 中断
    Xil_Out32(XPAR_AXI_INTC_0_BASEADDR+XIN_MER_OFFSET,XIN_INT_MASTER_ENABLE_MASK|XIN_INT_HAR
DWARE_ENABLE_MASK);
    //处理器中断使能
    microblaze_enable_interrupts();
    //启动传输，发送数据 0
    Xil_Out16(XPAR_AXI_QUAD_SPI_0_BASEADDR+XSP_DTR_OFFSET,0); //启动 SPI 传输，产生时钟和片选信号
    //while(1);
```

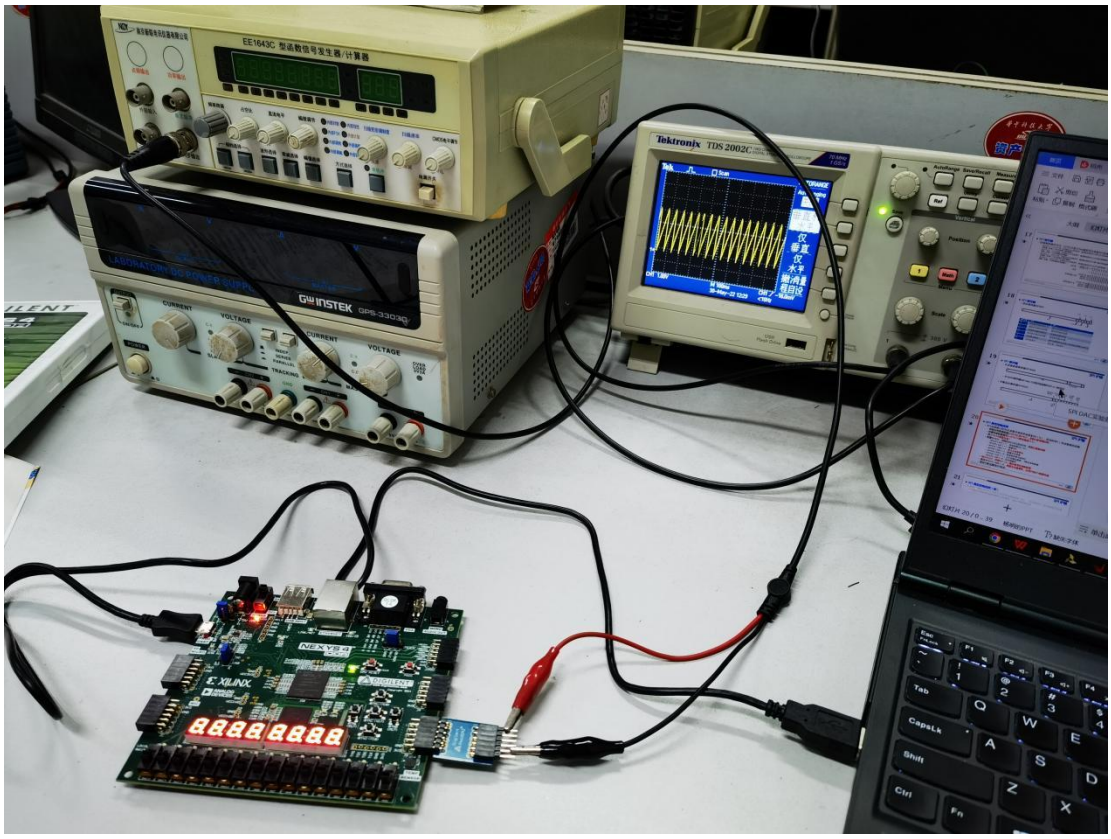
```
        return 0;
}
void My_ISR()
{
      int status;
      status=Xil_In32(XPAR_AXI_INTC_0_BASEADDR+XIN_ISR_OFFSET);     //读入中断状态
     if((status&XPAR_AXI_GPIO_0_IP2INTC_IRPT_MASK)==XPAR_AXI_GPIO_0_IP2INTC_IRPT_MASK)
  {
           switchHandler();     //如果开关产生了中断，则进入开关中断服务函数
      }
      else if((status&XPAR_AXI_TIMER_0_INTERRUPT_MASK)==XPAR_AXI_TIMER_0_INTERRUPT_MASK)
     {
         timerHandler();      //如果定时器产生了中断，则进入定时器中断服务函数

}

    Xil_Out32(XPAR_AXI_INTC_0_BASEADDR+XIN_IAR_OFFSET,status);
}

void switchHandler() //开关中断服务程序
{
       int sw;
      sw = Xil_In16(XPAR_AXI_GPIO_0_BASEADDR+XGPIO_DATA_OFFSET);     //读入开关值
        Xil_Out32(XPAR_AXI_GPIO_0_BASEADDR+XGPIO_DATA2_OFFSET,sw);     //把开关的状态反映到 LED 上
        Xil_Out16(XPAR_AXI_GPIO_0_BASEADDR+XGPIO_ISR_OFFSET,0x01);
        int min=6000000;     //最短时间 60ms
        RESET_VALUE=((sw&0x0000ffff)*1434+min)/0xfff-2;     //步进值 1434=（最大时长 100000000-最小时长
6000000）/2^16（=65536） 每拨动一个开关加一个步进时长
        //读入的开关值 sw 一定要与上 0x0000ffff 保存低 16 位，否则会自动有符号数扩展，装载进去的值就会是个负的
        //主程序中的是定时器初始化，此处开关改变了定时器的预置值，故需要重新装载
        int status=Xil_In32(XPAR_TMRCTR_0_BASEADDR+XTC_TCSR_OFFSET);
        Xil_Out32(XPAR_TMRCTR_0_BASEADDR+XTC_TCSR_OFFSET,status&(~XTC_CSR_ENABLE_TMR_MASK));
        Xil_Out32(XPAR_TMRCTR_0_BASEADDR+XTC_TLR_OFFSET,RESET_VALUE);   //为定时器装载改变后的预置
值

        Xil_Out32(XPAR_TMRCTR_0_BASEADDR+XTC_TCSR_OFFSET,Xil_In32(XPAR_TMRCTR_0_BASEADDR+XTC_T
CSR_OFFSET)|XTC_CSR_LOAD_MASK);
        status=(status&(~XTC_CSR_LOAD_MASK))|XTC_CSR_ENABLE_TMR_MASK;
        Xil_Out32(XPAR_TMRCTR_0_BASEADDR+XTC_TCSR_OFFSET,status);
}

void timerHandler() //锯齿波形成
{
              volt++;     //输出锯齿波，每中断一次，输出的数字信号＋1
```

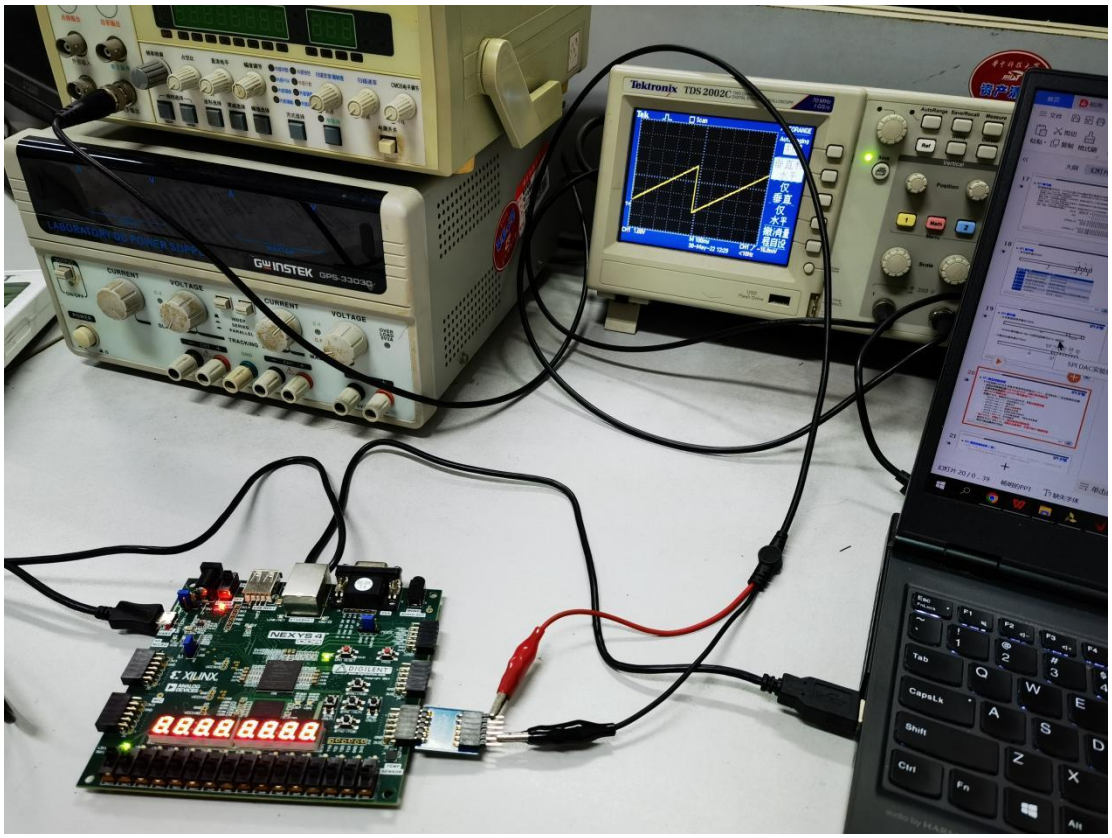
```
        Xil_Out32(XPAR_TMRCTR_0_BASEADDR+XTC_TCSR_OFFSET,Xil_In32(XPAR_TMRCTR_0_BASEADDR+XTC_TCSR_OFFSET));//清定时器中断，不然一直中断周期会不对
            Xil_Out32(XPAR_INTC_0_BASEADDR+XIN_IAR_OFFSET,0x8);  //普通中断模式，手动清中断
            Xil_Out16(XPAR_SPI_0_BASEADDR+XSP_DTR_OFFSET,volt&0xfff);//启动SPI传输,产生时钟和片选信号，发送数据，有效数据为低 12 位
}
```

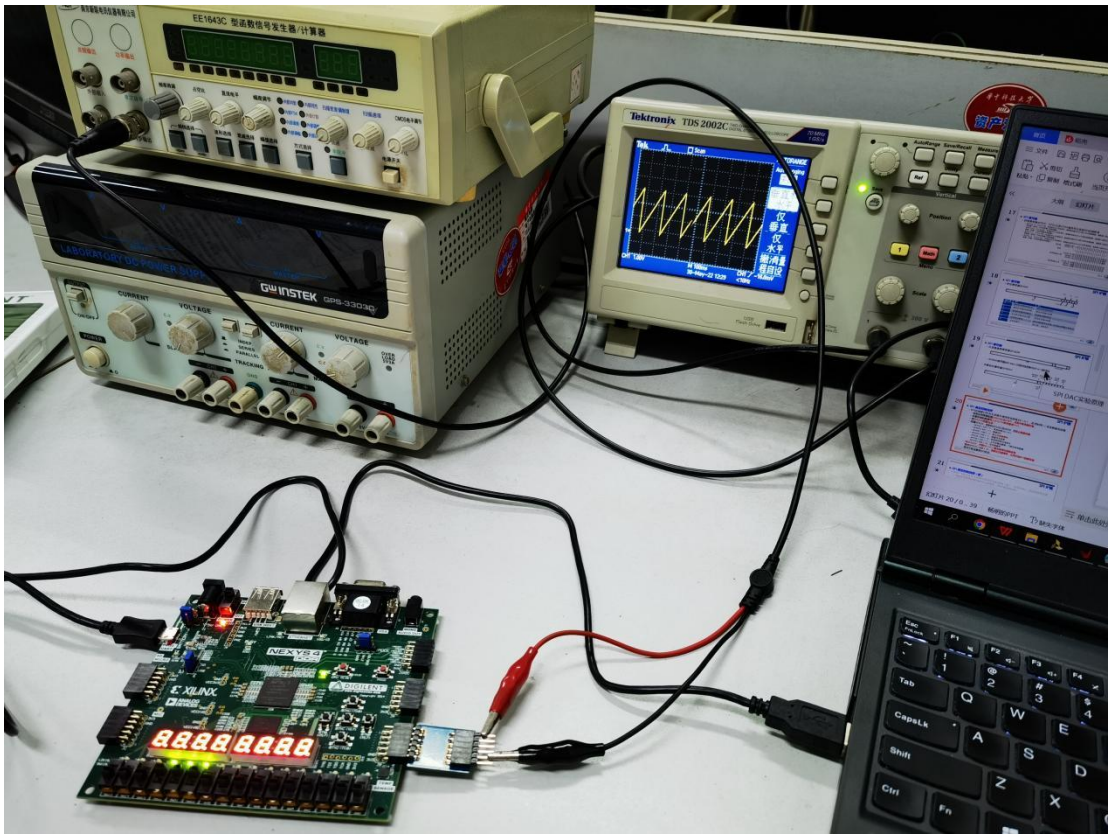
六、实验结果

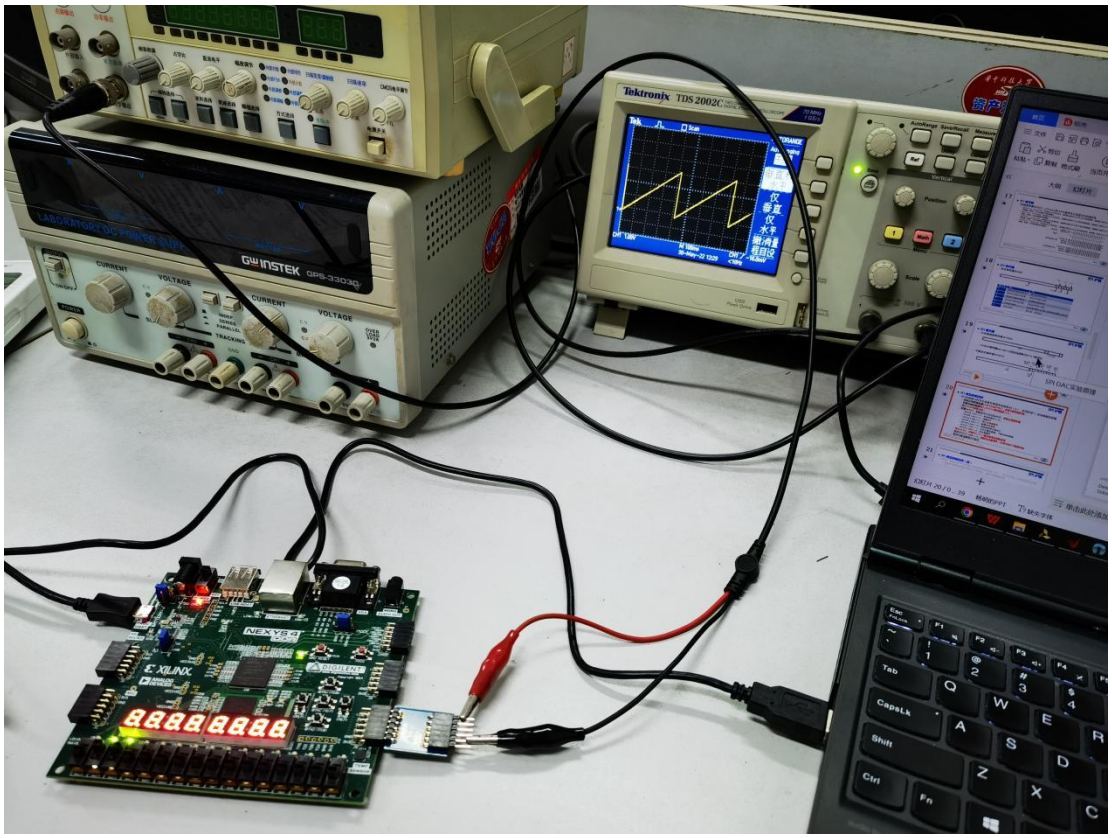
用 DA 转换器输出波形到示波器，得到如下测试结果：

拨动开关，可以输出不同周期的锯齿波。




Tektronix TDS 2002C
GW INSTEK GPS-3303C
XILINX


EE1643C 型函数信号发生器/计算器
Tektronix TDS 2002C
GW INSTEK GPS-3303C
XILINX



七、心得体会

在本次实验中，由于理论课上对 UARTLite 和 SPI 都学的不是很透彻，一开始做这个实验的时候很是困难。对着实验书学习了很久，才开始写代码。

平台建立是遇到了一些问题的，一开始我跟着学习通上的视频建立平台，最终导出 bit 流文件总是出错，原来是引脚约束出了点问题。可能是版本不一样的问题，我用 2019.2 版本按照视频上的做法无法约束引脚，最后只好自己在 IOPort 里面进行引脚约束。

中断服务函数中，那个周期的设置并不简单，遇到了不少的问题，sw 的值一定要&上 0x0000ffff，否则会自动有符号扩展。然后一定要记得清除中断！本次实验中我在写计时器的时候，最后忘了清除中断，最终的周期就是错误的结果。

本次实验对中断和串行输入输出有了更深的理解，收获良多。