计算机组成原理与接口技术（实验）——基于MIPS架构

May, 2021

实验6 串行接口实验

杨明
华中科技大学电子信息与通信学院
myang@hust.edu.cn

Agenda

- 实验内容
  - 目的
  - 任务及时间安排
  - 报告要求

- 原理回顾
  - UART IP Core
  - UART 实验原理

  - SPI IP Core
  - DAC实验原理
  - ADC实验原理

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

实验任务及时间安排

- 任务
  - 所有实验任务要求采用中断方式实现，中断方式时，GPIO输入、延时、UART\SPI接口通信都采用中断方式实现。每位同学仅完成其中的一个任务，且需完成的实验任务编号与座位号模3的取值一致。
  - [实验任务0]
    - 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。
- 时间安排：2次课

实验任务及时间安排

- 任务
  - 所有实验任务要求采用中断方式实现，中断方式时，GPIO输入、延时、UART\SPI接口通信都采用中断方式实现。每位同学仅完成其中的一个任务，且需完成的实验任务编号与座位号模3的取值一致。
  - [实验任务1]
  - 利用SPI IP 核，timer 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 控制。锯齿波周期最长约为1s，最短约为60ms
  - 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。
- 时间安排：2次课

实验任务及时间安排

- 任务
  - 所有实验任务要求采用中断方式实现，中断方式时，GPIO输入、延时、UART\SPI接口通信都采用中断方式实现。每位同学仅完成其中的一个任务，且需完成的实验任务编号与座位号模3的取值一致。
  - [实验任务2]
  - 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。
- 时间安排：2次课

实验报告要求

- 实验任务
- 硬件电路框图、各模块参数设计、步骤
- 软件流程图、源码及注释
- 实验结果及调试过程
- 实验心得
- 14-15周综合项目验收

Agenda

- 实验内容
  - 目的
  - 任务及时间安排
  - 报告要求

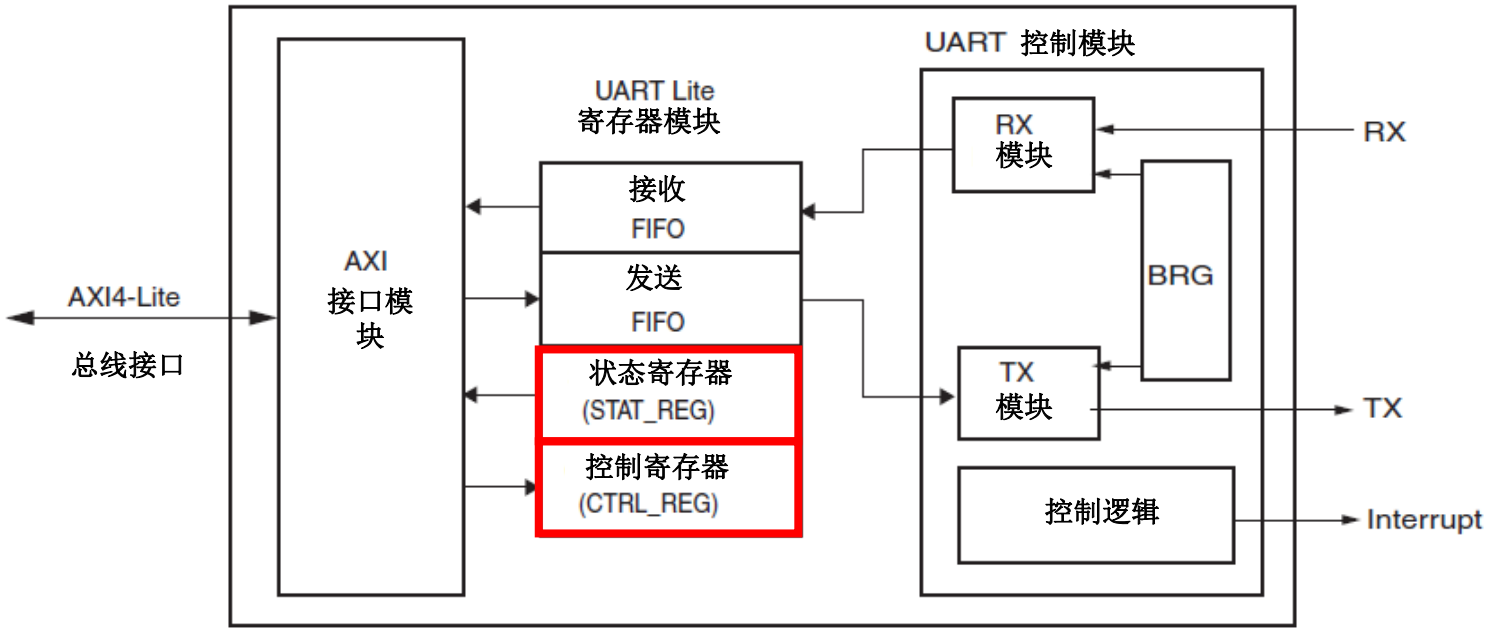
- 原理回顾
  - UART IP Core
  - UART 实验原理

  - SPI IP Core
  - DAC实验原理
  - ADC实验原理

UART lite IP核

- UART
  - UART（ Universal Asynchronous Receiver/Transmitter，通用异步收发器）作为异步串口通信协议的一种，工作原理是将传输的数据的一位接一位地串行传输。

- UART lite IP核



寄存器名称	偏移地址	初始值	功能
RX FIFO	0x0	0x0	接收数据FIFO
TX FIFO	0x4	0x0	发送数据FIFO
STAT_REG	0x8	0x4	状态寄存器
CTRL_REG	0xc	0x0	控制寄存器

UART lite IP核

▶ UART状态寄存器STAT_REG

bit位置	名称	含义
bit0	Rx FIFO Valid Data	1：接收FIFO接收到有效数据
bit1	Rx FIFO Full	1：接收FIFO满
bit2	Tx FIFO Empty	1：发送FIFO空
bit3	Tx FIFO Full	1：发送FIFO满
bit4	Intr Enabled	1：使能中断
bit5	Overrun Error	1：FIFO溢出
bit6	Frame Error	1：数据帧错误
bit7	Parity Error	1：奇偶校验错
8~31	无	保留

▶ UART控制寄存器CTRL_REG

bit位置	名称	含义
bit0	RstTx FIFO	1：复位发送缓冲区
bit1	Rst Rx FIFO	1：复位接收缓冲区
bit4	Enable Intr	1：使能硬件中断
2-3, 5~31	无	保留

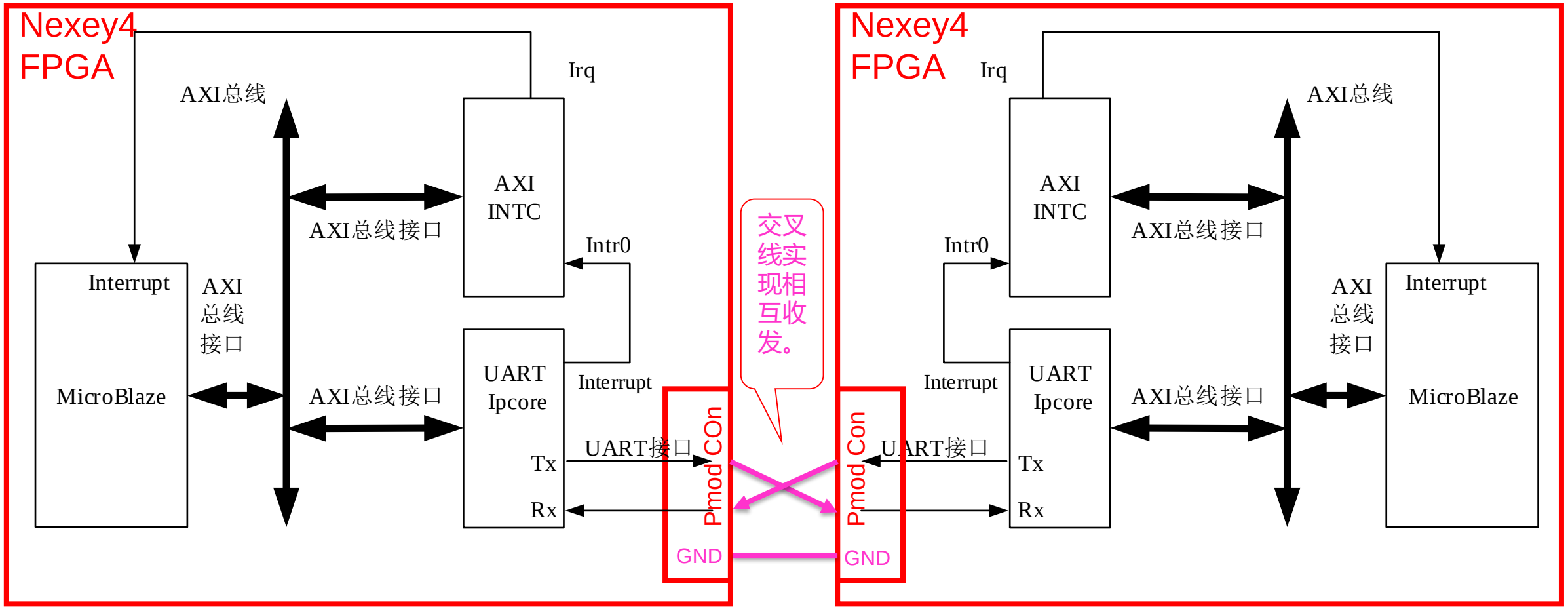
波特率、帧格式等如何设定？
—— 通过XPS在硬件配置中设定。

▶ XUartLite API

- XUartLite_Initialize(XUartLite*, u16) : int
- XUartLite_LookupConfig(u16) : XUartLite_Config*
- XUartLite_CfgInitialize(XUartLite*, XUartLite_Config*, u32) : int
- XUartLite_ResetFifos(XUartLite*) : void
- XUartLite_Send(XUartLite*, u8*, unsigned int) : unsigned int
- XUartLite_Recv(XUartLite*, u8*, unsigned int) : unsigned int
- XUartLite_IsSending(XUartLite*) : int
- XUartLite_GetStats(XUartLite*, XUartLite_Stats*) : void
- XUartLite_ClearStats(XUartLite*) : void
- XUartLite_SelfTest(XUartLite*) : int
- XUartLite_EnableInterrupt(XUartLite*) : void
- XUartLite_DisableInterrupt(XUartLite*) : void
- XUartLite_SetRecvHandler(XUartLite*, XUartLite_Handler, void*) : void
- XUartLite_SetSendHandler(XUartLite*, XUartLite_Handler, void*) : void
- XUartLite_InterruptHandler(XUartLite*) : void

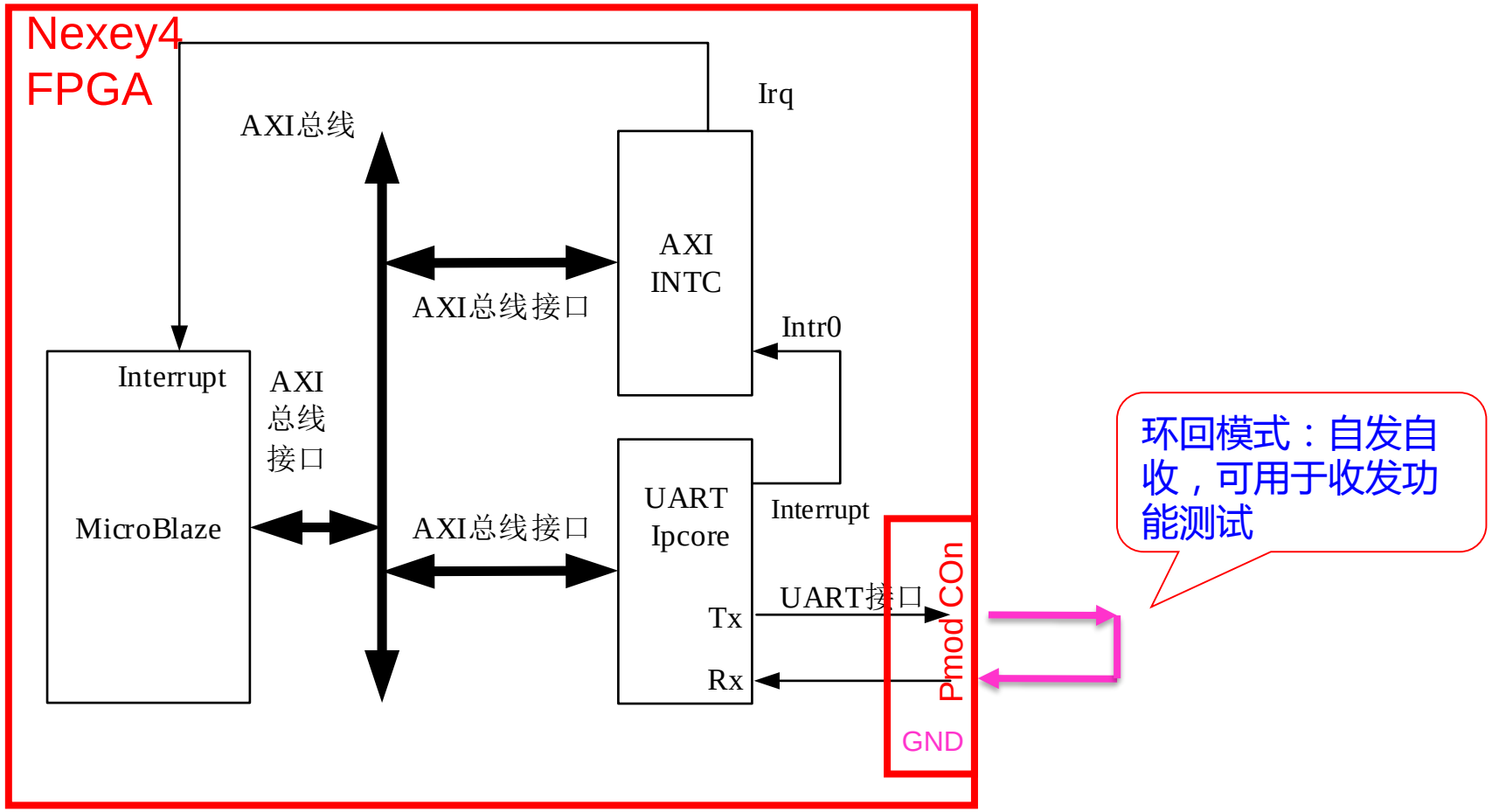
UART实验原理

- 两块Nexey4开发板通过UART，实现数据的相互收发
  - 通信双方需要设置正确的波特率，帧格式（硬件配置）



UART实验原理

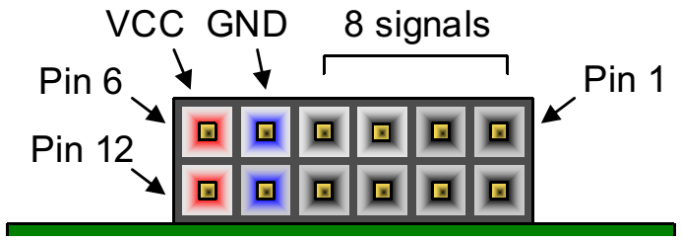
- 两块Nexey4开发板通过UART，实现数据的相互收发
  - 通信双方需要设置正确的波特率，帧格式（硬件配置）



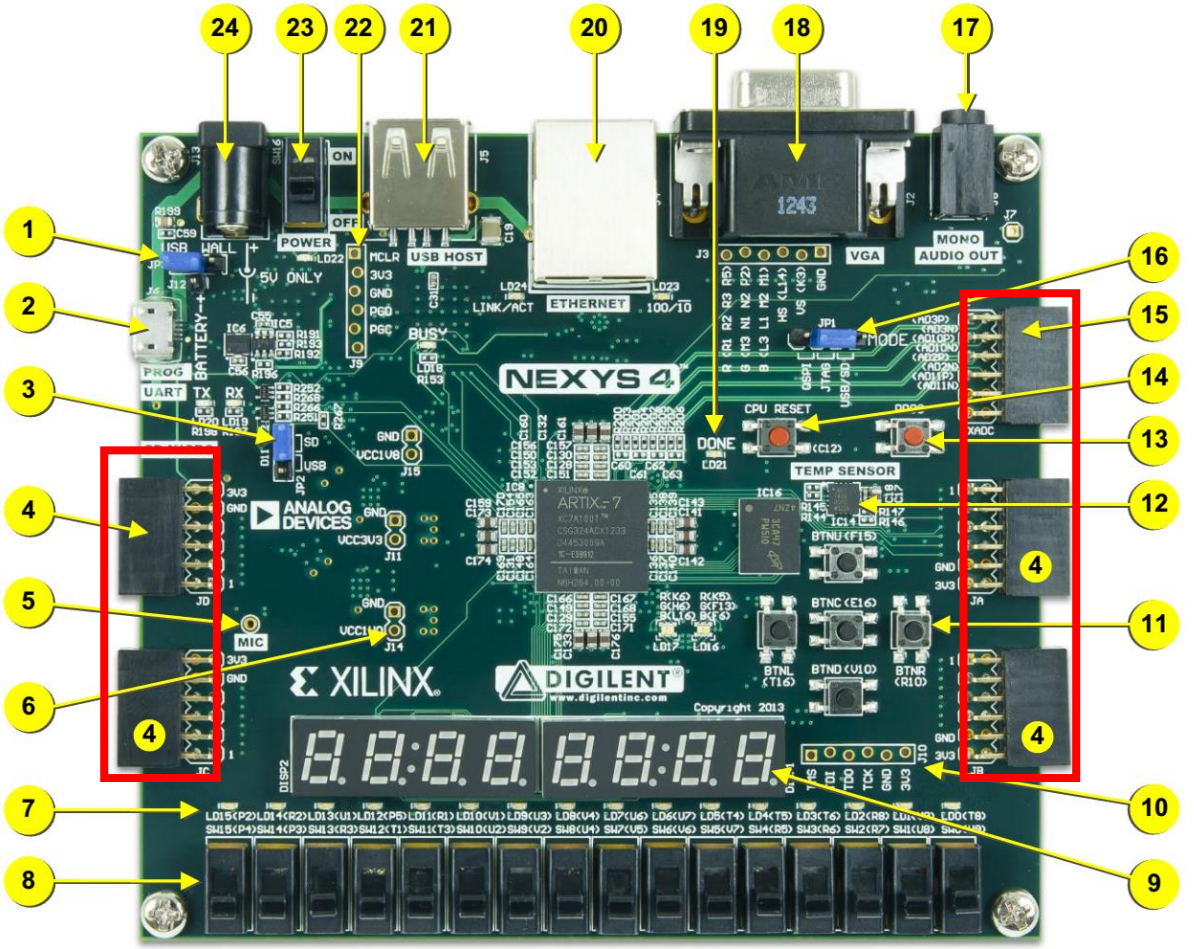
UART实验原理

▶ Pmod Connectors

[1] 提供外部扩展能力
[2] 本实验中可用

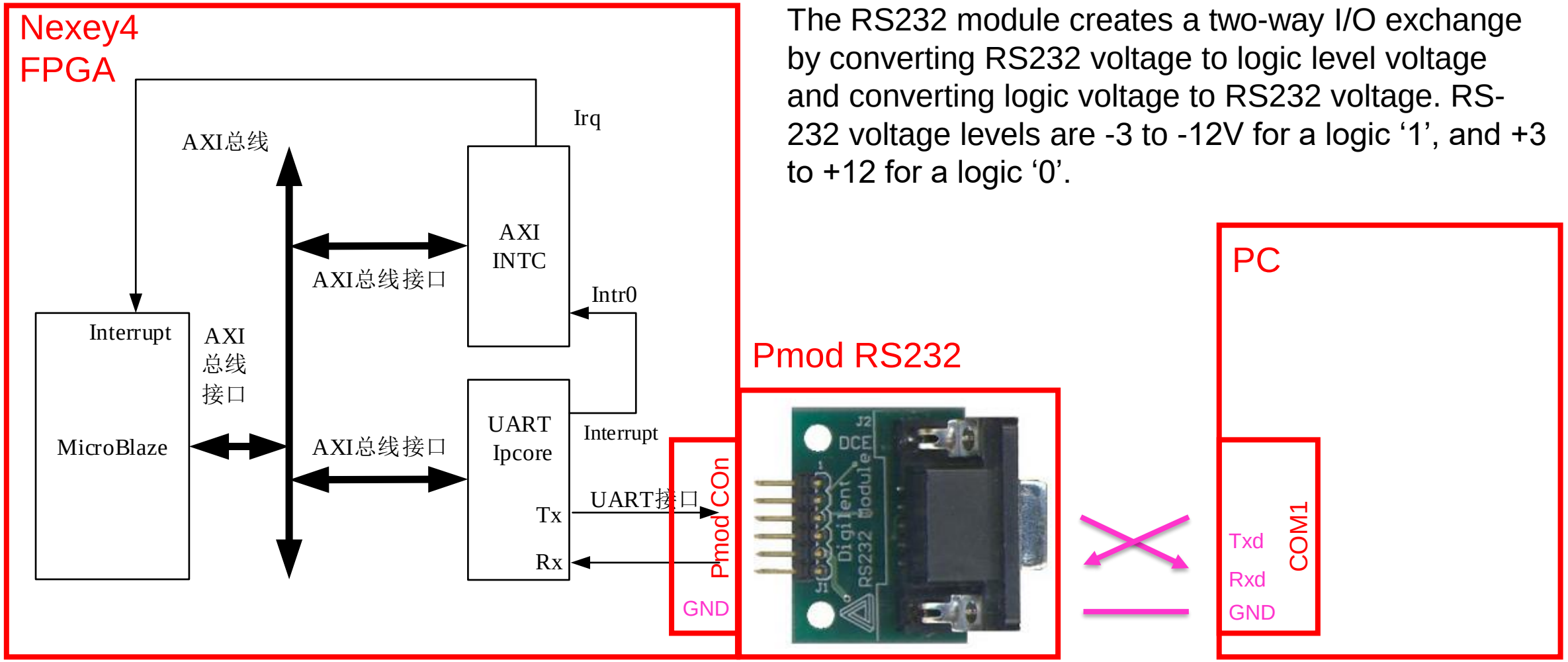


Pmod JA	Pmod JB	Pmod JC	Pmod JD	Pmod XDAC
JA1: B13	JP1: G14	JC1: K2	JD1: H4	JXADC1: A13
JA2: F14	JB2: P15	JC2: E7	JD2: H1	JXADC2: A15
JA3: D17	JB3: V11	JC3: J3	JD3: G1	JXADC3: B16
JA4: E17	JB4: V15	JC4: J4	JD4: G3	JXADC4: B18
JA7: G13	JB7: K16	JC7: K1	JD7: H2	JXADC7: A14
JA8: C17	JB8: R16	JC8: E6	JD8: G4	JXADC8: A16
JA9: D18	JB9: T9	JC9: J2	JD9: G2	JXADC9: B17
JA10: E18	JB10: U11	JC10: G6	JD10: F3	JXADC10: A18



UART实验原理

- 若Nexey4要通过UART和PC实现RS232互联、通信，则需要RS232电平转换



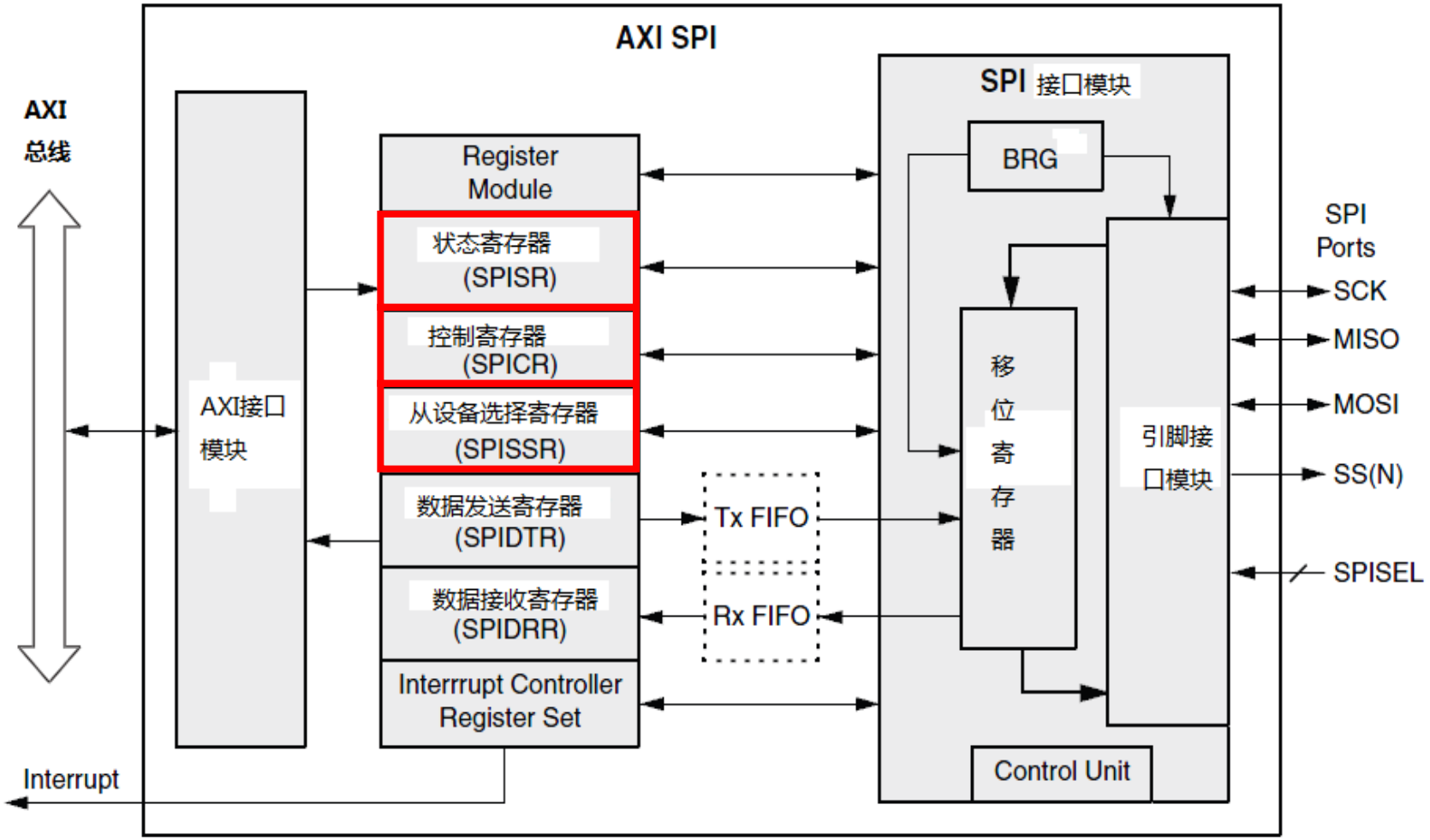
The RS232 module creates a two-way I/O exchange by converting RS232 voltage to logic level voltage and converting logic voltage to RS232 voltage. RS-232 voltage levels are -3 to -12V for a logic '1', and +3 to +12 for a logic '0'.

Agenda

- 实验内容
  - 目的
  - 任务及时间安排
  - 报告要求

- 原理回顾
  - UART IP Core
  - UART 实验原理
  - SPI IP Core
  - DAC实验原理
  - ADC实验原理

▶ SPI IP核

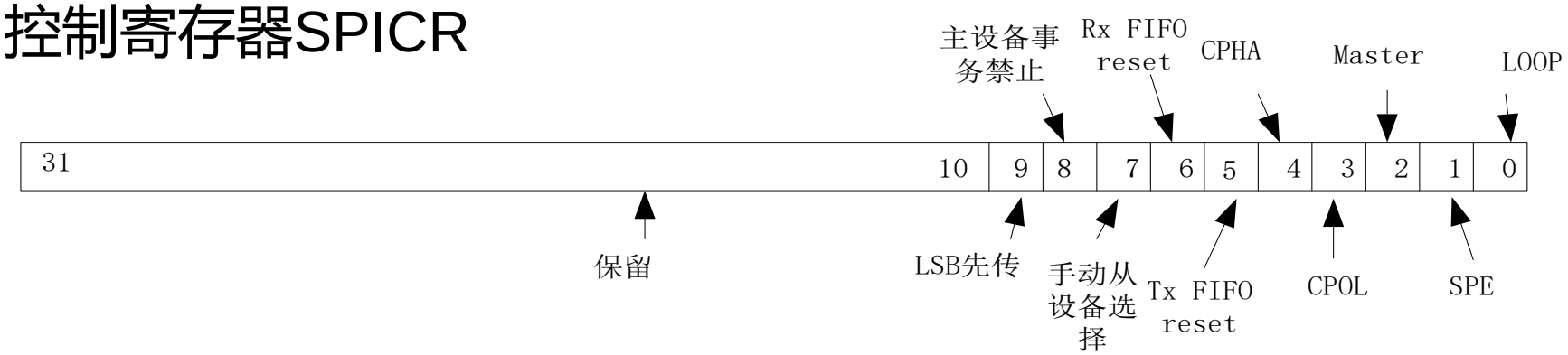

AXI SPI
AXI
总线
AXI接口
模块
Register
Module
状态寄存器
(SPISR)
控制寄存器
(SPICR)
从设备选择寄存器
(SPISSR)
数据发送寄存器
(SPIDTR)
数据接收寄存器
(SPIDRR)
Interrrupt Controller
Register Set
Interrupt
Tx FIFO
Rx FIFO
SPI 接口模块
BRG
移
位
寄
存
器
引脚接
口模块
Control Unit
SPI
Ports
SCK
MISO
MOSI
SS(N)
SPISEL

▶ SPI 寄存器

寄存器名称	偏移地址	含义	操作类型	初始值
SRR	40	软件复位寄存器，向该寄存器写0x0000000A，复位接口	写	N/A
SPICR	60	控制寄存器	读写	0x180
SPISR	64	状态寄存器	读	0x25
SPIDTR	68	发送寄存器或FIFO（可为8，16，32位）	写	0x0
SPIDRR	6C	接收寄存器或FIFO（可为8，16，32位）	读	N/A
SPISSR	70	从设备选择寄存器	读写	未选中
Tx_FIFO_OCY	74	发送FIFO占用长度指示，低4位的值+1表示FIFO有效数据的长度	读	0x0
Rx_FIFO_OCY	78	接收FIFO占用长度指示，低4位的值+1表示FIFO有效数据的长度	读	0x0
DGIER	1C	设备总中断请求使能寄存器，仅最高位有效，bit31=1使能设备中断	读写	0x0
IPISR	20	中断状态寄存器	读/写	0x0
IPIER	28	中断使能寄存器	读写	0x0

SPI 寄存器

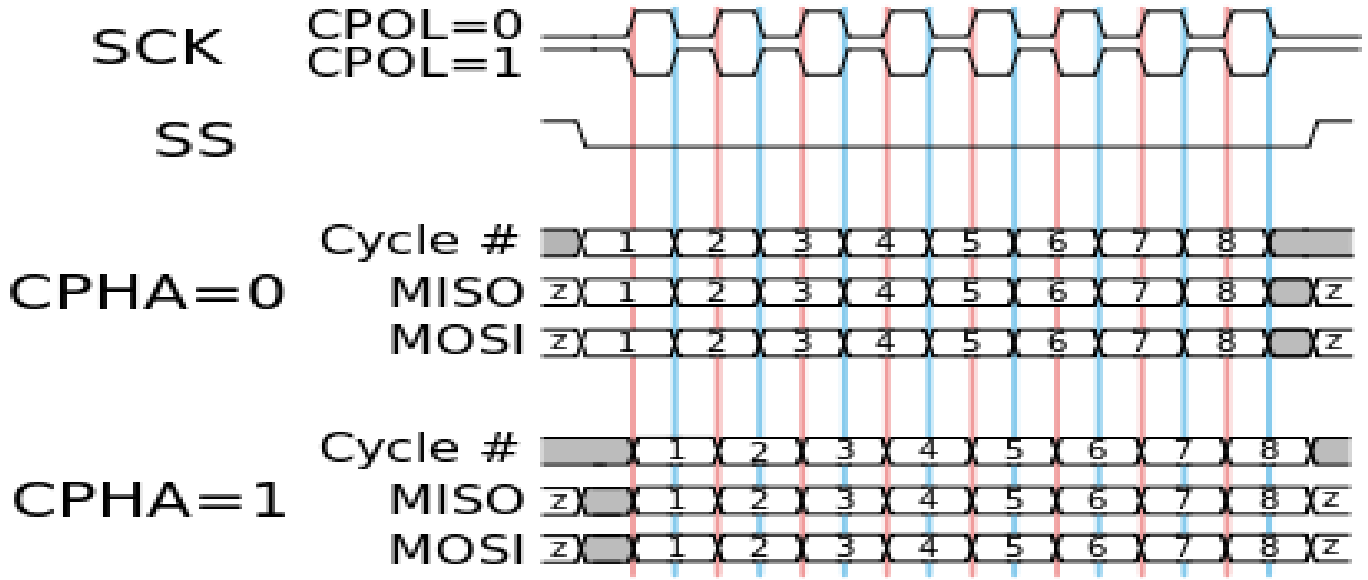
- 控制寄存器SPICR



Bit位置	写1	写0
0	SPI发送端与接收端在内部形成环路	正常工作
1	使能SPI接口	停止SPI接口
2	配置为主设备	配置为从设备
3	空闲时时钟为高电平	空闲时时钟为低电平
4	数据在第二个时钟周期开始有效	数据在第一个时钟开始有效
5	复位发送FIFO指针	无影响
6	复位接收FIFO指针	无影响
7	配置为手动控制，根据SPISSR寄存器的值输出SS	根据内部逻辑自动输出SS
8	禁止主设备事务；若为从设备则无影响	使能主设备事务
9	串行数据低位优先传送	串行数据高位优先传送

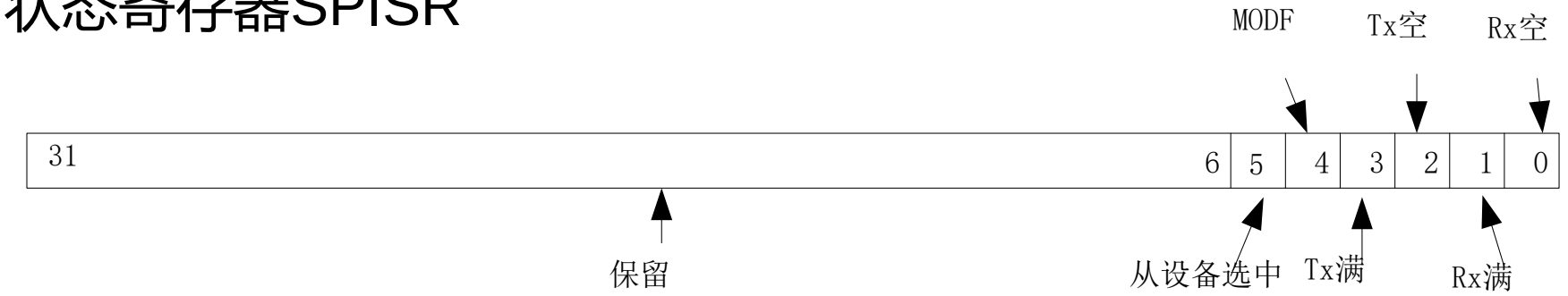
▶ SPI 寄存器

- 控制寄存器SPICR：CPOL以及CPHA组合可以设定SPI总线时序
  - 当CPOL=0，CPHA=1时，空闲时SCLK为低电平，数据输出端在SCLK的上升沿转换数据，数据输入端在SCLK下降沿采样数据（即第二个时钟周期才采样数据）
  - 当CPOL=0，CPHA=0时，空闲时SCLK为低电平，数据输出端在SCLK下降沿转换数据，数据输入端在SCLK上升沿采样数据（即第一个时钟周期采样数据）
  - 当CPOL=1，CPHA=1时，空闲时SCLK为高电平，数据输出端在SCLK的下降沿转换数据，数据输入端在SCLK上升沿采样数据（即第二个时钟周期才采样数据）
  - 当CPOL=1，CPHA=0时，空闲时SCLK为高电平，数据输出端在SCLK上升沿转换数据，数据输入端在SCLK下降沿采样数据（即第一个时钟周期采样数据）



▶ SPI 寄存器

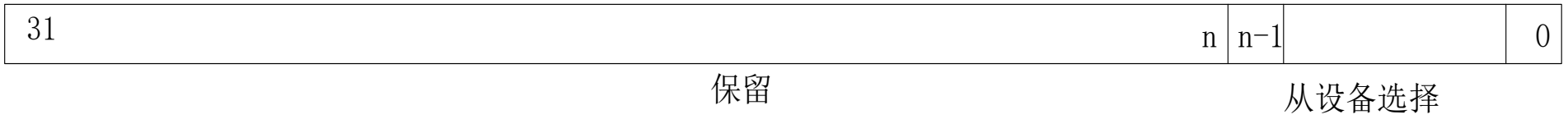
- 状态寄存器SPISR



Bit位置	读0	读1
0	接收寄存器/FIFO非空	接收寄存器/FIFO空
1	接收寄存器/FIFO未满	接收寄存器/FIFO满
2	发送寄存器/FIFO非空	发送寄存器/FIFO空
3	发送寄存器/FIFO未满	发送寄存器/FIFO满
4	没有模式错误	SPI设备配置为主设备，但是SS引脚输入低电平
5	从设备选中	默认值，未被选中

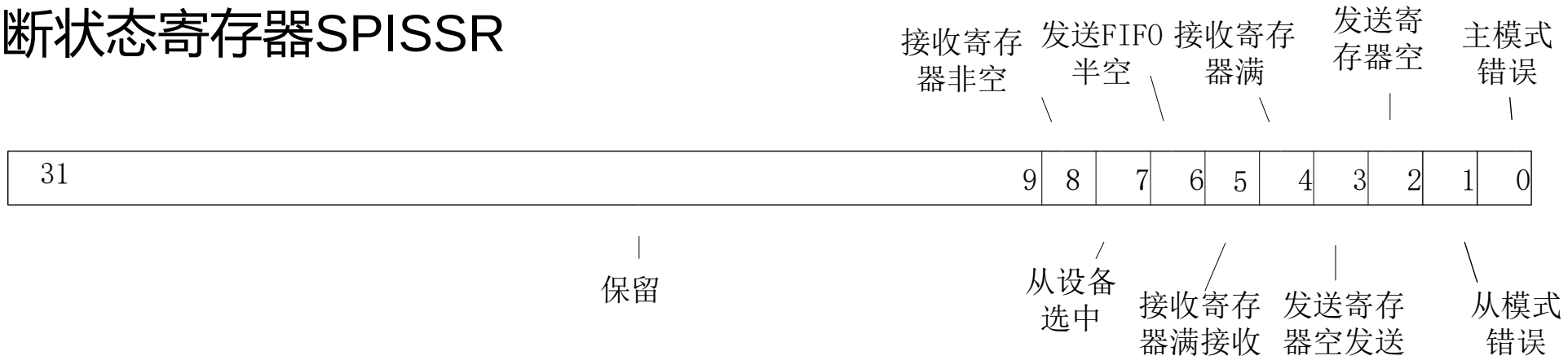
SPI 寄存器

- 从设备选择寄存器SPISSR



  - SPISSR寄存器bit0~bitn-1分别对应控制SS(0~n-1)的输出

- 中断状态寄存器SPISSR



SPI 典型控制流程

- SPI总线接口作为主设备并采用手动控制SS（N），应用时序1）发送数据的流程
  - 根据应用需要配置DGIER以及IPIER，实现中断使能控制；
  - 将要传输的数据写入SPIDTR寄存器或FIFO；
  - 将SPISSR预置为全1
  - 配置SPICR，初始化SCLK以及MOSI，但禁止数据传输
    - SPICR（bit7）=1，手动控制SS（N）
    - SPICR（bit1）=1，使能SPI
    - SPICR（bit8）=1，禁止主设备事务
    - SPICR（bit2）=1，配置为主设备
    - SPICR（bit3）=0，空闲时时钟为低电平
    - SPICR（bit4）=1，上升沿输出数据，下降沿采样数据
    - SPICR（bit0）=0，非内部循环
  - 写SPISSR，控制SS（N）输出信号的使能信号
  - 修改SPICR（bit8）=0，使能主设备事务，从而开始SPI数据传输
  - 等待中断或查询SPI状态

▶ SPI 典型控制流程（续）

- SPI总线接口作为主设备并采用手动控制SS（N），应用时序1）发送数据的流程
  - 根据应用需要配置DGIER以及IPIER，实现中断使能控制；
  - 将要传输的数据写入SPIDTR寄存器或FIFO；
  - 将SPISSR预置为全1
  - 配置SPICR，初始化SCLK以及MOSI，但禁止数据传输
  - 写SPISSR，控制SS（N）输出信号的使能信号
  - 修改SPICR（bit8）=0，使能主设备事务，从而开始SPI数据传输
  - 等待中断或查询SPI状态
  - 进入中断服务，修改SPICR（bit8）=1，禁止主设备事务，将新的数据写入数据寄存器或FIFO之后，再修改SPICR（bit8）=0，使能主设备事务，从而开始SPI数据传输
  - 重复6），7）直到所有数据传送完毕
  - 写SPISSR，控制SS（N）输出全1
  - 禁止SPI设备。

▶ SPI API函数

XSpi_IntrGlobalEnable	XSpi_Initialize(XSpi*, u16) : int
XSpi_IntrGlobalDisable	XSpi_LookupConfig(u16) : XSpi_Config*
XSpi_IsIntrGlobalEnabled	XSpi_CfgInitialize(XSpi*, XSpi_Config*, u32) : int
XSpi_IntrGetStatus	XSpi_Start(XSpi*) : int
XSpi_IntrClear	XSpi_Stop(XSpi*) : int
XSpi_IntrEnable	XSpi_Reset(XSpi*) : void
XSpi_IntrDisable	XSpi_SetSlaveSelect(XSpi*, u32) : int
XSpi_IntrGetEnabled	XSpi_GetSlaveSelect(XSpi*) : u32
XSpi_SetControlReg	XSpi_Transfer(XSpi*, u8*, u8*, unsigned int) : int
XSpi_GetControlReg	XSpi_SetStatusHandler(XSpi*, void*, XSpi_StatusHandler) : void
XSpi_GetStatusReg	XSpi_InterruptHandler(void*) : void
XSpi_SetSlaveSelectReg	XSpi_SelfTest(XSpi*) : int
XSpi_GetSlaveSelectReg	XSpi_GetStats(XSpi*, XSpi_Stats*) : void
XSpi_Enable	XSpi_ClearStats(XSpi*) : void
XSpi_Disable	XSpi_SetOptions(XSpi*, u32) : int
	XSpi_GetOptions(XSpi*) : u32

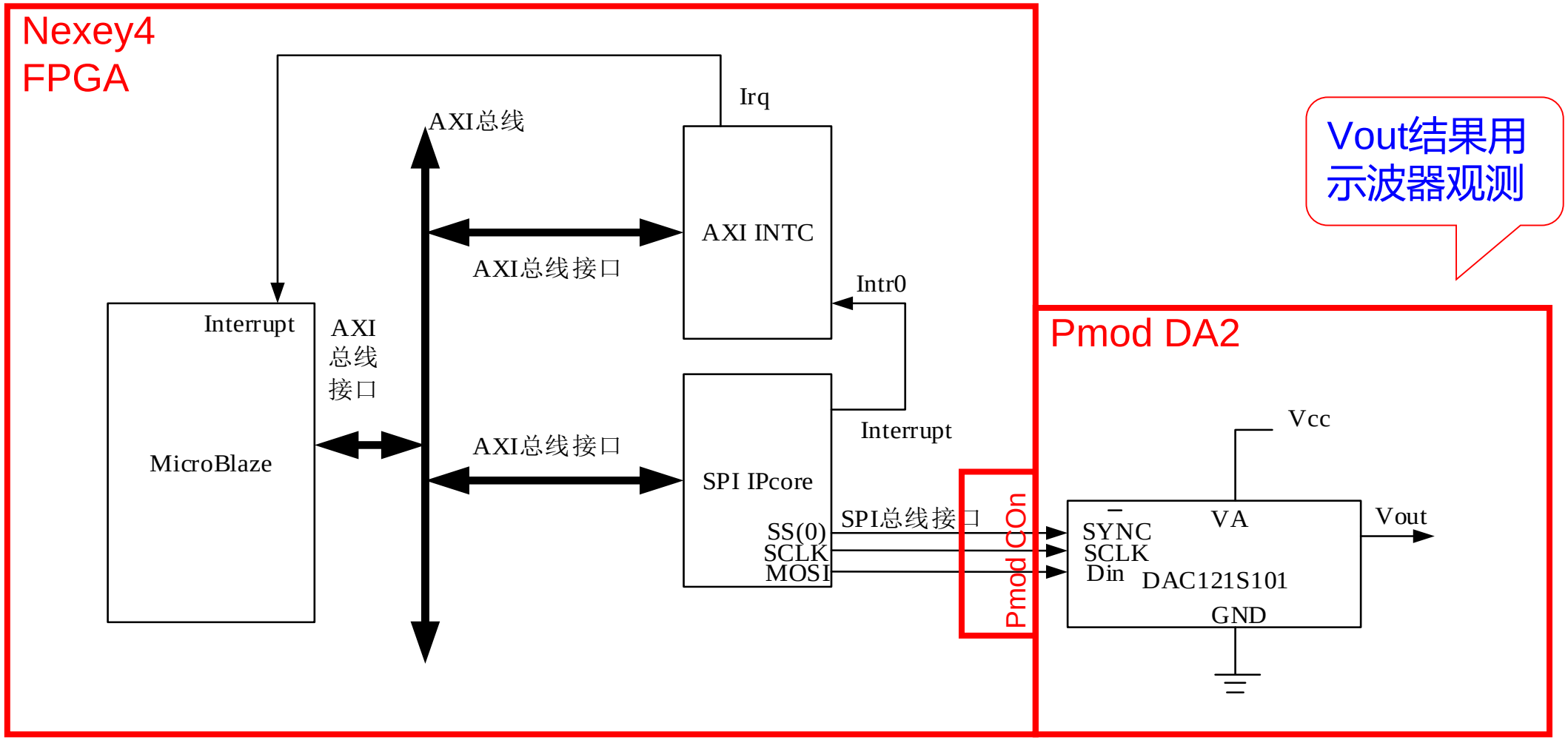
Agenda

- 实验内容
  - 目的
  - 任务及时间安排
  - 报告要求

- 原理回顾
  - UART IP Core
  - UART 实验原理
  - SPI IP Core
  - DAC实验原理
  - ADC实验原理

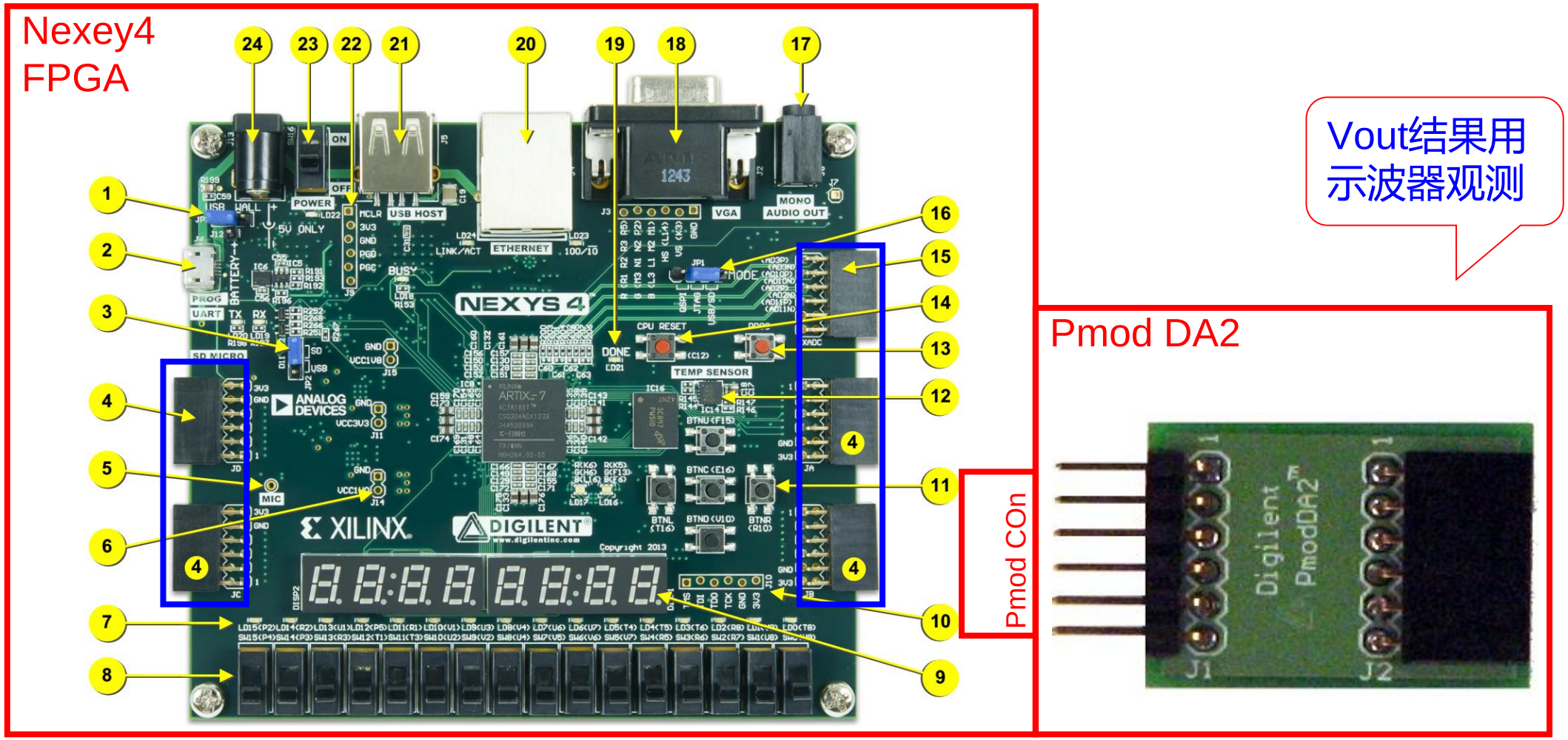
SPI DAC实验原理

- 利用DAC121S101DA转换芯片，基于SPI总线控制其Dout输出锯齿波



SPI DAC实验原理

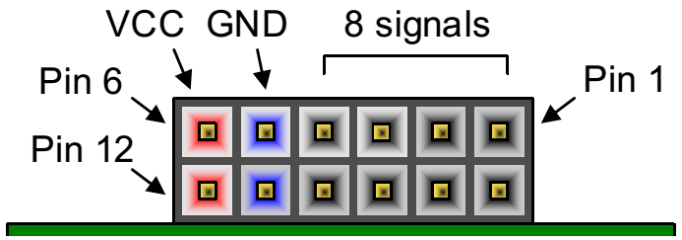
- 利用DAC121S101DA转换芯片，基于SPI总线控制其Dout输出锯齿波



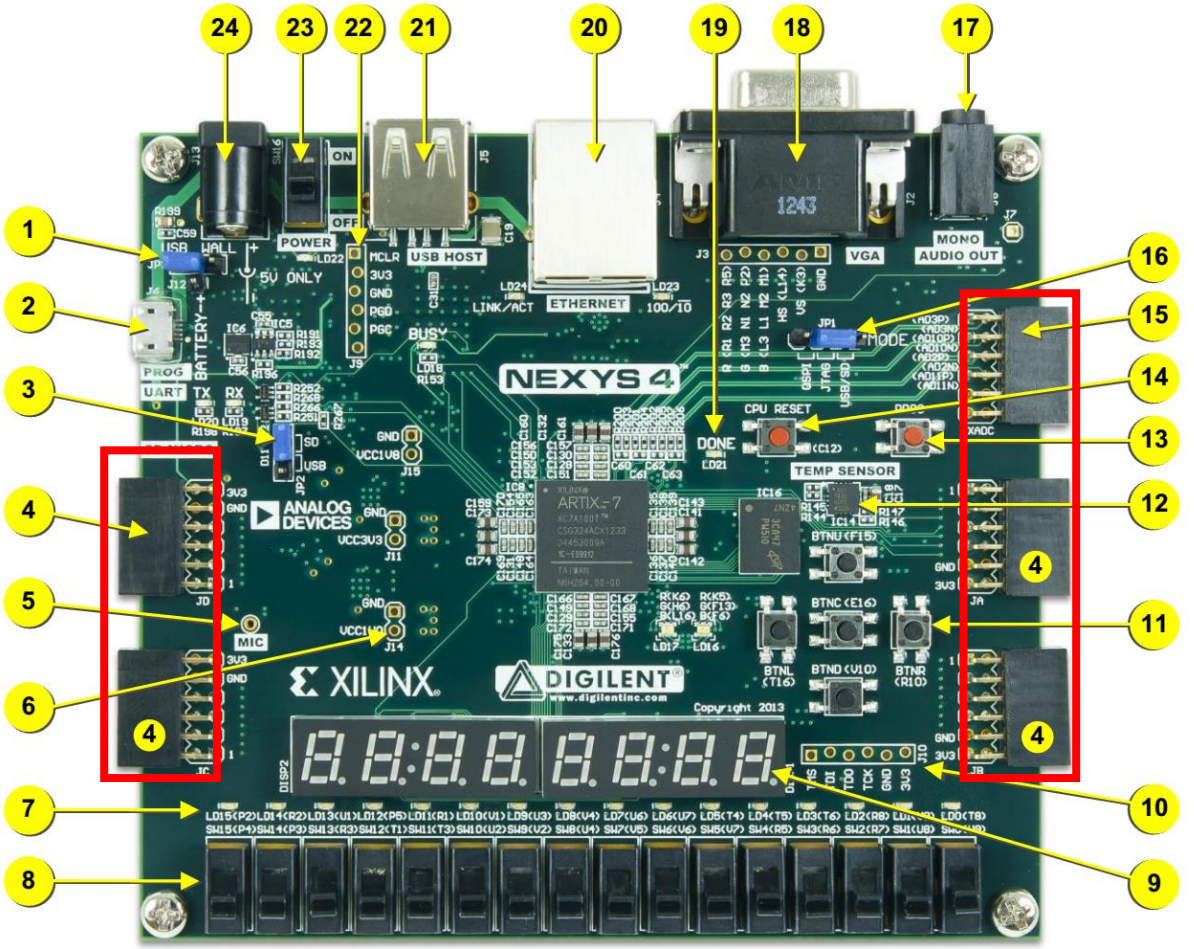
SPI DAC实验原理

▶ Pmod Connectors

[1] 提供外部扩展能力
[2] 本实验中可用

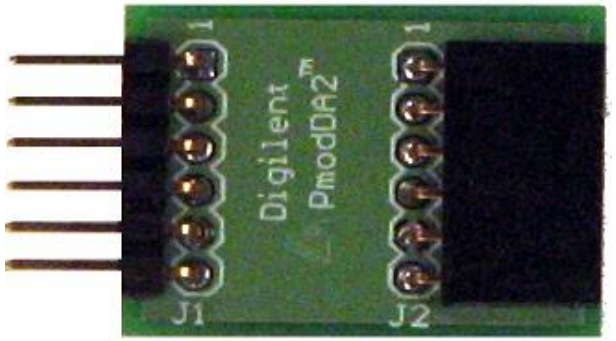


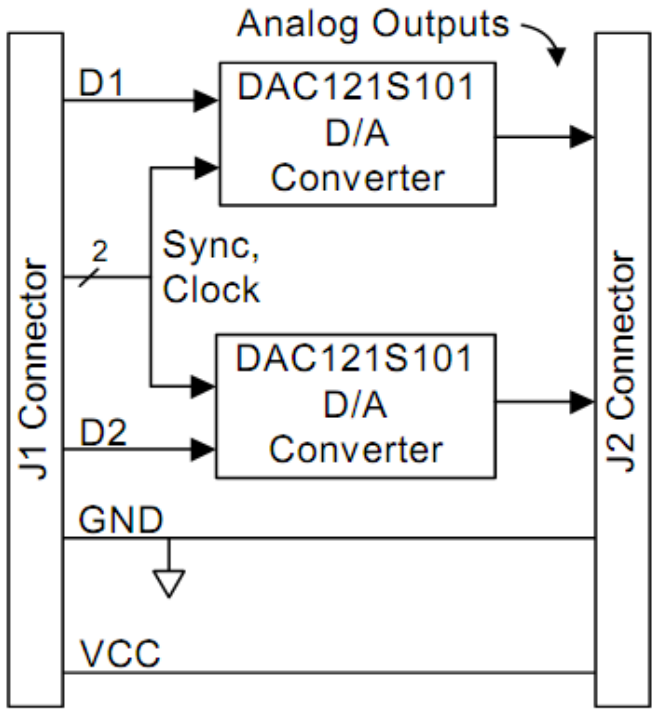
Pmod JA	Pmod JB	Pmod JC	Pmod JD	Pmod XDAC
JA1: B13	JP1: G14	JC1: K2	JD1: H4	JXADC1: A13
JA2: F14	JB2: P15	JC2: E7	JD2: H1	JXADC2: A15
JA3: D17	JB3: V11	JC3: J3	JD3: G1	JXADC3: B16
JA4: E17	JB4: V15	JC4: J4	JD4: G3	JXADC4: B18
JA7: G13	JB7: K16	JC7: K1	JD7: H2	JXADC7: A14
JA8: C17	JB8: R16	JC8: E6	JD8: G4	JXADC8: A16
JA9: D18	JB9: T9	JC9: J2	JD9: G2	JXADC9: B17
JA10: E18	JB10: U11	JC10: G6	JD10: F3	JXADC10: A18



SPI DAC实验原理

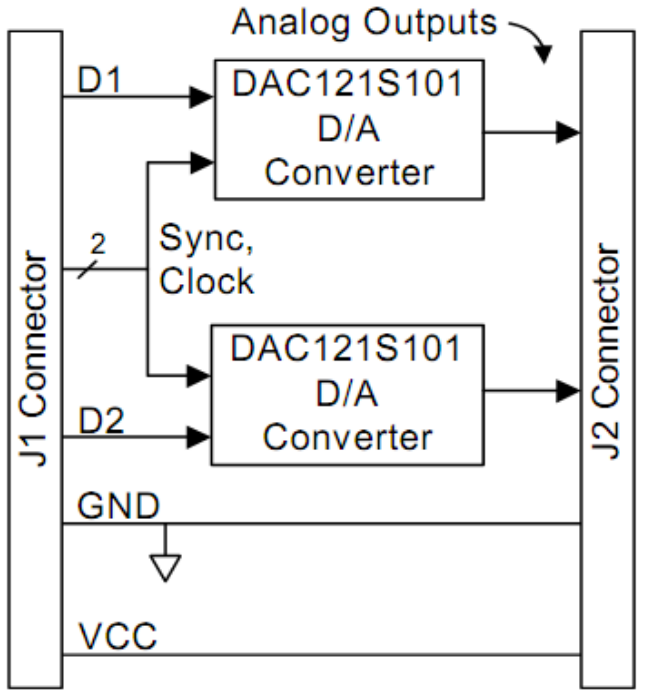
- Pmod DA2
  - two National Semiconductor DAC121S101, 12-bit D/A converters
  - a 6-pin header and 6-pin connector
  - two simultaneous D/A conversion channels
  - very low power consumption
  - small form factor (0.80” x 0.80”)





SPI DAC实验原理

- Pmod DA2
  - two National Semiconductor DAC121S101, 12-bit D/A converters
  - a 6-pin header and 6-pin connector
  - two simultaneous D/A conversion channels
  - very low power consumption
  - small form factor (0.80” x 0.80”)



Digital Interface – J1	
1	SYNC (common)
2	DINA (converter IC1)
3	DINB (converter IC2)
4	SCLK (common)
5	GND
6	VCC

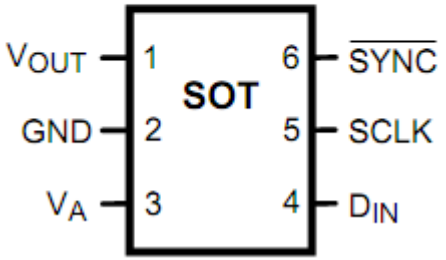
Analog Interface – J2	
1	VOUTA (converter IC1)
2	N/C
3	VOUTB (converter IC2)
4	N/C
5	GND
6	VCC

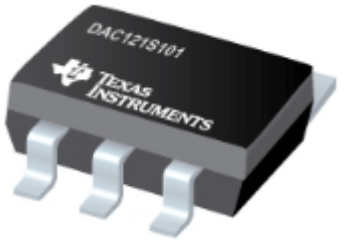
SPI DAC实验原理

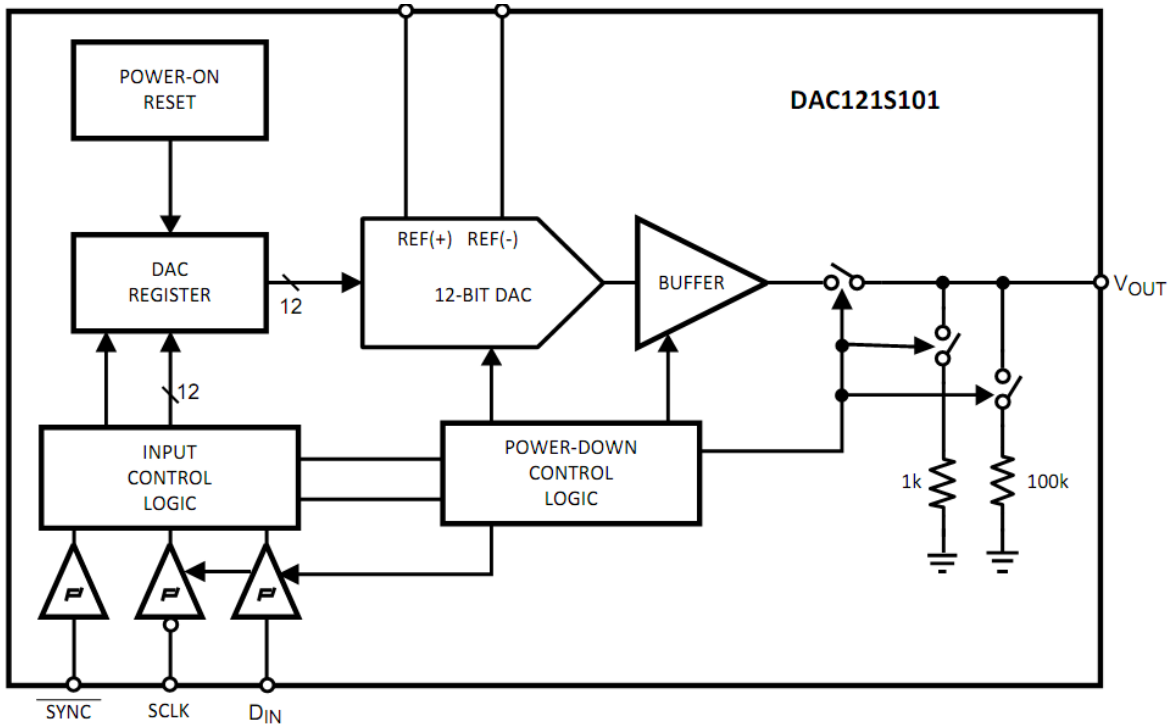
▶ DAC121S101 概述

管脚	含义
V_{OUT}	模拟电压输出
GND	地
V_A	模拟参考电压
$\overline{SYNC}$	帧数据同步，当该引脚为低电平时，数据在SCLK的下降沿输入，并且16个时钟周期之后，移位寄存器的数据进入DAC寄存器，开始DA转换；若该引脚在16个时钟周期之前变为高电平，那么之前传入的数据都将被忽略。
SCLK	SPI总线时钟，数据在该时钟的下降沿采样。时钟频率最高为30MHZ
D_{IN}	SPI总线从设备数据输入线，相当于MOSI

DDC Package
6-Pin SOT
Top View







SPI DAC实验原理

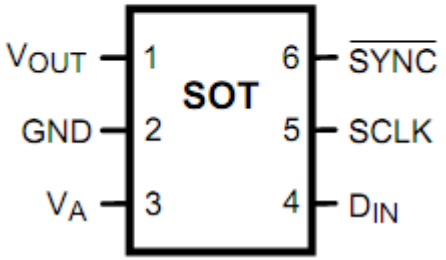
▶ DAC121S101 概述

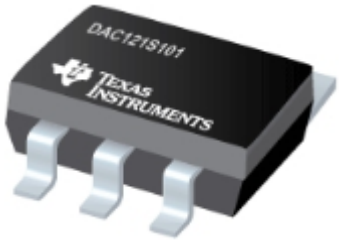
- The input coding is straight binary with an ideal output voltage of:

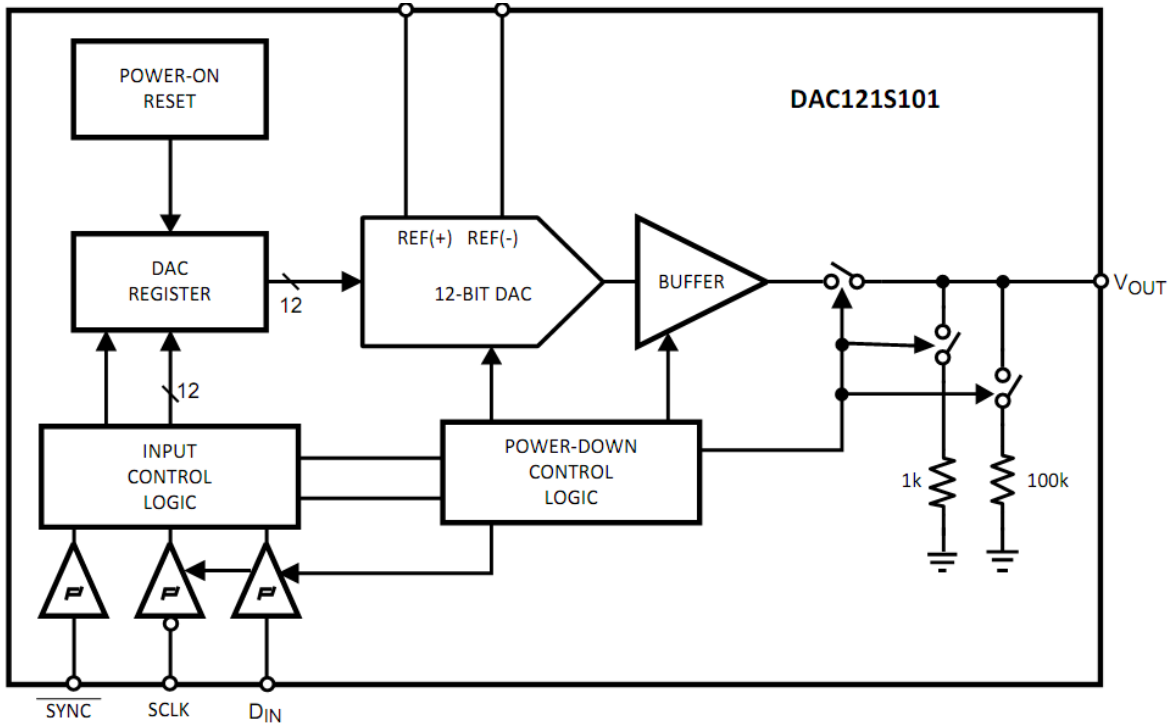
  VOUT = VA × (D / 4096)

  - where D is the decimal equivalent of the binary code that is loaded into the DAC register and can take on any value between 0 and 4095.

DDC Package
6-Pin SOT
Top View

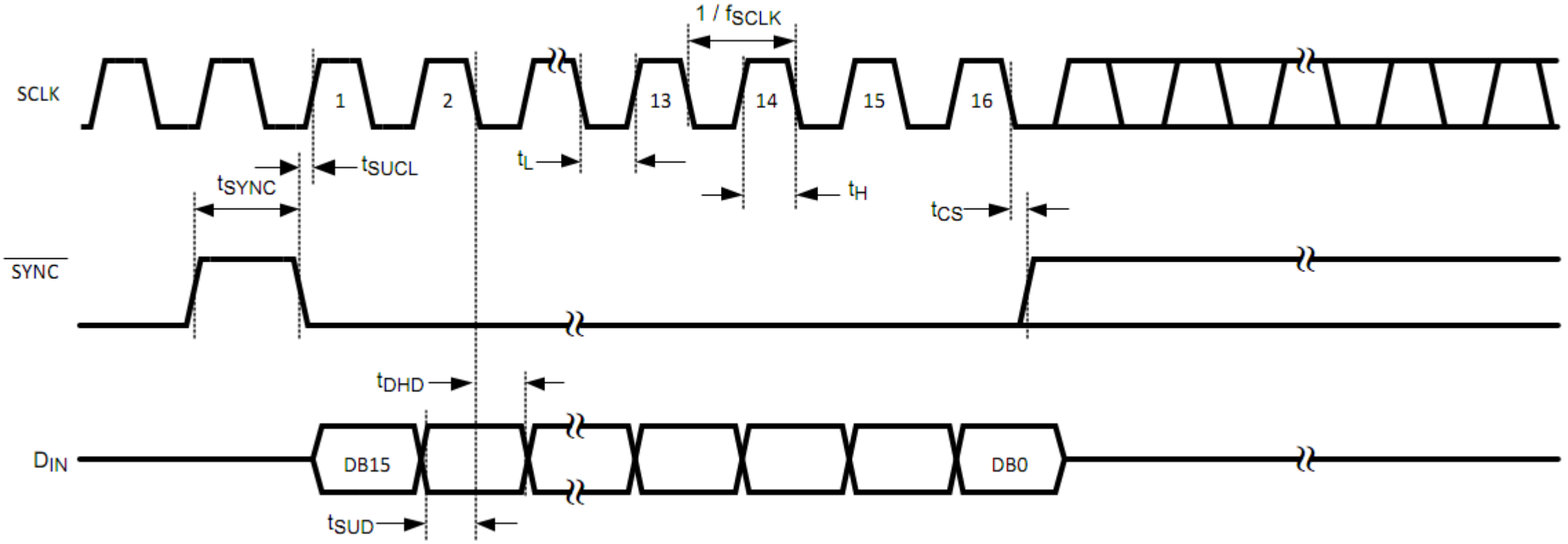






SPI DAC实验原理

▶ DAC121S101 DA转换接口时序



- 任何两次写操作之间必须使(SYNC#) 维持一段时间的高电平，以便启动下一次数据传输。该芯片支持SCLK的最高时钟频率为30MHz

SPI DAC实验原理

▶ DAC121S101 DA转换接口时序

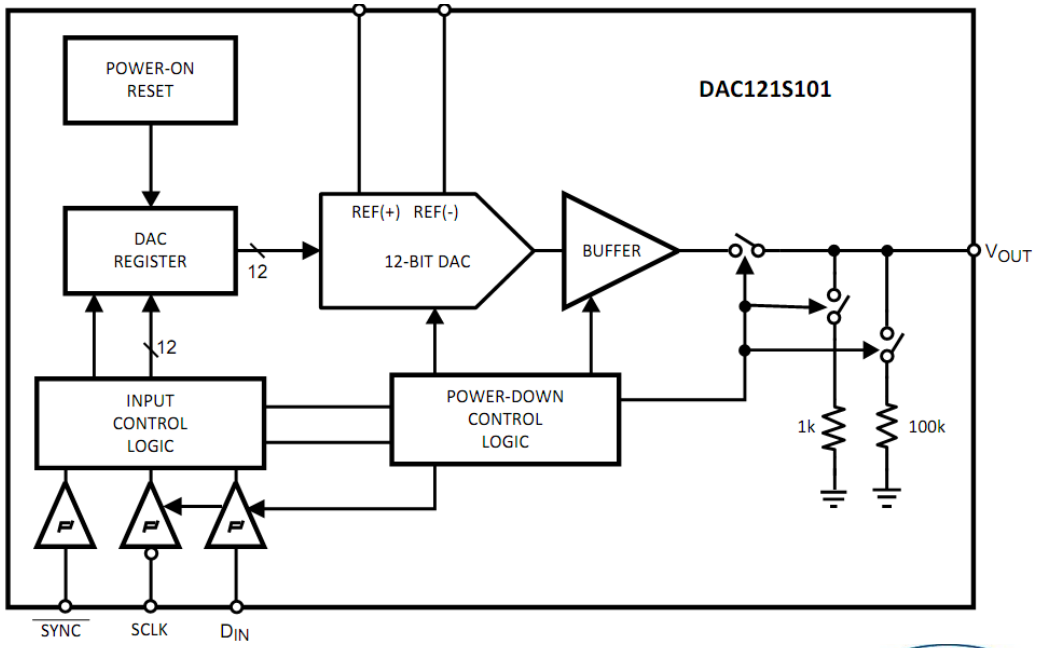
- 16位串行数据的含义

DB15 (MSB) DB0 (LSB)

X	X	PD1	PD0	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0

- D0~D11为12位DA转换的数字量，
- PD0~PD1为电源下拉控制逻辑的输入，控制电源下拉模块的工作方式，改变输出Vout的输出连接方式

PD1	PD0	电源下拉模块工作方式
0	0	正常工作（不下拉），Vout正常输出
0	1	Vout通过1K电阻下拉
1	0	Vout通过100K电阻下拉
1	1	Vout为高阻状态



SPI DAC实验原理

▶ DAC121S101 SPI 时序控制要求

- DAC121S101要求SPI总线时序空闲时SCLK为低电平，并且在SCLK的下降沿采样数据，因此SPI总线接口控制寄存器SPICR的CPOL需设置为0，CPHA需设置为1。
- DAC121S101要求SPI总线高位优先传送，因此SPI总线接口控制寄存器的LSB优先需设置为0。 此SPI总线接口需设置为主设备，并且使能SPI接口。
- 由于DAC121S101要求每次传送16位数据，而且在两次数据的传送过程中必须使(SYNC)维持一定时间的高电平，因此可以配置SPI总线接口采用16位数据、自动控制SS（N）的数据传送方式，其中N=1。AXI SPI接口通过硬件配置为采用16位数据传送，并且不使用FIFO的模式。
- 由于DAC121S101支持的最高时钟频率为30MHz，如果AXI总线时钟为100MHz，那么可以将该频率4分频，得到SPI输出时钟频率为25MHz。

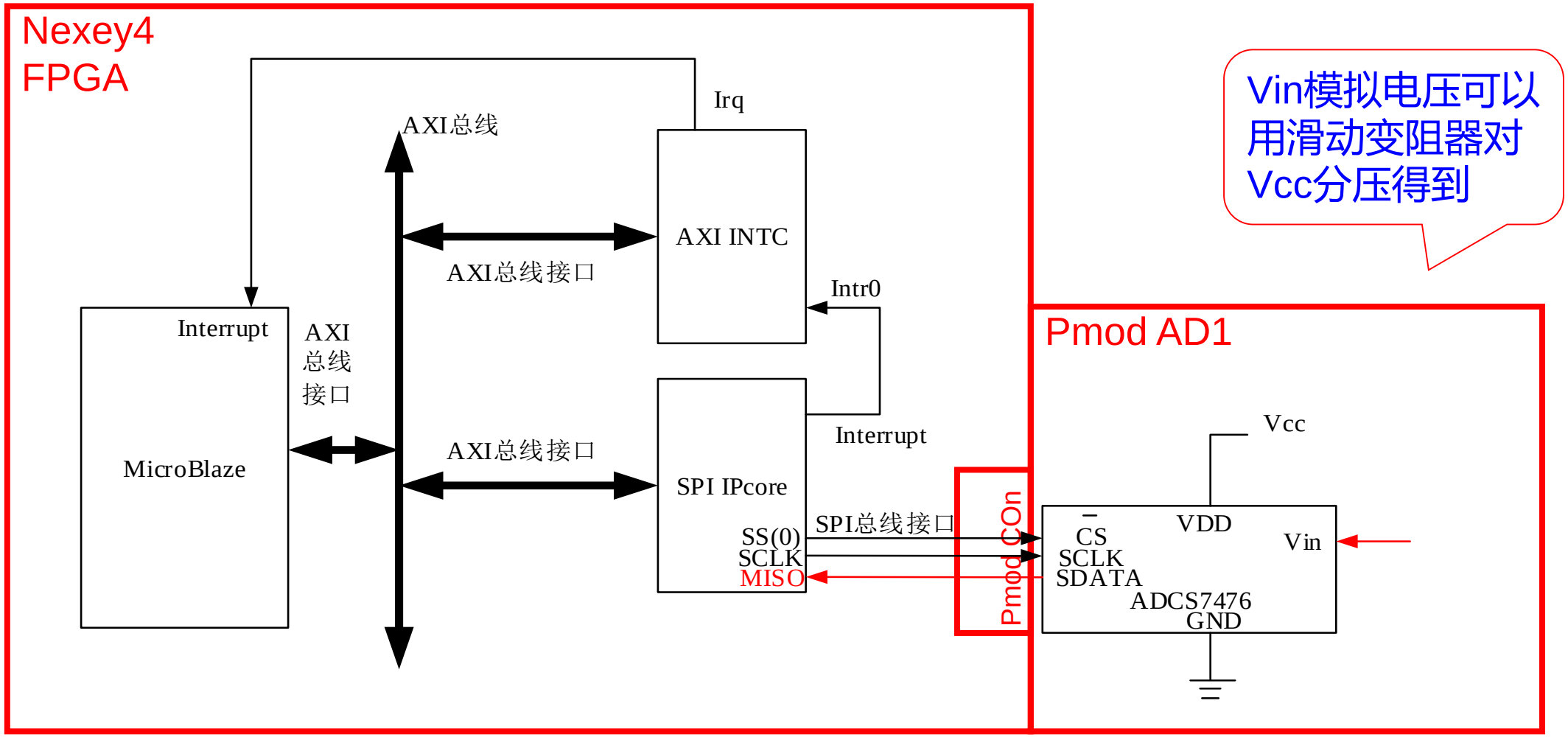
Agenda

- 实验内容
  - 目的
  - 任务及时间安排
  - 报告要求

- 原理回顾
  - UART IP Core
  - UART 实验原理
  - SPI IP Core
  - DAC实验原理
  - ADC实验原理

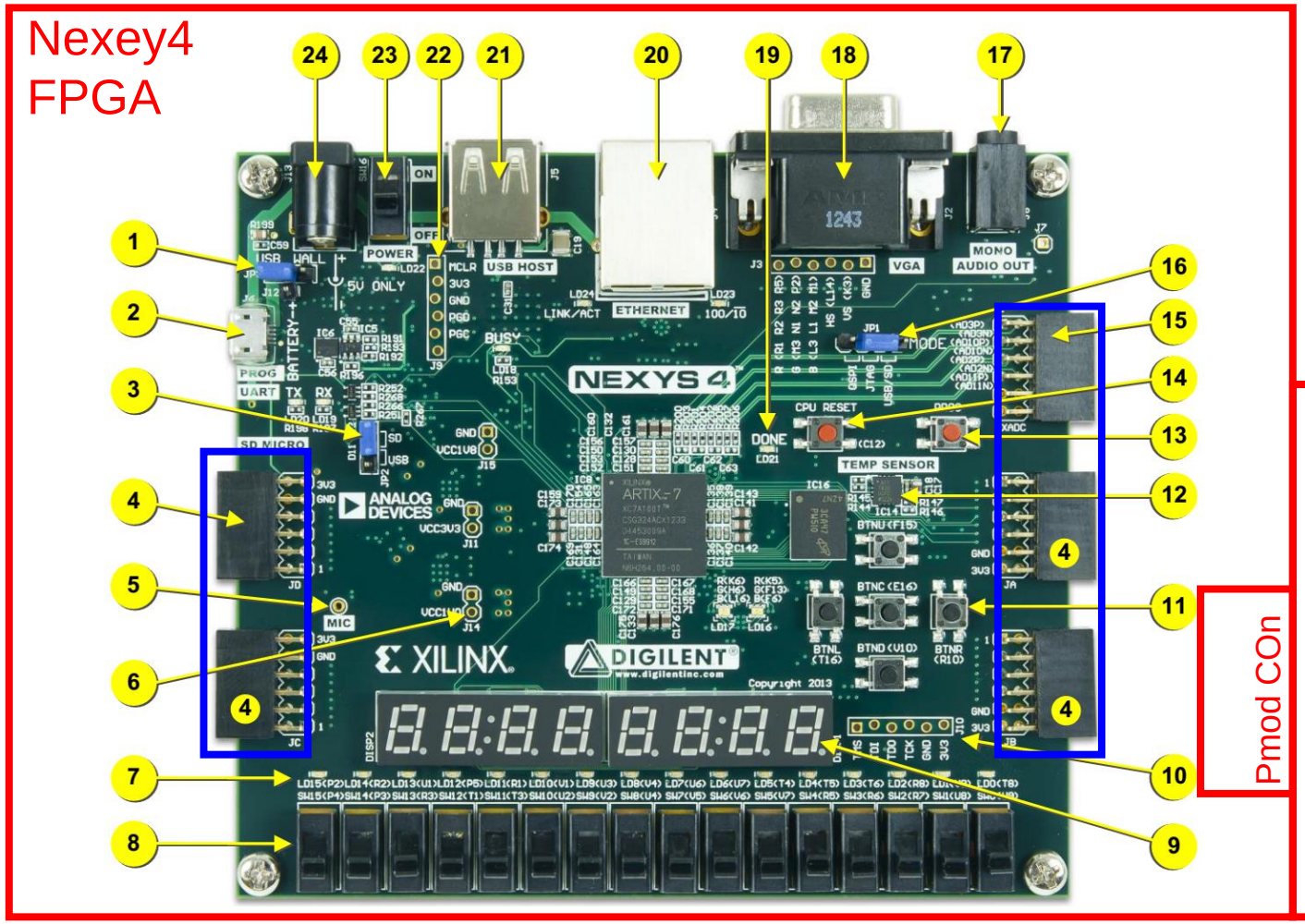
SPI ADC实验原理

- 利用ADCS7476转换芯片，基于SPI总线控制其Vin管脚上的模拟电压

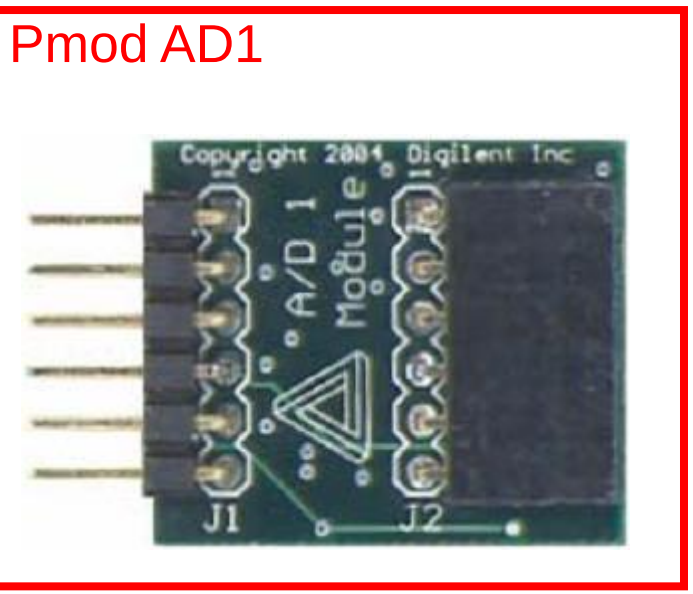


SPI ADC实验原理

- 利用ADCS7476转换芯片，基于SPI总线控制其Vin管脚上的模拟电压



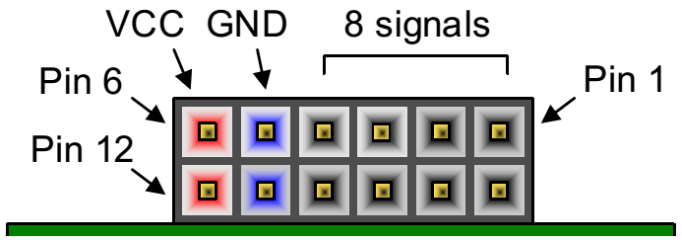
Vin模拟电压可以用滑动变阻器对Vcc分压得到



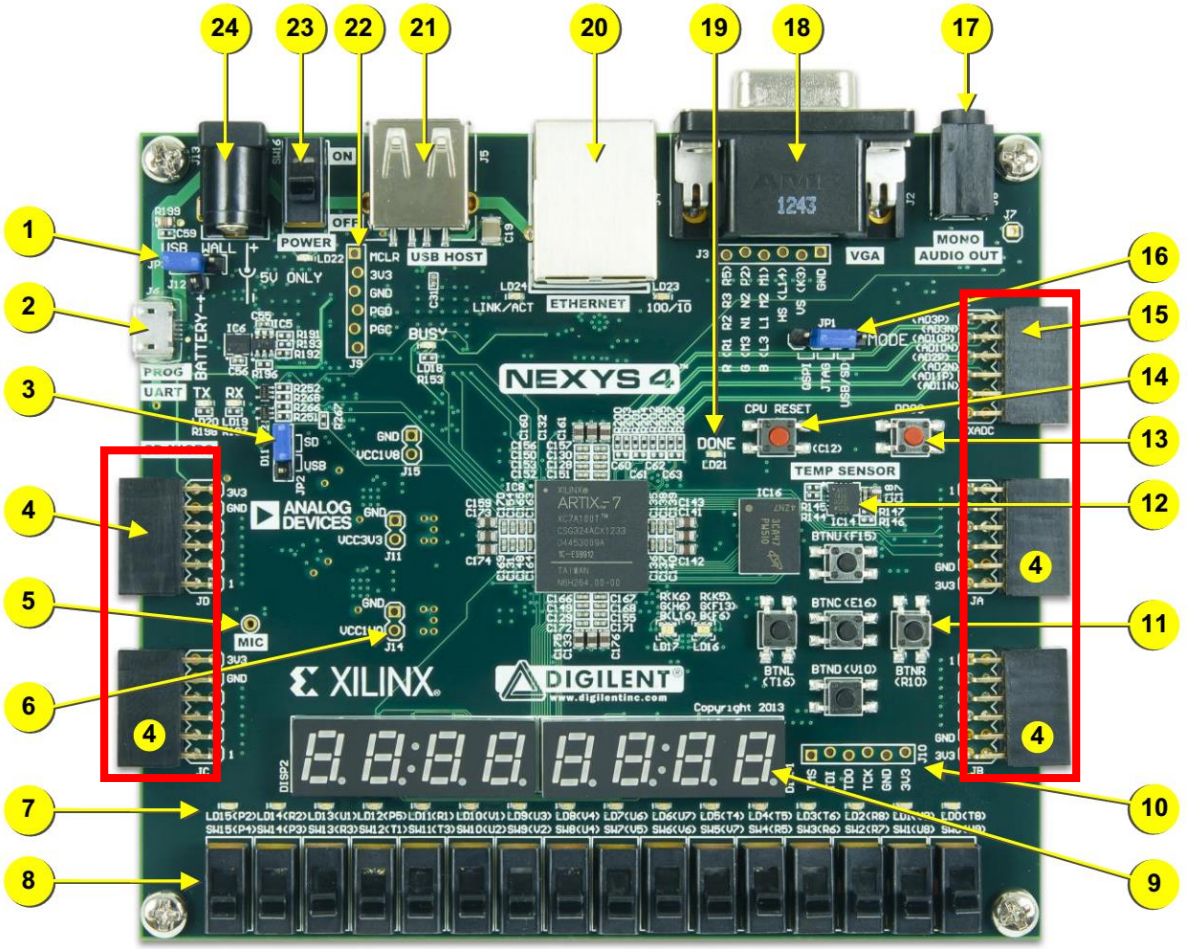
SPI ADC实验原理

▶ Pmod Connectors

[1] 提供外部扩展能力
[2] 本实验中可用



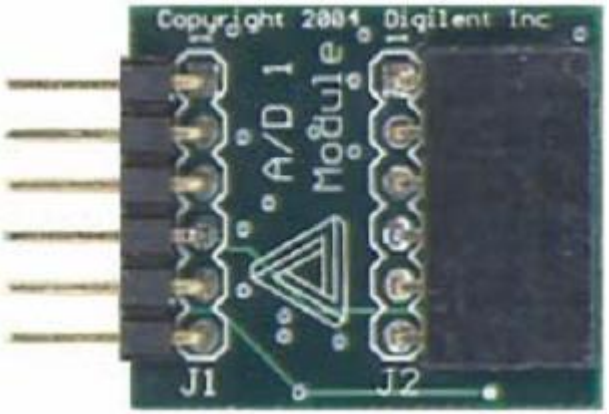
Pmod JA	Pmod JB	Pmod JC	Pmod JD	Pmod XDAC
JA1: B13	JP1: G14	JC1: K2	JD1: H4	JXADC1: A13
JA2: F14	JB2: P15	JC2: E7	JD2: H1	JXADC2: A15
JA3: D17	JB3: V11	JC3: J3	JD3: G1	JXADC3: B16
JA4: E17	JB4: V15	JC4: J4	JD4: G3	JXADC4: B18
JA7: G13	JB7: K16	JC7: K1	JD7: H2	JXADC7: A14
JA8: C17	JB8: R16	JC8: E6	JD8: G4	JXADC8: A16
JA9: D18	JB9: T9	JC9: J2	JD9: G2	JXADC9: B17
JA10: E18	JB10: U11	JC10: G6	JD10: F3	JXADC10: A18

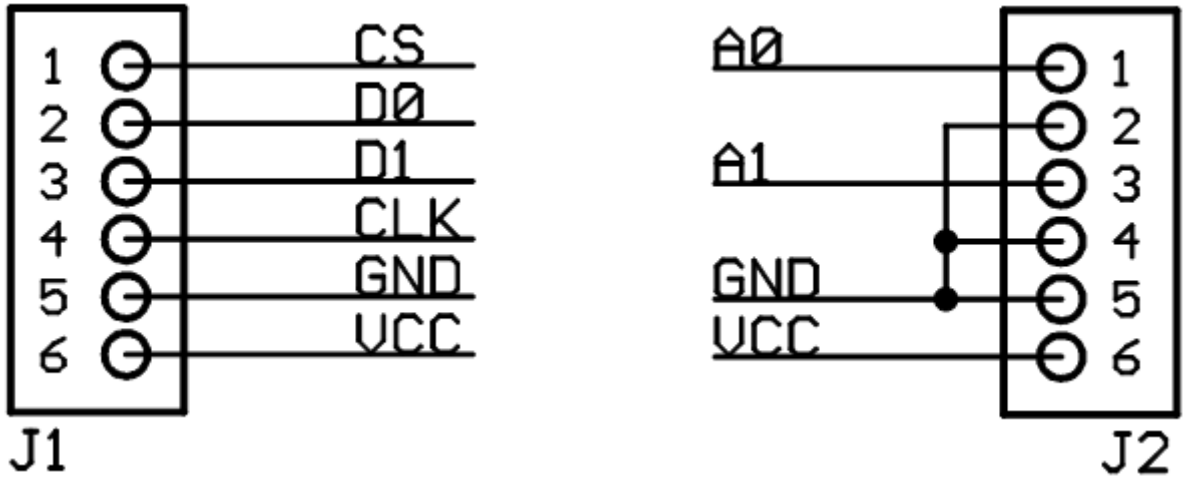


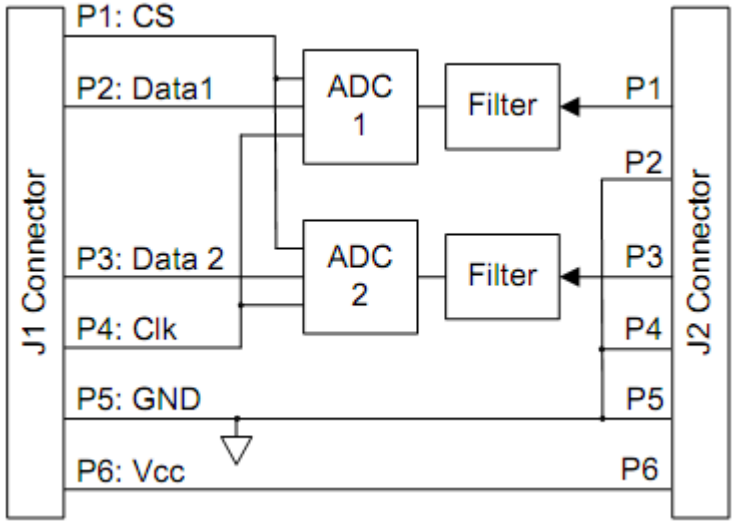
SPI ADC实验原理

▶ Pmod AD1

- two ADCS7476MSPS 12-bit A/D converter chips
- a 6-pin header connector
- a 6-pin connector
- two 2-pole Sallen-Key anti-alias filters
- two simultaneous A/D conversion channels at up to one MSa per channel
- very low power consumption
- small form factor (0.95” x 0.80”).

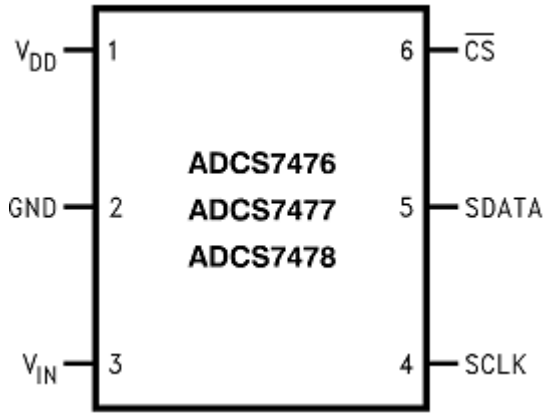


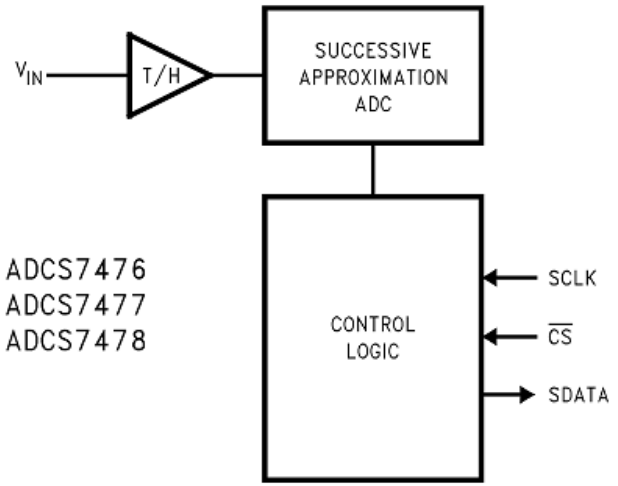




SPI ADC实验原理

▶ ADCS7476 概述



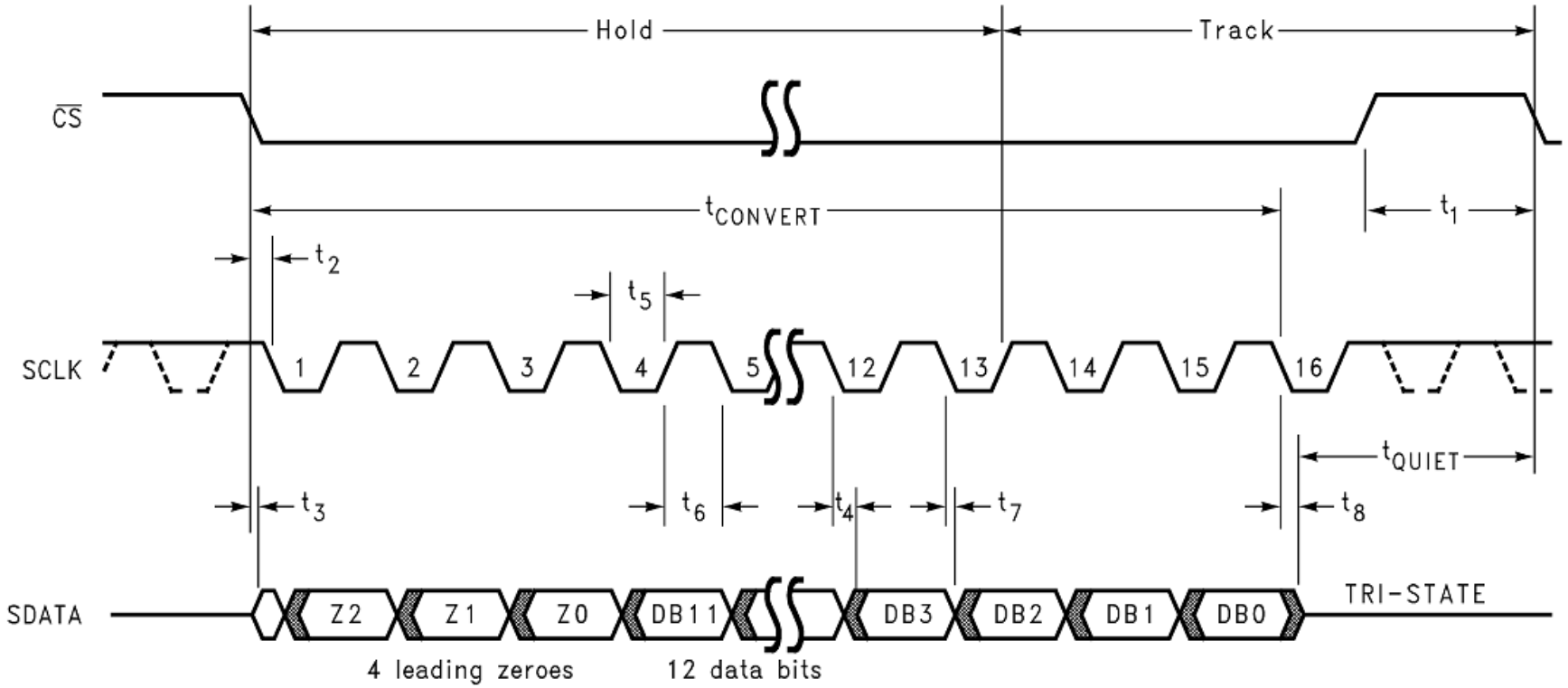


PIN DESCRIPTIONS

Pin No.	Symbol	Description
ANALOG I/O		
3	V_{IN}	Analog input. This signal can range from 0V to V_{DD}.
DIGITAL I/O		
4	SCLK	Digital clock input. The range of frequencies for this input is 10 kHz to 20 MHz, with ensured performance at 20 MHz. This clock directly controls the conversion and readout processes.
5	SDATA	Digital data output. The output words are clocked out of this pin by the SCLK pin.
6	$\overline{CS}$	Chip select. A conversion process begins on the falling edge of $\overline{CS}$.
POWER SUPPLY		
1	V_{DD}	Positive supply pin. These pins should be connected to a quiet +2.7V to +5.25V source and bypassed to GND with 0.1 µF and 1 µF monolithic capacitors located within 1 cm of the power pin. The ADCS7476/77/78 uses this power supply as a reference, so it should be thoroughly bypassed.
2	GND	The ground return for the supply.

SPI ADC实验原理

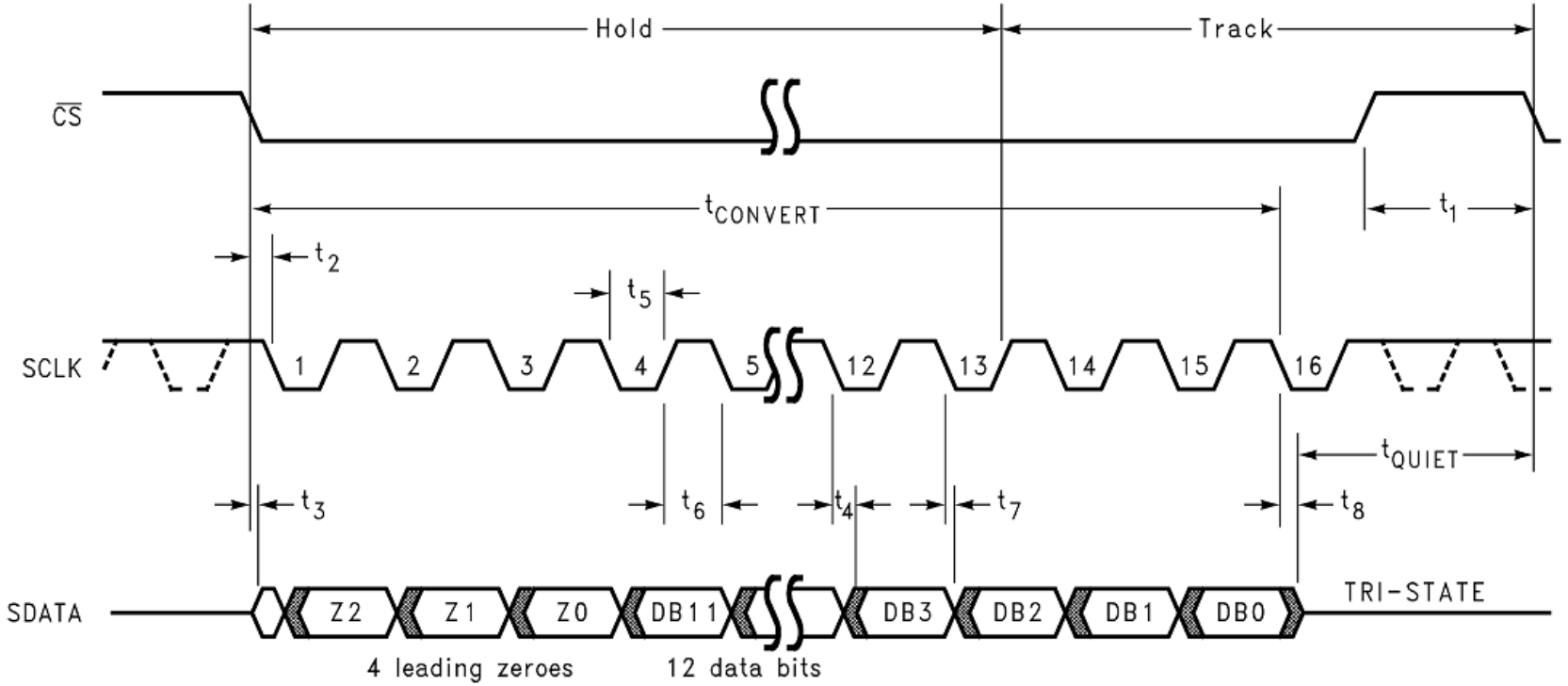
▶ ADCS7476 ADC转换接口时序



- CS#低电平启动AD转换，同时输出转换结果
- 转换结果包含4个前导0，12位有效数据，MSB在前，LSB在后
- 数据在SCLK的下降沿输出，要求接收设备在上升沿采样数据线

SPI ADC实验原理

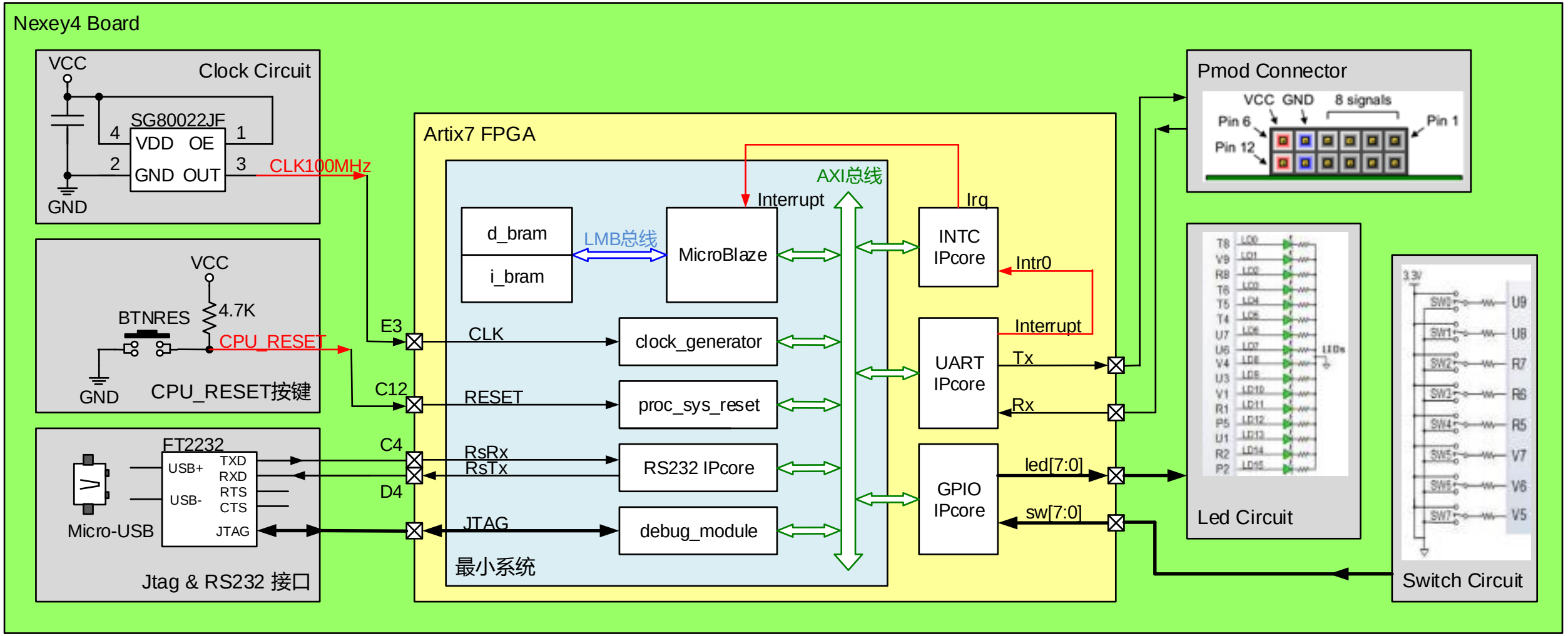
- ADC转换接口对SPI时序的要求
  - 当CPOL=0，CPHA=1时，空闲时SCLK为低电平，SCLK下降沿采样数据
  - 当CPOL=0，CPHA=0时，空闲时SCLK为低电平，SCLK上升沿采样数据
  - 当CPOL=1，CPHA=1时，空闲时SCLK为高电平，SCLK下降沿采样数据
  - 当CPOL=1，CPHA=0时，空闲时SCLK为高电平，SCLK上升沿采样数据



Agenda

- 实验内容
  - 目的
  - 任务及时间安排
  - 报告要求
- 原理回顾
  - UART IP Core
  - UART 实验原理
  - SPI IP Core
  - DAC实验原理
  - ADC实验原理
- 参考设计

▶ UART 参考设计：硬件


Nexey4 Board
Clock Circuit
VCC
SG80022JF
4
VDD
OE
1
2
GND
OUT
3
CLK100MHz
GND
VCC
4.7K
BTNRES
CPU_RESET
GND
CPU_RESET按键
FT2232
USB+
USB-
TXD
RXD
RTS
CTS
JTAG
Micro-USB
Jtag & RS232 接口
E3
C12
C4
D4
Artix7 FPGA
AXI总线
Interrupt
Irq
d_bram
i_bram
LMB总线
MicroBlaze
CLK
clock_generator
RESET
proc_sys_reset
RsRx
RsTx
RS232 IPcore
JTAG
debug_module
最小系统
INTC
IPcore
Intr0
Interrupt
UART
IPcore
Tx
Rx
GPIO
IPcore
led[7:0]
sw[7:0]
Pmod Connector
VCC GND 8 signals
Pin 6
Pin 12
Pin 1
Led Circuit
Switch Circuit

▶ UART 参考设计：软件

```
// UART的测试程序
#include "xparameters.h"       //The hardware configuration describing constants
#include "xintc.h"             //Interrupt Controller API functions
#include "xil_io.h"            // IO functions
#include "xil_types.h"
#include "mb_interface.h"

#define sw_DATA_OFFSET         0x0000  // switch 数据寄存器偏移地址
#define sw_TRI_OFFSET          0x0004  // switch 控制寄存器偏移地址
#define sw_ISR_OFFSET          0x0120  // switch 中断状态寄存器偏移地址

#define led_DATA_OFFSET        0x0008  // led 数据寄存器偏移地址
#define led_TRI_OFFSET         0x000C  // led 控制寄存器偏移地址
#define led_ISR_OFFSET         0x0120  // led 中断状态寄存器偏移地址

#define intc_ISR_OFFSET        0x00    // 中断控制器中断状态寄存器偏移地址
#define intc_IER_OFFSET        0x08    // 中断控制器中断允许寄存器偏移地址
#define intc_IAR_OFFSET        0x0C
#define intc_MER_OFFSET        0x1C

#define uart_Rx_OFFSET         0x00    // UART接收数据寄存器偏移地址
#define uart_Tx_OFFSET         0x04    // UART发送数据寄存器偏移地址
#define uart_STAT_OFFSET       0x08    // UART状态寄存器偏移地址
#define uart_CTRL_OFFSET       0x0c    // UART控制寄存器偏移地址
```

```

// 注册总中断服务程序地址
void My_ISR (void) __attribute__ ((interrupt_handler));

void Initialize();                      //初始化函数（包含中断初始化）
void UartHandler();                     //UART处理函数
void Delay_50ms();                      //延时函数

short                   flag_RX;        // 接收标志
unsigned char           rxData;         // 接收到的数据
int                     txTime;         // 发送时刻

```

▶ UART 参考设计：软件

```
36
37  int main(void)
38  {
39          xil_printf("\r\nRunning UART Test(No APP)!\r\n");
40          Initialize();
41          while(1)
42          {
43                  if(flag_RX)             //若接收到数据，则打印相关信息
44                  {
45                          xil_printf("Uart receives data!!! the data is 0x%X\n\r",rxData);
46                          flag_RX = 0;
47                  }
48                  Delay_50ms();
49                  txTime = txTime + 1;
50                  if(txTime == 10000)     txTime = 0x00;
51
52                  int     flag_Uart;
53                  flag_Uart = Xil_In32(XPAR_UART_BASEADDR + uart_STAT_OFFSET);//读取UART的状态Reg
54                  if(flag_Uart&0x04)      // bit2 = 1,说明是UART发送FIFO空
55                  {
56                          if(txTime%2)    // 控制发送速度
57                          {
58                                  int     data_Tx;
59                                  data_Tx = Xil_In32(XPAR_GPIO_BASEADDR + sw_DATA_OFFSET);     // 读取SW
60                                  Xil_Out32(XPAR_UART_BASEADDR + uart_Tx_OFFSET,data_Tx);      // 通过UART发送SW值
61                          }
62                  }
63          }
64          return 0;
65  }
66
```

▶ UART 参考设计：软件

```
66
67  void Initialize()
68  {
69          flag_RX = 0x00;
70          rxData = 0x00;
71          txTime = 0x00;
72
73          Xil_Out8(XPAR_GPIO_BASEADDR + sw_TRI_OFFSET,0xff);          // sw is used as input
74
75          Xil_Out8(XPAR_GPIO_BASEADDR + led_DATA_OFFSET,0x5A);
76          Xil_Out8(XPAR_GPIO_BASEADDR + led_TRI_OFFSET,0x00);         // led is used as output
77
78          Xil_Out32(XPAR_UART_BASEADDR + uart_CTRL_OFFSET,0x13);      // UART Interrupt enable and clear FIFO
79
80          Xil_Out32(XPAR_INTC_BASEADDR + intc_IAR_OFFSET,0xffffffff); // claer all irq requests
81          Xil_Out32(XPAR_INTC_BASEADDR + intc_IER_OFFSET,0x80000003); // Intr[0] Interrupt enable
82          Xil_Out32(XPAR_INTC_BASEADDR + intc_MER_OFFSET,0x03);       // INTC Interrupt enable
83
84          microblaze_enable_interrupts();       // CPU interrupt enable
85  }
86
87  void My_ISR(void)
88  {
89          int status;
90          status = Xil_In32(XPAR_INTC_BASEADDR + intc_ISR_OFFSET);    // 读取ISR
91          if(status&0x01)                                             // ISR[0]=1,说明是UART中断
92          {
93                  UartHandler();
94          }
95
96          Xil_Out32(XPAR_INTC_BASEADDR + intc_IAR_OFFSET,status);     // 写IAR清INTC中断标志
97  }
```

▶ UART 参考设计：软件

```
98
99  void UartHandler()
100 {
101         int    flag_Uart;
102         flag_Uart = Xil_In32(XPAR_UART_BASEADDR + uart_STAT_OFFSET);        //读取UART的状态Reg
103         if(flag_Uart&0x01)                      // bit0 = 1,说明是UART接收FIFO收到有效数据
104         {
105                 flag_RX = 1;
106                 rxData = Xil_In32(XPAR_UART_BASEADDR + uart_Rx_OFFSET);     //读取UART的接收Reg
107                 Xil_Out8(XPAR_GPIO_BASEADDR + led_DATA_OFFSET,rxData);      //输出到Led
108         }
109         Xil_Out32(XPAR_UART_BASEADDR + uart_CTRL_OFFSET,0x13);              // UART Interrupt enable and clear FIFO
110 }
111
112 void Delay_50ms()
113 {
114         int i;
115         for(i=0;i<5000000;i++);
116 }
```

Thanks