

**基于FPGA的Modbus协议的实现**

院 系 自动化学院

专业班级 硕1601班

姓 名 柳 伟

学 号 M201672296

2016年 11月 12日

**摘 要**

目录

[1、绪论 2](#_Toc466796897)

[1.1 现场总线概述 2](#_Toc466796898)

[1.2 Modbus的发展 2](#_Toc466796899)

[1.3 Modbus总线产品的技术特征 2](#_Toc466796900)

[2、Modbus协议研究 2](#_Toc466796901)

[2.1 Modbus网络体系结构 2](#_Toc466796902)

[2.2 Modbus协议概述 2](#_Toc466796903)

[2.3 Modbus的两种传输方式 2](#_Toc466796904)

[2.4 Modbus消息桢 2](#_Toc466796905)

[2.5 错误校验方法 2](#_Toc466796906)

[3 基于FPGA的Modbus协议的实现原理 2](#_Toc466796907)

[3.1 实验平台 2](#_Toc466796908)

[3.2 FPGA开发板介绍 2](#_Toc466796909)

[3.3 串行通信的基本原理 2](#_Toc466796910)

[3.4 基于FPGA的Modbus协议的实现 2](#_Toc466796911)

[3.4.1 FPGA串口接收模块 2](#_Toc466796912)

[3.4.2 FPGA串口发送模块 2](#_Toc466796913)

[3.5 Modbus消息桢格式 2](#_Toc466796914)

# 1、绪论

## 1.1 现场总线概述

现场总线是安装在制造和过程区域的现场装置与控制室内的自动化装置之间的数字式串行多点通信的数据总线，是计算机技术、通信技术和控制技术发展的结晶。现场总线控制系统（FCS）是一种全数字化、全分散、全开放、可互操作和开放式互连的新一代控制系统。由于采用了全数字化技术，FCS极大低简化了传统控制系统繁杂的布线工作量，并使系统检测和控制单元的分布更加合理。与传统的分布式控制系统（DCS）相比，FCS具有高可靠性、可维护性好、成本低、实时性好，以及实现了管控一体化的结构体系等优点。

现场总线技术需要处理自动化行业千变万化的现场仪表设备，需要实现不同厂家、不同种类产品的互连，现场总线技术标准化工作显得至关重要。为此，国际电工委员会（IEC）于1984年提出制定现场总线技术标准IEC1158，它的目标是制定面向整个工业自动化的现场总线标准。经过不断的发展，各国的大公司都投入了大量的人力、物力、财力，在市场上展开了激烈的竞争，产生了几十种现场总线标准。这些协议和标准有许多的共同之处，又有各自的特色和技术细节。IECTC65于1999年12月通过IEC61158国际技术标准。

当今国际现场总线的技术已经相当成熟，但仍与其他技术（计算机、网络技术）融合发展，广泛应用于各个行业和各个领域，涌现出大批的总线技术产品，各种总线相互竞争，试图夺取主动权，建立自身的优势。经过相互的争夺、协商和妥协，已经形成了几大主流的现场总线。然而，一些非主流的总线仍具有各自的独特市场和发展潜力。

## 1.2 Modbus的发展

Modbus是由Modicon（现为施耐德电气公司的一个品牌）在1979年发明的，是全球第一个真正用于工业现场的总线协议。Modbus网络是一个工业通信系统，由带智能终端的可编程序控制器和计算机通过公用线路或局部专用线路连接而成，其系统结构既包括硬件、亦包括软件，可应用于各种数据采集和过程监控。

Modbus协议已被IEC承认为公开有效规范（PAS），Modbus/TCP和它的一个配套协议RTPS一起，作为实时工业以太网协议簇，被提交给IEC SC65C，IEC已经批准了Modbus-IDA处于D类联络状态。2005年3月，Modbus-IDA和CAN in Automation（CIA）宣布了一项专利合作协定，由该协定产生了CIA DSP 309-2规范：第一部分为CANopen和TCP/IP的交互；第二部分为Modbus/TCP映射。这对Modbus与CANopen进行通讯的方法做了标准化的规定，从而为用户采用Modbus/TCP通讯增加了更多的选择性，也减小了发生不兼容现场的可能性。

## 1.3 Modbus总线产品的技术特征

Modbus通讯协议是一种工业现场总线通讯协议，它定义的是一种设备控制器可以识别和使用的信息帧结构，独立于物理层介质，可以承载于多种网络类型中。Modbus协议把通信参与者规定为“主站”(Master)和“从站”(Slave)，数据和信息的通信遵从主／从模式，当它应用于标准Modbus网络时，信息被直接传送。Modbus总线网络中的各个智能设备通过异步串行总线连接起来，只允许一个控制器作为主站，其余智能设备作为从站。采用命令／应答的通信方式，主站发出请求，从站应答请求并送回数据或状态信息，从站不能够自己发送信息。Modbus协议定义的各种信息帧格式，描述了主站控制器访问从站设备的过程，规定从站怎样做出应答响应，以及检查和报告传输错误等。网络中的每个从设备都必须分配给一个唯一的地址，只有符合地址要求的从设备才会响应主设备发出的命令。

由于Modbus总线产品开发成本低，简单易用，并且现在已有很多工控器、PLC、变频器、显示屏等都具有Modbus通信接口，所以它已经成为一种公认的通信标准。通过Modbus总线，可以很方便地将不同厂商生产的控制设备连成工业网络，进行集中监控。Modbus最初为PLC通信而设计，它通过24种总线命令实现PLC与外界的信息交换。这些总线命令对应的通信功能主要包括AI/AO、DI／DO的数据传送。但是很多Modbus设备的控制只使用其中的几条命令，对其余命令不做反应。MODBUS总线产品具有通信协议简洁，系统结构简单，性能稳定，应用方便，适宜应用在信息量不大工业控制系统中。

# 2、Modbus协议研究

## 2.1 Modbus网络体系结构

Modbus是一个应用层报文传输协议，用于通讯不同类型的总线或网络连接的设备之间的客户机（主站）/服务器（从站）通信。目前主要通过以太网上的TCP/IP、Modbus Plus令牌传递网络以及各种介质上的异步串行传输实现Modbus通信。图2-1是一个Modbus网络体系结构的实例。

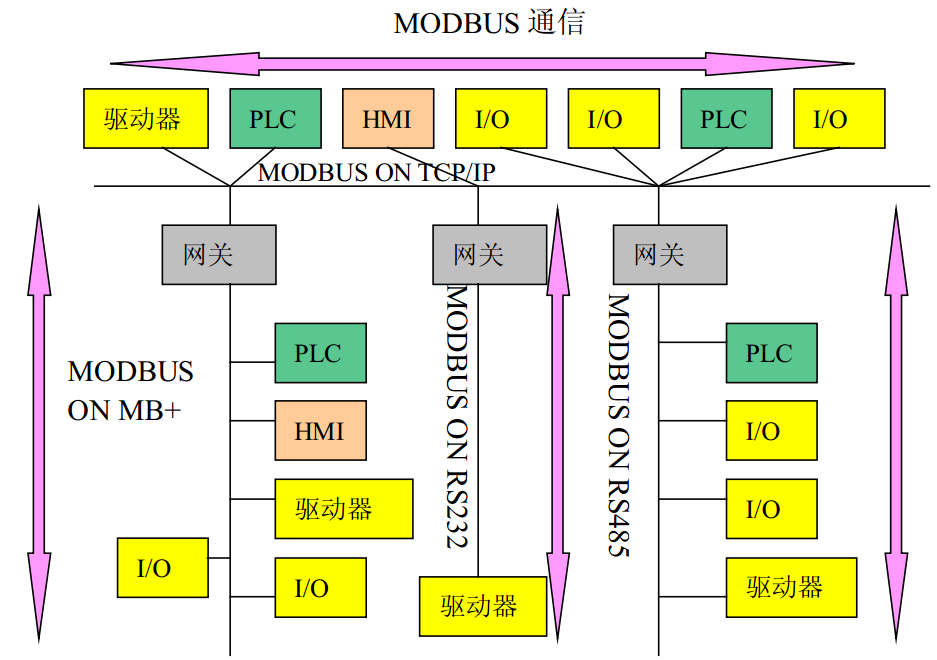


图2-1 一个Modbus网络体系结构的实例

## 2.2 Modbus协议概述

**2.2.1 总体概述**

Modbus协议是工业控制系统采用的一种通用协议。通过此协议，采集器相互之间、采集器经由网络(例如以太网)和其它设备之间可以通信，己经成为一种通用工业标准。不同厂商生产的采集设备可以通过协议连成工业网络，进行集中监控。Modbus协议描述了采集器请求访问其它设备的过程，如何回应来自其它设备的请求，以及怎样侦测错误并记录，制定了消息域格式和内容的公共格式。当在一个Modbus网络上通信时，此协议决定了每个采集器必须知道自己的设备地址，根据地址识别消息的归属，决定执行何种功能。如果需要回应，采集器将生成报文并以Modbus协议帧方式发送报文。

Modbus协议包括ASCII、RTU、TCP等协议，标准的Modbus采集器使用RS232实现串行通讯。Modbus的ASCII、RTU协议规定了消息、命令和应答的方式，数据通讯采用Maser／Slave方式，Master发出数据请求消息，slave端接收到正确消息后就可以发送数据到Master端以响应请求，Master端也可以直接发消息修改Slave端的数据，实现双向读写。Modbus协议需要对数据进行校验，串行协议中除有奇偶校验外，ASCII模式主要采用LRC校验，RTU模式主要采用16位CRC校验。Modbus是OSI模型第7层上的应用层报文传输协议，它在连接不同类型网络的设备之间提供数据的通信，如图2-2所示。

目前，可以通过下列三种方式实现Modbus通信：

(1)以太网上的TCP/IP。

(2)各种介质(有线：EIA／TIA232-F，EIA-422，EiAMA485-A，光纤，

无线等)上的异步串行传输。

(3)Modbus Plus，一种高速令牌传递网络。

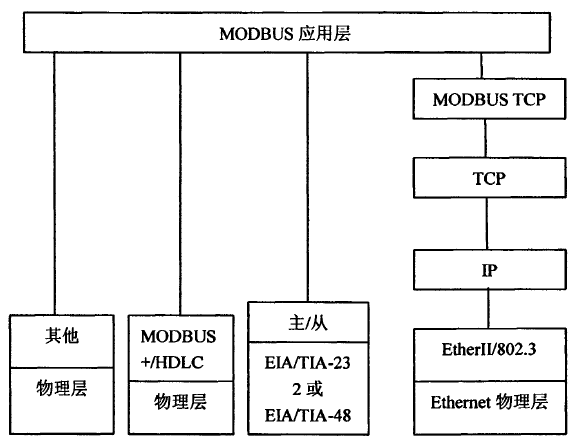


图2-2 Modbus通信线路

**2.2.2 Modbus数据单元**

Modbus协议定义了一个与基础通信层无关的简单协议数据单元（PDU），特定总线或网络上的Modbus协议映射能在应用数据单元（ADU）上引入一些附加域，通用Modbus桢如图2-3所示。

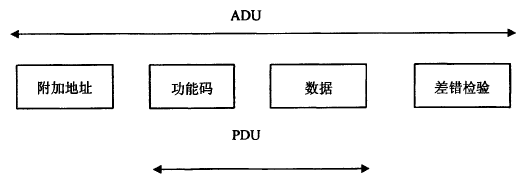


图2-3 通用Modbus桢

**2.2.3 Modbus通信网络**

Modbus协议可以方波地在各种网络体系结构内进行通信，每种设备（PLC、HMI、采集面板、变频器、运动采集、I/O设备等）都使用Modbus协议来启动远程操作，同样的通信能在串行链路和TCP/IP以太网网络上进行。而网关则能实现使用Modbus协议的总线或者网络之间的通信。

标准的Modbus是使用一个RS-232C兼容串行接口，它定义了连接口针脚、电缆、信号位、传输波特率、奇偶校验。控制器能直接经由Modem组网。控制器通信使用主—从技术，即仅一设备（主设备）能初始化传输（查询）。其他设备（从设备）根据主设备查询提供的数据做出相应的反应。

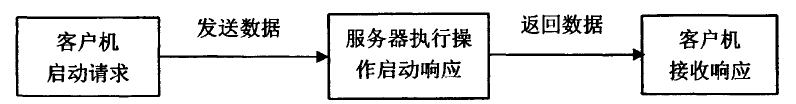


图2-4 Modbus事务处理（无差错）

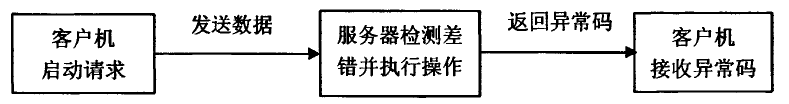


图2-5 Modbus事务处理（异常响应）

## 2.3 Modbus的两种传输方式

Modbus定义了两种串行传输模式：RTU（远程终端单元）模式和ASCII模式。

**2.3.1 ASCII模式**

在MODBUS网络上以ASCII(美国标准信息交换代码)模式通信时，在消息中，每个8bit字节对应的两位16进制数都以ASCII字符值发送。这种方式的主要优点是字符发送的时间间隔可达到1秒而不容易产生错误。消息中ASCII字符采用十六进制字符0~9，A~F；采用LRC(纵向冗长检测)进行检测。

**2.3.2 RTU模式**

在MODBUS网络上以RTU(远程终端单元)模式通信，在消息中的每个8bit字节包含两个4bit的十六进制字符。这种方式的主要优点是：在同样的波特率下，可比ASCII方式传送更多的数据。消息是8位二进制数，采用十六进制数字符0-9，A-F；采用CRC(循环冗余检测)进行检测。

## 2.4 Modbus消息桢

**2.4.1 ASCII桢**

一个典型的ASCII消息桢如表2-1所示。

表2-1 ASCII消息桢

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 起始位 | 设备地址 | 功能代码 | 数据 | LRC校验 | 结束符 |
| 1字符 | 2字符 | 2字符 | n字符 | 2字符 | 2字符 |

**2.4.2 RTU桢**

一个典型的RTU消息桢如表2-1所示。

表2-2 RTU消息桢

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 起始位 | 设备地址 | 功能代码 | 数据 | CRC校验 | 结束符 |
| T1-T2-T3-T4 | 8bit | 8bit | n 8bit | 16bit | T1-T2-T3-T4 |

## 2.5 错误校验方法

标准的MODBUS串行网络采用两种错误检测方法。帧的测分为LRC校验和CRC校验，它们都是在消息发送前由主设备产生的，从设备在接收过程中检测每个字符和整个消息帧。用户要给主设备配置一个先定义的超时时间间隔，这个时间间隔要足够长，以使任何从设备都能作为正常反应。如果从设备测到一个输错误，消息将不会接收，也不会向主设备做出回应。这样超时事件将触发主设备来处理错误。发往不存在的从设备的地址也会产生超时。。

**2.5.1 LRC校验**

使用ASCII模式，消息包括了一个基于LRC方法的错误检测域。LRC域检测了消息域中除去开始的冒号及结束的回车换行号外的内容。LRC域是一个包含一个8bit二进制值的字节。LRC值由传输设备来计算并放到消息帧中，接收设备在接收消息的过程中计算LRC，并将它和接收到消息中LRC域中的值比较，如果两个数值不相等，说明有误。LRC方法是将消息中的8bR的字节连续累加，丢弃了进位。

**2.5.2 CRC校验**

CRC即循环冗余校验码（Cyclic Redundancy Check）是数据通信领域中最常用的一种查错校验码，其特征是信息字段和校验字段的长度可以任意选定。循环冗余检查（CRC）是一种数据传输检错功能，对数据进行多项式计算，并将得到的结果附在帧的后面，接收设备也执行类似的算法，以保证数据传输的正确性和完整性。

循环冗余校验码的基本原理是：在K位信息码后再拼接R位的校验码，整个编码长度为N位，因此，这种编码也叫（N，K）码。对于一个给定的（N，K）码，可以证明存在一个最高次为N-K=R的多项式G(x)。根据G(x)可以生成K位信息的校验码，而G(x)叫做这个CRC码的生成多项式。校验码的具体生成过程为：假设要发送的信息用多项式C(X)表示，将C(x)左移R位（可表示成C(x)\*2R），这样C(x)的低位就会空出R位，这就是校验码的位置。用C(x)\*2R除以生成多项式G(x)得到的余数就是校验码。

CRC-16校验方法是：先调入一个值是全“1”的16位寄存器，然后将消息中连续的8位字节当前寄存器中的值进行处理。每个字节的8位数据位对CRC有效，起始位和停止位以及奇偶校验位均无效。CRC产生过程中，每个8位字符都单独和寄存器的值相异或，结果向最低有效位方向移动，最高有效位以0填充，异或后寄存器的值的LSB(最低有效位)被提取出来检测，如果LSB为1，寄存器单独和预置数值异或，如果LSB为O，则不进行。整个过程要重复8次。在最后一位(第8位)完成后，下一个8位字节又单独和寄存器的当前数值异或，最终寄存器中的值，就是消息中所有的字节都执行之后的CRC的值。

# 3 基于FPGA的Modbus协议的实现原理

本次实验是实现基于Modbus协议的具有差错校验和桢丢失重传功能的RS232点到点通信。

实验平台是FPGA对PC实现点到点的半双工通信。

## 3.1 实验平台

1、FPGA开发板（Altera Cyclone IV，EP4CE6F17C8）

2、USB Blaster

3、RS232串口线

4、PS/2键盘

4、上位机（串口调试助手，Serial Port Unity）

## 3.2 FPGA开发板介绍

FPGA的开发环境为Quartus II 15.0，仿真环境为Modelsim AE.

本次实验所用FPGA开发板为ALINX AX301开发板，开发板资源如下：

1）EP4CE6F17C8

2）256MB SRAM

3）4层PCB沉金工艺

4）7寸液晶屏接口

5）OV7670摄像头接口

6）SD卡座

7）40针扩展口

8）实时时钟

9）EEPROM

10）65536色VGA接口

11）PS/2接口

12）USB转串口

开发板实物图如图3-1所示，开发板的结构图如图3-2所示，USB Blaster如图3-3所示。

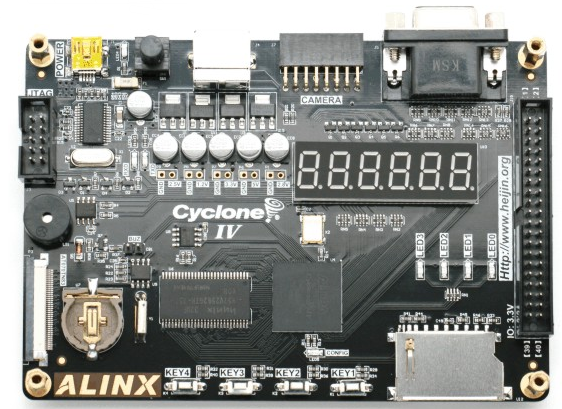


图3-1 AX301 FPGA开发板实物图

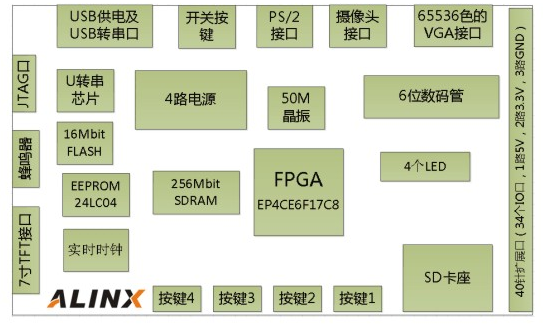


图3-2 AX301 FPGA开发板结构图图



图3-3 USB Blaster

## 3.3 串行通信的基本原理

串行通信是在通信线路上既传输数据信息也传输联络控制信息，硬件开销小，传输成本低，但是传输速度慢，且收发双方需要通信协议，可用于远距离通信。

本次实验采用的串行通信的传输方式为半双工（Half-duplex）方式，也就是数据传输是双向的，但是在这种传输方式下，除了地线之外，一般只有一根数据线。任何一个时刻，半双工的通讯方式只能由一方发送数据，另一方接收数据，不能同时手法。

本次实验FPAG与PC的通信原理如图3-4所示。



图3-4 FPGA与PC通信原理

本次实验使用的USB转RS232控制器为PL2303型，转换接口设计如图3-5所示。

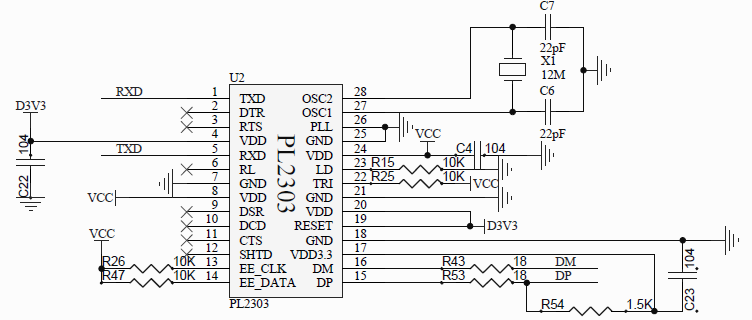


图3-5 PL2303控制器

串口传输的时序图如图3-6所示。

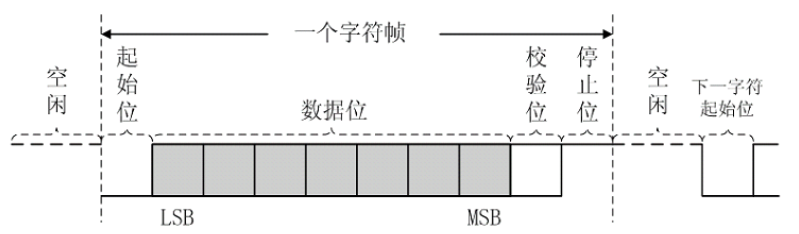
、

图3-6 串口传输时序图

串口传输数据都是一帧数据10位，如表3-1所示。

表3-1 串口传输数据位

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 位 | 0 | 1~7 | 8 | 9 |
| 位功能 | 起始位 | 数据位 | 校验位 | 停止位 |

## 3.4 Modbus消息桢（RTU桢）格式

本次实验使用RTU桢模式，消息桢格式如表3-2所示。

表3-2 RTU消息桢

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 起始位 | 设备地址 | 功能代码 | 数据 | CRC校验 | 结束符 |
| T1-T2-T3-T4 | 8bit | 8bit | 4\*8bit | 16bit | T1-T2-T3-T4 |

在本次实验中，Modbus消息桢为**8Byte**，其中数据位为**4Byte**。

## 3.5 基于FPGA的Modbus协议的实现

基于FPGA的Modbus协议由串口接收模块和串口发送模块组成，顶层模块即Modbus\_demo.v如下：

module Modbus\_demo

(

CLK,RSTn,

RX\_Pin\_In, //串口输入

PS2\_CLK\_Pin\_In, //PS\_2键盘时钟

PS2\_Data\_Pin\_In, //PS\_2键盘输入

TX\_Pin\_Out, //串口输出

LED, //4个发光二极管

DIG,SEL //6个数码管

);

input CLK;

input RSTn;

input PS2\_CLK\_Pin\_In, PS2\_Data\_Pin\_In;

input RX\_Pin\_In;

output TX\_Pin\_Out;

output [3:0]LED;

output [7:0]DIG;

output [5:0]SEL;

rx\_module\_demo RX\_U1

(

.CLK( CLK ),

.RSTn( RSTn ),

.RX\_Pin\_In(RX\_Pin\_In),

.SEL( SEL ),

.DIG( DIG )

);

tx\_module\_demo TX\_U2

(

.CLK( CLK ),

.RSTn( RSTn ),

.PS2\_CLK\_Pin\_In( PS2\_CLK\_Pin\_In ),

.PS2\_Data\_Pin\_In( PS2\_Data\_Pin\_In ),

.TX\_Pin\_Out(TX\_Pin\_Out),

.LED(LED)

);

endmodule

### 3.5.1 FPGA串口接收模块

FPGA的串口接收模块如图3-7所示。

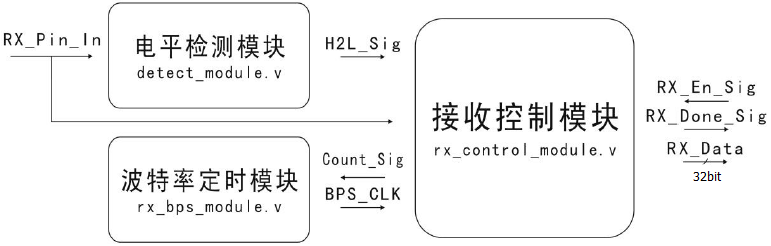


图3-7 rx\_module.v

其中RX\_Pin\_In接开发板RXD引脚（PIN\_M2），RX\_Data是接收的一个Modbus桢的数据部分（4Byte），通过开发板的数码管将RX\_Data经过处理后的低24位显示出来（因为该开发板只有6个数码管）。

如图3-7所示，rx\_module模块由detect\_module.v，rx\_bps\_module.v和rx\_control\_module.v 组成。其中rx\_bps\_module.v是产生波特率定时的模块（9600kbps）；detect\_module.v 的输入是连结物理引脚rx，它主要检测一帧数据的第0位，也就是起始位，然后产生一个高脉冲经H2L\_Sig给rx\_control\_module.v，以表示一帧数据接收工作已经开始。

FPGA串口接收模块的程序如下：

**（1）rx\_module.v**

module rx\_module //串口接收模块

(

CLK, RSTn,

RX\_Pin\_In, RX\_En\_Sig,

RX\_Done\_Sig, RX\_Data

);

input CLK;

input RSTn;

input RX\_Pin\_In;

input RX\_En\_Sig;

output [63:0]RX\_Data;

output RX\_Done\_Sig;

wire H2L\_Sig;

rx\_detect\_module U1

(

.CLK( CLK ),

.RSTn( RSTn ),

.RX\_Pin\_In( RX\_Pin\_In ), // input - from top

.H2L\_Sig( H2L\_Sig ) // output - to U3

);

wire BPS\_CLK;

rx\_bps\_module U2

(

.CLK( CLK ),

.RSTn( RSTn ),

.Count\_Sig( Count\_Sig ), // input - from U3

.BPS\_CLK( BPS\_CLK ) // output - to U3

);

wire Count\_Sig;

rx\_control\_module U3

(

.CLK( CLK ),

.RSTn( RSTn ),

.H2L\_Sig( H2L\_Sig ), // input - from U1

.RX\_En\_Sig( RX\_En\_Sig ), // input - from top

.RX\_Pin\_In( RX\_Pin\_In ), // input - from top

.BPS\_CLK( BPS\_CLK ), // input - from U2

.Count\_Sig( Count\_Sig ), // output - to U2

.RX\_Data( RX\_Data ), // output - to top

.RX\_Done\_Sig( RX\_Done\_Sig ) // output - to top

);

endmodule

**（2）detect\_module.v**

module rx\_detect\_module //下降沿检验模块

(

CLK, RSTn,

RX\_Pin\_In,

H2L\_Sig

);

input CLK;

input RSTn;

input RX\_Pin\_In;

output H2L\_Sig;

reg H2L\_F1;

reg H2L\_F2;

always @ ( posedge CLK or negedge RSTn )

if( !RSTn )

begin

H2L\_F1 <= 1'b1;

H2L\_F2 <= 1'b1;

end

else

begin

H2L\_F1 <= RX\_Pin\_In;

H2L\_F2 <= H2L\_F1;

end

assign H2L\_Sig = H2L\_F2 & !H2L\_F1;

endmodule

**（3）rx\_bps\_module.v**

//本次实验所选用的波特率为9600kbps

module rx\_bps\_module //波特率模块

(

CLK, RSTn,

Count\_Sig,

BPS\_CLK

);

input CLK;

input RSTn;

input Count\_Sig;

output BPS\_CLK;

reg [12:0]Count\_BPS;

always @ ( posedge CLK or negedge RSTn )

if( !RSTn )

Count\_BPS <= 13'd0;

else if( Count\_BPS == 13'd5207 )

Count\_BPS <= 13'd0;

else if( Count\_Sig )

Count\_BPS <= Count\_BPS + 1'b1;

else

Count\_BPS <= 13'd0;

assign BPS\_CLK = ( Count\_BPS == 12'd2604 ) ? 1'b1 : 1'b0;

endmodule

（4）**rx\_control\_module.v**

//Modbus桢接收程序

module rx\_control\_module

(

CLK, RSTn,

H2L\_Sig, RX\_Pin\_In, BPS\_CLK, RX\_En\_Sig,

Count\_Sig, RX\_Data, RX\_Done\_Sig

);

input CLK;

input RSTn;

input H2L\_Sig;

input RX\_En\_Sig;

input RX\_Pin\_In;

input BPS\_CLK;

output Count\_Sig;

output [63:0]RX\_Data;

output RX\_Done\_Sig;

reg [8:0]i;

reg [63:0]rData;

reg isCount;

reg isDone;

always @ ( posedge CLK or negedge RSTn )

if( !RSTn )

begin

i <= 8'd0;

rData <= 64'd0;

isCount <= 1'b0;

isDone <= 1'b0;

end

else if( RX\_En\_Sig )

case ( i )

// Adress Byte

8'd0 :

if( H2L\_Sig ) begin i <= i + 1'b1; isCount <= 1'b1; end

8'd1 :

if( BPS\_CLK ) begin i <= i + 1'b1; end

8'd2, 8'd3, 8'd4, 8'd5, 8'd6, 8'd7, 8'd8, 8'd9 :

if( BPS\_CLK )

begin

i <= i + 1'b1; rData[ i - 2 ] <= RX\_Pin\_In;

end

8'd10 :

if( BPS\_CLK ) begin i <= i + 1'b1; end

8'd11:

begin i <= i + 1'b1; end

8'd12 :

begin i <= i + 1'b1; end

8'd13 :

begin i <= 8'd14; end

/\*Function\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

8'd14 :

if( H2L\_Sig ) begin i <= i + 1'b1; end

8'd15 :

if( BPS\_CLK ) begin i <= i + 1'b1; end

8'd16, 8'd17, 8'd18, 8'd19, 8'd20, 8'd21, 8'd22, 8'd23 :

if( BPS\_CLK )

begin

i <= i + 1'b1; rData[ i - 16 + 8 ] <= RX\_Pin\_In; end

8'd24 :

if( BPS\_CLK ) begin i <= i + 1'b1; end

8'd25:

begin i <= i + 1'b1; end

8'd26 :

begin i <= i + 1'b1; end

8'd27 :

begin i <= 8'd28; end

/\*\*\* DATA 3 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

8'd28 :

if( H2L\_Sig ) begin i <= i + 1'b1; end

8'd29 :

if( BPS\_CLK ) begin i <= i + 1'b1; end

8'd30, 8'd31, 8'd32, 8'd33, 8'd34, 8'd35, 8'd36, 8'd37 :

if( BPS\_CLK )

begin

i <= i + 1'b1; rData[ i - 30 + 16 ] <= RX\_Pin\_In; end

8'd38 :

if( BPS\_CLK ) begin i <= i + 1'b1; end

8'd39:

begin i <= i + 1'b1; end

8'd40 :

begin i <= i + 1'b1; end

8'd41 :

begin i <= 8'd42; end

/\*\*\* DATA 2 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

8'd42 :

if( H2L\_Sig ) begin i <= i + 1'b1; end

8'd43 :

if( BPS\_CLK ) begin i <= i + 1'b1; end

8'd44, 8'd45, 8'd46, 8'd47, 8'd48, 8'd49, 8'd50, 8'd51 :

if( BPS\_CLK )

begin

i <= i + 1'b1; rData[ i - 44 + 24 ] <= RX\_Pin\_In; end

8'd52 :

if( BPS\_CLK ) begin i <= i + 1'b1; end

8'd53:

begin i <= i + 1'b1; end

8'd54 :

begin i <= i + 1'b1; end

8'd55 :

begin i <= 8'd56; end

/\*\*\* DATA 1 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

8'd56 :

if( H2L\_Sig ) begin i <= i + 1'b1; end

8'd57 :

if( BPS\_CLK ) begin i <= i + 1'b1; end

8'd58, 8'd59, 8'd60, 8'd61, 8'd62, 8'd63, 8'd64, 8'd65 :

if( BPS\_CLK )

begin i <= i + 1'b1; rData[ i - 58 + 32 ] <= RX\_Pin\_In;

end

8'd66 :

if( BPS\_CLK ) begin i <= i + 1'b1; end

8'd67:

begin i <= i + 1'b1; end

8'd68 :

begin i <= i + 1'b1; end

8'd69 :

begin i <= 8'd70; end

/\*\*\* DATA 0 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

8'd70 :

if( H2L\_Sig ) begin i <= i + 1'b1; end

8'd71 :

if( BPS\_CLK ) begin i <= i + 1'b1; end

8'd72, 8'd73, 8'd74, 8'd75, 8'd76, 8'd77, 8'd78, 8'd79 :

if( BPS\_CLK )

begin

i <= i + 1'b1; rData[ i - 72 + 40 ] <= RX\_Pin\_In; end

8'd80 :

if( BPS\_CLK ) begin i <= i + 1'b1; end

8'd81:

begin i <= i + 1'b1; end

8'd82 :

begin i <= i + 1'b1; end

8'd83 :

begin i <= 8'd84; end

/\*\*\* CRC 1 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

8'd84 :

if( H2L\_Sig ) begin i <= i + 1'b1; end

8'd85 :

if( BPS\_CLK ) begin i <= i + 1'b1; end

8'd86, 8'd87, 8'd88, 8'd89, 8'd90, 8'd91, 8'd92, 8'd93 :

if( BPS\_CLK )

begin

i <= i + 1'b1; rData[ i - 86 + 48 ] <= RX\_Pin\_In; end

8'd94 :

if( BPS\_CLK ) begin i <= i + 1'b1; end

8'd95:

begin i <= i + 1'b1; end

8'd96 :

begin i <= i + 1'b1; end

8'd97 :

begin i <= 8'd98; end

/\*\*\* CRC 2 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

8'd98 :

if( H2L\_Sig ) begin i <= i + 1'b1; end

8'd99 :

if( BPS\_CLK ) begin i <= i + 1'b1; end

8'd100, 8'd101, 8'd102, 8'd103, 8'd104, 8'd105, 8'd106, 8'd107 :

if( BPS\_CLK )

begin

i <= i + 1'b1; rData[ i - 100 + 56 ] <= RX\_Pin\_In;

end

8'd108 :

if( BPS\_CLK ) begin i <= i + 1'b1; end

8'd109:

begin i <= i + 1'b1; end

8'd110 :

begin

i <= i + 1'b1; isDone <= 1'b1; isCount <= 1'b0;

end

8'd111 :

begin i <= 8'd0; isDone <= 1'b0; end

endcase

assign Count\_Sig = isCount;

assign RX\_Data = rData;

assign RX\_Done\_Sig = isDone;

endmodule

### 3.5.2 FPGA串口发送模块