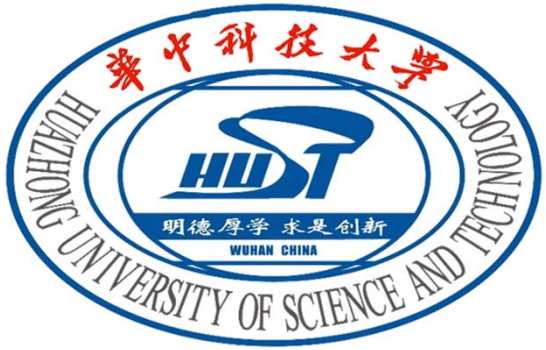
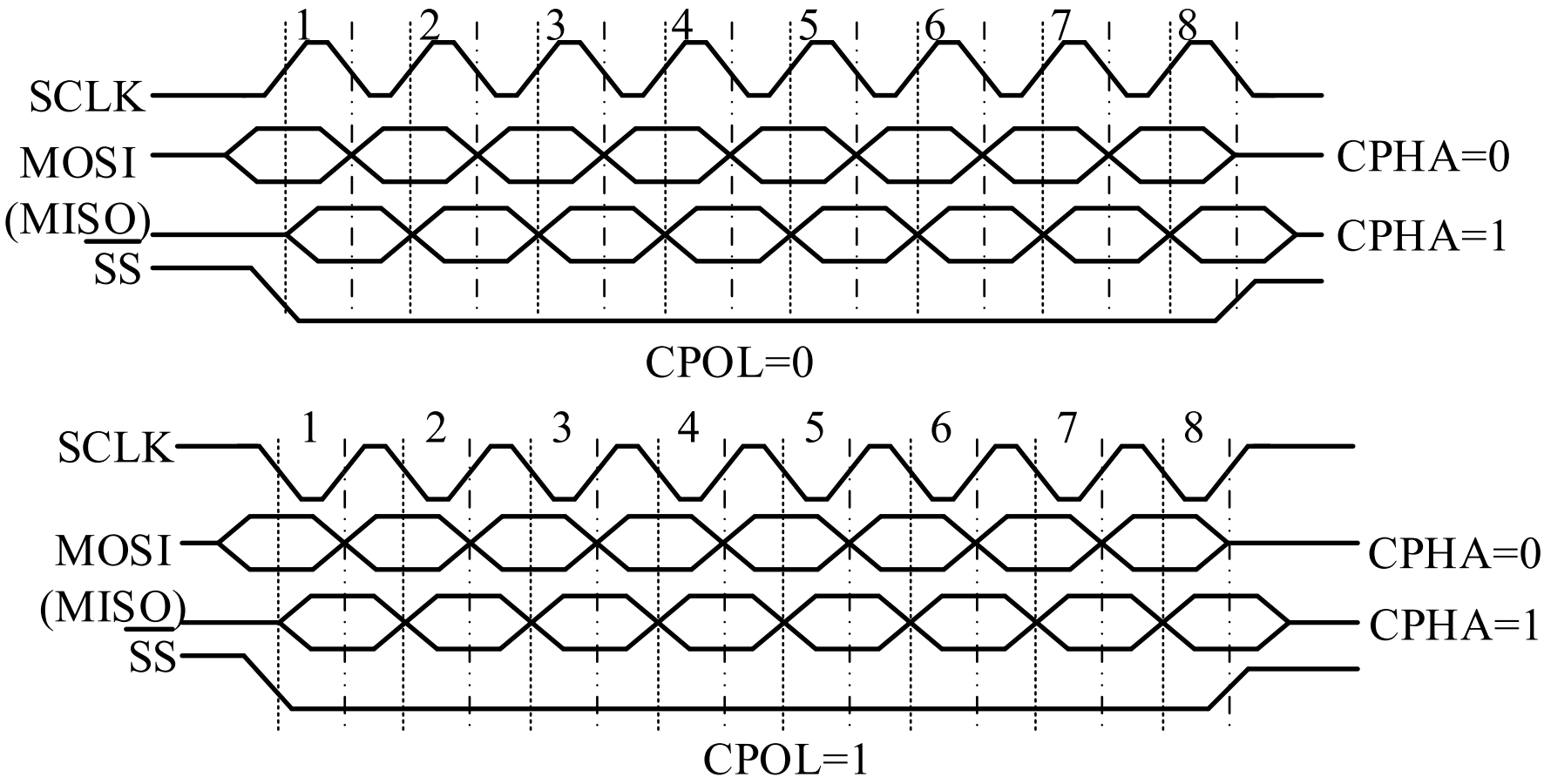
微机原理与接口技术

串行SPI接口中断程序设计

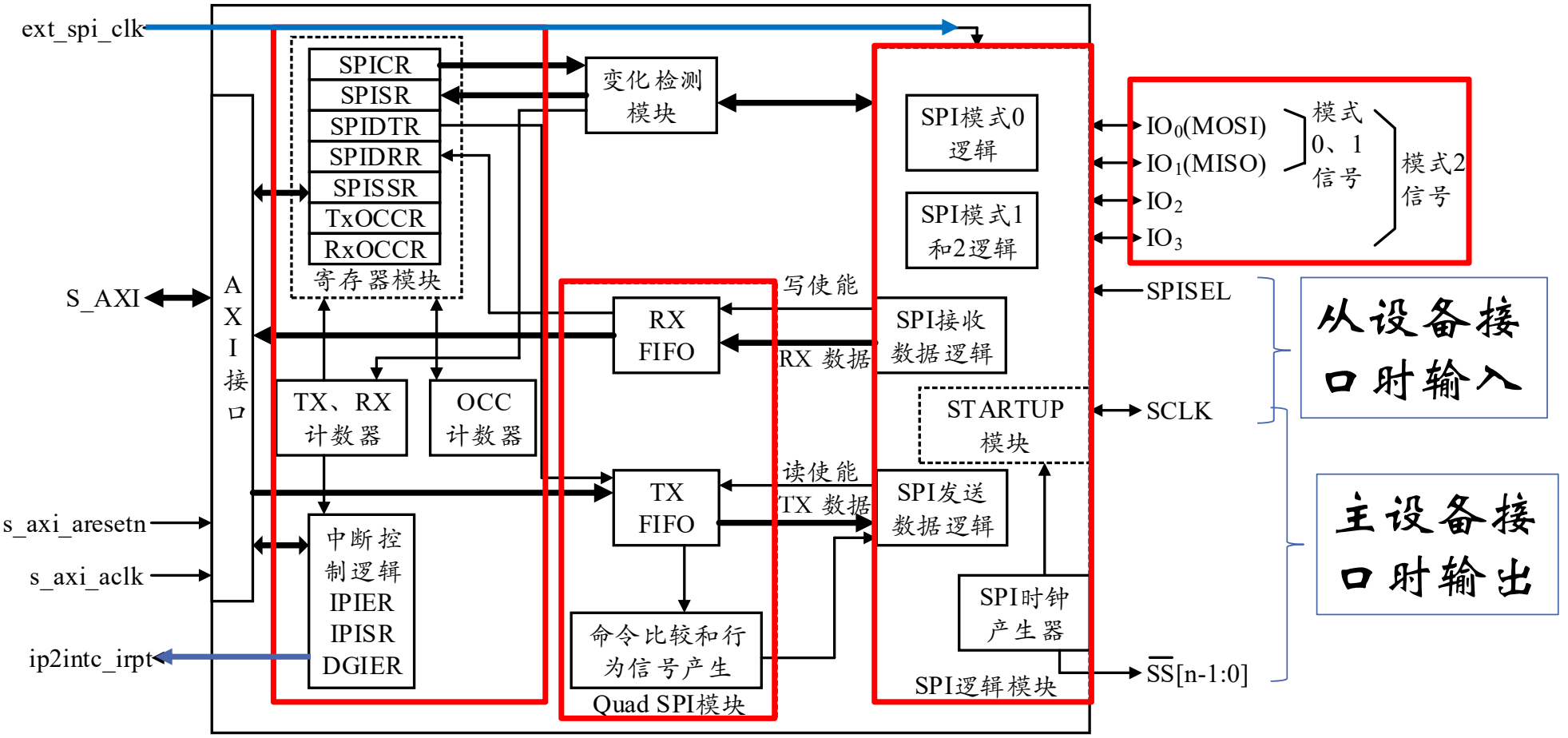
华中科技大学　左冬红



回顾SPI总线通信流程



AXI SPI IP核简介



SPI编程寄存器存储映像

寄存器	偏移地址	含义
SRR	0x40	软件复位寄存器，写0x0000000A复位接口
SPICR	0x60	控制寄存器，设定AXI Quad SPI IP核工作方式
SPISR	0x64	状态寄存器，指示AXI Quad SPI IP核工作状态
SPIDTR	0x68	发送数据寄存器或发送数据FIFO
SPIDRR	0x6C	接收数据寄存器或接收数据FIFO
SPISSR	0x70	从设备选择寄存器
TxOCCR	0x74	发送FIFO占用长度指示，值+1表示发送FIFO有效数据的长度
RxOCCR	0x78	接收FIFO占用长度指示，值+1表示接收FIFO有效数据的长度
DGIER	0x1C	设备总中断使能寄存器，仅最高位有效，D_{31}=1使能接口中断请求输出
IPISR	0x20	中断状态寄存器
IPIER	0x28	中断使能寄存器

SPICR寄存器含义

位	含义	1	0
0	回环	SPI发送端(MOSI)与接收端(MISO)内部连通形成环路	SPI发送端(MOSI)与接收端(MISO)独立工作
1	启用接口	启用SPI接口(与SPI总线连接)	禁用SPI接口(与SPI总线断开、高阻态)
2	主设备	SPI主设备接口	SPI从设备接口
3	CPOL	空闲时时钟为高电平	空闲时时钟为低电平
4	CPHA	数据信号在时钟第二个边沿(相位180°)稳定有效	数据信号在时钟第一个边沿(相位0°)稳定有效
5	Tx 复位	复位发送FIFO指针	无意义
6	Rx 复位	复位接收FIFO指针	无意义
7	手动控制从设备选择	程序控制从设备选择SS输出，即写SPISSR立即反映到$\overline{66}$[n-1:0]	协议逻辑控制SPISSR输出到$\overline{66}$[n-1:0]
8	禁止主设备事务	禁止主设备事务；若为从设备则无意义	使能主设备事务
9	低位优先	低位优先传送	高位优先传送
10~31	保留，无意义		

SPISR寄存器含义

位	含义	0	1
0	接收寄存器/FIFO空否	非空	空
1	接收寄存器/FIFO满否	未满	满
2	发送寄存器/FIFO空否	非空	空
3	发送寄存器/FIFO满否	未满	满
4	模式错误否	无错误	错误。主设备接口时，$\overline{66}$输入低电平有效信号则置位
5	从设备选中否	选中	未选中
6	CPOL_CPHA错误否	无	错误。DSPI或QSPI模式时，CPOL、CPHA仅支持配置为00或11，若配置为10或01则置位
7	从设备模式错误否	无	错误。DSPI或QSPI模式时，仅支持主设备模式，若配置为从设备模式则置位
8	高位优先错误否	无	错误。DSPI或QSPI模式时，仅支持高位优先传送，若配置为低位优先传送则置位
9	回环错误否	无	错误。DSPI或QSPI模式时，不支持回环，若配置为回环则置位
10	命令错误否	无	错误。DSPI或QSPI模式时，若复位后发送FIFO中的第一个数据不是支持的命令则置位
11~31	保留，无意义		

SPIDTR\DRR\FIFO有效数据位

SPIDTR、SPIDRR以及发送、接收FIFO有效数据位长度与SPI一帧数据位数n一致

n在标准SPI模式时可为8、16、32，其余模式时仅可为8

IPISR\IER寄存器含义

位	含义
31:14	保留，无意义
13	1：命令错误，DSPI或QSPI模式时，复位后发送FIFO中的第一个数据不是支持的命令；0：标准SPI或无错误
12	1：回环错误，DSPI或QSPI模式时，配置为回环；0：标准SPI或无回环错误
11	1：DSPI或QSPI模式时，配置为低位优先传送；0：标准SPI或无低位优先设置错误
10	1：DSPI或QSPI模式时，配置为从设备模式；0：标准SPI或为主设备模式
9	1：DSPI或QSPI模式时，CPOL、CPHA配置为10或01 0：标准SPI或CPOL、CPHA为00或11
8	1：数据接收FIFO非空，仅采用FIFO且从设备模式时有效；0：空或其他模式
7	1：从设备被选中，仅工作在从设备模式时有效；0：未被选中或其他模式
6	1：Tx FIFO半空，表示FIFO深度为16时，TxOCCR寄存器的值从8变为7或FIFO深度为256时，TxOCCR寄存器的值从128变为127；0：其他情况
5	1：数据接收寄存器/FIFO过载（满接收）；0：未溢出
4	1：数据接收寄存器/FIFO满；0：未满
3	1：数据发送寄存器/FIFO欠载（空发送）；0：未溢出
2	1：数据发送寄存器/FIFO空；0：非空
1	1：从设备模式错误，配置为从设备但未启动SPI接口时，$\overline{66}$引脚输入低电平； 0：无错
0	1：模式错误，配置为主设备但$\overline{66}$引脚输入低电平；0：未错

SPI主设备接口中断方式控制程序流程

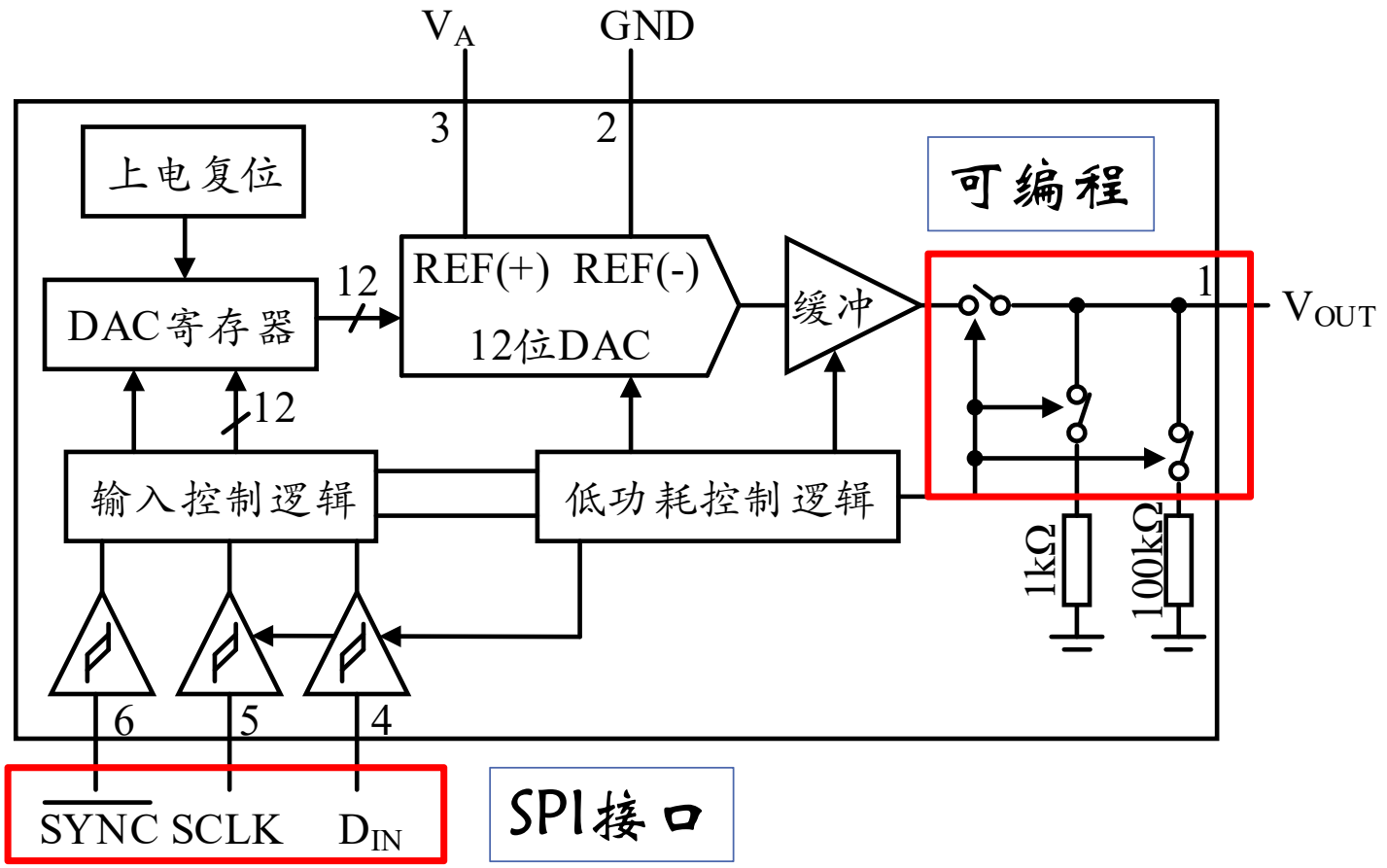
中断初始化

- 写SPICR设置SPI接口工作模式
- 自动输出SS
- 写SPISSR选择通信从设备
- 写IPIER\DGIER使能中断源
- 写SPIDTR发送数据，实现SPI总线全双工通信，触发中断源

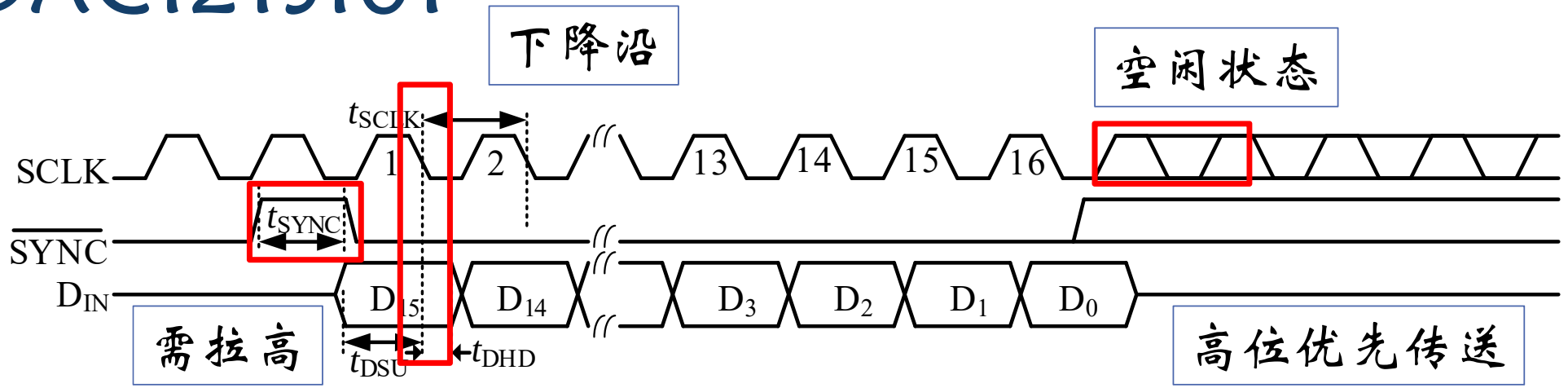
中断服务

- 读、写IPISR清除中断状态
- 读SPIDRR获取接收的数据
- 写SPIDTR发送下一个数据，触发下一次中断源

SPI应用示例——DAC121S101



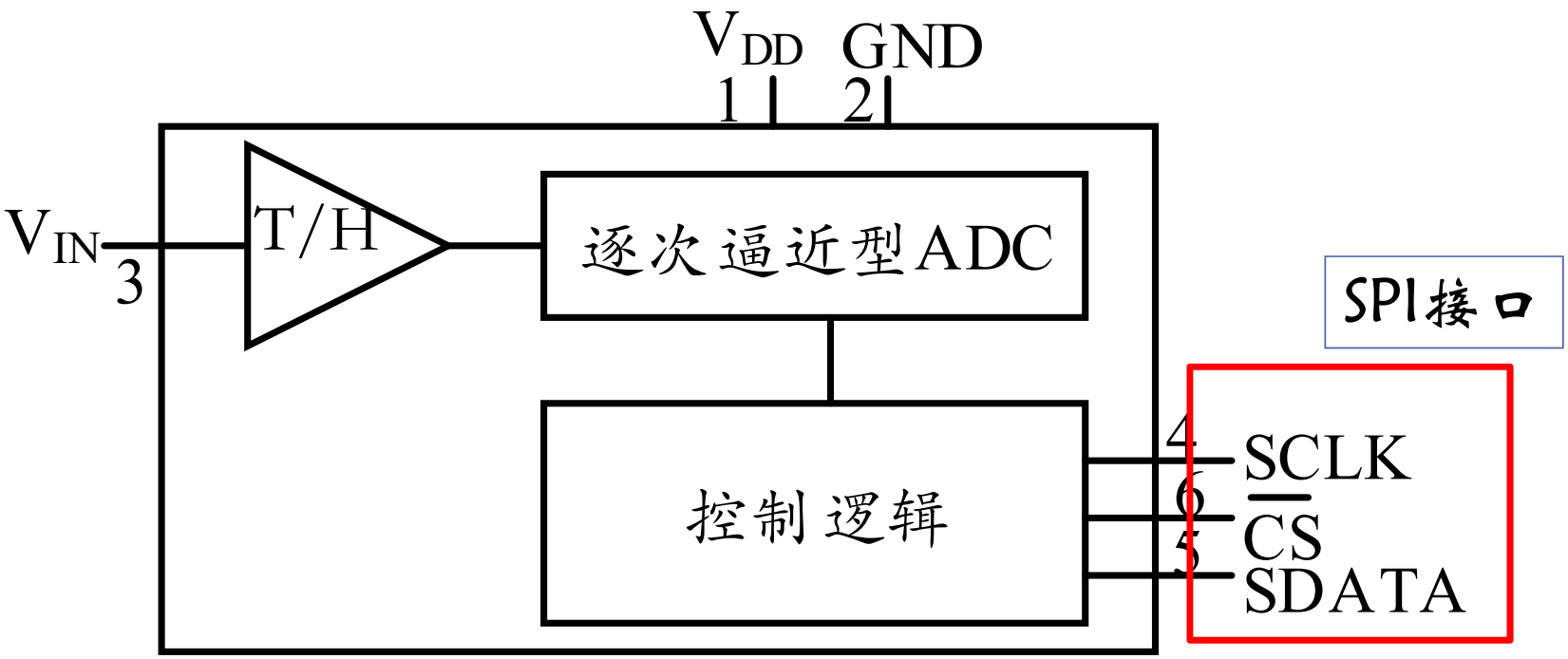
DAC121S101



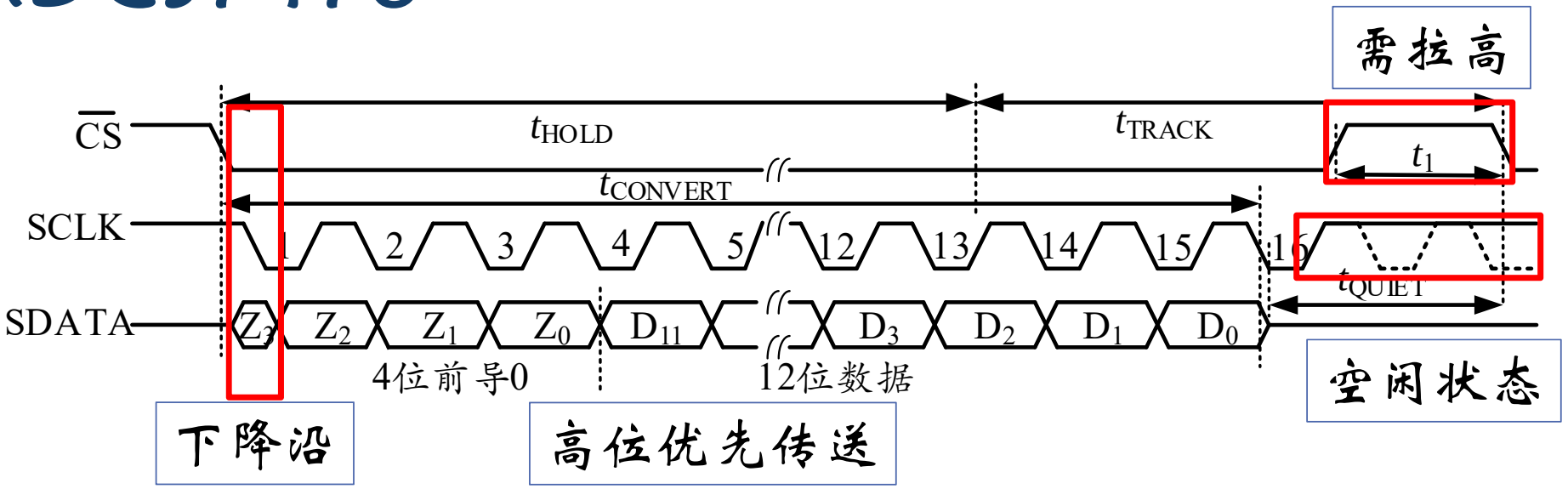
D_{15}	D_{14}	D_{13}	D_{12}	D_{11}	……	D_0
无意义		PD_1	PD_0	D		

PD_1	PD_0	V_{OUT}输出方式
0	0	V_{OUT}正常输出（不下拉）
0	1	V_{OUT}通过1kΩ电阻下拉
1	0	V_{OUT}通过100kΩ电阻下拉
1	1	V_{OUT}高阻（无输出）

SPI应用示例——ADCS7476



ADCS7476


需拉高
$\overline{CS}$
SCLK
SDATA
t_{HOLD}
t_{TRACK}
$t_{CONVERT}$
t_1
t_{QUIET}
1 2 3 4 5 12 13 14 15 16
Z_3 Z_2 Z_1 Z_0 D_{11} D_3 D_2 D_1 D_0
4位前导0
12位数据
下降沿
高位优先传送
空闲状态

SPI应用示例

某计算机系统要求利用ADCS7476采集某电路输出电压（范围为0~3.3V），采样频率为5kHz，连续采集100个数据，然后再利用DAC121S101 DA转换芯片将采集到的100个数据作为一个周期的数据样本以最快速度转换为模拟电压信号输出。试设计该数据采集及转换系统接口电路和控制程序。

定时采样采用定时器中断

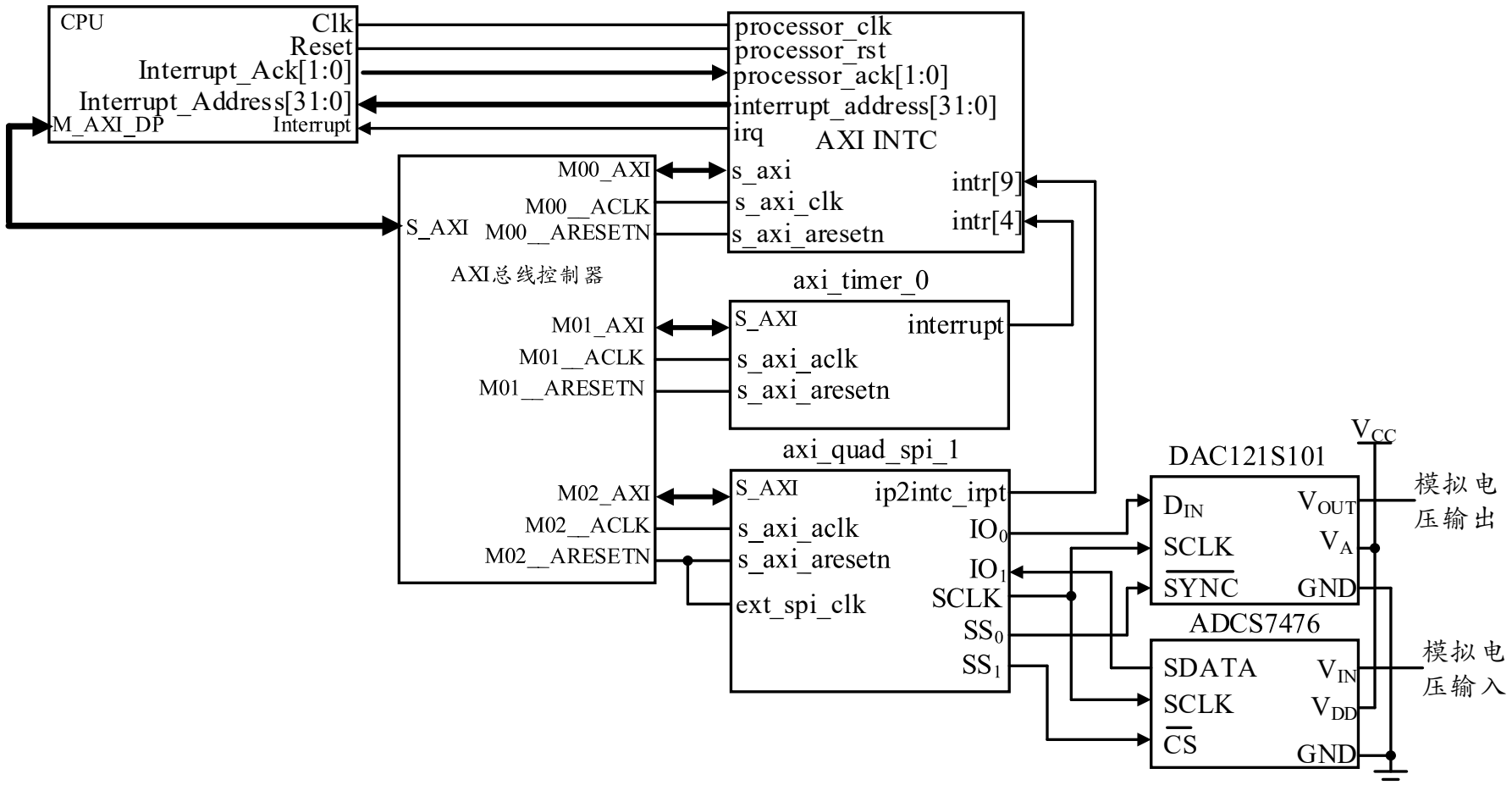
中断100次

数据发送完立即中断，继续发送下一个数据

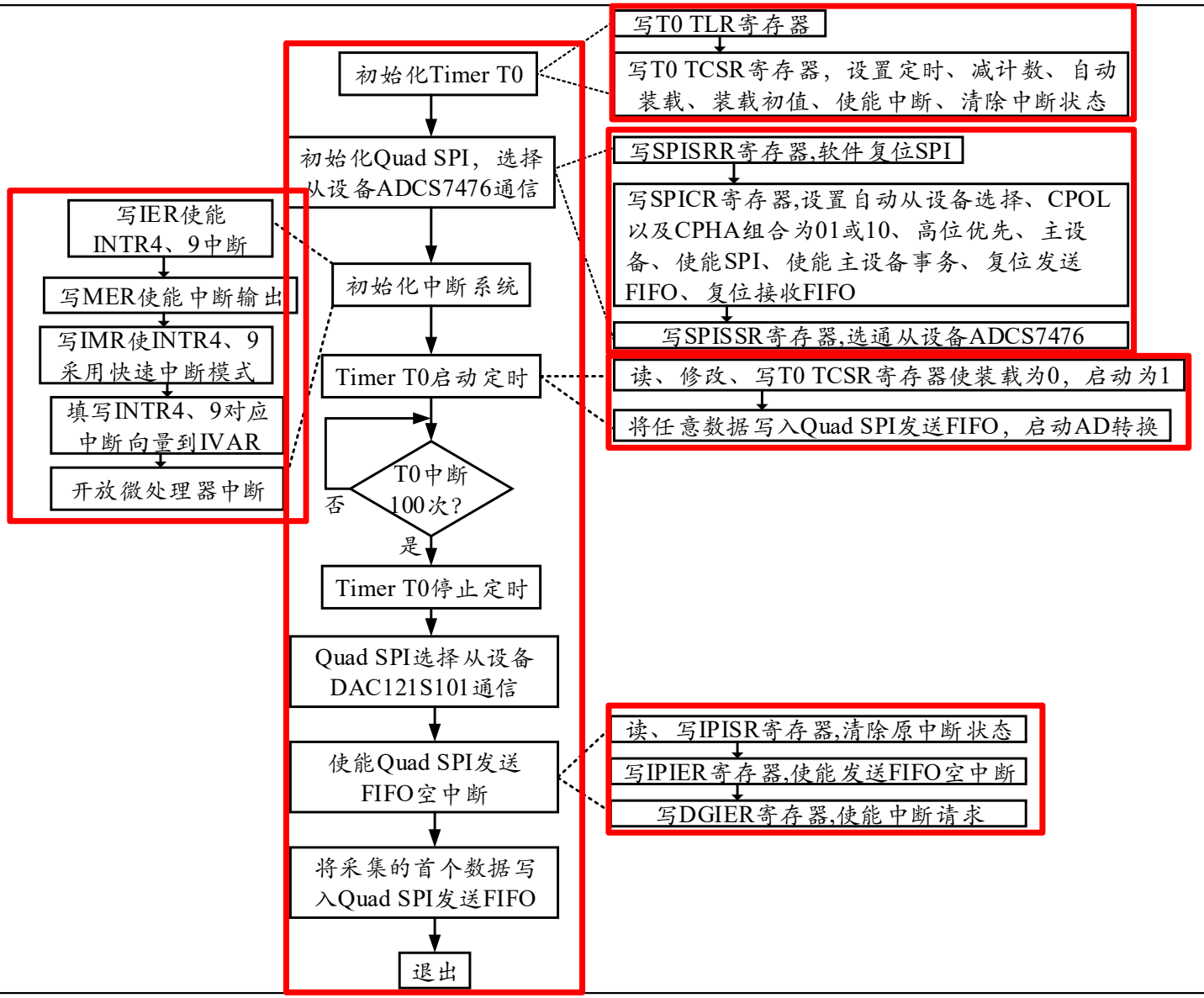
不停中断

定时器、中断控制器、SPI两个从设备（两个SPI）

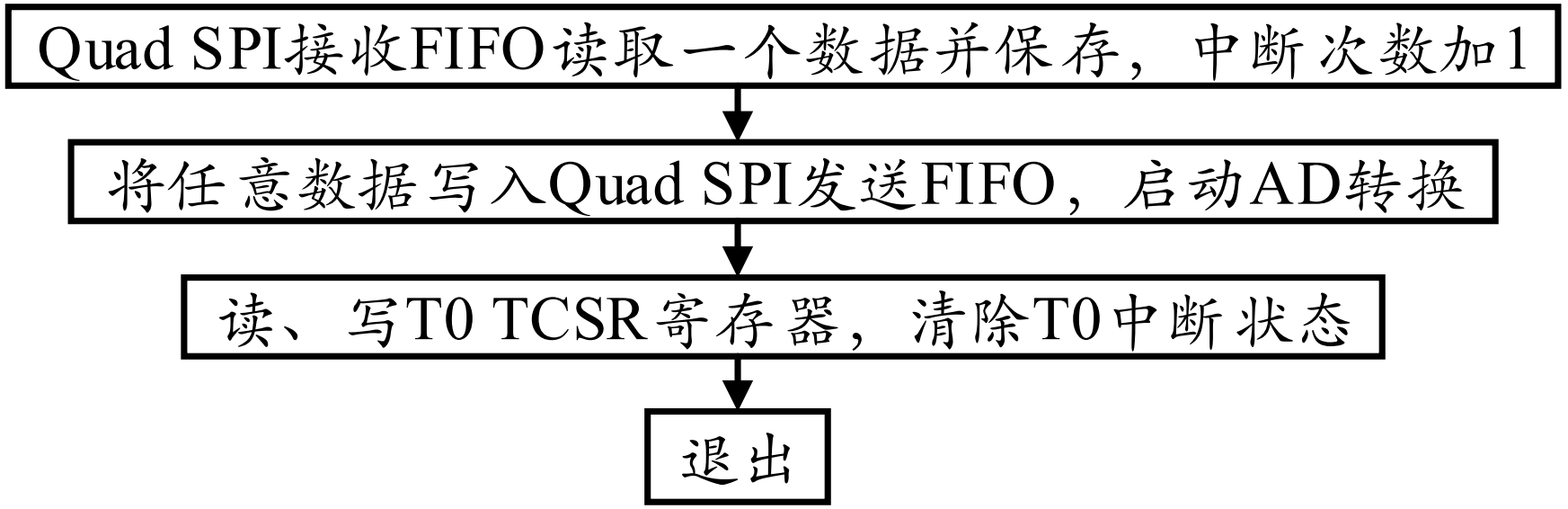
接口电路示例


CPU
Clk
Reset
Interrupt_Ack[1:0]
Interrupt_Address[31:0]
M_AXI_DP
Interrupt
processor_clk
processor_rst
processor_ack[1:0]
interrupt_address[31:0]
irq
AXI INTC
intr[9]
intr[4]
M00_AXI
M00__ACLK
S_AXI
M00__ARESETN
s_axi
s_axi_clk
s_axi_aresetn
AXI总线控制器
axi_timer_0
M01_AXI
M01__ACLK
M01__ARESETN
S_AXI
interrupt
s_axi_aclk
s_axi_aresetn
axi_quad_spi_1
M02_AXI
M02__ACLK
M02__ARESETN
S_AXI
ip2intc_irpt
s_axi_aclk
s_axi_aresetn
ext_spi_clk
IO0
IO1
SCLK
SS0
SS1
DAC121S101
DIN
SCLK
SYNC
VOUT
VA
GND
VCC
模拟电压输出
ADCS7476
SDATA
SCLK
CS
VIN
VDD
GND
模拟电压输入

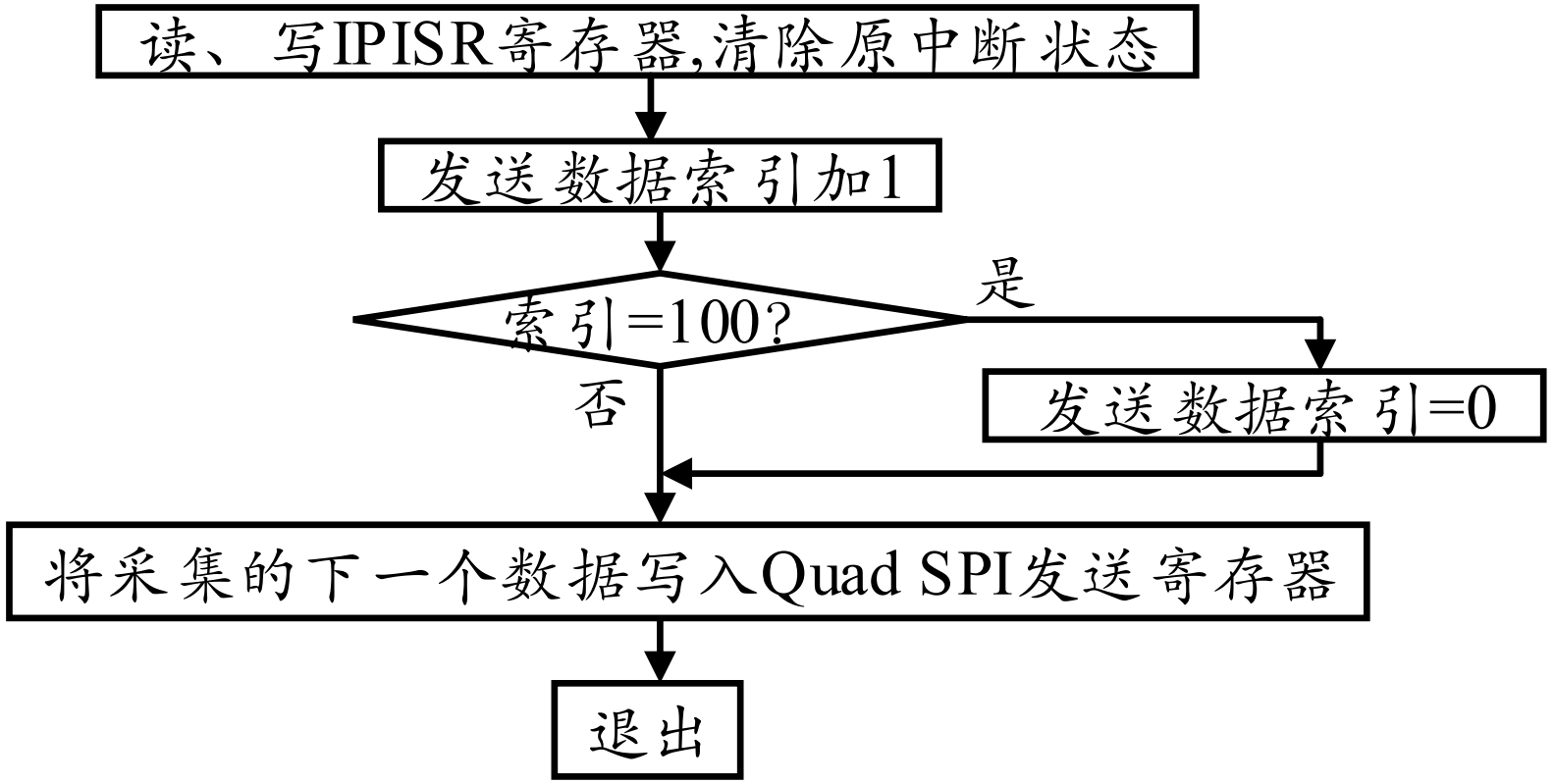
主程序流程


初始化Timer T0
写T0 TLR寄存器
写T0 TCSR寄存器，设置定时、减计数、自动装载、装载初值、使能中断、清除中断状态
初始化Quad SPI，选择从设备ADCS7476通信
写SPISRR寄存器,软件复位SPI
写SPICR寄存器,设置自动从设备选择、CPOL以及CPHA组合为01或10、高位优先、主设备、使能SPI、使能主设备事务、复位发送FIFO、复位接收FIFO
写SPISSR寄存器,选通从设备ADCS7476
写IER使能INTR4、9中断
写MER使能中断输出
写IMR使INTR4、9采用快速中断模式
填写INTR4、9对应中断向量到IVAR
开放微处理器中断
初始化中断系统
Timer T0启动定时
读、修改、写T0 TCSR寄存器使装载为0，启动为1
将任意数据写入Quad SPI发送FIFO，启动AD转换
T0中断100次？
否
是
Timer T0停止定时
Quad SPI选择从设备DAC121S101通信
使能Quad SPI发送FIFO空中断
读、写IPISR寄存器,清除原中断状态
写IPIER寄存器,使能发送FIFO空中断
写DGIER寄存器,使能中断请求
将采集的首个数据写入Quad SPI发送FIFO
退出

T0中断服务程序流程


Quad SPI接收FIFO读取一个数据并保存，中断次数加1
将任意数据写入Quad SPI发送FIFO，启动AD转换
读、写T0 TCSR寄存器，清除T0中断状态
退出

SPI发送中断服务程序流程



SPICR寄存器含义

位	含义	1	0
0	回环	SPI发送端(MOSI)与接收端(MISO)内部连通形成环路	SPI发送端(MOSI)与接收端(MISO)独立工作
1	启用接口	启用SPI接口(与SPI总线连接)	禁用SPI接口(与SPI总线断开、高阻态)
2	主设备	SPI主设备接口	SPI从设备接口
3	CPOL	空闲时时钟为高电平	空闲时时钟为低电平
4	CPHA	数据信号在时钟第二个边沿(相位180°)稳定有效	数据信号在时钟第一个边沿(相位0°)稳定有效
5	Tx 复位	复位发送FIFO指针	无意义
6	Rx 复位	复位接收FIFO指针	无意义
7	手动控制从设备选择	程序控制从设备选择SS输出，即写SPISSR立即反映到$\overline{66}$[n-1:0]	协议逻辑控制SPISSR输出到$\overline{66}$[n-1:0]
8	禁止主设备事务	禁止主设备事务；若为从设备则无意义	使能主设备事务
9	低位优先	低位优先传送	高位优先传送
10~31	保留，无意义		

中断初始化程序-快速中断

```
int main()
{    int status;
     Xil_Out32(XPAR_INTC_0_BASEADDR+XIN_ISR_OFFSET,
Xil_In32(XPAR_INTC_0_BASEADDR+XIN_ISR_OFFSET));
     Xil_Out32(XPAR_INTC_0_BASEADDR+XIN_IER_OFFSET,
XPAR_AXI_TIMER_0_INTERRUPT_MASK|XPAR_AXI_QUAD_SPI_1_IP2INTC_IRPT_MASK);
     Xil_Out32(XPAR_INTC_0_BASEADDR+XIN_IMR_OFFSET,
XPAR_AXI_TIMER_0_INTERRUPT_MASK|XPAR_AXI_QUAD_SPI_1_IP2INTC_IRPT_MASK);
     Xil_Out32(XPAR_INTC_0_BASEADDR+XIN_MER_OFFSET,XIN_INT_MASTER_ENABLE_MASK|XIN_INT_HARDW
ARE_ENABLE_MASK);
     Xil_Out32(XPAR_INTC_0_BASEADDR+XIN_IVAR_OFFSET+4*XPAR_INTC_0_TMRCTR_0_VEC_ID,(int)T0Handler);
     Xil_Out32(XPAR_INTC_0_BASEADDR+XIN_IVAR_OFFSET+4*XPAR_INTC_0_SPI_1_VEC_ID,(int)SPIHandler);
     microblaze_enable_interrupts();

     Xil_Out32(XPAR_TMRCTR_0_BASEADDR+XTC_TLR_OFFSET,RESET_VALUE);
     Xil_Out32(XPAR_TMRCTR_0_BASEADDR+XTC_TCSR_OFFSET,XTC_CSR_INT_OCCURED_MASK|XTC_CSR_AUT
O_RELOAD_MASK|XTC_CSR_DOWN_COUNT_MASK|XTC_CSR_LOAD_MASK  |XTC_CSR_ENABLE_INT_MASK);
     status=Xil_In32(XPAR_TMRCTR_0_BASEADDR+XTC_TCSR_OFFSET);
     status=(status&(~XTC_CSR_LOAD_MASK))|XTC_CSR_ENABLE_TMR_MASK;
     Xil_Out32(XPAR_TMRCTR_0_BASEADDR+XTC_TCSR_OFFSET,status);
```

中断初始化程序

```
Xil_Out32(XPAR_SPI_1_BASEADDR+XSP_SRR_OFFSET,XSP_SRR_RESET_MASK);
Xil_Out32(XPAR_SPI_1_BASEADDR+XSP_CR_OFFSET,XSP_CR_ENABLE_MASK|
        XSP_CR_MASTER_MODE_MASK|XSP_CR_CLK_POLARITY_MASK
        |XSP_CR_TXFIFO_RESET_MASK|XSP_CR_RXFIFO_RESET_MASK);
Xil_Out32(XPAR_SPI_1_BASEADDR+XSP_SSR_OFFSET,0x1);
Xil_Out32(XPAR_SPI_1_BASEADDR+XSP_DTR_OFFSET,0x0);
while(int_times<100);
status=Xil_In32(XPAR_TMRCTR_0_BASEADDR+XTC_TCSR_OFFSET);
status=status&(~XTC_CSR_ENABLE_TMR_MASK);
Xil_Out32(XPAR_TMRCTR_0_BASEADDR+XTC_TCSR_OFFSET,status);
Xil_Out32(XPAR_SPI_1_BASEADDR+XSP_SSR_OFFSET,0x2);
Xil_Out32(XPAR_SPI_1_BASEADDR+XSP_IISR_OFFSET,
        Xil_In32(XPAR_SPI_1_BASEADDR+XSP_IISR_OFFSET));
Xil_Out32(XPAR_SPI_1_BASEADDR+XSP_IIER_OFFSET,
        XSP_INTR_TX_EMPTY_MASK);
Xil_Out32(XPAR_SPI_1_BASEADDR+XSP_DGIER_OFFSET,
        XSP_GINTR_ENABLE_MASK);
int_times=0;
Xil_Out32(XPAR_SPI_1_BASEADDR+XSP_DTR_OFFSET,samples[int_times]);
return 0;}
```

中断初始化程序

```
#include "xparameters.h"
#include "xtmrctr_l.h"
#include "xspi_l.h"
#include "xintc_l.h"
#include "xil_io.h"
#include "xil_exception.h"
#define RESET_VALUE 2000-2
void T0Handler() __attribute__((fast_interrupt));
void SPIHandler() __attribute__((fast_interrupt));
short samples[100];
int int_times;
```

T0中断服务程序-读取AD转换结果并启动AD转换

```
void T0Handler()
{
samples[int_times]=(short)(Xil_In32(XPAR_SPI_1_BASEADDR+XSP_DRR_OFFSET)
            &0xfff);
int_times++;
Xil_Out32(XPAR_SPI_1_BASEADDR+XSP_DTR_OFFSET,0x0);
Xil_Out32(XPAR_TMRCTR_0_BASEADDR+XTC_TCSR_OFFSET,
            Xil_In32(XPAR_TMRCTR_0_BASEADDR+XTC_TCSR_OFFSET));
}
```

SPI发送结束中断服务程序

```
void SPIHandler()
{
    Xil_Out32(XPAR_SPI_1_BASEADDR+XSP_IISR_OFFSET,
            Xil_In32(XPAR_SPI_1_BASEADDR+XSP_IISR_OFFSET));
    int_times++;
    if(int_times==100)
        int_times=0;
    Xil_Out32(XPAR_SPI_1_BASEADDR+XSP_DTR_OFFSET,samples[int_times]);
}
```

中断初始化程序-普通中断

```
#include "xparameters.h"
#include "xtmrctr_l.h"
#include "xspi_l.h"
#include "xintc_l.h"
#include "xil_io.h"
#include "xil_exception.h"
#define RESET_VALUE 2000-2
void T0Handler()  __attribute__((fast_interrupt));
void SPIHandler()  __attribute__((fast_interrupt));
short samples[100];
int int_times;
```

```
Void My_ISR __attribute__(interrupt_handler)
```

中断初始化程序-普通中断

```
int main()
{    int status;
     Xil_Out32(XPAR_INTC_0_BASEADDR+XIN_ISR_OFFSET,
Xil_In32(XPAR_INTC_0_BASEADDR+XIN_ISR_OFFSET));
     Xil_Out32(XPAR_INTC_0_BASEADDR+XIN_IER_OFFSET,
XPAR_AXI_TIMER_0_INTERRUPT_MASK|XPAR_AXI_QUAD_SPI_1_IP2INTC_IRPT_MASK);
     Xil_Out32(XPAR_INTC_0_BASEADDR+XIN_IMR_OFFSET,
XPAR_AXI_TIMER_0_INTERRUPT_MASK|XPAR_AXI_QUAD_SPI_1_IP2INTC_IRPT_MASK);
     Xil_Out32(XPAR_INTC_0_BASEADDR+XIN_MER_OFFSET,XIN_INT_MASTER_ENABLE_MASK|XIN_INT_HARDW
ARE_ENABLE_MASK);
     Xil_Out32(XPAR_INTC_0_BASEADDR+XIN_IVAR_OFFSET+4*XPAR_INTC_0_TMRCTR_0_VEC_ID,(int)T0Handler);
     Xil_Out32(XPAR_INTC_0_BASEADDR+XIN_IVAR_OFFSET+4*XPAR_INTC_0_SPI_1_VEC_ID,(int)SPIHandler);
     microblaze_enable_interrupts();

     Xil_Out32(XPAR_TMRCTR_0_BASEADDR+XTC_TLR_OFFSET,RESET_VALUE);
     Xil_Out32(XPAR_TMRCTR_0_BASEADDR+XTC_TCSR_OFFSET,XTC_CSR_INT_OCCURED_MASK|XTC_CSR_AUT
O_RELOAD_MASK|XTC_CSR_DOWN_COUNT_MASK|XTC_CSR_LOAD_MASK  |XTC_CSR_ENABLE_INT_MASK);
     status=Xil_In32(XPAR_TMRCTR_0_BASEADDR+XTC_TCSR_OFFSET);
     status=(status&(~XTC_CSR_LOAD_MASK))|XTC_CSR_ENABLE_TMR_MASK;
     Xil_Out32(XPAR_TMRCTR_0_BASEADDR+XTC_TCSR_OFFSET,status);
```

总中断服务程序

```
void My_ISR()
{
      int status;
      status=Xil_In32(XPAR_AXI_INTC_0_BASEADDR+XIN_ISR_OFFSET);//读取ISR
      if((status&XPAR_AXI_TIMER_0_INTERRUPT_MASK)==XPAR_AXI_TIMER_0_INTERRUPT_MASK)
            T0Handler();//调用T0中断服务程序
      if((status&XPAR_AXI_QUAD_SPI_1_IP2INTC_IRPT_MASK)==
                        XPAR_AXI_QUAD_SPI_1_IP2INTC_IRPT_MASK)
           SPIHandler ();//调用SPI中断
      Xil_Out32(XPAR_AXI_INTC_0_BASEADDR+XIN_IAR_OFFSET,status);//写IAR
}
```

小结

- SPI中断方式编程控制
  - 写控制寄存器控制工作方式
  - 写IPIER\IPGIER开放中断
    - 发送空、接收满中断
  - 促使产生中断
    - 先发送数据，收发全双工
- 再次回顾快速中断、普通中断编程差别
- 再次回顾定时器中断编程控制

下一讲：多中断源程序设计