

目录

摘要	I
ABSTRACT	II
引言	1
1 相关协议介绍及论文内容安排	2
1.1 协议介绍	2
1.1.1 RS-485 协议	2
1.1.2 HDLC 协议	2
1.1.3 CRC 协议	3
1.1.4 串口通信相关说明	4
1.2 论文内容安排	4
2 软硬件平台介绍	5
2.1 软件平台	5
2.1.1 仿真环境	5
2.1.2 综合环境	5
2.2 硬件平台	5
3 模块设计及仿真结果	7
3.1 整体方案设计	7
3.2 整体电路设计	8
3.3 模块划分及仿真	9
3.3.1 485 串口部分	9
3.3.2 HDLC 协议部分	12
3.4 验证方案设计	20
3.5 验证方案仿真	20
3.5.1 使用 Quartus 进行综合设计	20
3.5.2 设置管脚	23
3.5.3 下载程序到开发板	24
3.5.4 通过串口调试助手进行调试	24
4 总结和展望	26
4.1 总结	26
4.2 展望	26
4.3 心得体会	26
致谢	27
参考文献	28

摘要

RS-485 这种串行通信方式由于其结构简单、通信速率较高和传输距离较远等优点，在多点通信方面使用非常广泛。本文首先介绍了 RS-485 通信标准，然后在 485 的基础上，使用 HDLC 协议实现 485 的接口电路，最终使用 Verilog 语言在 RTL 级描述，并在 FPGA 开发板上完成验证。

本文中的 485 接口电路包括一个发送模块和一个接收模块。发送模块主要完成的功能是对 485 串口数据的接收、并通过 HDLC 协议将数据封装后发送；接收模块主要完成的功能是接收通过 HDLC 协议封装的数据完成 HDLC 数据解封装后，将数据通过 485 串口进行发送。在仿真验证及 FPGA 调试的时候可将发送模块输出的 HDLC 封装的数据还回到接收模块的 HDLC 接收电路，并通过在 485 串口进行数据的发送和接收来完成仿真及 FPGA 验证。

关键词： 485 接口； HDLC 协议； FPGA； Verilog

ABSTRACT

RS-485 serial communication mode was widely applied in multipoint communication aspect due to its simple structure, higher communication speed, longer transmission distance and other advantages. This paper introduced RS-485 Communication Standard firstly, and then applied HDLC protocol to implement 485 interface circuit on the basis of 485. Finally, it used Verilog language to take RTL level description and complete verification on FPGA development board.

The 485 interface circuit in this paper included a sending module and a receiving module. Functions mainly completed by the sending module were to receive 485 serial data and send the data through HDLC protocol after being packaged. Functions mainly completed by the receiving module were to receive the data packaged through HDLC protocol and send the data through 485 serial after the HDLC data unpackaging was completed. In simulation verification and FPGA debugging, the HDLC packaged data output by sending module was also returned to HDLC receiving circuit of receiving module. Additionally, it took data sending and receiving through 485 serial to complete simulation and FPGA verification.

Keywords: the 485 interface circuit; the HDLC protocol; Field-Programmable Gate Array; Verilog Hardware Description Language

引言

在一些复杂系统中，需要一个合适的接口电路完成主系统和各个分系统之间的信息传递。RS-485 在物理层定义了数据如何传输的电气规范，是比较适合完成这种主从多点通信的一种接口形式。在选用数据链路层的通信协议的时候，考虑到 HDLC（High Level Data Link Control）高级数据链路控制规程协议具有：全双工传输，具有较高的传输效率、所有数据帧都经过 CRC 校验，传输可靠性高、传输控制功能和处理功能分离，灵活性高、“0 比特插入/删除”硬件比较容易实现等特点，因此在设计中选用了 HDLC 协议。

HDLC 协议的一般实现方法是采用专用的商用芯片（如 8273、8274 等），但是这些芯片在应用中灵活性较差。因此采用 FPGA 实现 HDLC 协议是一种值得采用的办法。Xilinx 等公司也推出了实现 HDLC 协议的 IP Core，但这些 IP Core 都需要付费购买许可（License）才可以使用^[1]。因此，本文采用 Verilog 语言设计了以 HDLC 协议为基础的 485 通信接口电路，并在 Altera 公司的 DE-2 FPGA 开发板上完成板级调试。

本文的 485 接口电路主要由三个大模块构成：分频模块、接收模块、发送模块。发送模块主要完成 485 串口的接收、HDLC 协议封装数据，接收模块主要完成解封装 HDLC 协议封装的数据、485 串口的发送，分频模块主要完成系统需要时钟的产生。

1 相关协议介绍及论文内容安排

1.1 协议介绍

1.1.1 RS-485 协议

RS-485 是 1983 年电子工业协会在 RS-422 的基础上增加了多点通信和双向通信的能力之后而制定的新规范。RS-485 是在 RS-422 的基础上发展的，所以 RS-485 的许多标准和 RS-422 都是相近的，不同之处就是 RS-485 是可以采用两线和四线的传输方式，在采用两线的时候可以实现真正的多点的双向通信^[2]。

由于 RS-422 和 RS-485 的电气规范是接近的，这里就对 RS-422 的电气规范做以说明。RS-422 标准的全称是“平衡电压数字接口电路的电气特性”，它定义了接口的电路特征。如图 1-1 所示是典型的 RS-422 的四线接口。RS-422 的最大传输距离约为 1219 米，最大传输速率约为 10Mb/s。

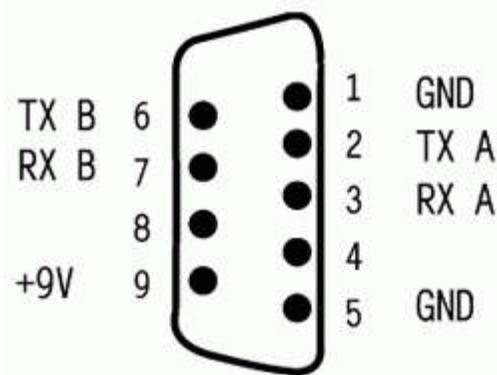


图 1-1 RS-422 接口示意图

RS-485 在电气特性上和 RS-422 是很相近的，在设计中设计的时候使用的两线的传输方式，也就是只有一个 RXD 和一个 TXD，这样也方便在后期进行调试。RS-485 和 RS-422 的最大传输距离和最大传输速率都是一样的。

1.1.2 HDLC 协议

HDLC（High Level Data Link Control）高级数据链路控制规程协议是面向比特的数据链路控制协议的典型代表；数据可以透明的进行传输，用于实现透明传输的“0 码插入法”更加容易的用硬件实现；全双工通信，有较高的数据链路的传输效率；所有数据帧均采用 CRC 检验，传输可靠性高；传输控制功能与处理功能分离，具有较大灵活性。

HDLC 数据帧主要是由帧头、帧尾、地址、控制、数据、CRC 校验几部分构成如图 1-2 所示^[3]：

帧头 01111110	地址 (8bit)	控制 (8bit)	数据 (nbit)	CRC校验 (16bit)	帧尾 01111110
----------------	--------------	--------------	--------------	------------------	----------------

图 1-2 HDLC 数据帧格式

帧头和帧尾是固定的 8bit 数据 (01111110)，用来标志数据帧的开头和结束。

地址字段的内容取决于操作方式，可以为本站地址、也可以为要传输的站的地址、也可以为 11111111 (所有站)、还可以为 00000000 (用来测试，不带地址)。

控制字段是用来对各种命令和响应，从而对数据链路做以监视和控制。

数据字段是(1-n bit)的数据，也就是要进行传输的数据。

CRC 校验字段是使用 CRC-CCITT 的校验方法对地址、控制、数据三个部分进行 CRC 校验。

在数据帧完成之后，在发送的时候还需要对地址、控制、数据及 CRC 校验这几个部分的数据进行“插入 0 码”用来区分标志帧，即遇到 5 个 1 的时候在其后插入一个“0”，同样的在接受的时候，就需要对上述的这几个部分的数据进行“删除 0 码”如果遇到 5 个 1，则删除其后的一个“0”，这里使用表 1-1 来进行解释。

表 1-1 插入“0”码和删除“0”码举例

	地址	控制	数据	CRC 校验
要发送数据	0111 1110	0000 0000	1111 1101	0011 1011
插入 0 码	01 1111 1000 0000 0001 1111 0101 0011 1011			
删除 0 码	0111 1110 0000 0000 1111 1101 0011 1011			
接收到数据	0111 1110	0000 0000	1111 1101	0011 1011

1.1.3 CRC 协议

CRC 循环冗余校验码 (Cyclic Redundancy Check)，是一种常见的查错校验码，CRC 循环冗余检查具有很强的检错能力，在数据进行发送的时候要对数据进行多项式计算，并将结果附到数据帧的最后，同样的在数据接收的时候也要进行 CRC 校验，从而确定数据传输的准确性。

CRC 校验的工作原理：生成一个 R 位的多项式，根据多项式可以生成校验码，将数据左移 R 位，然后将左移之后的数据除以多项式得到的余数就是校验码。

CRC 校验中生成多项式的选取是比较困难的，如果没有选择的非常合理，那么检查出错误的几率就会低很多，而这个问题已经被好多专家研究了好长时间了，所以对于某些位数的 CRC 校验就有了固定的多项式了，如表 1-2 列举了一些常见的 CRC 校验的多项式。

而在 HDLC 协议的 CRC 校验部分的时候使用的是 CRC-CCITT 对 24 位的数据进行校验，得到 16 位的校验结果。

表 1-2 几种常见的 CRC 校验多项式

校验标准	多项式	位数
CRC8	$x^8 + x^5 + x^4 + x^0$	8
CRC-CCITT	$x^{16} + x^{12} + x^5 + x^0$	16
CRC16	$x^{16} + x^{15} + x^2 + x^0$	16
CRC12	$x^{12} + x^{11} + x^3 + x^2 + x^0$	12

1.1.4 串口通信相关说明

在设计中由于要做到从串口的收发数据，因此串口通信时的相关实现方式就需要做以了解，在该部分大致介绍串口如何收发数据。

串口的发送或者接受是 1bit 接着 1bit 的方式进行发送或者接受的，同时它是具有一定的格式的，如图 1-3 所示。这个数据格式是从一个周期的低电平开始，以一个周期的高电平结束的，中间包含了 8 个周期的数据和一个周期的校验位，相关的时序图如图 1-3 所示^[4]。

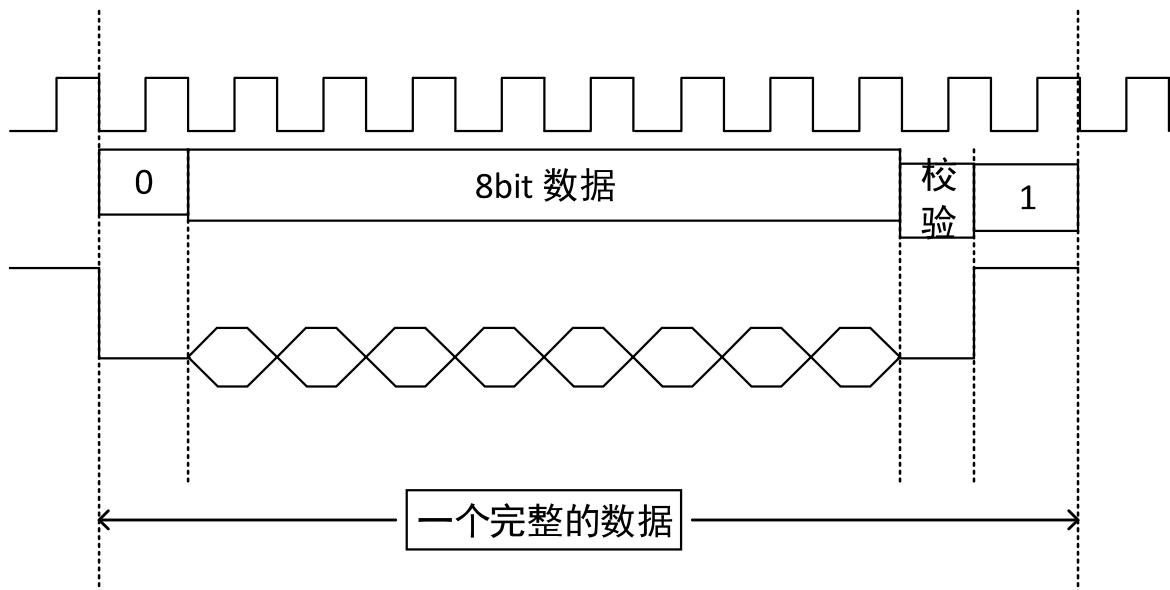


图 1-3 串口的收发数据时序图

1.2 论文内容安排

- 第一章 详细介绍 RS-485 标准、HDLC 协议、CRC 协议、串口通信等。
- 第二章 软硬平台的介绍。
- 第三章 整体介绍设计，并进行模块的划分，设计可以划分为两大部分：1、485 串口协议处理电路；2、HDLC 协议处理电路，包括：CRC 校验、“0”码插入、“0”码删除、串口接收/发送、HDLC 数据发送/接收。
- 第四章 总结展望及心得体会。

2 软硬件平台介绍

2.1 软件平台

2.1.1 仿真环境

在完成电路的硬件描述语言的编写之后需要对 verilog 代码进行仿真，这里使用的是 Mentor 公司的 modelsim 软件，版本是 v10.4。

Mentor 公司的 modeisim 软件是业界最优秀的硬件描述语言仿真软件，是 FPGA、ASIC 设计的 RTL 级和门级电路仿真的首选仿真软件。它支持 Windows、Unix、Linux 平台，是单一内核支持 VHDL 和 Verilog 混合仿真的仿真器。如图 2-1 所示是 Modelsim 的主界面。

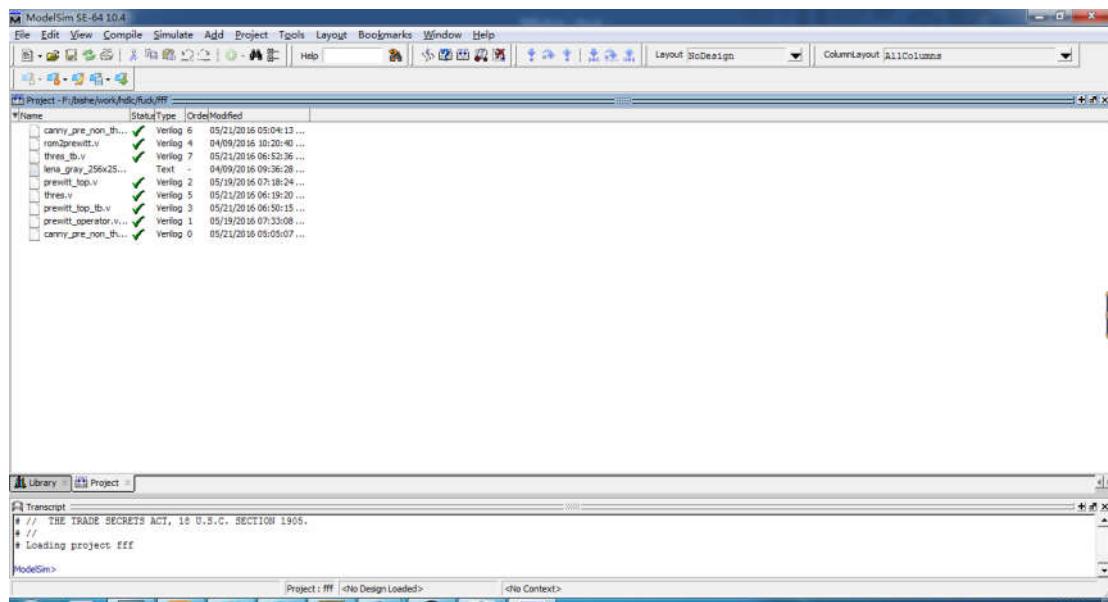


图 2-1 Modelsim 主界面示意图

2.1.2 综合环境

在仿真完成之后，就需要对 verilog 代码进行综合和布局布线，这里选用的是 Altera 公司的 Quartus v13.0，Quartus 是用于 system-on-a-programmable-chip(SOPC)的设计环境，由于其强大的设计能力和直观易用的接口，越来越受到 IC 设计者的青睐。

2.2 硬件平台

在进行设计验证的时候，使用了 Altera 公司的 DE2 开发板，它属于 Altera 公司的 Cyclone II 系列，在设计验证的时候使用的是 EP2C35F672C6N 类型的芯片。

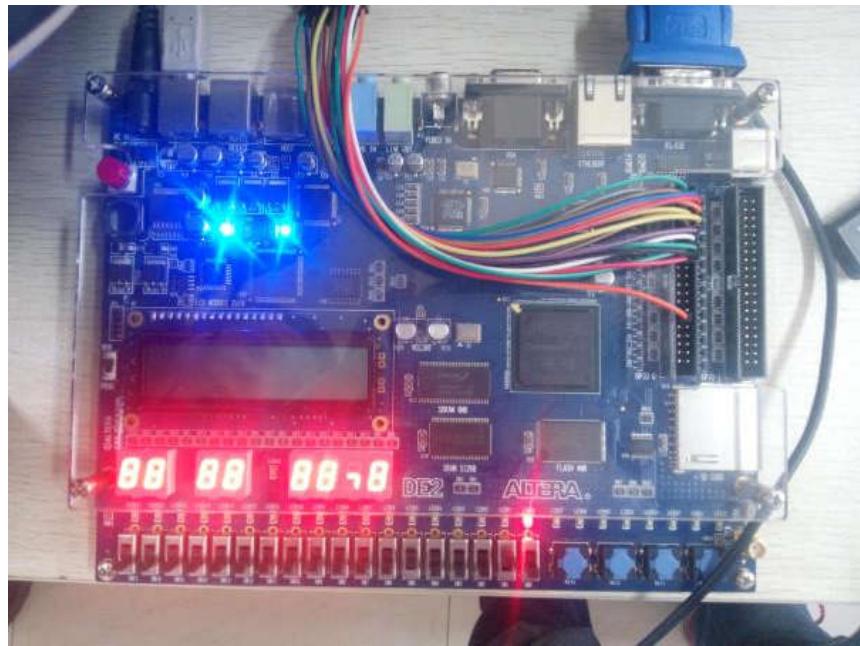


图 2-2 DE2 开发板图

3 模块设计及仿真结果

3.1 整体方案设计

本文提出的 485 接口电路主要是由 3 个大模块构成的：分频模块、接收模块、发送模块，而发送模块又是由 485 串口接收模块和 HDLC 封装数据模块完成，接收模块又是由 485 串口发送模块和 HDLC 解封装模块完成，如图 3-1 所示。

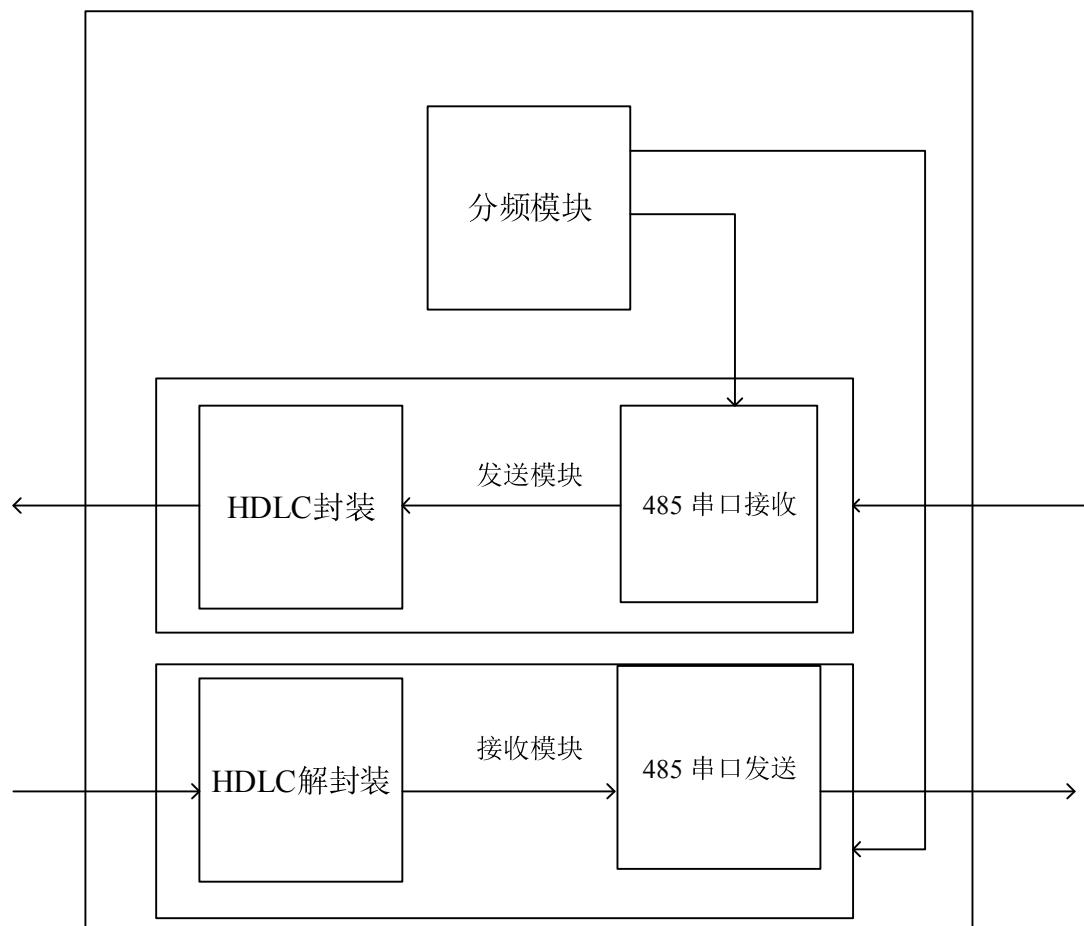


图 3-1 整体模块示意图

由于 RS-485 是一个半双工通信的，所以在设计的时候是有一个控制发送接收的信号的，但是由于验证的时候需要有发送的数据，所以在验证的时候设定是第一个数据接收完成之后就可以进行数据的发送，所以验证的时候是双向通信的。当接收模块接收数据完成之后，并将通过 HDLC 协议封装的数据帧发送成功之后，发送模块一直处于检测的状态，如果检测到有 0x7E 的出现，就进行接收数据，然后接受到的数据帧进行处理，然后通过串口将数据发送出去^[5]。

上文说到分为 3 个大模块，而接收模块和发送模块又分为了几个小模块。接收模块主要完成从串口接收数据、完成 CRC 校验、插入“0”码及数据串行发送出去，而发

送模块主要完成的是接收符合 HDLC 协议的数据帧、删除“0”码、CRC 校验及串口发送数据^[1]。

同时在本文设计时没有考虑到多基站，而是直接设计的点对点的通信。RS-485 是支持多设备的，本文在设计时考虑到每个基站的原理都是相同的，所以设计只考虑了点对点的通信。

而在 HDLC 协议中数据帧中是存在地址字段和控制字段的，由于本文设计的是一个点对点的通信，所以设计中把地址字段和控制字段默认地址字段为 0x01，控制字段为全“0”。

3.2 整体电路设计

在 3.1 章节中，主要说明了设计的大致思路，这里对 3.1 章节的描述做以更加深入的描述。设计中比较重要的模块就是发送模块、接收模块，这里先介绍发送模块：发送模块要完成的内容就是从串口接收数据、进行 CRC 验证、插入“0”码、串行将数据发送出去，如图 3-2 所示。

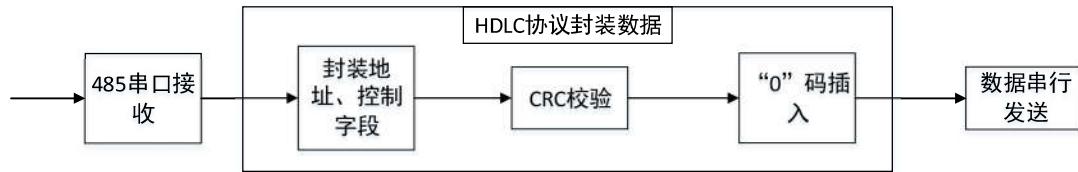


图 3-2 发送模块示意图

再介绍接收模块，接收模块主要完成的接收符合 HDLC 协议的数据，删除“0”码，CRC 校验、从串口发送，如图 3-3 所示。

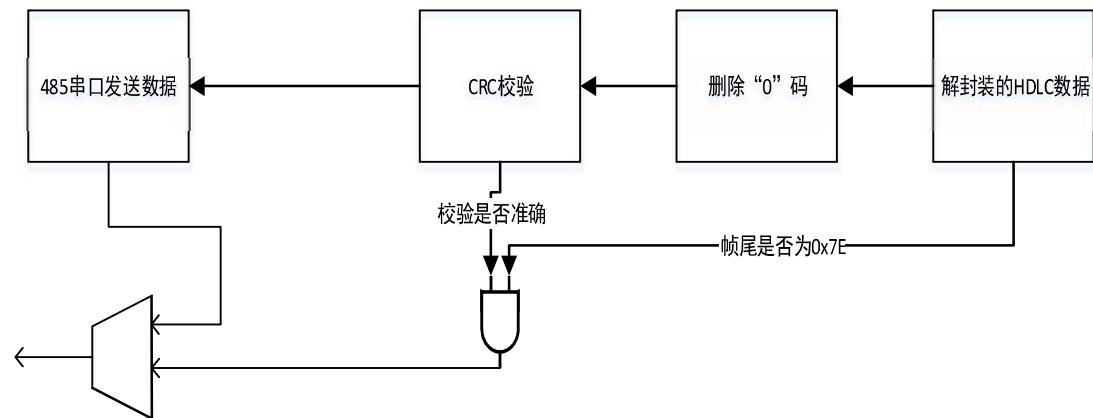


图 3-3 接收模块示意图

整体模块就是需要对上文的发送模块、接收模块连接起来。而在设计中使用的 25MHz 的时钟，FPGA 开发板上是 50M 的时钟，所以还需要用到一个分频模块，完成 25MHz 时钟的产生。在完成第一次发送之后，发送模块开始接收符合 HDLC 协议的数据帧的数据，然后进行检测，然后完成串口发送数据。

3.3 模块划分及仿真

设计在设计的时候主要考虑到设计一个点对点的通信，所以设计主要是划分为两个部分的，一个是发送模块，一个是接受模块。而这两个部分可以通过协议等划分为 HDLC 协议和串口部分两个部分，同时还需要有分频等模块协助从而实现整个电路功能。

HDLC 协议部分需要完成 CRC 校验、“0”码删除、“0”码插入、HDLC 发送封装数据、HDLC 接收封装数据。串口部分需要完成串口数据接收、串口数据发送。下面主要介绍上面提到的这 7 个模块。

3.3.1 485 串口部分

在笔记本电脑上现在是没有串口的，因此在调试的时候一般都会采用 USB 转串口的方案虚拟一个串口供笔记本使用。

a. 485 串口数据接收模块

在进行串口数据接收模块的设计的时候，在设计中假定了数据传送的波特率为 9600（也就是一秒发送 9600 位数据）。在接收数据的时候是串行的，也就是一位接一位的，对串口发送过来的数据进行检测，检测到有一个低电平的时候，开始进行接收数据，直到接收数据完成。

在设计接口之初，假定系统的时钟是 25MHz 的，所以串口发送一位数据的时候，FPGA 上都需要一个计数器进行计数： $25000000/9600 = 2604$ 次。在这个计数器计算到中间，也就是 1302 的时候，是最佳的采样时间，在这个时候，串口传送过来的数据的电平，就是需要记录下来的，如图 3-4 所示，是在串口接收模块的流程图。

串口数据接收模块的端口定义如表 3-1：

表 3-1 串口数据接收模块端口定义表

端口	I/O	位宽	端口说明
clk	in	1	时钟
rst	in	1	复位信号
rx	in	1	串口发送过来的数据
rx_vld	out	1	标志是否完成这个符合格式的数据的接收
rx_data	out	8	接收到的数据

如表 3-1 所示，是该模块的端口定义表，在到达采样时间的时候，开始采样 rx 的值，如果 rx 的值为 0，就开始进行接收，直到接收位数的计数器为 9 的时候，结束计数，得到接收到的数据，如图 3-5 所示，是串口数据接收模块在 ModelSim 工具下的仿真结果图。

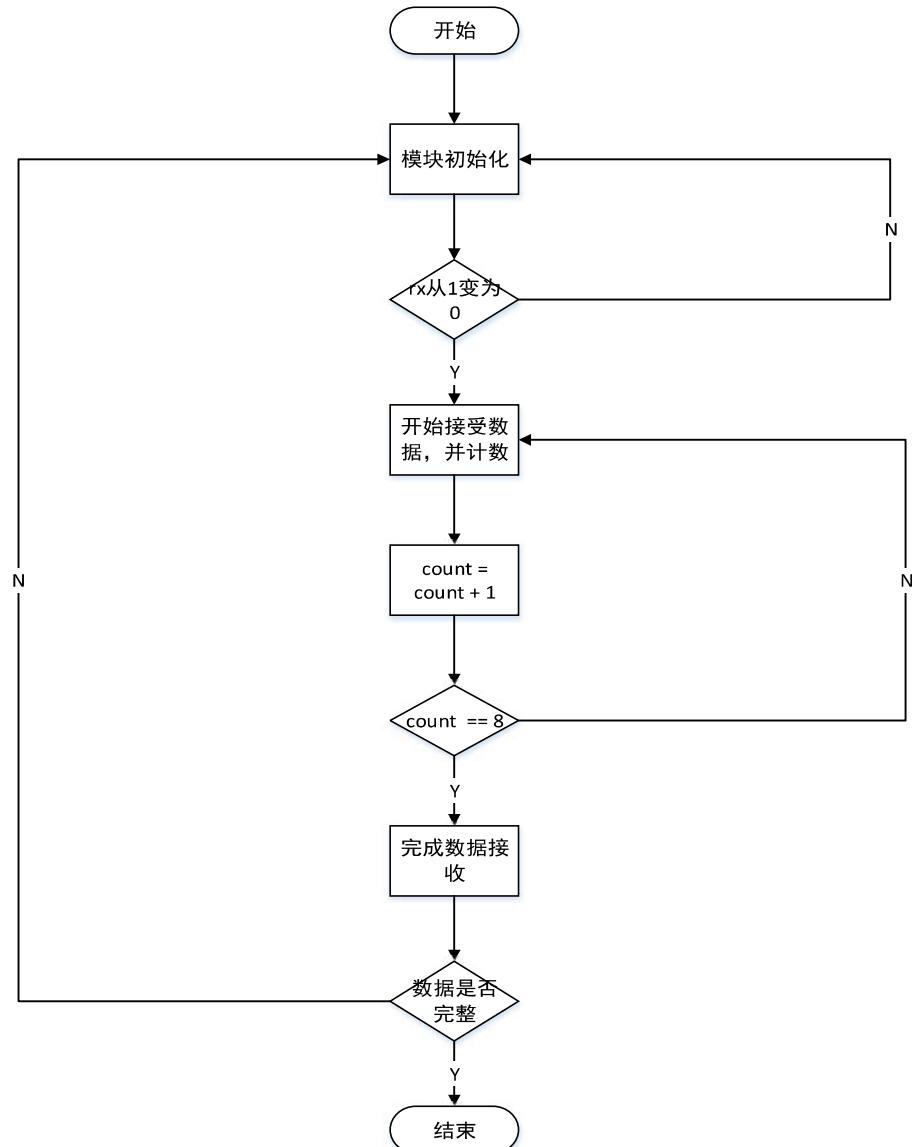


图 3-4 串口接收数据流程图

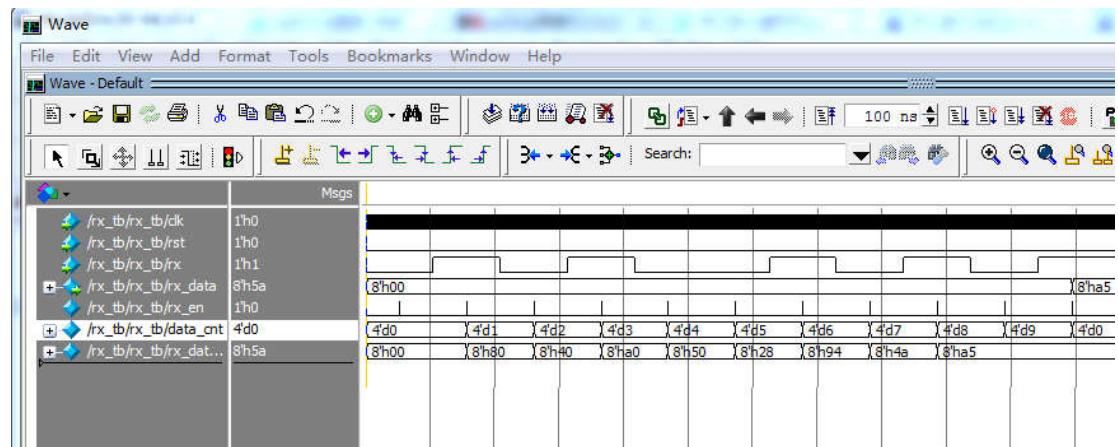


图 3-5 串口数据接收模块仿真结果图

b. 485 串口数据发送模块

串口发送模块就是将一个 8bit 的数据通过串口发送出去，同样的，这里假定波特率为 9600，而在发送数据的时候还需要发送 1bit 0，1bit 的校验位，然后开始拉高电平。这里也需要一个计数器，来确定采样时间，然后通过输入来确定，什么时候开始进行传输数据，然后再通过一个传送位数的计数器，确定传输什么数据，该模块的流程图如图 3-6 所示。

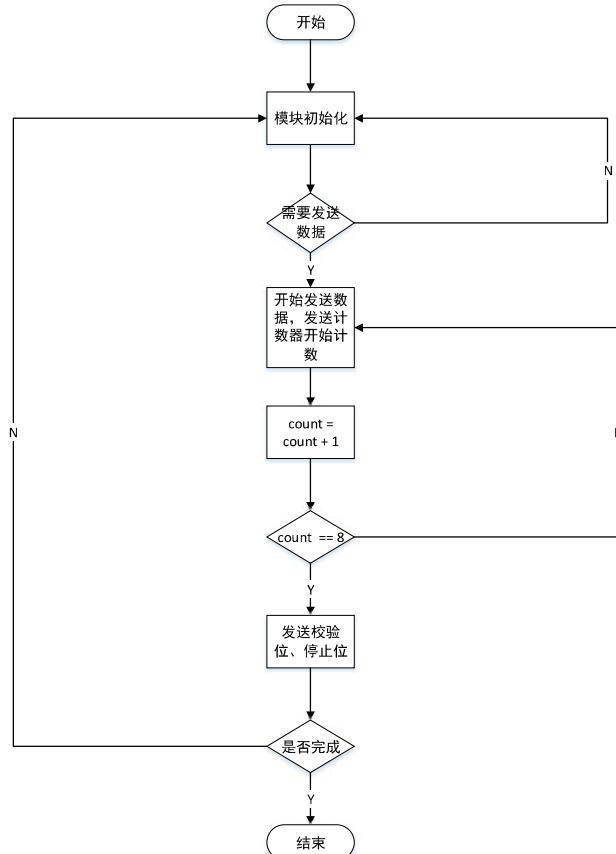


图 3-6 串口数据发送模块流程图

串口数据发送模块的端口定义如表 3-2 所示。

表 3-2 串口数据发送模块端口定义表

端口	I/O	位宽	端口说明
clk	in	1	时钟
rst	in	1	复位信号
tx_vld	in	1	是否进行传输的标志位
tx_data	in	8	要进行传输的数据
tx	out	1	传输出去的结果
txrdy	out	1	是否传输完成
is_busy	out	1	是否正在传输的标志位

如表 3-2 所示是串口数据发送模块的端口定义表，在 tx_vld 有效的时候，开始进行传输位数的计数，然后在采样时间进行发送，直到传输位数的计数为 10 的时候，完成传输。在这个过程中，is_busy 是高电平的，标志正在传输。图 3-7 是串口数据发送模块的仿真结果图：

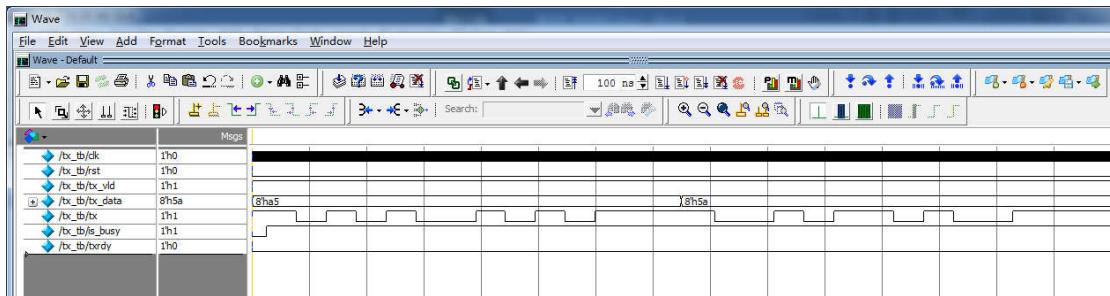


图 3-7 串口数据发送模块仿真结果

3.3.2 HDLC 协议部分

由于 485 标准本身只是规定了硬件层的相关标准，并没有规定如何去传输数据，所以在传输数据的时候还需要借助一些高级协议，如 Modbus 协议，HDLC 协议等等。在考虑到 HDLC 协议在硬件实现上是相对比较简单的，因此在本设计中就采用了 HDLC 协议。

HDLC 协议在设计中是相对比较重要的部分，由于 HDLC 的功能中比较重要的部分就是 CRC 校验、“0”码插入、“0”码删除、帧头和帧尾的封装等，而帧头和帧尾的封装^[6]，就不用再单独用一个模块去描述，剩下的三个功能都需要使用一个模块完成。在 HDLC 协议处理完成之后还需要进行数据的发送，而在接收模块的时候是需要解封装 HDLC 协议封装的数据帧的，所以还需要两个模块，因此在该模块总共有 5 个小模块。

a. CRC 校验模块

数据在传输的过程中，由于各种不确定的原因，可能会导致传输的数据发生错误，为了减少和避免上述错误的发生，常见的做法就是在要传输的数据中加入若干位的校验位，使得能有效的发现和纠正错误的数据。常见的校验算法有：奇偶校验、海明码校验、CRC 校验等。

由于 HDLC 协议在数据的校验部分使用的是 CRC 校验，所以就需要去实现 CRC 校验。CRC 校验码是一种典型的分组码，码组由 k 个信息码元和 $n - k$ 个检验码元构成，如图 3-8 所示，所以 CRC 码也称为 (n, k) 码^[7]。

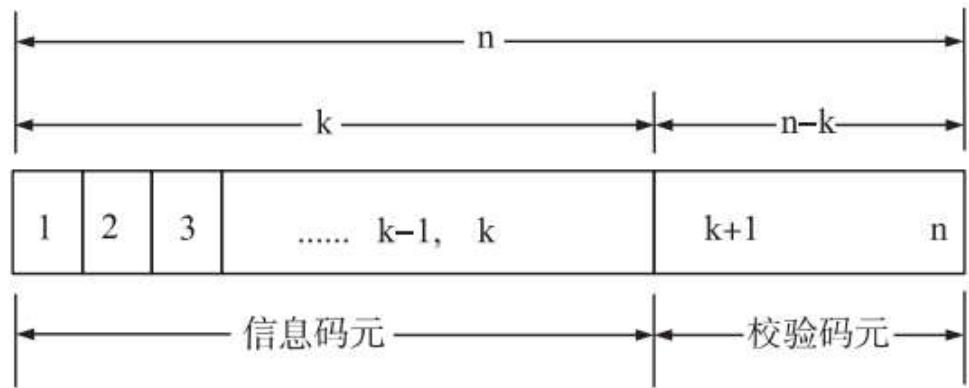


图 3-8 CRC 校验码示意图

上文介绍了 CRC 的相关基本概念，这里以一个例子为例说明 CRC 的计算过程，这里假设要传输的数据为 01111000，多项式为 $x^3 + x^0$ 。

$$\begin{array}{r}
 & & 1110 \\
 \hline
 1001) & 1111000 \\
 & \underline{1001} \\
 & & 1100 \\
 \hline
 & \underline{1001} \\
 & & 1010 \\
 \hline
 & \underline{1001} \\
 & & 110
 \end{array}$$

图 3-9 CRC 校验码计算示例

如图 3-9 所示,校验码就是 110,所以这里数据传输的时候就应该是 01111000 110。HDLC 协议中使用的是 CRC-CCITT 的多项式 $x^{16} + x^{12} + x^5 + x^0$,也就说明了在设计串路的时候使用的多项式就是 $x^{16} + x^{12} + x^5 + x^0$ 。

在 HDLC 协议中规定了数据帧中的地址、控制和数据字段需要进行 CRC 校验，而在设计中数据是从串口中得到的，所以这里要进行 CRC 校验的数据的位宽是 24 位，使用 CRC-CCITT 的多项式生成一个 16 位的 CRC 校验码。

使用 Verilog 进行描述 CRC-CCITT 的时候，如图 3-10 所示，是一个简单的电路示意图^[8]：

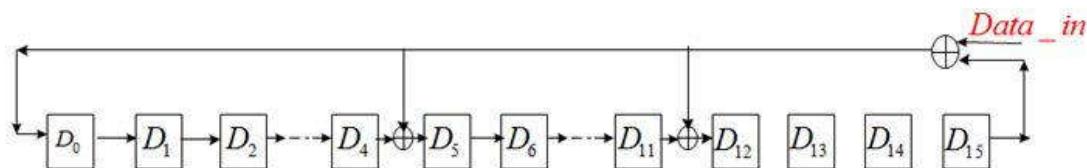


图 3-10 CRC-CCITT 电路示意图

在设计中使用 Verilog 进行描述电路的时候如果使用时序串行实现，则 24bit 的数据要进行移位 24 次，就需要 24 个 clk，效率低下，为了能在一个时钟周期内输出结果，就需要采用组合电路实现，这是一种以空间换取时间的方法，由于使用了 for 循环 24 次，所以电路的规模可能将会扩大 24 倍。

表 3-3 CRC 模块端口定义

端口	I/O	位宽	端口说明
clk	in	1	时钟
rst	in	1	复位信号
data_in	in	24	8bit 地址，8bit 控制码，8bit 数据
out_data	out	40	输入的 24 位数据拼接上 CRC 校验的结果

如表 3-3 所示，将串口得到的 8bit 数据和地址、控制字段进行拼接，然后交给 CRC 校验模块处理，CRC 校验得到的是 16bit 的校验码，然后将输入的 24bit 的数据和校验码拼接起来进行输出。如图 3-11 所示，是仿真结果。

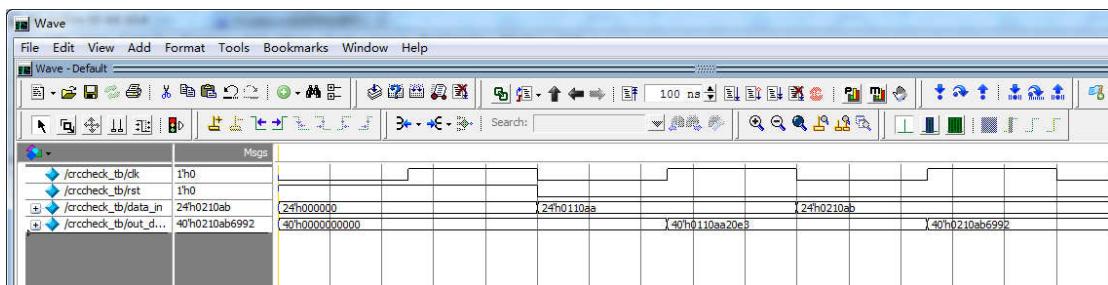


图 3-11 CRC 校验模块仿真结果图

如图 3-11 所示，分别输入了两个数 0x0110aa 和 0x0210ab，其 CRC 的结果分别为：0x20e3 和 0x6992。下面是用一款 CRC 的校验工具对这两个数分别做以校验，查看是否和设计中的结果一致。

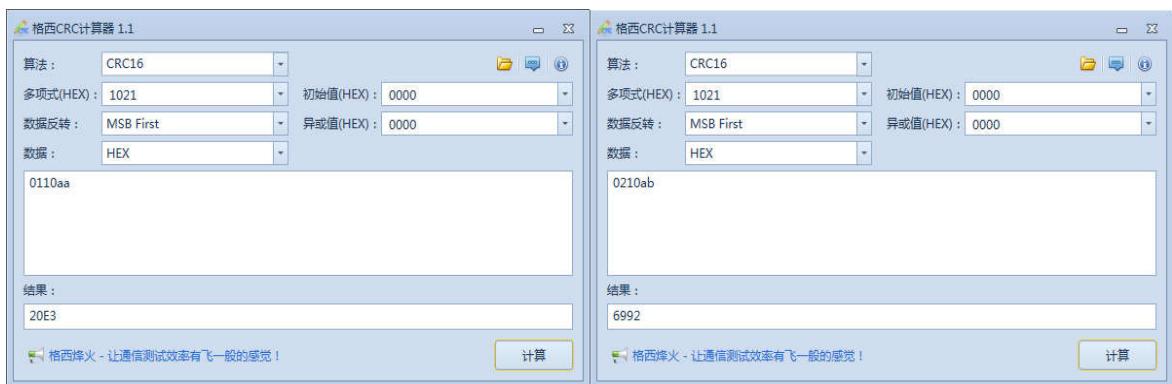


图 3-12 使用 CRC 校验工具进行计算

如图 3-12 所示，使用格西 CRC 计算器对上面的 0x0110aa 和 0x0210ab 分别计算 CRC16-CCITT 结果发现和设计仿真计算的结果一致，说明这里的设计没有问题。

b. “0”码插入

HDLC 的数据帧结构中，如果在帧头和帧尾之间的有效信息字段中，碰巧出现了帧头、帧尾标志字段(0x7E)一样的组合，那么就会误认为这就是数据帧的边界。为了避免这种情况的发生，HDLC 协议采用零比特填充法使一帧中帧头和帧尾两个字段之间不会出现 6 个连续的“1”。

在使用 Verilog 实现电路的时候，在想到需要进行零比特填充的数据位宽为 40 位，分别包括：地址字段、控制字段、数据字段、CRC-CCITT 校验字段四个部分，总共 40 位。在实现的时候使用一个 3 位的计数器进行计数，每一个“1”计数器就加 1，计数到 5 的时候就在其后插入一个“0”。

在进行设计的时候考虑到上一部分已经将数据转换成并行的了，所以这里就不再进行串行判断，而是直接将 40 位的数据一次性输入，然后去判断，遇到 5 个连续的“1”就在其后插入一个“0”，否则不进行插入，最终输出的为 48 位的一个数据。如果在插入的“0”的位数不是 8 位时，其他不足的位数补“0”实现，如表 3-4 所示是一个例子：

表 3-4 “0”码插入示例

原数据		插入“0”码后的数据	
十六进制	二进制	十六进制	二进制
0xFFFFF FFFFF	11111111111111111111111111 1111111111111111	0xFBEBE FBEBE	111110111110111110111110111 110111110111110111110
0xFA4A 52A53F	1111101001001010010100 101010010100111111	0x03E494 A54A7D	11111001001001010010100101 0100101001111101

上面大致介绍了“0”码插入的相关原理，下面就对模块的相关设计做以说明，在设计该模块的时候也是使用到了 for 循环实现的，同样的为了节省时间，使用了组合电路完成“0”码的插入，其中完成“0”码插入的代码如下所示：

```
always @(data_in) begin
    flag = 3'd0;
    data = 48'd0;
    for(i = 39; i >= 0; i = i-1) begin
        if (data_in[i] == 1) begin
            flag = flag + 1;
            if (flag == 3'd5) begin
                data = {data, 2'b10};
                flag = 3'd0;
            end
        end
    end
end
```

```

    end

    else begin
        data = {data, data_in[i]};
    end

    else begin
        flag = 3'd0;
        data = {data, data_in[i]};
    end
end
end

```

表 3-5 是“0”码删除模块的端口定义：

表 3-5 “0”码删除模块端口定义

端口	I/O	位宽	端口说明
clk	in	1	时钟
rst	in	1	复位信号
data_in	in	40	8bit 地址，8bit 控制码，8bit 数据，16 位 CRC 校验
out_data	out	48	插入“0”码后的结果，如果不够 8 个“1”剩下的用“0”补齐

仿真结果：

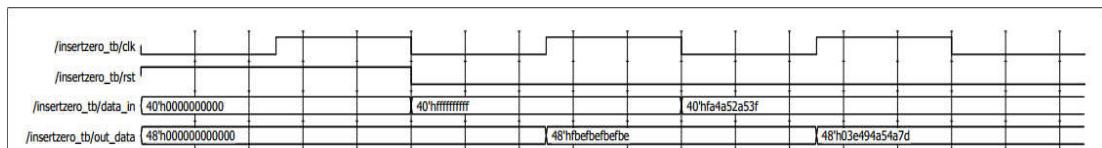


图 3-13 “0”码插入模块仿真结果

c. “0”码删除

“0”码删除是在接收数据的时候，删除掉在发送数据的时候插入的“0”码，从而还原数据。“0”码插入和“0”码删除的原理是相通的，所以这里就不再贴该模块的 verilog 代码，如表 3-6 所示是“0”码删除模块的端口定义，图 3-14 是“0”码删除模块的仿真结果。

仿真结果：

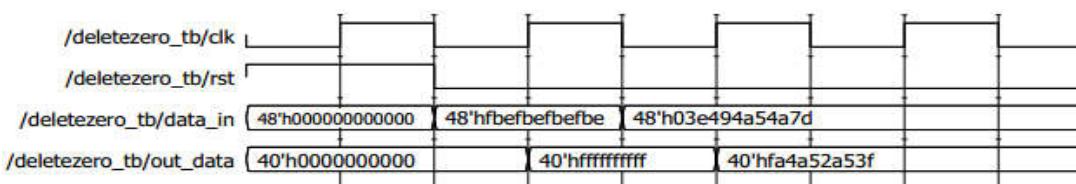


图 3-14 删减“0”码模块仿真结果

表 3-6 “0”码删除模块端口定义

端口	I/O	位宽	端口说明
clk	in	1	时钟
rst	in	1	复位信号
data_in	in	48	“0”码插入的结果
out_data	out	40	8bit 地址， 8bit 控制码， 8bit 数据， 16 位 CRC 校验

d. HDLC 发送封装数据

接口的发送部分需要将收到的串口的数据通过 HDLC 协议^[9]将数据封装起来，然后串行发送出去，串口接收到一个 8bit 的数据，然后对其进行封装地址、控制码，然后进行 CRC 校验，插入“0”码，封装帧头和帧尾，然后进行发送。

在本模块，完成的就是将一个 64 位的通过 HDLC 协议封装好的数据串行发送出去，端口说明如表 3-7。

表 3-7 HDLC 串行发送数据端口定义表

端口	I/O	位宽	端口说明
clk	in	1	时钟
rst	in	1	复位信号
is_tran	in	1	标志是否进行传输
data	in	64	通过 HDLC 协议封装过的数据帧
tx	out	1	串行将数据发送出去

如图 3-15 所示，就是该模块的仿真结果图。

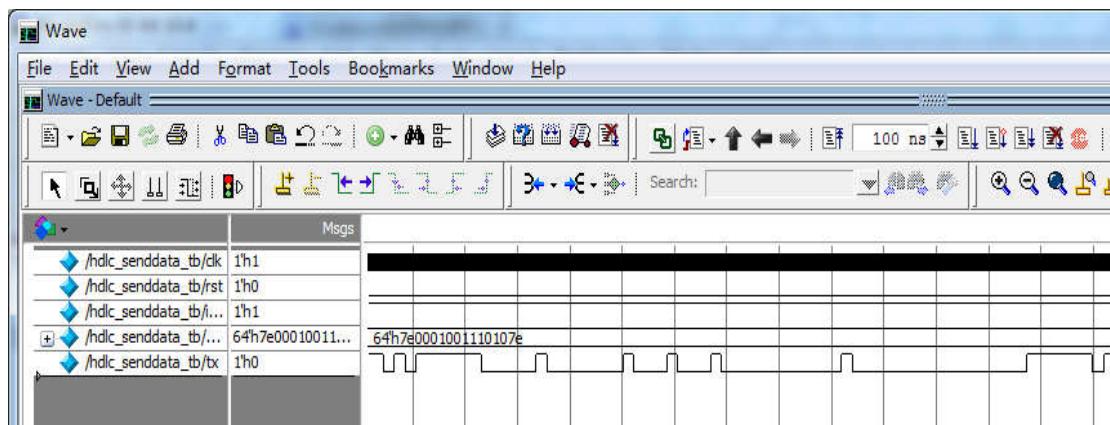


图 3-15 发送数据模块仿真结果图

e. HDLC 接收封装数据

接口的接收部分需要接受到 HDLC 封装过的数据帧，然后将其进行“0”码删除、CRC 校验，然后按照串口的格式将数据发送出去。

在接收数据的时候，需要先进行帧头的识别（也就是 0x7E 标志的识别），在这里识别的时候需要用到一个简单的状态机^[10]，如图 3-16 所示。其转换过程如下：

- a. 复位信号有效时，进入初始化状态 (s_count)。
- b. 当检测到当前输入为“0”时，进入 s0 状态。
- c. 当检测到当前输入为“1”时，进入 s1 状态。
- d. 当检测到当前输入为“1”时，进入 s2 状态。
- e. 当检测到当前输入为“1”时，进入 s3 状态。
- f. 当检测到当前输入为“1”时，进入 s4 状态。
- g. 当检测到当前输入为“1”时，进入 s5 状态。
- h. 当检测到当前输入为“1”时，标志检测到 0x7E，帧头标志位赋值为 1。

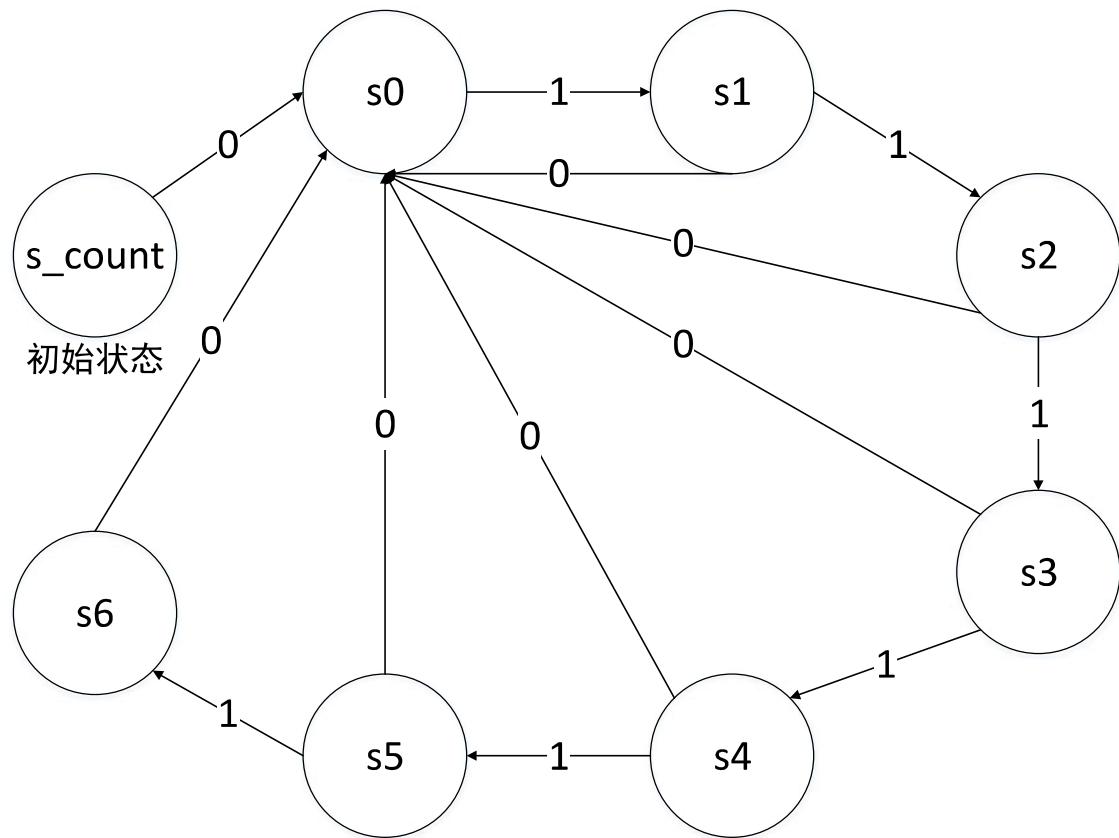


图 3-16 识别帧头标志状态迁移图

如表 3-8 所示，是状态迁移图中的每个状态的输入、下一状态以及是否检测到帧头/尾标志位的标志。

在识别到帧头的时候，就可以开始接收数据了，直到下一个帧头标志（也就是帧尾）结束接收数据，将接收到的数据发送出去进行删除“0”码、CRC 校验，然后将其中的数据部分通过串口发送出去，然后电脑接收。

表 3-8 状态迁移解释表

当前状态	输入	下一个状态	是否检测到帧头标志位
s_count	0	s0	0
s0	1	s1	0
s1	1	s2	0
s2	1	s3	0
s3	1	s4	0
s4	1	s5	0
s5	1	s6	0
s6	0	s0	1

在这个处理的过程中，可能会有些错误的出现，所以这里还需要对错误进行处理，如每帧数据都是 64 位的，帧头和帧尾都是 0x7E，所以就需要对帧尾进行检测是否为 0x7E，如果不是，就是数据帧出错；还有一个可能出错的地方就是 CRC 校验的时候，如果 CRC 校验出错，可能就是数据传输出了问题。除了这些问题，在接收的时候还需要判断地址是否为当前基站的地址，尽管本设计中只是设计的只是一个点对点的通信，但是 485 是可以多基站的，所以必须进行基站地址的判断。

表 3-9 接收数据模块端口定义

端口	I/O	位宽	端口说明
clk	in	1	时钟
rst	in	1	复位信号
rx	in	1	输入的数据
is_send	in	1	标志当前是属于发送状态还是属于接收状态
out_data	out	64	将接收到的数据封装起来，共 64 位，8bit 帧头，8bit 地址，8bit 控制，8bit 数据，16bitCRC 校验码，8bit 帧尾
is_receive	out	1	是否接收，标志是否处于接收状态

图 3-17 是该模块的仿真结果图：

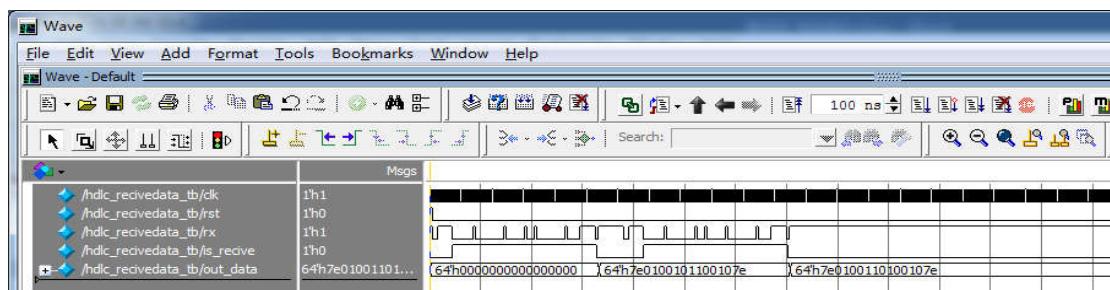


图 3-17 HDLC 接收数据仿真图

3.4 验证方案设计

在设计验证方案的时候，设计了一个点对点的通信，借助 DE2 FPGA 开发板上的 RS-232 接口完成方案的验证。设计中就是利用一个接收模块和一个发送模块，然后通过串口发送数据，然后接收，处理，最后再将数据发送到串口，这个过程需要使用到串口调试助手协助完成，其原理框图如图 3-18 所示：

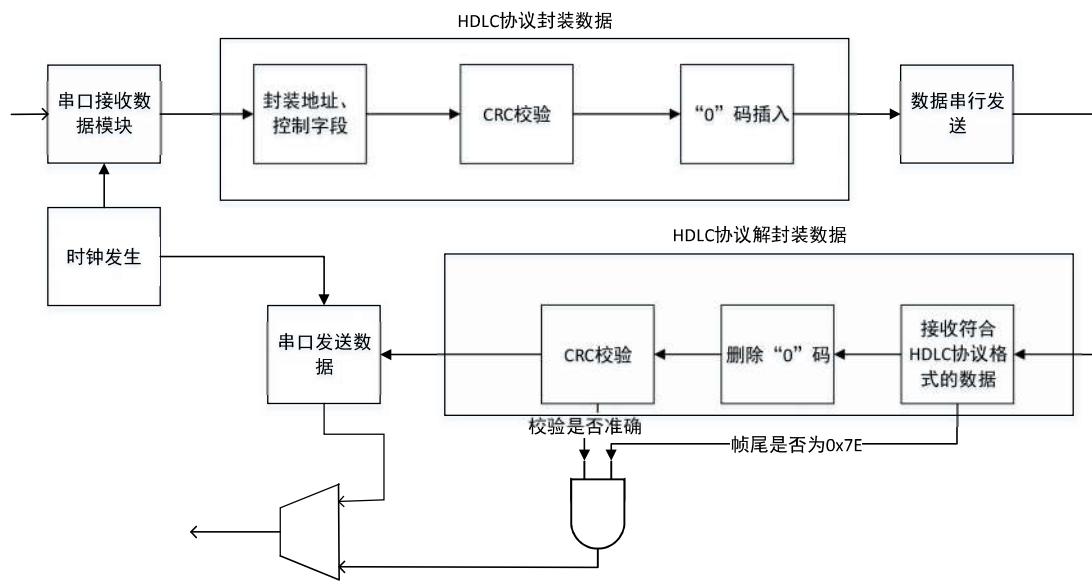


图 3-18 验证方案模块图

3.5 验证方案仿真

3.5.1 使用 Quartus 进行综合设计

在代码完成仿真调试之后，需要进行板级调试，在综合 Verilog 代码的时候，选用的是 Altera 公司的 Quartus 工具，其综合过程如下：

a. 新建工程

在打开 Quartus 之后，点击 File->New->New Quartus II Project，输入工程名，如图 3-19 所示：



图 3-19 新建工程示意图

b. 添加源代码

在新建工程完成之后，需要将已经写好的 Verilog 代码添加到工程中来，如图 3-20 所示：

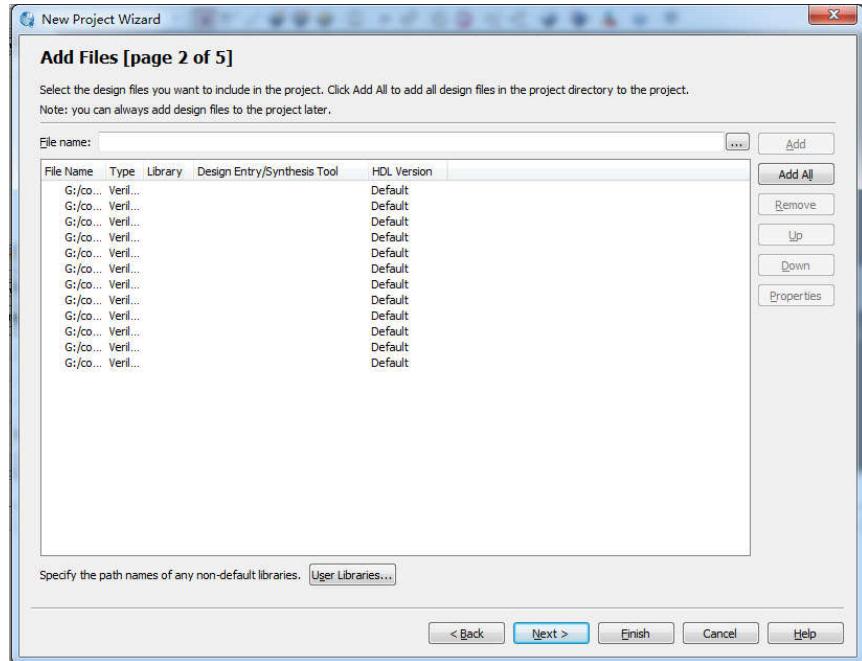


图 3-20 添加代码步骤示意图

c. 选择开发板类型

代码添加完成之后，需要选择 FPGA 开发板的芯片类型，从而确定，要进行板级调试的设备，这里选用的是 DE2 开发板，芯片类型为：EP2C35F672C6N，如图 3-21 所示：

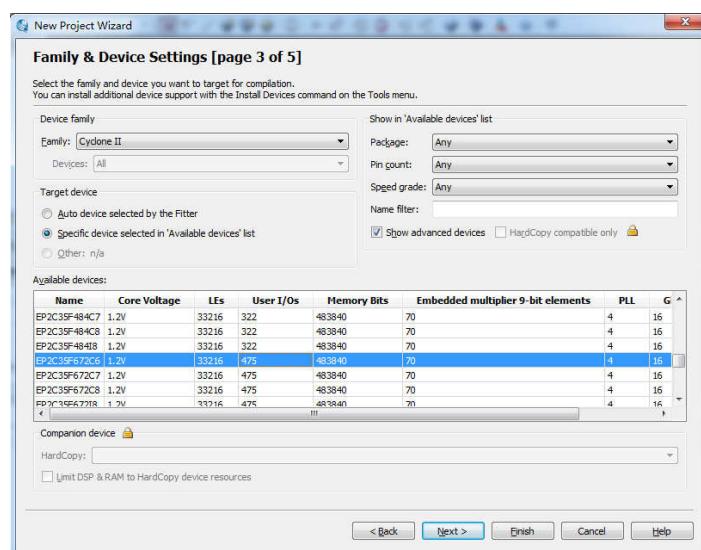


图 3-21 选择开发板类型示意图

d. 进行编译

在完成 FPGA 开发板的芯片选择之后，就可以点击 Finish 完成工程的创建了。在工程创建完成之后，对代码进行编译，确定是否可综合，如果不能综合，就进行修改相关的错误代码，直到代码可以综合。

在 Task 窗口中双击 Compile Design 进行编译代码，如图 3-22 所示：

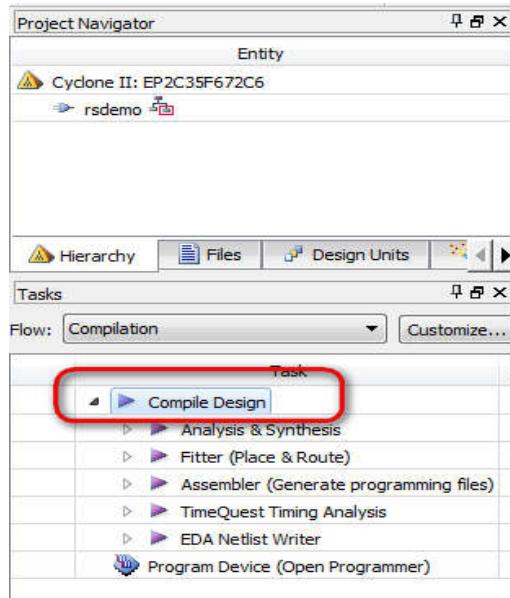


图 3-22 编译按钮示意图

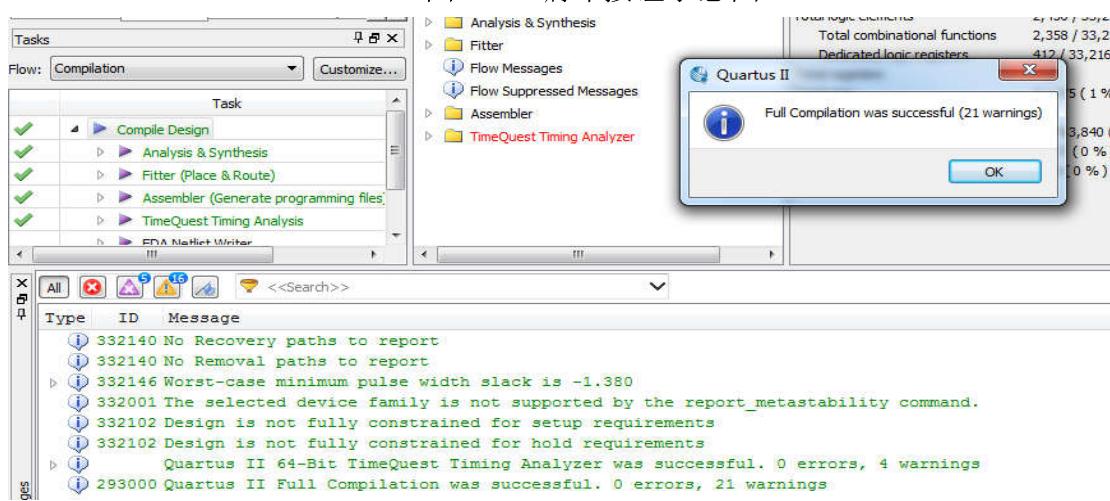


图 3-23 编译结果图

编译完成之后，在综合报告中可以看到电路的占用 FPGA 开发板的资源（LE 资源），电路的工作频率等信息，如表 3-9 所示：

表 3-10 电路的相关信息

项目	值
占用资源	2475 个(LE)
工作频率	34.63M Hz

表 3-9 是在 Quartus 的综合报告中看到的电路的相关信息，具体截图如图 3-24：

Flow Summary	
Flow Status	Successful - Fri May 20 15:49:54 2016
Quartus II 64-Bit Version	13.0.1 Build 232 06/12/2013 SP 1 SJ Web Edition
Revision Name	rs
Top-level Entity Name	rs_top
Family	Cyclone II
Device	EP2C35F672C6
Timing Models	Final
Total logic elements	2,475 / 33,216 (7 %)
Total combinational functions	2,400 / 33,216 (7 %)
Dedicated logic registers	412 / 33,216 (1 %)
Total registers	412
Total pins	6 / 475 (1 %)
Total virtual pins	0
Total memory bits	0 / 483,840 (0 %)
Embedded Multiplier 9-bit elements	0 / 70 (0 %)
Total PLLs	0 / 4 (0 %)

图 3-24 电路占用资源图

Slow Model Fmax Summary				
	Fmax	Restricted Fmax	Clock Name	Note
1	34.63 MHz	34.63 MHz	clkdiv:clk25 clk	

图 3-25 电路工作频率图

3.5.2 设置管脚

在代码完成综合之后，就需要进行下载到 FPGA 开发板进行调试了，在下载程序到 FPGA 开发板之前，是需要进行管脚的设置的，如图 3-26 所示，是在验证时候的管脚设置的示意图。

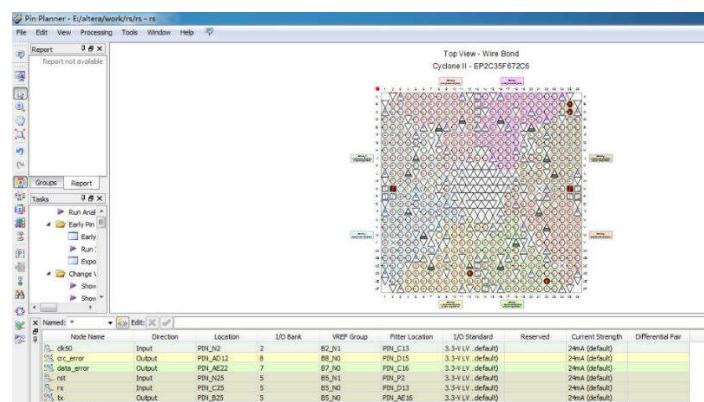


图 3-26 管脚分配示意图

3.5.3 下载程序到开发板

在完成管脚的分配之后，需要再次进行电路的综合，综合完成之后，点击 Tools->Programmer，进入下载程序到开发板的步骤，如图 3-27 所示。

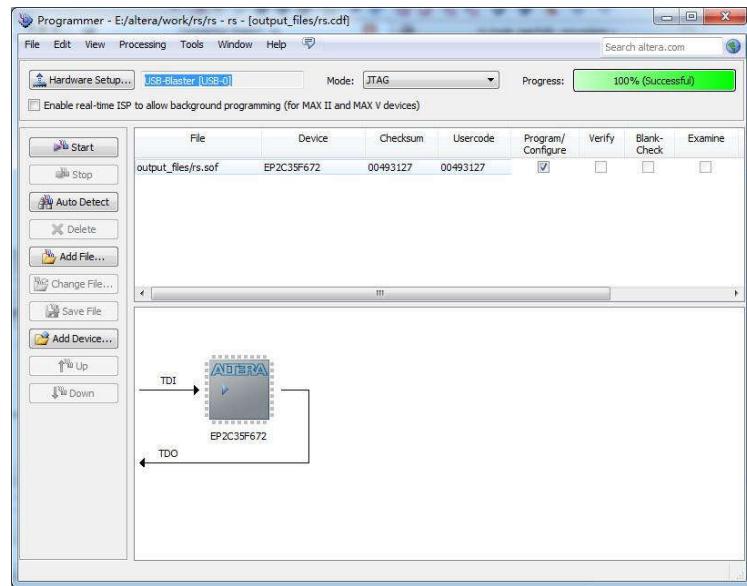


图 3-27 下载程序到 FPGA 开发板示意图

3.5.4 通过串口调试助手进行调试

完成下载程序之后，就需要用串口调试助手进行调试了，查看调试结果。在选用串口调试工具的时候，这里选用的是“友善串口调试助手”，如图 3-28 所示是调试结果。

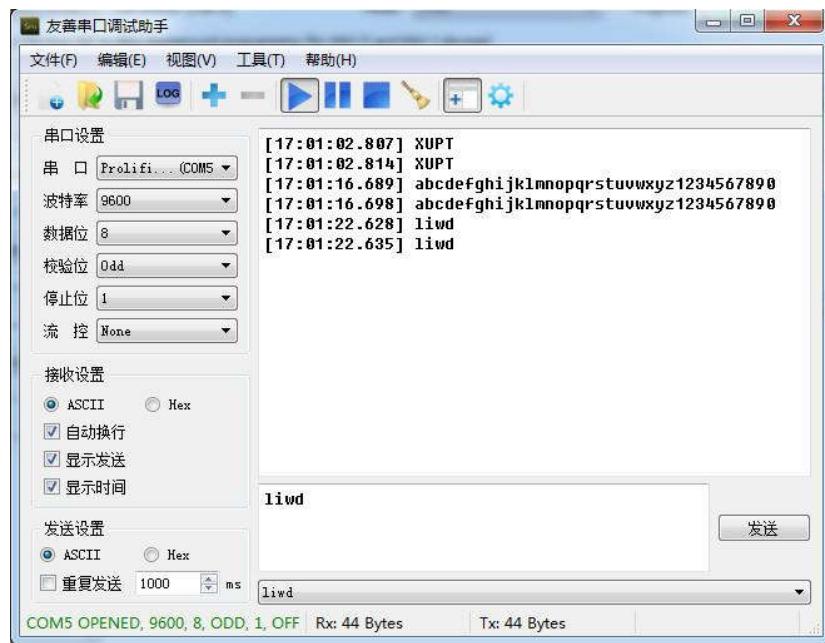


图 3-28 串口调试结果示意图

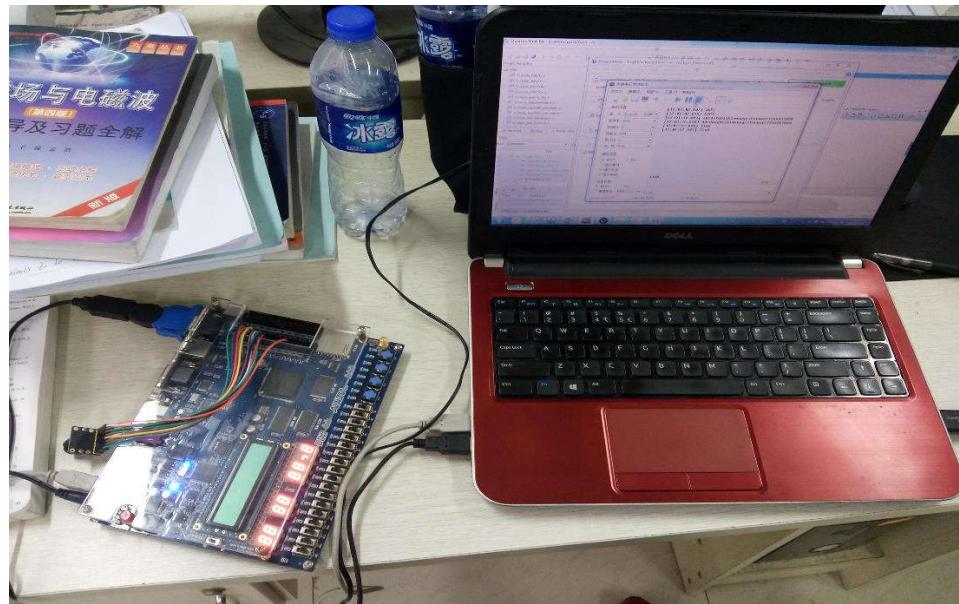


图 3-29 板级调试示意图

4 总结和展望

4.1 总结

本文的目的是用 FPGA 实现一个 485 通信接口，在实现接口的时候可以选择 Modbus 协议、HDLC 协议及自制协议等方案，由于 HDLC 而本文在实现接口的时候采用了 HDLC 协议，实现了一个接收模块和一个发送模块，接收模块完成了数据通过 HDLC 协议处理后串行发送，发送模块完成了数据接收后通过 HDLC 协议然后确定接收的数据。本文设计的接口电路具有如下的特点：自动完成插入“0”码、自动完成删除“0”码、串口的接收数据、串口的发送数据、自动进行 CRC 校验(CRC-CCITT)，自动生成和识别 HDLC 协议的帧标志（0x7E）等。

最终实现的接口电路通过 Quartus 综合后发现电路的速率最快可以跑到 34.63MHz，使用了 2475 个 (LE)。

4.2 展望

本文的设计还存在着许多不足的地方，如错误处理，这里在错误处理的时候只是将相关的错误标志拉高而并没有进行后续的操作，并且还是需要 64 个时钟完成数据接收后才判断的，而成熟的方案应该是需要及时判断，出错就结束这个数据的接收，而开始接收下一个数据。同时在设计之初没有做控制单元的设计，如发送或者接收的控制，地址的产生等。

4.3 心得体会

在完成本次毕业设计的过程中，身边同学不厌其烦的给我修改 bug 的过程，让我看到自己和同学之间还差了好多，如不够细心，没有耐心、毅力等缺点，让我能及时的更正过来，为我以后的工作打好了坚实的基础。

和老师的不断沟通才发现不论做什么东西沟通是非常重要的，遇到不懂的问题先在互联网上进行搜索，实在搜索不到就咨询身边的同学和老师。同时资料的查询能力也是非常重要的，不论是在学习或者以后的工作中，遇到的问题可能大多都是没有接触到过的，所以资料的查询能力是非常必要的，通过本次毕业设计也加强了我的资料的查询能力。

致谢

在我进行毕业设计和论文编写的过程中，首先要感谢刘宇老师，刘老师为我指定了题目，并且给我提供了相关的参考论文，在我进行方案设计的过程中刘老师不断的修改的我的方案，直到可以正常的进行下一步编写 Verilog 代码，让我甚是感激。

同时在毕业设计过程中，要感谢西邮 FPGA 创新实验室，为我提供了做毕业设计的场所，同时提供了 FPGA 开发板，从而我的毕业设计才能得以进行下去。

最后感谢我的同学，在我做毕业设计的过程中，每次遇到问题，他们都会详细的给我讲解，直到问题解决。

参考文献

- [1] 高振斌, 陈禾, 韩月球. HDLC 协议 RS-485 总线控制器的 FPGA 实现. 河北工业大学学报, 2004 Vol.33 No.5:29~32
- [2] 郑国灿等. RS-232、RS-422 与 RS-485 标准及应用技术. 中国电影电视技术学会学术研讨会, 2002
- [3] 陈金华. 基于 HDLC 协议的 RS-485 通信设备的研制. 测控技术, 2010 Vol.29 No.6:98~101
- [4] 李新兵. 基于 ARM9 的软核处理器设计（基于 FPGA）. 北京: 机械工业出版社 Report.
- [5] 梁士龙, 王力男, 杨嘉伟. 用 FPGA 实现 RS-485 通信接口芯片. 系统工程与电子技术, 2002 Vol.24 No.3:103~106
- [6] Armaan Hasan Nagpurwala, Sundaresan C, Chaitanya CVS. Implementation of HDLC Controller Design using Verilog HDL. International Conference on Electrical, Electronics and System Engineering, 2013 7~10
- [7] 张炎, 任勇峰, 齐蕾, 姚宗. 基于 FPGA 的 CRC 校验算法的实现. 2015 Vol.38 No.1:222~226
- [8] Jeng-Liang Rsai, Jie Zhang, Jui-Ning Cheng. HDLC Controller Design ECE 551 Final Project
- [9] 王雅荣, 鲍民权, 邱智亮. HDLC 协议控制器 IP 核的设计与实现. 集成电路应用, 2006 33~36
- [10] 夏宇闻. Verilog 数字系统设计教程. 北京: 北京航空航天大学出版社