# 1 SPI接口

## 1.1 概念和原理

SPI是Serial Peripheral Interface Bus的缩写，是一种高速的，全双工，同步的通信总线。它是一种用于短距通信的同步串行通信接口标准，主要用于嵌入式系统。这个接口是Motorola在1980年末开发的，之后变成一种约定俗成的通信标准，SPI协议使用单个Master的主-从（Master-Slave）结构，以全双工的方式工作，其接口连接方式如图1.1所示。

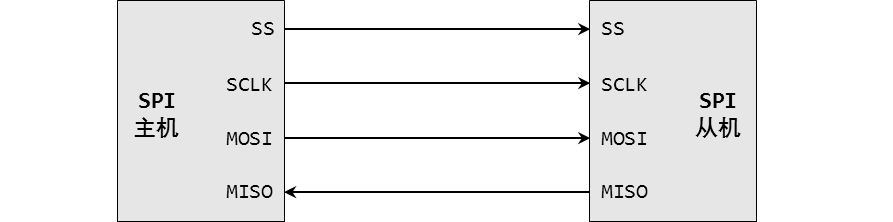


图1.1 SPI接口示意图

SPI接口是在CPU和外围低速器件之间进行同步串行数据传输，在主器件的移位脉冲下，数据按位传输，高位在前，低位在后，为全双工通信，数据传输速度总体来说比I2C总线要快，速度可达到几Mbps。采用SPI协议通信的设备通常只需要四条线就可以完成数据的传输，因此，这种占用端口资源少的优点也被称为SPI协议的一个亮点，但是SPI接口的缺点是没有指定的流控制，没有应答机制确认是否接收到数据。

* **SCLK**：串行时钟，由Master输出，从机接受SCLK信号。它控制着数据传输的节拍，进而影响数据交换的快慢。
* **MOSI**：（Master output Slave input）主器件数据输出，从器件数据输入。
* **MISO**：（Master input Slave output）主器件数据输入，从器件数据输出。
* **SS**：（Slave Select）片选信号，只有该Slave上的SS信号有效时，该Slave才被选中，通常情况下为低电平有效。

由于SPI接口具有片选信号，因此单个SPI主设备可以控制多个从设备进行读写，其连接图如图1.2所示。

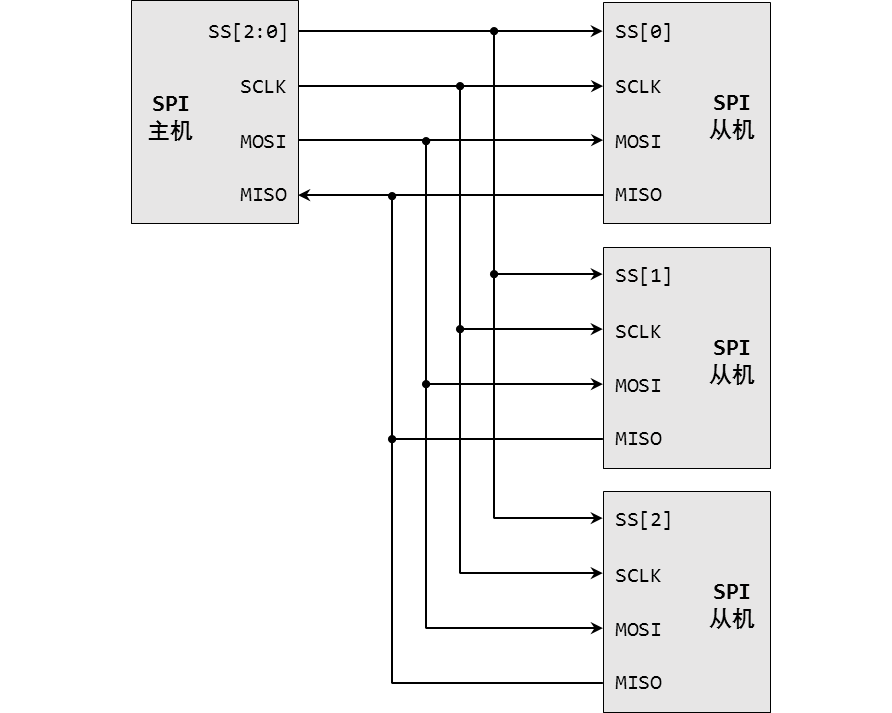


图1.2 SPI接口片选示意图

SPI通信过程本质上来讲，就是数据的交换。在数据交换的过程中完成数据的发送和接收。主机控制SS信号和SCLK信号的产生，在SS信号有效时，相应的从机被选中，在SCLK的节拍下完成数据交换。

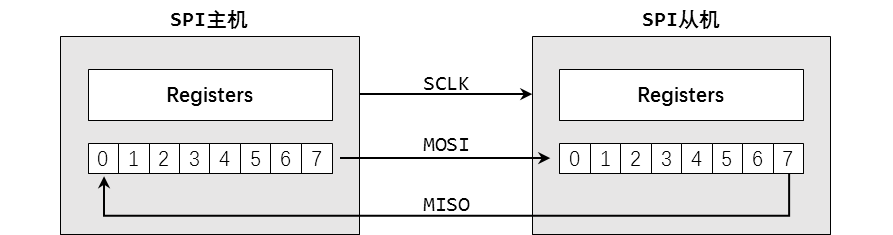


图1.3 SPI接口数据交换示意图

SPI可根据SCLK的不同形式可以分为四种工作模式，四种工作模式受控于CPOL和CPHA，即串行时钟SCLK的极性和相位。

表1.1 SPI接口工作模块

| **PI模式** | **时钟极性（CPOL）** | **时钟相位（CPHA）** |
| --- | --- | --- |
| 0 | 0 | 0 |
| 1 | 0 | 1 |
| 2 | 1 | 0 |
| 3 | 1 | 1 |

SPI 模块为了和外设进行数据交换，根据外设工作要求，其输出串行同步时钟极性和相位可以进行配置，时钟极性（CPOL）对传输协议没有重大的影响。

若 CPOL=0，串行同步时钟的空闲状态为低电平；

若 CPOL=1，串行同步时钟的空闲状态为高电平。时钟相位（CPHA）能够配置用于选择两种不同的传输协议之一进行数据传输；

若 CPHA=0，在串行同步时钟的第一个跳变沿（上升或下降）数据被采样；

若 CPHA=1，在串行同步时钟的第二个跳变沿（上升或下降）数据被采样。SPI主模块和与之通信的外设备时钟相位和极性应该一致。

四种工作模式现如下：

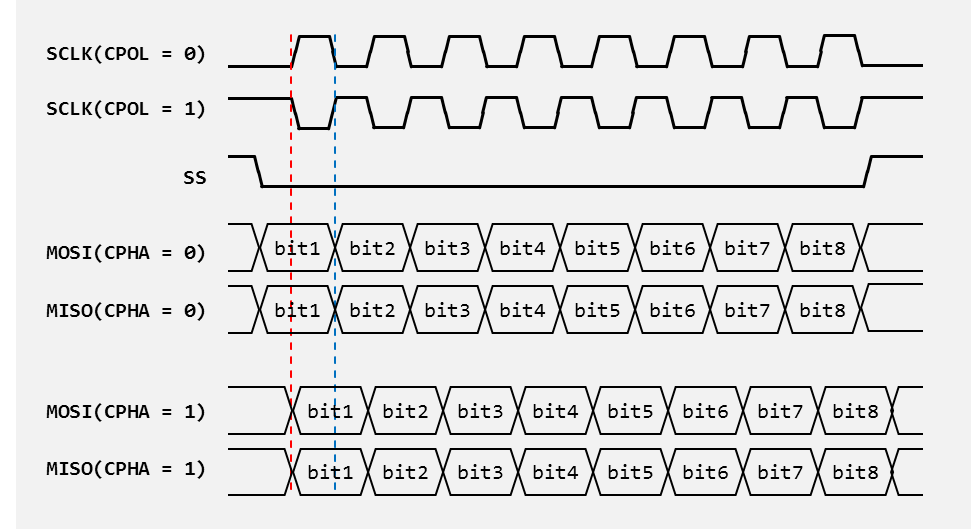


图1.4 SPI接口时序

## 1.2 SPI接口设计

根据以上概念和原理，使用Verilog描述一对SPI主机和从机模块，要求能够实现主机对从机内部任意寄存器进行读写，设计满足自定义数据位宽以及地址寻址位宽，最后通过testbench进行相关测试，完成SPI接口模块的设计。

### 1.2.1 端口定义

SPI主机与从机模块端口都必须具有四个基本端口，即SCLK、SS、MISO以及MOSI，除此之外，SPI主机还需要控制端口以实现对从机的数据交换，下表是SPI主机与从机的端口设计表。

表1.2 主机端口定义

|  |  |  |
| --- | --- | --- |
| 名称 | 类型 | 说明 |
| CLOCK | input | 模块输入时钟 |
| DATAO | output | 输出数据 |
| DATAI | input | 输入数据 |
| ADDR | input | 地址 |
| WR | input | 写使能 |
| RD | input | 读使能 |
| BUSY | output | 繁忙 |
| SS | output | 片选 |
| SCLK | output | 时钟 |
| MOSI | output | 主出从入 |
| MISO | input | 主入从出 |

表1.3 从机端口定义

|  |  |  |
| --- | --- | --- |
| 名称 | 类型 | 说明 |
| SS | output | 片选 |
| SCLK | output | 时钟 |
| MOSI | output | 主出从入 |
| MISO | input | 主入从出 |

两者模块连接方式如下图所示：

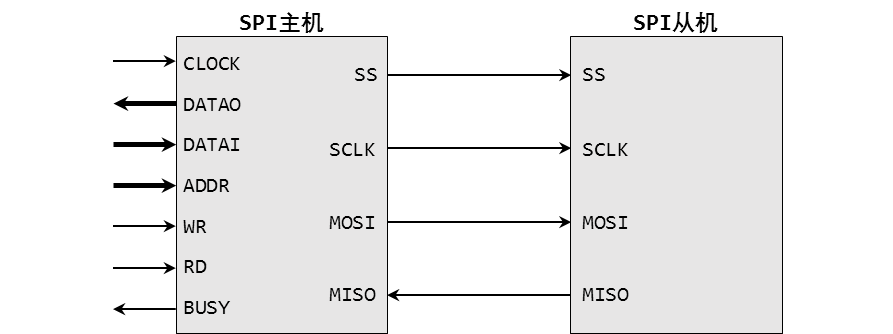


图1.5 模块连接示意图

该SPI接口模块成对使用，CLOCK作为整个模块的驱动时钟；SPI主机具有两路数据端口，分别是读数据DATAO与写数据DATAI；ADDR为读写数据对应的寄存器偏移地址；WR与RD为写使能与读使能，两者同为上升沿有效；BUSY为繁忙指示信号，当数据交换完成后回归低电平。

### 1.2.2 模块代码

下面给出用Verilog HDL语言实现SPI接口的关键程序，包括SPI主机、SPI从机以及tesnbench测试程序：

|  |
| --- |
| // SPI 主机代码  module SPI\_MASTER #(parameter D = 8, A = 8)(  input wire CLOCK,  output reg [D-1:0] DATAO,  input wire [D-1:0] DATAI,  input wire [A-1:0] ADDR,  input wire WR,  input wire RD,  output reg BUSY,    output reg SS,  output wire SCLK,  output reg MOSI,  input wire MISO  );  reg [7 :0] bit;  reg [D-1:0] data;  reg [A-1:0] addr;  reg wr;  reg clken;  initial begin  SS <= 1;  MOSI <= 0;  clken <= 0;  BUSY <= 0;  DATAO <= 0;  data <= 0;  addr <= 0;  bit <= 0;  wr <= 0;  end  assign SCLK = clken & ~CLOCK;  always @(posedge CLOCK or posedge (WR|RD)) begin  if (WR|RD) begin  BUSY <= 1;  data <= DATAI;  addr <= ADDR;  wr <= WR;  bit <= 0;  SS <= 0;  clken <= 0;  end else begin  if (wr) begin  if (bit == 0) begin  MOSI <= 1;  clken <= 1;  end else if (bit <= A) begin  MOSI <= addr[bit-1];  end else if (bit <= A + D ) begin  MOSI <= data[bit-A-1];  end else if (bit == A + D + 1) begin  clken <= 0;  MOSI <= 0;  end else if (bit > A + D + 1) begin  SS <= 1;  BUSY <= 0;  end  end else begin  if (bit == 0) begin  clken <= 1;  end else if (bit <= A) begin  MOSI <= addr[bit-1];  end else if (bit == A+1) begin  MOSI <= 0;  end else if (bit <= A + D + 1) begin  DATAO[bit-A-2] <= MISO;  end else if (bit == A + D + 2) begin  clken <= 0;  end else if (bit > A + D + 2) begin  SS <= 1;  BUSY <= 0;  end  end  bit <= bit +1;  end  end  endmodule |
|  |
| // SPI 从机代码  module SPI\_SLAVE #(parameter D = 8, A = 8)(  input wire SS,  input wire SCLK,  input wire MOSI,  output reg MISO  );  localparam a = 2\*\*A;  reg [D-1:0] regs[a-1:0];  reg [D-1:0] data;  reg [A :0] addr;  reg [7 :0] bit;  integer i;  initial begin  data <= 0;  addr <= 0;  bit <= 0;  MISO <= 0;  for (i=0; i<a; i=i+1) begin  regs[i] <= 0;  end  end  always @(posedge SCLK or negedge SS) begin  if (!SCLK) begin  data <= 0;  addr <= 0;  bit <= 0;  MISO <= 0;  end else begin  if (bit <= A) begin  addr[bit] <= MOSI;  end else begin  if (addr[0]) begin // 0: rd , 1:wr  regs[addr[A:1]][bit-A-1] <= MOSI;  end else begin  if (bit <= A+D) begin  MISO <= regs[addr[A:1]][bit-A-1];  end else begin  MISO <= 0;  end  end  end  bit <= bit + 1;  end  end  endmodule |
|  |
| //testbench 代码  `timescale 1 ps/ 1 ps  module testbench();  localparam D = 8; // 设置数据位宽为8位，即1byte  localparam A = 4; // 设置地址位宽为4位，即16个寄存器  wire SS;  wire SCLK;  wire MOSI;  wire MISO;  wire BUSY;  reg CLK = 0;  reg WR = 0;  reg RD = 0;  reg [D-1:0] DATAI = 0;  reg [A-1:0] ADDR = 0;  always #50 CLK <= ~CLK;  initial begin  // 往偏移地址为7的寄存器写入数据205  #2000  DATAI = 205;  ADDR = 7;  #100  WR = 1;  #100  WR = 0;  // 从偏移地址为7的寄存器读取数据  #2000  ADDR = 7;  #100  RD = 1;  #100  RD = 0;  end  SPI\_MASTER #(.D(D), .A(A)) M(  .CLOCK (CLK),  .DATAO (),  .DATAI (DATAI),  .ADDR (ADDR),  .WR (WR),  .RD (RD),  .BUSY (BUSY),  .SS (SS),  .SCLK (SCLK),  .MOSI (MOSI),  .MISO (MISO)  );  SPI\_SLAVE #(.D(D), .A(A)) S(  .SS (SS ),  .SCLK (SCLK),  .MOSI (MOSI),  .MISO (MISO)  );  endmodule |

### 1.2.3 仿真测试

利用Modelsim工具进行模块仿真的结果如下图所示，仿真测试文件tesnbench在上一小节已经表示出来，仿真实例化的SPI主机与从机模块的数据位宽设为8位，即1byte，寄存器寻找位宽为4位，即对应16个可访问的寄存器，测试过程为对偏移地址为7的寄存器进行数据写入，并读取该寄存器的值进行数据对比。

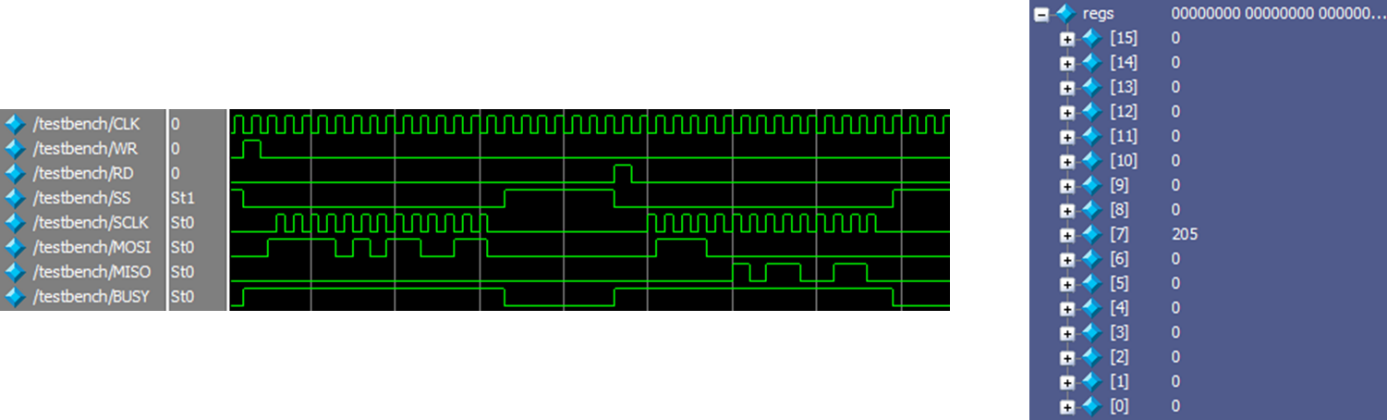


图1.6 仿真时序结果

根据图1.6，当写使能信号WR跳变后，片选信号SS拉低，繁忙指示信号BUSY拉高，随后串行时钟SCLK开始工作，MOSI信号根据地址和数据由低位到高位按位发送，发送完成之后，BUSY信号置0提示数据已经完成传输。当读使能信号RD跳变后，同样SS信号与BUSY信号随之跳变，当地址数据传输完成后，SPI主机的MISO接收到来自从机的数据信息，如图所知，读写数据一致。

# 2 UART接口

## 2.1 定义和原理

通用异步收发传输器（Universal Asynchronous Receiver/Transmitter)，通常称作UART。它将要传输的数据在串行通信与并行通信之间加以转换。作为把并行输入信号转成串行输出信号的芯片，UART通常被集成于其他通讯接口的连结上。UART是一种通用串行数据总线，用于异步通信，该总线双向通信，可以实现全双工传输和接收。在嵌入式设计中，UART用于主机与辅助设备通信，如汽车音响与外接AP之间的通信，与PC机通信包括与监控调试器和其它器件，如EEPROM通信。

在单片机中UART可以说是一种最基本的配置，很多与电脑进行通信的设备都采用到它，按计算机最常规的说法就是串行通信，其接口连接示意图如2.1所示。

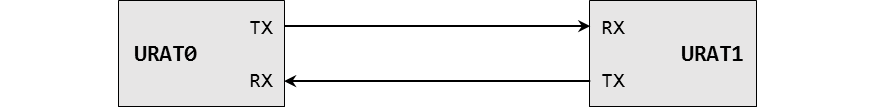


图2.1 接口连接示意图

两个设备间将数据发送接口TX与数据接受接口RX相连，RX与TX相连即可正常工作。最常用到的就是我们电脑上的USB那就是个最典型的UART接口。UART作为异步串口通信协议的一种，工作原理是将传输数据的每个字符一位接一位地传输，如下图所示。

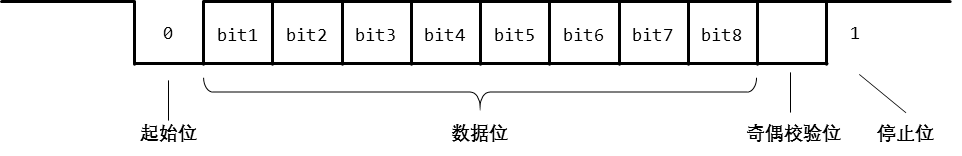


图2.2 UART时序图

* **起始位：**先产生逻辑”0”信号表示传输字符的开始。
* **数据位：**紧接着起始位之后是数据位，其个数可定制，通常采用8位构成一个字符，即ASCII码，从最低位开始传送。
* **奇偶校验位：**数据位之后是校验位，通常使用奇偶检验数据传输的正确性。
* **停止位：**它是一个字符数据的结束标志。可以是1位、1.5位、2位的高电平。 由于数据是在传输线上定时的，并且每一个设备有其自己的时钟，很可能在通信中两台设备间出现了小小的不同步。因此停止位不仅仅是表示传输的结束，并且提供计算机校正时钟同步的机会。适用于停止位的位数越多，不同时钟同步的容忍程度越大，但是数据传输率同时也越慢。
* **空闲位：**处于逻辑“1”状态，表示当前线路上没有数据传送。
* **波特率：**是衡量数据传送速率的指标。表示每秒钟传送的符号数（symbol）。一个符号代表的信息量（比特数）与符号的阶数有关。例如传输使用256阶符号，每8bit代表一个符号，数据传送速率为120字符/秒，则波特率就是120baud，比特率是120\*8=960bit/s。

## 2.2 UART接口设计

### 2.2.1 端口定义

UART接口模块的端口相对简单，没有主从之分，因此一个UART接口模块可复用，作为数据的发送端和接收端，模块接口首先需要包含数据发送端TX与数据接收端RX，其详细的端口定义如下表所示。

表2.1 UART模块端口定义

|  |  |  |
| --- | --- | --- |
| 名称 | 类型 | 说明 |
| CLOCK | input | 时钟 |
| RESET\_N | input | 复位 |
| DATAI | input | 发送数据 |
| DATAO | output | 接收数据 |
| WR | input | 发送使能 |
| BUSY | output | 繁忙指示 |
| VALID | output | 接收有效 |
| TX | output | 数据发送端 |
| RX | input | 数据接收端 |

时钟信号CLOCK为整个模块的驱动时钟，复位信号RESET\_N在低电平时有效，数据端分为发送数据DATAI和接收数据DATAO，发送使能WR在上升沿时有效，在发送阶段繁忙指示信号BUSY置高电平，当数据接收完成时接收有效信号置高并在下一次数据来临时拉低。

### 2.2.2 模块代码

下面给出用Verilog HDL语言实现UART接口的关键程序，包括UART接口模块程序以及tesnbench测试程序：

|  |
| --- |
| // UART代码  module UART #(parameter BaudRate = 9600, RefFrequency = 10\_000\_000, N = 8)(  input wire CLOCK,  input wire RESET\_N,  input wire [N-1:0] DATAI,  output reg [N-1:0] DATAO,  input wire WR,  output wire BUSY,  output reg VALID,  output reg TX,  input wire RX  );  localparam cycle = RefFrequency/BaudRate;  reg signed [ 15:0] varCount = 0;  reg [ 15:0] count0 = 0;  reg [ 15:0] count1 = 0;  reg [ 7:0] bit0 = 0;  reg [ 7:0] bit1 = 0;  reg [N-1:0] datai = 0;  reg [N-1:0] datao = 0;  reg recving = 0;  reg sending = 0;  initial begin  DATAO <= 0;  VALID <= 0;  TX <= 0;  end  always @(posedge CLOCK or negedge RESET\_N) begin  if (!RESET\_N) begin  count0 <= 0;  bit0 <= 0;  recving <= 0;  varCount <= 0;  VALID <= 0;  end else begin  if (recving) begin  if (count0 < cycle) begin  count0 <= count0+1;  varCount = RX ? varCount+1 : varCount-1;  end else if(count0 == cycle) begin  count0 <= count0+1;  bit0 <= bit0+1;  if (bit0 == 0) begin  if (varCount>0) begin  recving <= 0;  end  end else if (bit0 <= N) begin  datao[bit0-1] <= varCount>0?1:0;  end else begin  DATAO <= datao;  VALID <= 1;  recving <= 0;  end  end else begin  count0 <= 1;  varCount <= 0;  end  end else begin  if (RX==0) begin  recving <= 1;  count0 <= 1;  bit0 <= 0;  varCount <= 0;  VALID <= 0;  datao <= 0;  end  end  end  end  assign BUSY = sending;  always @(posedge CLOCK or negedge RESET\_N) begin  if (!RESET\_N) begin  count1 <= 0;  bit1 <= 0;  sending <= 0;  TX <= 1;  end else begin  if (sending) begin  if (count1 < cycle) begin  count1 <= count1+1;  end else if(count1 == cycle) begin  count1 <= count1+1;  bit1 <= bit1+1;  if (bit1 < N) begin  TX <= datai[bit1];  end else if (bit1 == N) begin  TX <= 1;  end else begin  sending <= 0;  end  end else begin  count1 <= 1;  end  end else begin  if (WR) begin  sending <= 1;  count1 <= 1;  datai <= DATAI;  bit1 <= 0;  TX <= 0;  end  end  end  end  endmodule |
|  |
| //testbench 代码  `timescale 1 ps/ 1 ps  module testbench();  localparam n = 8; //定义数据位宽为8  wire TX;  wire RX;  reg CLK = 0;  reg WR0 = 0;  reg WR1 = 0;  reg [n-1:0] DATA0 = 0;  reg [n-1:0] DATA1 = 0;  reg RESET\_N = 1;  always #1 CLK <= ~CLK;  initial begin  #100  RESET\_N <= 0;  #100  RESET\_N <= 1;    //两端分别发送 一对数据  #1000  DATA0 <= 77;  DATA1 <= 88;  #2000  WR0 <= 1;  WR1 <= 1;  #200  WR0 <= 0;  WR1 <= 0;  //两端分别发送 一对数据  #30000  DATA0 <= 55;  DATA1 <= 66;  #2000  WR0 <= 1;  WR1 <= 1;  #200  WR0 <= 0;  WR1 <= 0;  end  UART #(.BaudRate(9600), .RefFrequency(10\_000\_000), .N(n)) U1(  .CLOCK (CLK),  .RESET\_N (RESET\_N),  .DATAI (DATA0),  .DATAO (),  .WR (WR0),  .BUSY (),  .VALID (),  .TX (TX),  .RX (RX)  );  UART #(.BaudRate(9600), .RefFrequency(10\_000\_000), .N(n)) U2(  .CLOCK (CLK),  .RESET\_N (RESET\_N),  .DATAI (DATA1),  .DATAO (),  .WR (WR1),  .BUSY (),  .VALID (),  .TX (RX),  .RX (TX)  );  endmodule |

### 2.2.3 仿真测试

利用Modelsim工具进行模块仿真的结果如下图所示，仿真测试文件tesnbench在上一小节已经表示出来，仿真实例化的UART接口模块数据位宽设为8位。测试过程为UART接口一端向另一端发送并同时接收数据，判断数据是否一致以验证模块能够满足UART的基本数据传输功能，仿真实验测试的波特率为9600，参考时钟频率为10MHz。

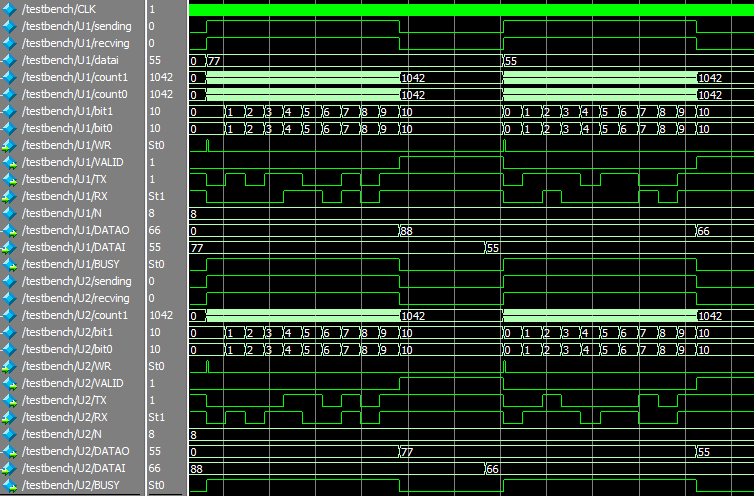


图2.3 仿真结果

由上图可知，实验进行了两次仿真实以验验证数据传输的正确性，当发送使能WR信号跳变时，数据开始向外发送，并将繁忙指示信号BUSY置1，当8位数据全部传输完成后再经过1位的停止位结束传输过程，最后在接口另一端得到正确的数据并将数据有效信号VALID置1。通过两次仿真测试的结果，可以看出模块按照UART通信协议的方式完成了正确的数据传输。

**3模拟信号与数字信号**

**3.1 信号转换基础**

随着现代数字技术的飞速发展与普及，在自动控制系统，通信系统等大量领域中，信号的处理广泛采用计算机处理技术。将自然界的各种信号运用计算机技术处理，首先涉及到的则是信号的转换。信号转换从形态变化上可以分为三种：自然界信号转换为电信号；电信号转换为自然界信号；电信号转换为另一种电信号。

自然界信号如温度、湿度、速度等，在时间上每时每刻都存在，在大小上都是连续变化的，我们将这类在时域上数学形式为连续函数的信号叫做模拟信号。我们知道，计算机只能识别0与1，要处理在时间与幅度上连续的模拟信号则必须要将模拟信号转换为计算机能够识别的0或1组成的信号，这种计算机能够识别的信号通常被称为数字信号。数字信号由一个个独立的点组成，在时间与幅度上都是离散形式。要对自然界的某些信号进行检测，控制等，往往需要一种能在模拟信号与数字信号之间起转换作用装置：模数转换器与数模转换器。

能把模拟信号转换为数字信号的装置称为模数转换器（简称ADC或A/D转换器）；反过来，将数字信号转换为模拟信号的装置称为数模转换器（简称DAC或D/A转换器）。如今几乎所有计算机系统都不可或缺的存在着ADC或DAC，例如话筒，数音响，摄像头等。为了保证数据处理结果的准确性，ADC和DAC必须有足够的转换精度。同时，为了适应快速过程的控制与检测的需要，ADC与DAC还必须有足够快的转换速度。因此，转换精度和转换速度乃是衡量ADC和DAC转换性能优劣的主要标志。

常见的ADC类型有很多种，可以分为直接ADC与间接ADC两大类。在直接ADC中，输入的模拟电压信号直接被转换成相应的数字信号；接ADC中，输入的模拟信号首先被转换成某种中间变量（例如时间、频率等），后再将这个中间变最转换成输出的数字信号。对于DAC来说，常用的有倒T形电阻网络DAC、权电流DAC、开关树型DAC以及权电容网络DAC等几种类型。此外，在ADC的输入方式上，又有并行输出和串行输输出两种类型。相对应的DAC有并行输入与串行输入两种类型。

**3.2 采样与滤波**

数字信号处理系统中，采样和滤波电路是常用的两个功能模块，采样是在波形上抽取足够多的离散点来描绘出波形的形状的过程，采样点越多，描绘的波形越精确，采样将模拟信号转换成一系列脉冲，每个脉冲表示在给定的时刻的信号幅度。采样包括两个过程，其过程如图3.1所示。



图3.1 采样过程示意图

当模拟信号被采样时，必须满足某种可以精确表示模拟信号的条件，在对周期信号采样时，必须满足奈奎斯特采样定理：若信号的频率带宽有限，要从抽样信号中无失真地恢复信号，抽样频率应大于两倍信号的最高频率。通常取采用频率Fs=(3~5)Fmax，其中Fmax为信号成分的最高频率。

对于采样后的信号，有用信号是低于奈奎斯特频率的信号。如果信号中存在高于奈奎斯特频率的信号，在采样时就会出现一个不期望出现的信号，称为混叠。混叠的产生是由于采样频率不到信号频率的两倍时产生的一个信号，混叠信号有比被采样信号的最好频率低的频率，所以会出现在输入信号的频率中而产生失真，这个信号是虚假信号，在实际中是不需要的，因此采样前的滤波十分必要。

采样的第二步是保持。经过滤波和采样后，数据必须保持恒定一段时间直到下一个采样脉冲到来，从而让模数转换器有足够的处理时间处理采样值。采样保持过程如图3.2。由于转换是在采样结束后的保持时间内完成的，多以最后转换输出的结果是模拟电压每次采样结束时的电压值。

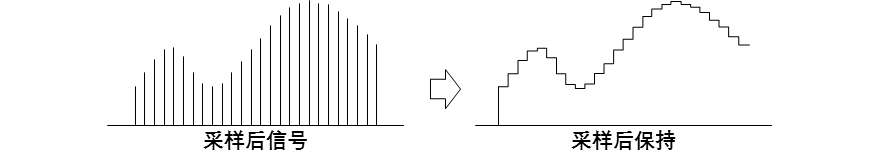


图3.2 采样保持示意图

**3.3 模数转换**

模数转换是将某一时刻采样保持过程的输出转换为代表模拟输入幅度的二进制码的过程。采样保持操作时模拟信号幅值在采样脉冲之间保持恒定，这样模数转换器在脉冲之间就可以使用一个不变的数值进行模数转换而不是变化的模拟信号。图3.3示意说明了模数转换电路的基本功能。

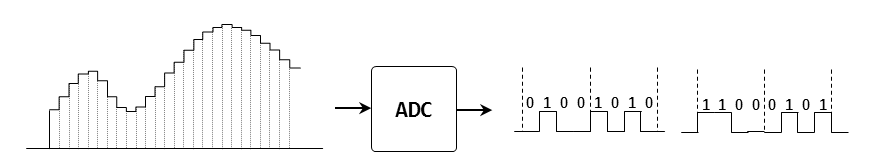


图3.3 模数转换及编码

将模拟数值转化为编码的过程被称为量化，在量化的过程中，ADC（模-数）转换器将模拟信号的每个采样数据转换为二进制代码。所用代表采样值的数据位越多，代表数据越精确。为了模拟量化过程，将一个波形量化成四个电平。 四个电平需要两位码表示。如图3.4所示，垂直方向每个量化电平用2位码表示，水平方向是用数字标识的采样间隔，在整个采样周期中，采样数据保持不变。数据量化为相邻的较低的电平，如表3.1所示，例如，比较采样3和4，它们被分配不同的电平。

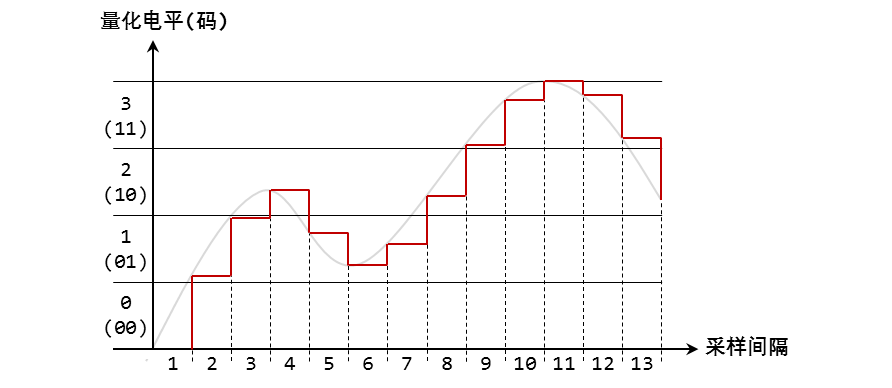


图3.4 具有四个量化电平的采样-保持输出波形

|  |  |  |
| --- | --- | --- |
| 采样间隔 | 量化电平 | 编码 |
| 1 | 0 | 00 |
| 2 | 1 | 01 |
| 3 | 1 | 01 |
| 4 | 2 | 10 |
| 5 | 1 | 01 |
| 6 | 1 | 01 |
| 7 | 1 | 01 |
| 8 | 2 | 10 |
| 9 | 3 | 11 |
| 10 | 3 | 11 |
| 11 | 3 | 11 |
| 12 | 3 | 11 |
| 13 | 3 | 11 |

表3.1 图1.4中波形采用两位码量化

如果使用2位数字码来重现原始波形，将得到图3.5给出的波形。这个操作过程被称为数-模转换（DAC），DAC就是实现数字到模拟转换的电路。从图中可以看到如果只使用2位数据表示采样值，精度相当低。下面来看增加位数如何提高准确度。图3.6给出采用4位码16个电平来量化同样的波形，4位量化过程汇总在表3.2中。

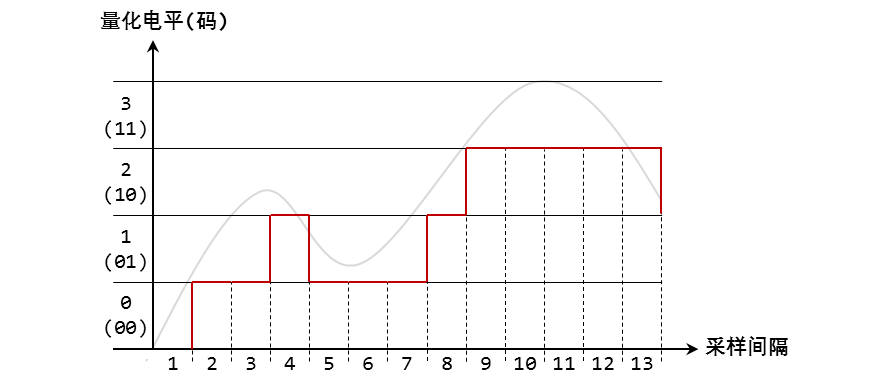


图3.5 使用两位码重建的波形

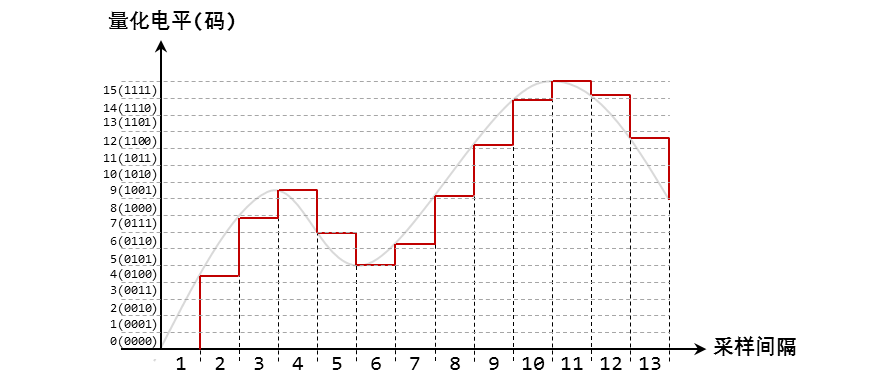


图3.6 具有16个量化电平的采样-保持输出波形

|  |  |  |
| --- | --- | --- |
| 采样间隔 | 量化电平 | 编码 |
| 1 | 0 | 0000 |
| 2 | 4 | 0100 |
| 3 | 7 | 0111 |
| 4 | 9 | 1001 |
| 5 | 7 | 0111 |
| 6 | 5 | 0101 |
| 7 | 6 | 0110 |
| 8 | 9 | 1001 |
| 9 | 12 | 1100 |
| 10 | 14 | 1110 |
| 11 | 15 | 1111 |
| 12 | 15 | 1111 |
| 13 | 12 | 1100 |

表3.2 图3.6中波形采用四位码量化

如果采用4位数字码来重建波形，将会得到图3.7中的波形。从图中可以看出使用4位数字码16个电平比使用2位数字码4个电平的重建波形更像原始波形。这说明量化位数越高，量化精度越好。大部分集成ADC的量化位数为8~24位，采样-保持功能模块在ADC内部。

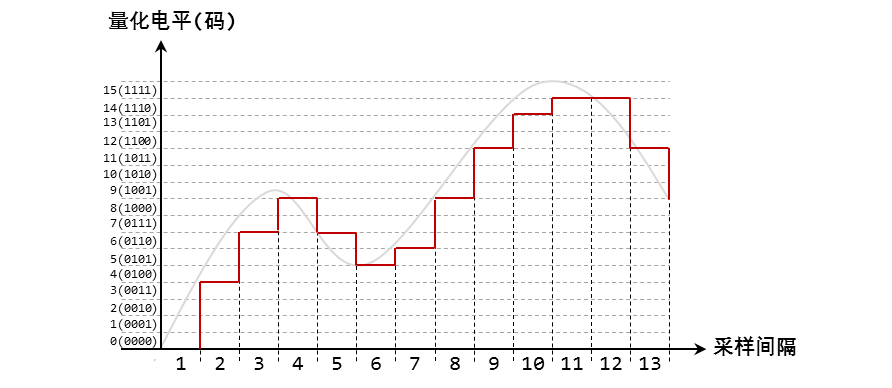


图3.7 使用四位码重建的波形

**3.4 数模转换**

数模转换就是将离散的数字量转换为连接变化的模拟量。与数模转换相对应的就是模数转换，模数转换是数模转换的逆过程，把数字量转换成模拟量，称为数模转换器（DAC）。DAC是将一组n位二进制数转换为模拟信号输出的一种器件，如图3.8所示，其中为n位二进制输入端，为模拟输出端。

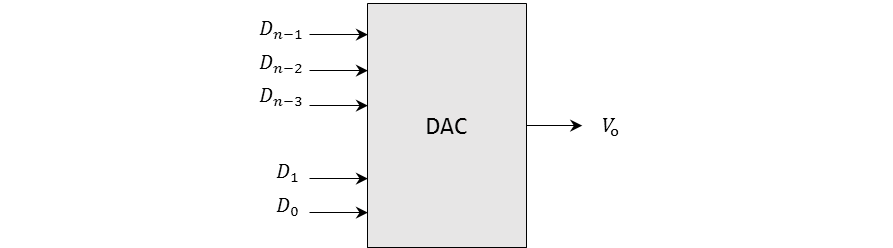


图3.8 DAC示意图

输入与输出的关系如下：

试中比例系数G为常数。

其中可以转换为十进制数，由此看出，数模转换实质是将输入的二进制数按照每一位所有的权值转化为模拟信号，然后将这些信号叠加输出，即DAC的基本原理。

n位DAC的一般结构框图如图3.9所示，按照解码网络的不同，DAC有不同结构的转换器如倒T型电阻网络DAC、权电流DAC等等。

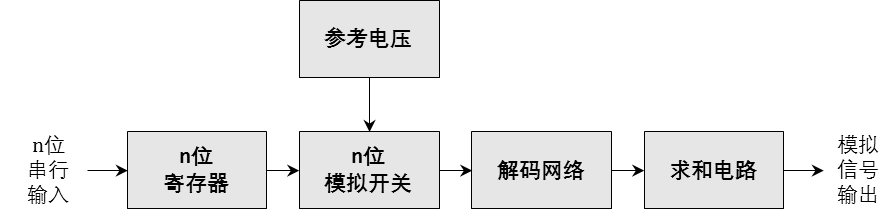


图3.9 n位DAC结构框图

**3.5 数字信号处理**

数字信号处理是将通常以模拟形式出现的信号比如声音、图像或者其他传感器信号，转换成数字形式，并根据不同的应用使用数字技术增强或者调整模拟信号数据。

在进行数字信号处理之前，首先需要通过ADC将模拟信号转换为数字信号，一旦模拟信号转换成二进制编码形式的数字信号，就可以输入到数字信号处理器（DSP）芯片中进行相关处理，DSP可以对输入信号做多种操作，如滤波去噪、纠错检测、信号提取加密等，经过DSP处理之后的信号可能需要被转换回增强形式的模拟信号，这个过程需要借助数模转换器（DAC）来实现，下图给出了一套常见的数字信号处理系统框图。

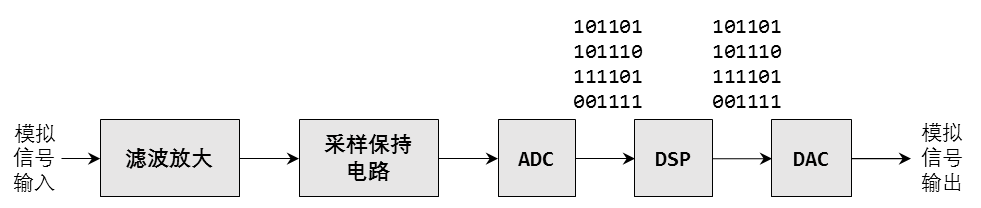


图3.10 数字信号处理系统框图

数字信号处理系统中ADC将数字信号传入DSP中进行处理和DSP将数字信号传入DAC的过程需要借助接口实现数据的传输，通常情况下，这三者提供并行或者串行的传输协议，如SPI、UART等。

接下来按照上图数字信号处理系统框图对模拟信号进行去噪并将处理后的数字信号转换为模拟信号，输入的信号是带有高频脉冲噪声的一维模拟信号，而输出信号则是经过去噪处理之后的模拟信号，其系统处理过如下图所示。

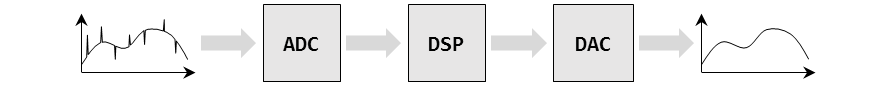


图3.11 数字信号去噪并转换为模拟信号

如上图所示，输入模拟信号带有高频噪声，如果需要在数字领域滤除这些噪声，信号首先通过ADC采样才能被DSP处理，然后得到转换之后的数字序列，对于以上这种高频噪声，在DSP中进行中值滤波就可以完好滤除这些噪声，之后得到处理后的数字序列，最后序列传输到DAC中将数字信号转换成模拟信号，实现以上过程。