北京航空航天大學
BEIHANG UNIVERSITY

实验二、串口程序设计——串口收发

2018 年 12 月 12 日 （版本 v01）

序号	组长标记	教学班	学号	姓名	签名（无手签无效）
1	*	162321	16231275	刘瀚骋	
2					
3					
4					
提交日期	2019-01-18		签收：	评价日期	
评阅 1	评阅 2		评阅 3	评阅 4	平均（百分制）

高等理工学院

目录

实验四、串口程序设计——串口收发实验（实验指导书部分）

1 实验指导

通过对串口协议的了解，利用 EELAB-FPGACORE2 实验硬件实验平台（以下简称“实验板”）通过 Verilog HDL 语言实现基本的串口收发代码，加深对组合逻辑设计和时序逻辑设计理念的认识。

1.1 背景知识

1.1.1 串口协议

串口是计算机上一种非常通用的设备通信协议。大多数计算机包含两个基于 RS232 的串口。串口同时也是各仪器必备接口，串口通信协议也可以用于获取远程采集设备的数据。

串口通信的概念较简单，串口按位（bit）发送和接收字节。尽管比按字节（byte）的并行通信慢，但串口可在使用一根线发送数据的同时使用另一根线进行接收数据，此外差分模式的编码还可以进一步减少干扰，以提高传输速率。典型地，串口采用 ASCII 码字符进行传输，硬件上一般包括地线、发送引脚和接收引脚。串口通信属于异步通信，可在接收数据的同时进行数据发送。串口通信最重要的参数包括比特率、数据位、停止位和奇偶校验。对于两个进行通信的端口或设备，其配置参数必须匹配，相关参数如图 1 所示。

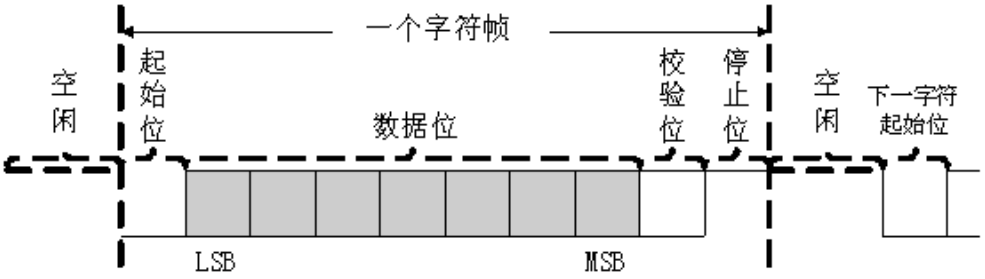

图 1 串口通信时序图

在图 1 为串口通信中发送或接收的一个数据帧，包括起始位、数据位、奇偶校验位和停止位等，详细如下：

1）比特率：衡量通信速度的参数，表示每秒钟传送的 bit 的个数，例如 300bps 表示每秒钟发送 300bit。在协议中，时钟周期一般指比特率，例如：如果协议需要 4800bps，那么时钟是 4800Hz 也表示数据线上的采样率为 4800Hz。

2）数据位：这是衡量通信中实际数据位的参数。当计算机发送一个信息包，实际的数据不总是 8

位，可能是 5、7 或 8 位，与传送的信息相关。比如，标准的 ASCII 码是 0～127（7 位）。扩展的 ASCII 码是 0～255（8 位）。如果数据使用简单的文本（标准 ASCII 码），那么每个数据包使用 7 位数据。每个包（一个字节）包括开始/停止位，数据位和奇偶校验位。

3）停止位：用于表示单个包的最后一位。典型的值为 1，1.5 和 2 位。由于串口的两端可能由两种不同的时钟驱动，导致数据传输过程中通信不同步。因此停止位不仅表示传输的结束，并且提供控制器校正时钟同步的机会。通常停止位位数越多，不同时钟同步容忍程度越大，但数据传输率变慢。

4）奇偶校验位：在串口通信中一种简单的检错方式。有四种检错方式：偶、奇、高和低。对于偶和奇校验的情况，串口会设置校验位（数据位后面的一位），用一个值确保传输的数据有偶个或奇个逻辑高位。例如，如果数据是 011，那么对于偶校验，校验位为 0，保证逻辑高的位数是偶数个。如果是奇校验，校验位为 1，这样就有 3 个逻辑高位。高位和低位不真正的检查数据，简单置位逻辑高或者逻辑低校验。这样使得接收设备能够知道一个位的状态，有机会判断是否有噪声干扰了通信或者是否传输和接收数据是否不同步。

1.1.2 RS-232

RS-232（ANSI/EIA-232 标准）是 IBM-PC 及其兼容机上的串行连接标准。可用于许多用途，比如连接鼠标、打印机或者 Modem，同时也可以接工业仪器仪表。用于驱动和连线的改进，实际应用中 RS-232 的传输长度或者速度常常超过标准的值。RS-232 只限于 PC 串口和设备间点对点的通信。RS-232 串口通信最远距离是 50 英尺。如图 2 为串口一般接口。

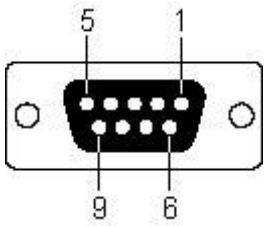

图 2 DB-9 针串口接口

图 2 中只显示出串口接口的大致，根据图中针脚排序，RS-232 针脚的功能：

1）数据：

TXD（pin 3）：串口数据输出(Transmit Data)

RXD（pin 2）：串口数据输入(Receive Data)

2）握手：

RTS（pin 7）：发送数据请求(Request to Send)

CTS（pin 8）：清除发送(Clear to Send)

DSR（pin 6）：数据发送就绪(Data Send Ready)

DCD（pin 1）：数据载波检测(Data Carrier Detect)

DTR（pin 4）：数据终端就绪(Data Terminal Ready)

3）地线：

GND（pin 5）：地线

串口连线示意图：

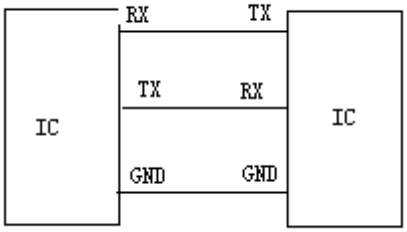

图 3 UART 连线示意图

在实际的电路中，一般只连接 RXD、TXD、电源线（VCC）和地线（GND），其他引脚可以悬空。

1.1.3 FIFO

FIFO，英文全称 First In First out，是一种先进先出的数据缓存器，与普通存储器相比，其区别是不需要外部写地址线，因此 FIFO 使用过程较简单，但由于只能顺序写入地址线，按顺序读出数据，其数据地址由内部读写指针自动加 1 完成,而普通存储器可由地址线决定读取或写入某个指定的地址。FIFO 可分为同步 FIFO 和异步 FIFO，其区别在于读写时钟是否为同一时钟，若采用时钟为同一个则为同步 FIFO，否则为异步 FIFO。本实验为简化实验难度，采用同步时钟实现一个 FIFO，其模块可包括：

1）FIFO_IN：输入数据；

2）FIFO_OUT：输出数据；

3）FIFO_FULL：写数据满标志位，fifo 写满置 1；

4）FIFO_EMPTY：fifo 空标志位，空时置 1；

5）FIFO_HALF：fifo 写数据达到 8 个，或读数据时，fifo 数据小于 8 个；

6）write：写数据使能信号，高电平有效；

7）read：读使能信号，高电平有效；

8）clock：时钟信号。

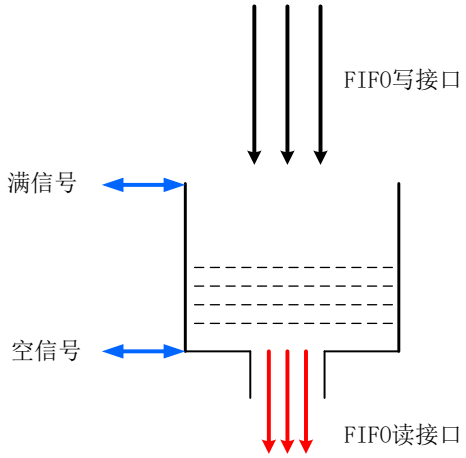

图 4 UART 连线示意图

上图所示为 FIFO 的模型，可以看做一个漏桶模型，其中标出输入、输出、满信号、空信号、可编程满等信号可用于在设计 FIFO 时进行参考。对于空满标志的判断，当读指针等于写指针加 1，且同时有一个写使能，此时认定 fifo 已满。空标志也是同样的原理。

以上 FIFO 模块相关参考可按实际需求进行更改。

注意：读、写操作可同步发生。

1.2 实验串口电路

目前，由于现在笔记本端接口一般没有串口，为使开发方便，一般开发板上的串口都采用相关芯片转换为 USB 口，在实际使用过程中，需采用 USB 线与开发平台进行连接。经过开发板与 PC 端由 USB 线进行通信，但是串口协议如何转换为 USB 协议由图中 CP2102 负责，在编写串口程序时不用考虑。

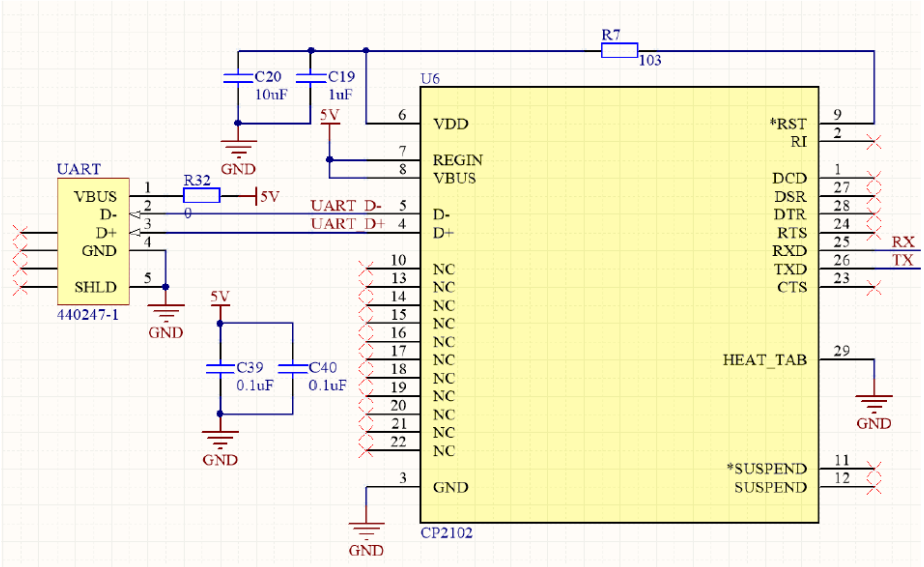

图 5 USB 转串口原理图

1.2 案例：串口收发程序

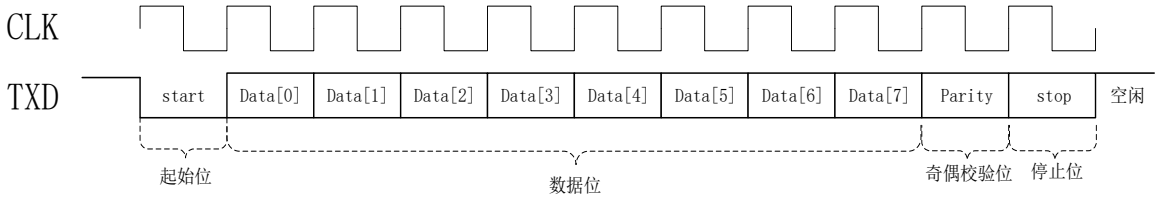

图 6 串口数据时序图

如图 6 中串口数据收发时序图，其中起始位（1 位，低电平）、八位的数据位、奇偶检验位（1 位）和停止位（1 位，高电平），需使用 Verilog 对起始位、数据位、奇偶校验位和停止位进行识别，在识别中需考虑不同波特率下每比特数据的时间宽度，通过每比特时间宽度设置每比特读取时间，由于开发板提供的时钟 Sys_CLK=50MHz，考虑在波特率为 9600bps 的传输下，每比特数据时间宽度为 1/9600s=100us，按照以下代码进行分频：

```
always@(posedge Sys_CLK)
begin
    if(Uart_CLK_Cnt <= 8'd161)
        Uart_CLK_Cnt = Uart_CLK_Cnt + 1'b1;
    else
    begin
        Uart_CLK = ~Uart_CLK;
        Uart_CLK_Cnt = 8'd0;
    end
end
```

那么 Uart_CLK = 0.15M，采用 0.15M 时钟提供给串口数据发送，那么需大约每 100us/6us=16 个串口时钟周期接收一个比特（其中 6us=1/0.15M）。因此对实验板提供的主频采用不同分频数值和设置相应的串口时钟周期个数可实现不同串口波特率的收发。

在实验板提供的 FPGAXC3_Test 测试程序中提供了串口收发程序，可在串口助手上进行测试，测试程序串口功能部分为 FPGA 实验板接收来自串口助手的数据并同时将接收的数据返回串口助手，如图 7 所示：

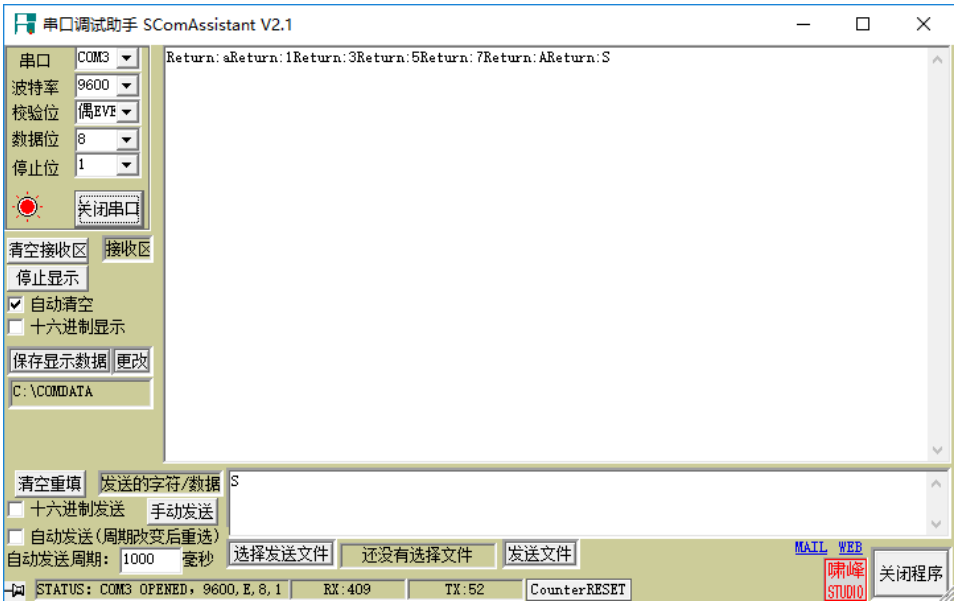

图 7 串口收发程序演示

1.3 基本实验要求

基本实验要求为：

a) 通过查阅串口协议，在开发板上实现基本的串口收发功能，具体包括在 PC 端通过串口助手发送数字 0-9 的编码，开发板接收到数字并返回显示到串口助手 return:x（x=0,…,9），若发送其他除数字以外字符，返回显示到串口助手 error，实现的串口配置为：

i）八位数据位；

ii）需具有校验位和 1 位停止位。

b) 添加数码管显示功能，要求发送到开发板的数字更新显示到数码管；

c) 学习 FPGA 中 FIFO 的使用，利用开发板实现一个基本的 FIFO，需满足：

i）当 FIFO 写满时，输出满标志位，写满后禁止向 RAM 写数据，防止数据混乱；

ii）当 FIFO 读空时，输出空标志位；

iii）合理利用板载资源，显示空满标志，以及当前 FIFO 存储数值的大小，FIFO 深度暂定为为 16Byte。

d) 将要求 c 中完成的 FIFO 模块添加到实验 a 的基础上，需要满足：存储串口助手十次发送的 8bit 数字编码，当存储到十个 8bit 数字编码以后，一并返回发送到串口助手上；

e) 对 FIFO 模块的功能进行仿真验证，典型的需要仿真半满、空、满，以及读写同步操作下的可

能发生的边界条件；

f) 对上述设计结果综合后的资源占用情况进行分析，了解或估算所消耗的可编程逻辑资源。

g) 修改波特率，实现其他波特率下的串口通信；

1.4 扩展要求

考虑优先级策略下的调度设计，假设 5~9 的优先级为高，0~4 的优先级为低，高优先级数值将先获得机会得到发送，而低优先级数值将等到高优先级数值发送完毕后才能得到发送，同一优先级的数字按照先进先出调度。一个简单的设计办法是采用两个 FIFO 分别进行 0~4 和 5~9 对应的数值编码的存储，在发送时优先发送 5~9 对应的 FIFO 里面的数值编码。

a) 完成上述优先级调度发送设计，并设计合适的案例，通过仿真检查设计结果的正确性。

b) 将设计代码综合后下载到 FPGA 中，修改波特率，利用串口通信工具检查设计结果的正确性，可以用两个数码管分别显示高低优先级 FIFO 队列存数数值的大小；

1.5 其它提示

在实验中心提供的样例程序中，UART 的样例源代码具有固定的 9600 波特率、8 位数据位、偶校验、1 位停止位，缺点是一个只能每次读写一个字节的；更好的 UART（注：更稳定可靠、波特率可配置、有 Telnet 等上层协议）需要用户自行开发，或者在网上搜索更好的代码加以改造。

“实验板”的 I/O 接口和人机接口资源请参考《digiC2018 课程实验实验指导书(01)》。

实验四、串口程序设计——串口收发实验（实验报告部分）

2 实验报告

2.1 实验背景与需求分析

本次实验需要于 FPGA 上实现一个 8 位数据位，1 位校验位，1 位停止位的的 UART 控制器。并通过 PC 机实现回环数据传输。

本次实验完成了基本实验要求中的内容，由于时间有限，未能完成扩展试验。

2.2 系统设计

2.2.1 总体设计思路

首先，实现遵照 Avalon-ST 总线协议的 FIFO。接着通过状态机和相应的组合逻辑电路实现串口的控制（由时序电路控制数据发送的时序，组合逻辑主要用于生成奇偶校验位），并在串口控制器的出方向(TX)方向和入(RX 方向)分别接入两个独立的 FIFO；串口控制器应该分别轮询接入其出方和入方向的 FIFO。将 TX 方向 FIFO 的写入端和 RX 方向 FIFO 的读取端口引出，当需要接收数据时，从 FIFO 读取数据；需要发送数据时，从 FIFO 发送。

2.2.3 FIFO 模块

FIFO 模块的接口实现 Avalaon-ST[1]总线的规范。Avalon-ST 总线是 Altera 公司颁布的一款专门用于芯片级数据传输的标准，比较适合流式数据的传输。

FIFO 的出方向(也称 Source 方向)预留三个接口: ready/valid/data. Data 为数据总线，ready 相对 FIFO 模块为输入，表示接收者是否已经准备好接收数据，valid 相对 FIFO 模块为输出，表示当前数据总线上的数据是一个有效字节。当满足以下条件时:

1. ready 置位(接收机准备就绪)
2. valid 置位(数据总线上为有效数据)
3. 时钟上升沿

将发生一次数据的读取。

FIFO 的入方向(也称 Sink 方向)预留和出方向相同的三个接口，不同点在于，ready 信号相对 FIFO

模块为输出。在 FIFO 非满时，ready 必然置位表示 FIFO 可以随时接收数据写入。显然，ready 可用作 FIFO 的满标志位。其他接口含义和 Source 方向的一致。

FIFO 的内部使用头尾两个指针，读操作时尾指针加一，写操作头指针加一，实现一个简单的优先队列数据结构。

这部分的代码参见:

2.2.4 UART 模块

串口模块亦按照 Avalon-ST 规范实现。

串口模块的发送部分实例化一个 TX-FIFO，并在模块内部连接到该 FIFO 的 Source 部分。当 TX-FIFO 的 Source valid 置位时，发送部分的时序逻辑部分按照串口通信规范驱动串口的 TX 线。TX-FIFO 的 Sink 端口引出，当用户需要发送数据时，则通过该 Sink 口向 FIFO 内写入发送数据。

串口模块的接收部分实例化一个 RX-FIFO, 并在模块内部连接到该 FIFO 的 Sink 部分。接收部分的时序逻辑按照串口通信规范解码串口 RX 线上的数据，并在内部实现一个简单的 Avalon-ST Source，将该 Source 和 RX-FIFO 的 Sink 相连，当总线上有数据时，将数据移入 FIFO。RX-FIFO 的 Source 端口引出，当串口收到数据后，用户可以从 Source 中读取接收数据。

该模块的代码可以参见 Uart.v

2.2.5 校验位产生模块

校验位的产生使用纯组合逻辑实现。该模块的代码可以参见 Parity.v

2.3 功能仿真测试

2.3.1 测试程序设计

使用 Verilog 编写 Testbench，并在 Modelsim 中进行仿真。本次实验对两个重要的模块(串口模块，FIFO 模块)进行了仿真。并按照 1.3 基本实验要求中的第 e 项，仿真了 FIFO 半满、空、满，以及读写同步操作下的可能发生的边界条件。

2.3.2 功能仿真过程

2.3.2.1 FIFO 模块仿真

首先仿真半满的情况。向 FIFO 内连续写入若干数据(小于 FIFO 深度)，然后驱动 FIFO 的 Source 接口读取数据，测试连续读取以及间歇读取的情况，测试结果如图 1 FIFO 半满仿真结果所示。可见，FillLevel 信号，空信号可以正常工作，且数据能够按照期望的数据被读取。

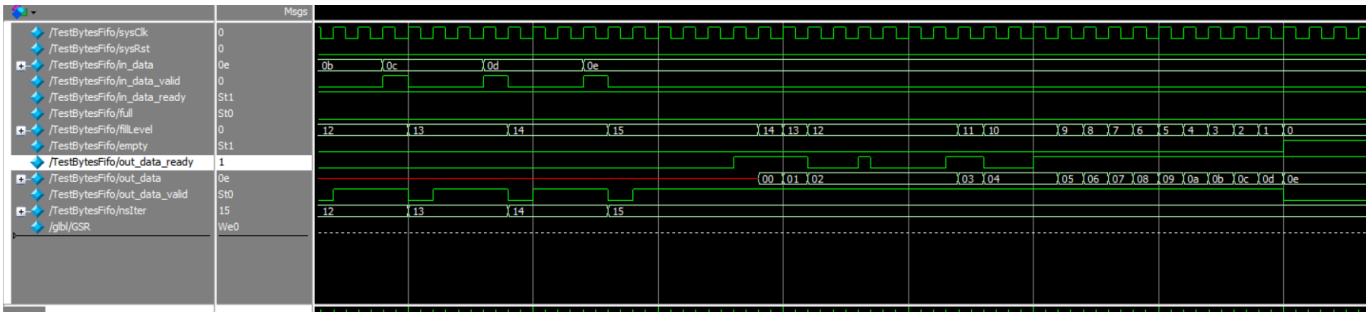

图 1 FIFO 半满仿真结果

接着测试了当 FIFO 写满时的情况，测试结果如图 2 FIFO 全满仿真结果所示。FIFO 写满后，Full 位被拉高，表示此时 FIFO 已经被写满；同时其 Sink 方向的 ready 位被拉低，阻塞了其他模块的读取操作。

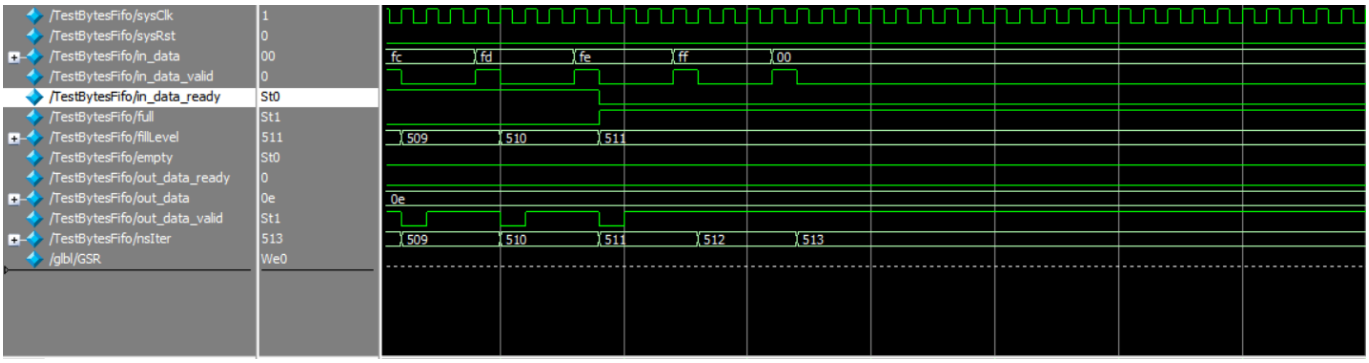

图 2 FIFO 全满仿真结果

最后测试 FIFO 同时读写的情况，仿真结果如图 3 FIFO 同时读写仿真结果所示。在实现 FIFO 时为了保证数据的一致性，读写操作会在 Source 端 Valid 信号的控制下，相差一个时钟周期，该延迟由一个 FIFO 内部的自旋锁产生，仿真结果可以证实这一点。此外从仿真结果可以看出，fillLevel，即 FIFO 的填充水平也在 0 和 1 间跳动，表示读写同时有效的情况下，写入一个字符后，该字符会立马被读取，符合设计期望。

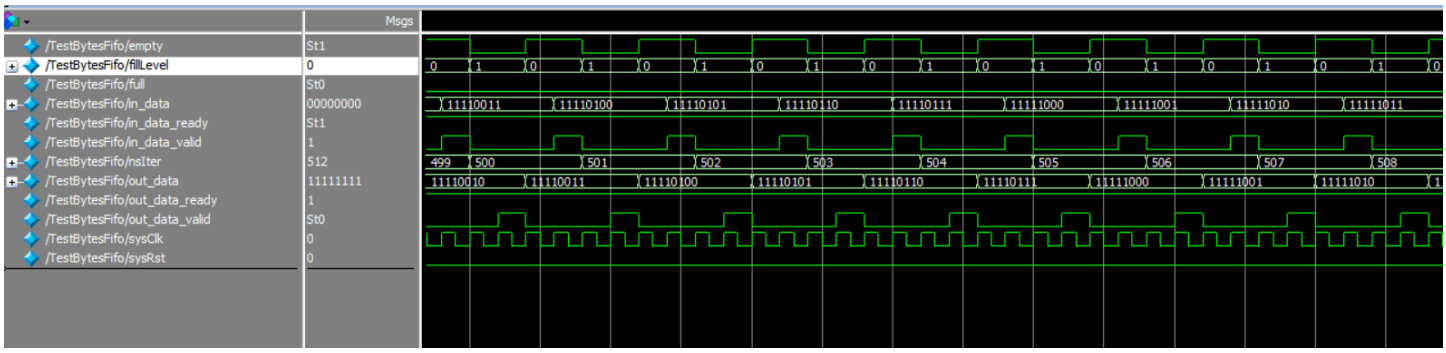

图 3 FIFO 同时读写仿真结果

2.3.2.2 UART 模块仿真

测试了 UART 模块的发送功能，测试结果如图 4 UART 发送仿真结果。在测试中，使用一个较高的时钟，向 UART 的入方向的 FIFO，以较高的速度连续写入多个数据。UART 控制器会从 FIFO 中读取数据，然后以较低的速度降这些数据从 TX 线上送出。

注意 UART 的读操作和用户的写操作是完全异步的。从仿真结果可以看出，当输入低六个符号时，UART 控制器已经开始在 TX 线上发送数据了，这符合设计预期。

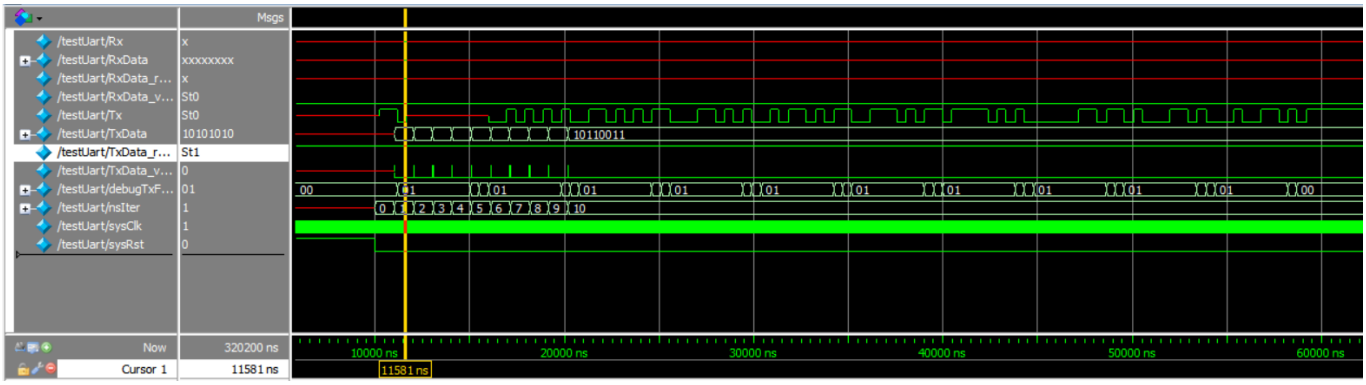

图 4 UART 发送仿真结果

2.4 设计实现

2.4.1 综合和下载过程

2.4.2 实验关键结果及其解释

2.4.2.1 基本实验 a

1.3 基本实验要求中的第一个实验，实验结果如图 5 实验 a 结果所示。当输入一个数字时，将其传回；输入一个非数字时，返回 error。

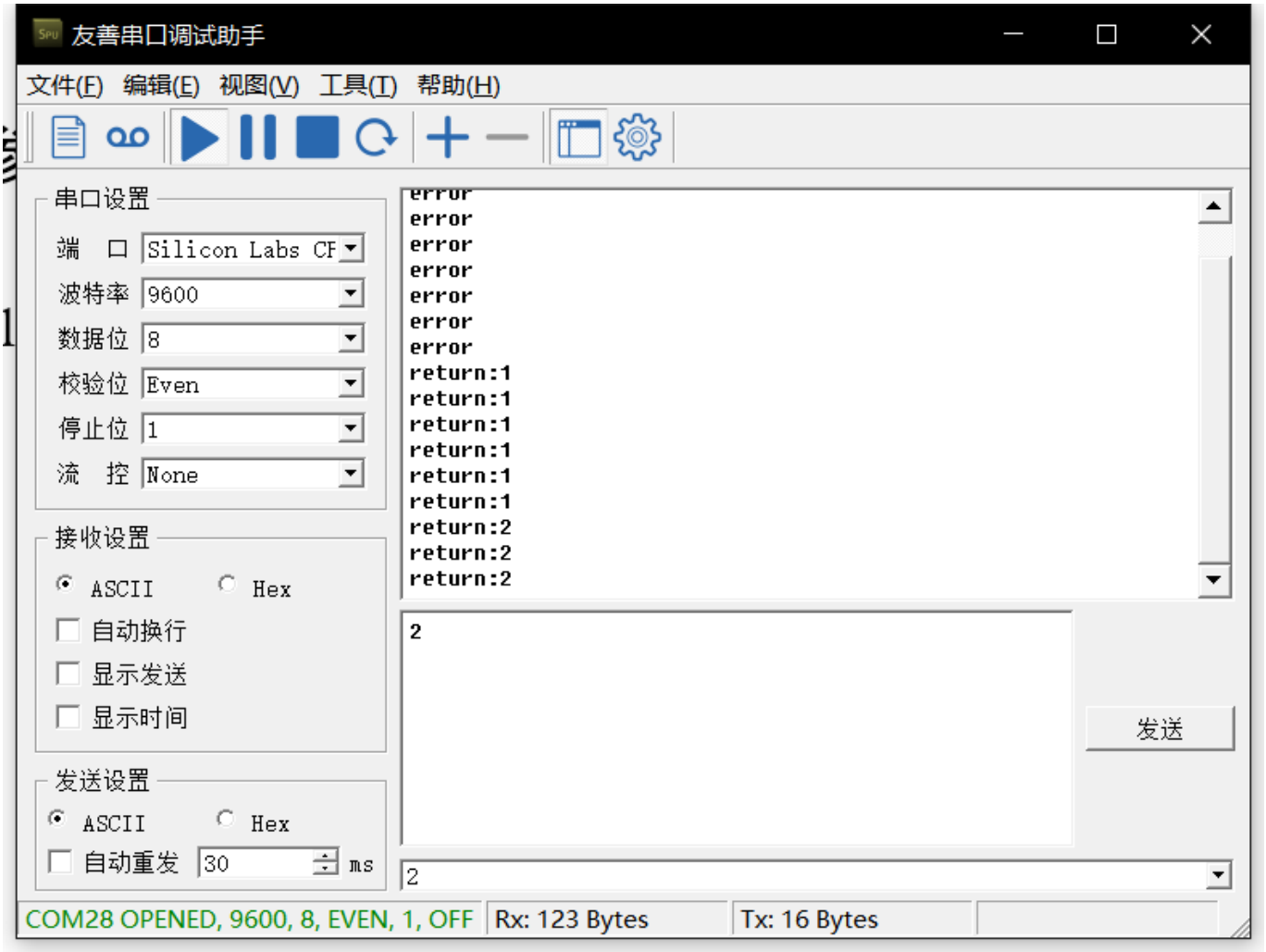

图 5 实验 a 结果

2.4.2.2 基本实验 bc

将 FIFO 的状态信息引出，并用数码管上的资源显示。由于前几个实验已经多次使用数码管/LED 灯显示数据，此处实验现象亦是平凡的，不多赘述。

2.4.2.3 基本实验 d

由于前文中实现的 Uart 模块是自带缓冲区的，因此无需额外实例化新的 FIFO 即可实现本实验的要求。仅需要根据引出的 RX Fifo 的大小，控制 UART 控制器的 Sink 端口即可完成实验。具体实验的结果如图 6 基本实验 d 结果所示。

为了让实验现象更加明显，在实验时

1. 显示接收信息的时间
2. 发送了一个长度无法被 10 整除的字串

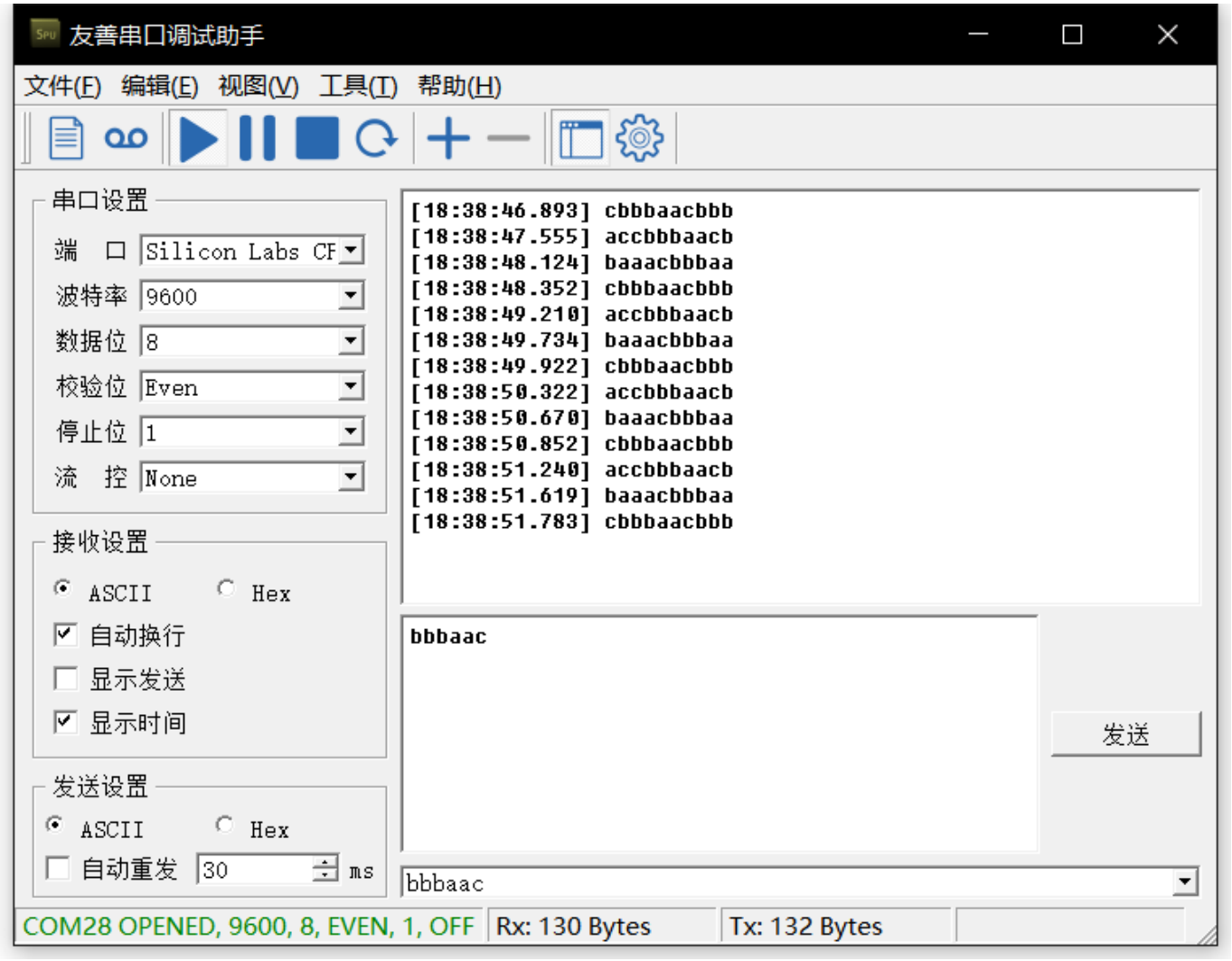

图 6 基本实验 d 结果

2.4.2.4 基本实验 g

改变波特率仅需要改变串口时钟即可。上文中实现 Uart 模块的同时，实现了一个时钟生成模块（具体代码参见 UartClockGenerator.v），该模块利用了 Verilog 的 Paramater 特性控制分频参数，因此只需对程序略作修改即可实现不同波特率下的数据传输。

使用 115200 波特率重新进行实验 a，结果如图 7 基本实验 e 结果所示。

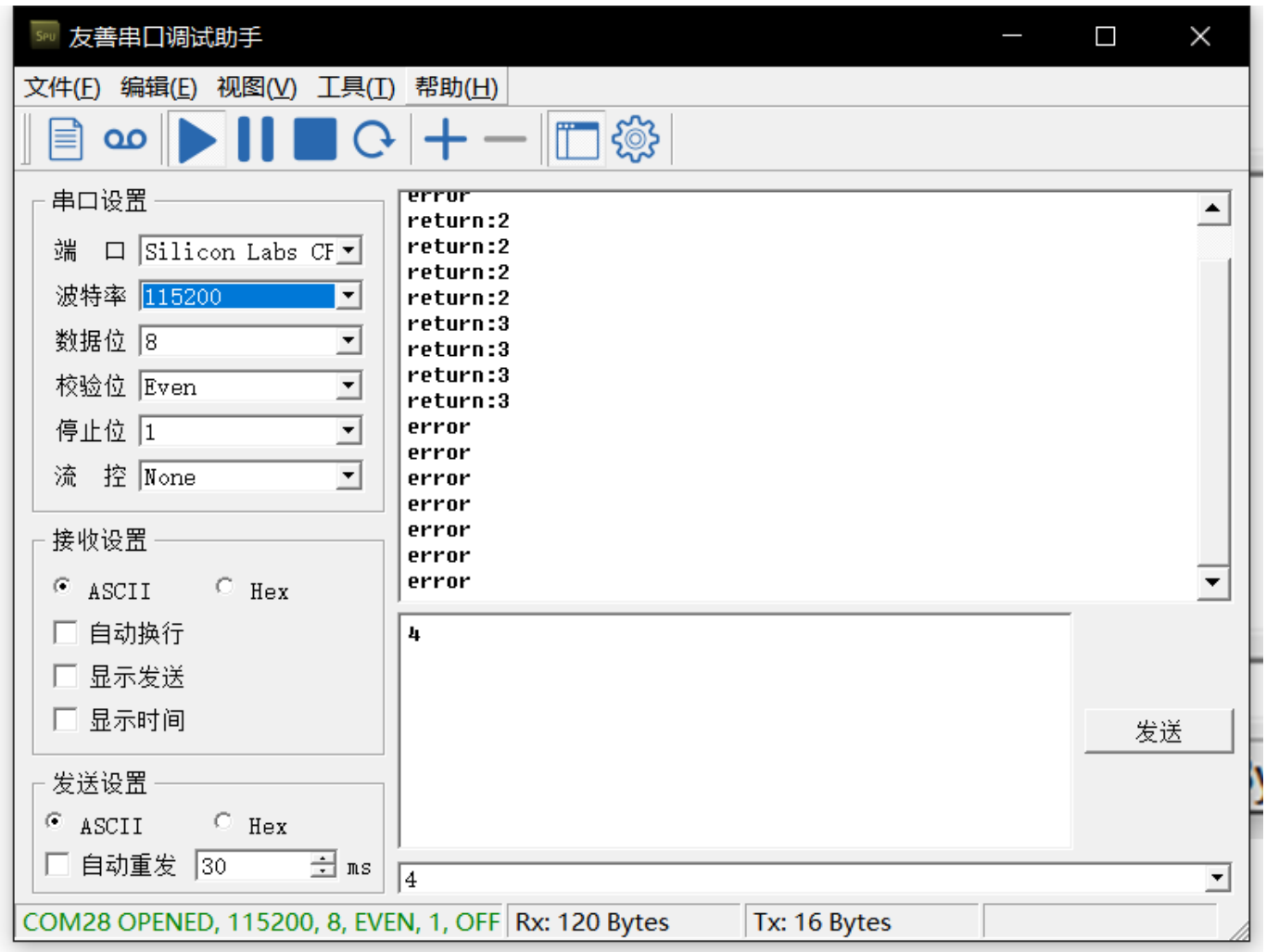

图 7 基本实验 e 结果

2.5小结

本次实验，使用 Verilog 实现了 UART 串口控制器和 FIFO。

参考文献

[1] Avalon® Interface Specifications – Intel

参考代码

注:直接访问 Github 网站可能较慢。如必需检查代码，亦可以将仓库 clone 到本地进行检查。仓库地址: https://github.com/CNLHC/DigiC2018

[1] BytesFIFO.v

[2] Uart.v

[3] Parity.v

[4] UartClockGenerator.v