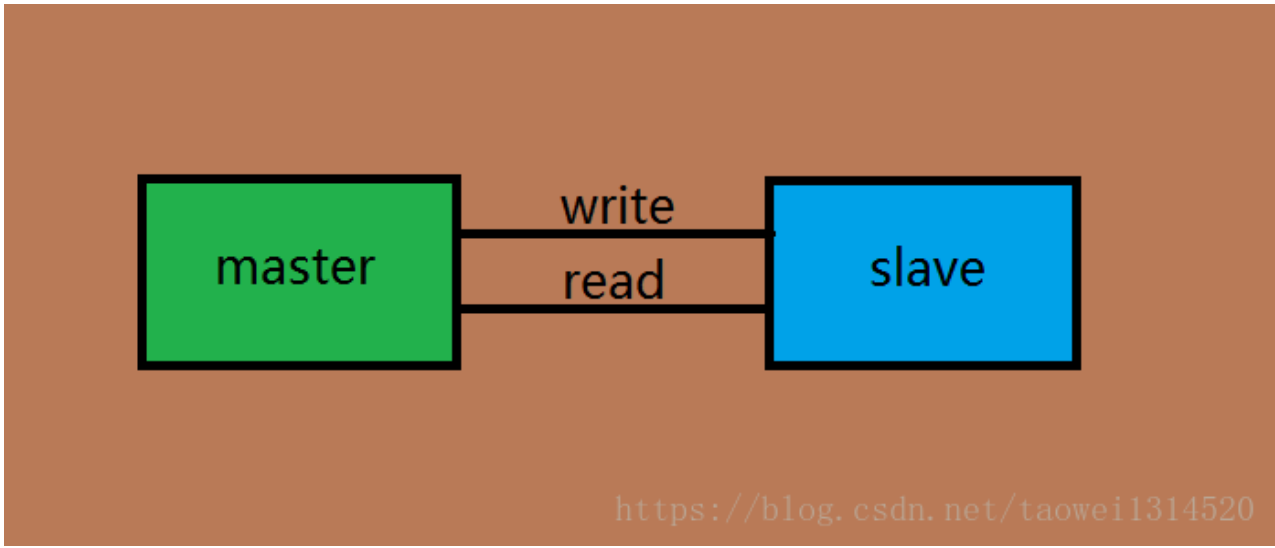


# MYIR-ZYNQ7000系列-zturn教程(16)：对axi\_lite IP核进行仿真以及axi总线的初步解

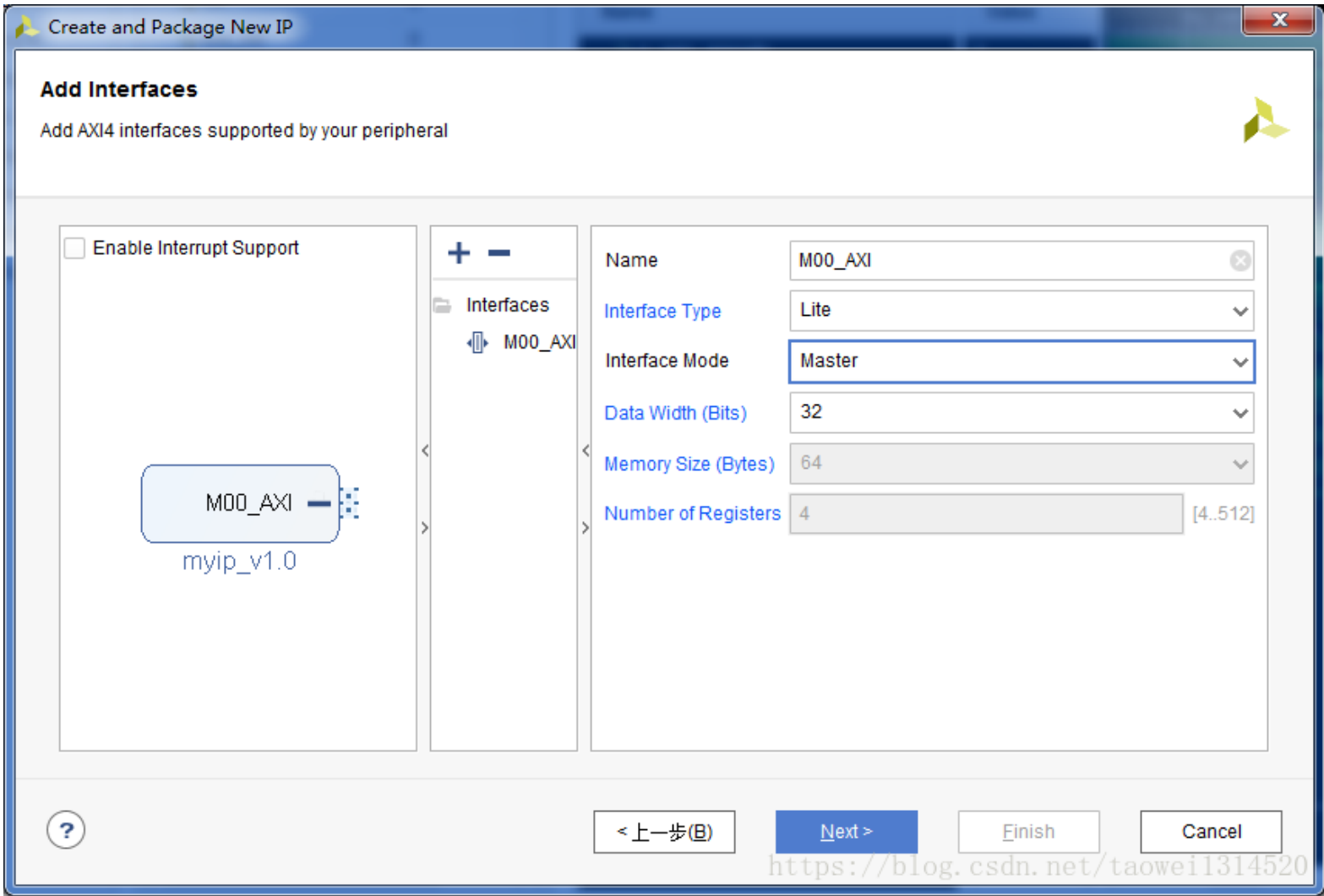
原创 虚无缥缈vs威武 最后发布于2018-06-06 19:11:51 阅读数 1516 ☆ 收藏

我这里一共调用了两个自定义的IP都是基于axi\_lite的IP核，一个是主机master一个是从机slave，然后将这两个调用的IP例化到一个新创建的fpga工程，最好写一个仿真脚本让这个master主机对这个从机slave进行读写。

链接：<https://pan.baidu.com/s/1WFCazNaUaXBwKuJtAZNKZQ> 密码：ex8l



主机：

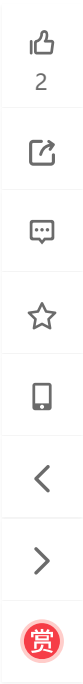
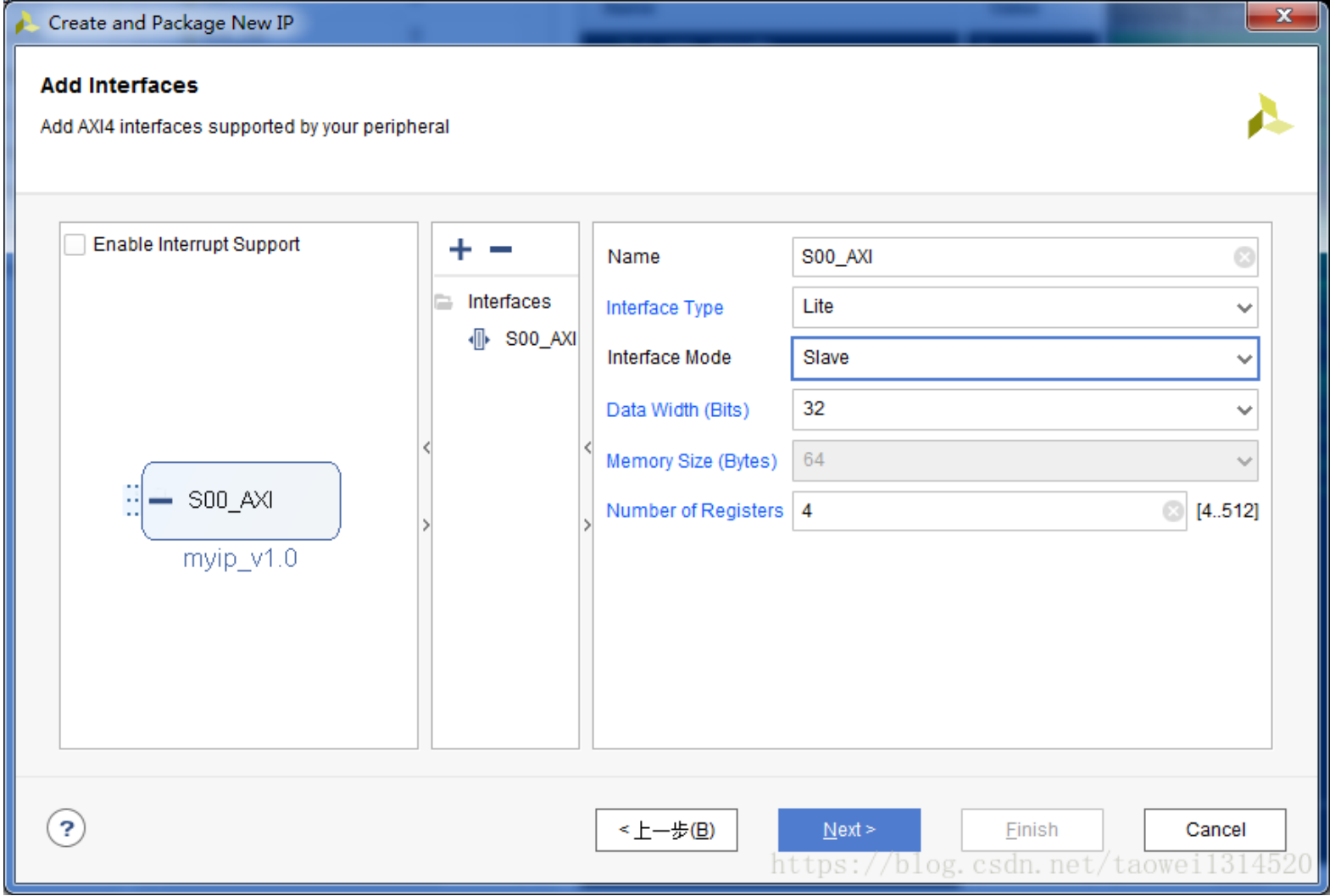


从机：

2

展开

赏



将master和slave都例化到fpga工程的顶层文件如下图所示

```
1  `timescale 1ns / 1ps
2  ///////////////////////////////////////////////////////////////////
3  // Company:
4  // Engineer:
5  //
6  // Create Date: 2018/05/21 10:27:22
7  // Design Name:
8  // Module Name: test_axi
9  // Project Name:
10 // Target Devices:
11 // Tool Versions:
12 // Description:
13 //
14 // Dependencies:
15 //
16 // Revision:
17 // Revision 0.01 - File Created
18 // Additional Comments:
19 //
20 ///////////////////////////////////////////////////////////////////
21
22
23 module test_axi(
24
25     input axi_aclk,
26     input axi_aresetn,
27     input app_txn,
28     output state_err,
29     output state_done
30
31 );
32
33 parameter integer C_AXI_ADDR_WIDTH  = 32;
34 parameter integer C_AXI_DATA_WIDTH  = 32;
35
36
37     wire w_err;          // 状态指示, 出现错误
38     wire w_txn_done;     // 状态指示, 发送完毕
39
40     assign state_err  = w_err;
41     assign state_done = w_txn_done;
42
43     wire [C_AXI_ADDR_WIDTH-1 : 0]    axi_AWADDR;    // AXI总线信号: AWADDR
44     wire [2 : 0]                     axi_AWPROT;     // AXI总线信号: AWPROT
45     wire                             axi_AWVALID;    // AXI总线信号: AWVALID
46     wire                             axi_AWREADY;    // AXI总线信号: AWREADY
47
```



48

wire [C\_AXI\_DATA\_WIDTH-1 : 0] axi\_WDATA; // AXI总线信号: WDATA

49

50

wire [C\_AXI\_DATA\_WIDTH/8-1 : 0] axi\_WSTRB; // AXI总线信号: WSTRB 50

51

52

wire axi\_WVALID; // AXI总线信号: WVALID 51

53

54

wire axi\_WREADY; // AXI总线信号: WREADY 52

55

56

wire [1 : 0] axi\_BRESP; // AXI总线信号: BRESP

57

58

wire axi\_BVALID; // AXI总线信号: BVALID

59

60

wire axi\_BREADY; // AXI总线信号: BREADY

61

62

wire [C\_AXI\_ADDR\_WIDTH-1 : 0] axi\_ARADDR; // AXI总线信号: ARADDR

63

64

wire [2 : 0] axi\_ARPROT; // AXI总线信号: ARPROT

65

66

wire axi\_ARVALID; // AXI总线信号: ARVALID

67

68

wire axi\_ARREADY; // AXI总线信号: ARREADY

69

70

wire [C\_AXI\_DATA\_WIDTH-1 : 0] axi\_RDATA; // AXI总线信号: RDATA

71

72

wire [1 : 0] axi\_RRESP; // AXI总线信号: RRESP

73

74

wire axi\_RVAILD; // AXI总线信号: RVAILD

75

76

wire axi\_RREADY; // AXI总线信号: RREADY

77

78

myip\_master\_0 u1 (

79

80

.m00\_axi\_awaddr(axi\_AWADDR), // output wire [31 : 0] m00\_axi\_awaddr

81

82

.m00\_axi\_awprot(axi\_AWPROT), // output wire [2 : 0] m00\_axi\_awprot

83

84

.m00\_axi\_awvalid(axi\_AWVALID), // output wire m00\_axi\_awvalid

85

86

.m00\_axi\_awready(axi\_AWREADY), // input wire m00\_axi\_awready

87

88

.m00\_axi\_wdata(axi\_WDATA), // output wire [31 : 0] m00\_axi\_wdata

89

90

.m00\_axi\_wstrb(axi\_WSTRB), // output wire [3 : 0] m00\_axi\_wstrb

91

92

.m00\_axi\_wvalid(axi\_WVALID), // output wire m00\_axi\_wvalid

93

94

.m00\_axi\_wready(axi\_WREADY), // input wire m00\_axi\_wready

95

96

.m00\_axi\_bresp(axi\_BRESP), // input wire [1 : 0] m00\_axi\_bresp

97

98

.m00\_axi\_bvalid(axi\_BVALID), // input wire m00\_axi\_bvalid

99

100

.m00\_axi\_bready(axi\_BREADY), // output wire m00\_axi\_bready

101

102

.m00\_axi\_araddr(axi\_ARADDR), // output wire [31 : 0] m00\_axi\_araddr

103

104

.m00\_axi\_arprot(axi\_ARPROT), // output wire [2 : 0] m00\_axi\_arprot

105

106

.m00\_axi\_arvalid(axi\_ARVALID), // output wire m00\_axi\_arvalid

107

108

.m00\_axi\_arready(axi\_ARREADY), // input wire m00\_axi\_arready

109

110

.m00\_axi\_rdata(axi\_RDATA), // input wire [31 : 0] m00\_axi\_rdata

111

112

.m00\_axi\_rresp(axi\_RRESP), // input wire [1 : 0] m00\_axi\_rresp

113

114

.m00\_axi\_rvalid(axi\_RVAILD), // input wire m00\_axi\_rvalid

115

116

.m00\_axi\_rready(axi\_RREADY), // output wire m00\_axi\_rready

117

118

.m00\_axi\_aclk(axi\_aclk), // input wire m00\_axi\_aclk

119

120

.m00\_axi\_aresetn(axi\_aresetn), // input wire m00\_axi\_aresetn

121

122

.m00\_axi\_init\_axi\_txn(app\_txn), // input wire m00\_axi\_init\_axi\_txn

123

124

.m00\_axi\_error(w\_err), // output wire m00\_axi\_error

125

126

.m00\_axi\_txn\_done(w\_txn\_done) // output wire m00\_axi\_txn\_done

127

128

);

129

130

myip\_Slave\_0 u2 (

131

132

.s00\_axi\_awaddr(axi\_AWADDR), // input wire [3 : 0] s00\_axi\_awaddr

133

134

.s00\_axi\_awprot(axi\_AWPROT), // input wire [2 : 0] s00\_axi\_awprot

135

136

.s00\_axi\_awvalid(axi\_AWVALID), // input wire s00\_axi\_awvalid

137

138

.s00\_axi\_awready(axi\_AWREADY), // output wire s00\_axi\_awready

139

140

.s00\_axi\_wdata(axi\_WDATA), // input wire [31 : 0] s00\_axi\_wdata

141

142

.s00\_axi\_wstrb(axi\_WSTRB), // input wire [3 : 0] s00\_axi\_wstrb

143

144

.s00\_axi\_wvalid(axi\_WVALID), // input wire s00\_axi\_wvalid

145

146

.s00\_axi\_wready(axi\_WREADY), // output wire s00\_axi\_wready

147

148

.s00\_axi\_bresp(axi\_BRESP), // output wire [1 : 0] s00\_axi\_bresp

149

150

.s00\_axi\_bvalid(axi\_BVALID), // output wire s00\_axi\_bvalid

151

152

.s00\_axi\_bready(axi\_BREADY), // input wire s00\_axi\_bready

153

154

.s00\_axi\_araddr(axi\_ARADDR), // input wire [3 : 0] s00\_axi\_araddr

155

156

.s00\_axi\_arprot(axi\_ARPROT), // input wire [2 : 0] s00\_axi\_arprot

157

158

.s00\_axi\_arvalid(axi\_ARVALID), // input wire s00\_axi\_arvalid

159

160

.s00\_axi\_arready(axi\_ARREADY), // output wire s00\_axi\_arready

161

162

.s00\_axi\_rdata(axi\_RDATA), // output wire [31 : 0] s00\_axi\_rdata

163

164

.s00\_axi\_rresp(axi\_RRESP), // output wire [1 : 0] s00\_axi\_rresp

165

166

.s00\_axi\_rvalid(axi\_RVAILD), // output wire s00\_axi\_rvalid

167

168

.s00\_axi\_rready(axi\_RREADY), // input wire s00\_axi\_rready

169

170

.s00\_axi\_aclk(axi\_aclk), // input wire s00\_axi\_aclk

171

172

.s00\_axi\_aresetn(axi\_aresetn) // input wire s00\_axi\_aresetn

173

174

);

175

176

endmodule

177

2

赏

举报

下面这个这个fpga工程的仿真脚本

```
1 `timescale 1ns /1ps
```

```

4 // Engineer:
5 //
6 // Create Date: 2018/05/21 11:10:59
7 // Design Name:
8 // Module Name: test_tb
9 // Project Name:
10 // Target Devices:
11 // Tool Versions:
12 // Description:
13 //
14 // Dependencies:
15 //
16 // Revision:
17 // Revision 0.01 - File Created
18 // Additional Comments:
19 //
20 //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
21
22
23 module test_tb();
24
25     reg axi_aclk;    // AXI总线时钟
26     reg axi_aresetn; // 系统复位信号
27
28     reg r_app_txn;
29     wire w_err;      // 状态指示, 出现错误
30     wire w_txn_done; // 状态指示, 发送完毕
31
32     test_axi u1 (
33         .axi_aclk(axi_aclk),
34         .axi_aresetn(axi_aresetn),
35         .app_txn(r_app_txn),
36         .state_err(w_err),
37         .state_done(w_txn_done)
38     );
39
40
41     always begin
42         #10;
43         axi_aclk = ~axi_aclk;
44     end
45
46     initial begin
47         axi_aclk    = 1'b0;
48         axi_aresetn = 1'b1;
49         r_app_txn   = 1'b1;
50
51         #10;
52         axi_aresetn = 1'b0;
53         #5;
54         r_app_txn = 1'b0;
55         #5;
56         axi_aresetn = 1'b1;
57         #5;
58         r_app_txn = 1'b1;
59     end
60
61 endmodule

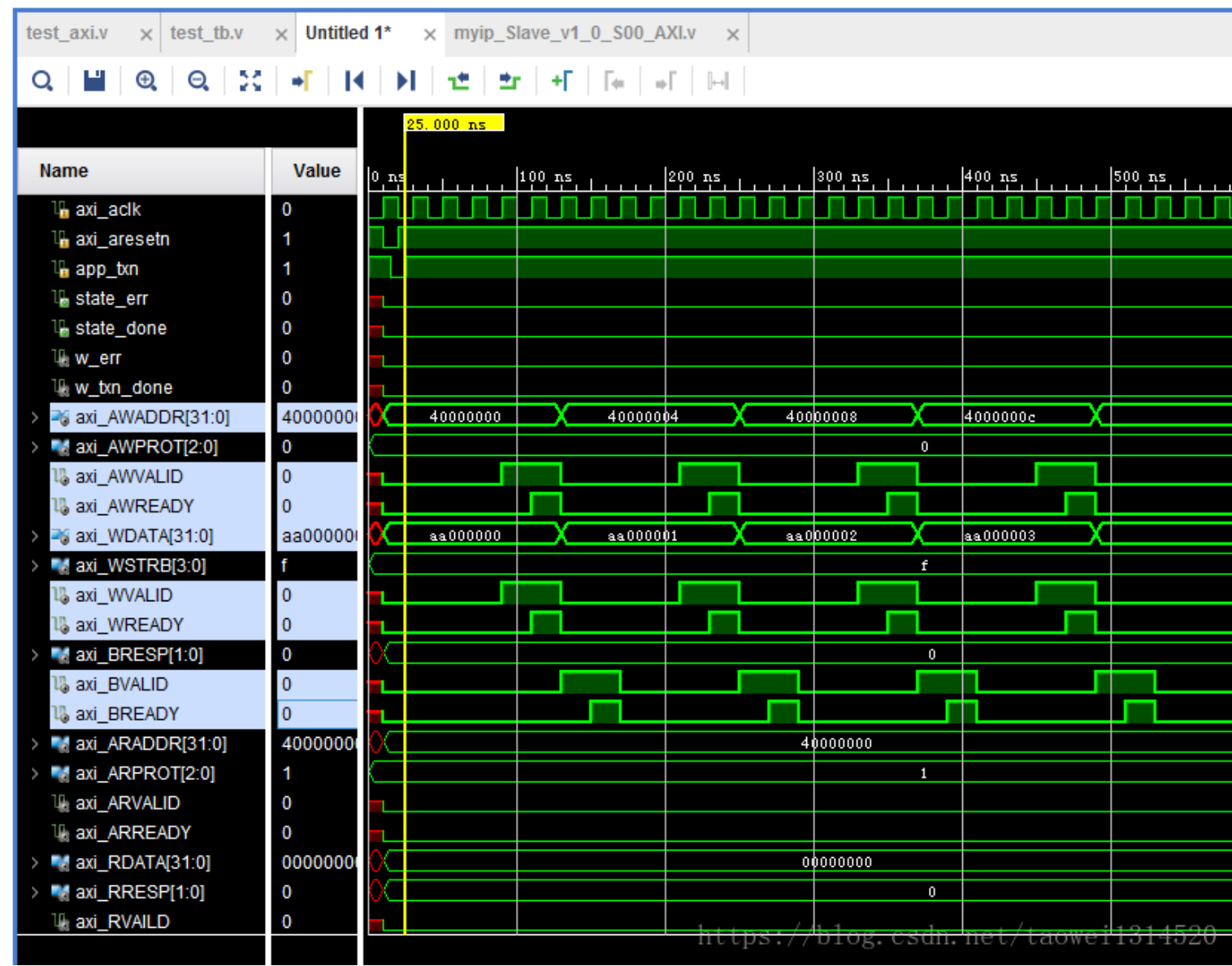
```



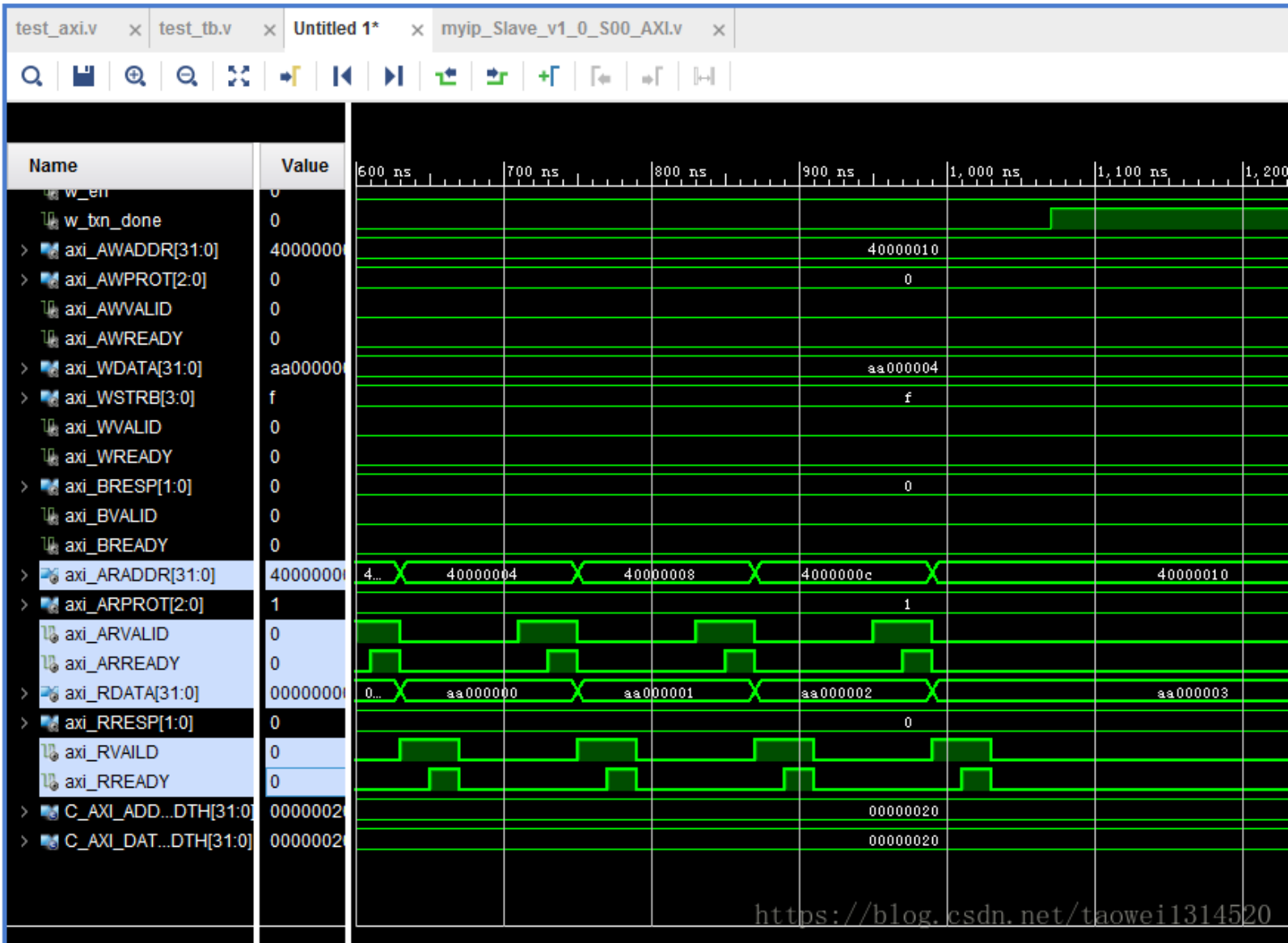
这个是运行仿真脚本后的master写的波形



举报





这个是运行仿真脚本后master读的波形





对比写和读可以发现写入的和读出的是一致的说明这个master对这个slave进行读写是成功的


对于这个axi协议的各个引脚代表的含义我这里不做一一介绍，下面的截图里面已经说得比较清楚了


2



















举报

写通道		和控制信息。		通 信 号 都 可 用。		
	AWADDR	写地址	WDATA	写数据	BREADY	响应准备。该信号指示在主主机可以接受一个响应信号
	AWREADY	写地址准备好了。该信号指示从器件准备好接受一个地址和相关联的控制信号	WSTRB	写选通。这个信号表明该字节通道持有效数据。每一bit对 应 WDATA 一个字节	BRESP	写响应。这个信号表示写事务处理的状态。
	AWPROT	写通道保护类型。这个信号表示该事务的特权和安全级别，并确定是否该事务是一个数据存取或指令的访问	WREADY	写准备好了。该信号指示从器件可以接受写数据。		

👍  
2

🔗

💬

☆

📱

<

>

赏

读通道	地址通道		数据通道			
	ARVALID	读地址有效。此信号表明该信道此时能有效读出地址和控制信息	RVALID		读数据有效。此信号表明该信道此时能有效读出数据	
	ARADDR	读地址	RDATA		读数据	
	ARREADY	读地址准备好了。该信号指示从器件准备好接受一个地址和相关联的控制信号	RREADY		读数据准备好了。该信号指示从器件准备好接收数据	
	ARPROT	保护类型。这个信号表示该事务的特权和安全级别，并确定是否该事务是一个数据存取或指令的访问	RRESP		读取响应。这个信号表明读事务处理的状态。	
	地址通道		数据通道		应答通道	
	AWVALID	写地址有效。这个信号表示该主信令有效的写地址	WVALID	写有效。这个信号表示有效的写数据和选	BVALID	写响应有效。此信号表明写命令的有效写入响应。

我这里主要介绍一个主机master对这个从机slave进行读写的详细过程

master

slave

🔄

举报



AWADDR信号：主机向

从机发送地址

AWVALID信号：当这个  
信号为高电平时代表从机  
可以采集来自主机  
的地址

WDATA：主机要写入的数  
据

WVALID信号：当这个  
信号为高电平代表从机  
可以采集来自主机的数  
据

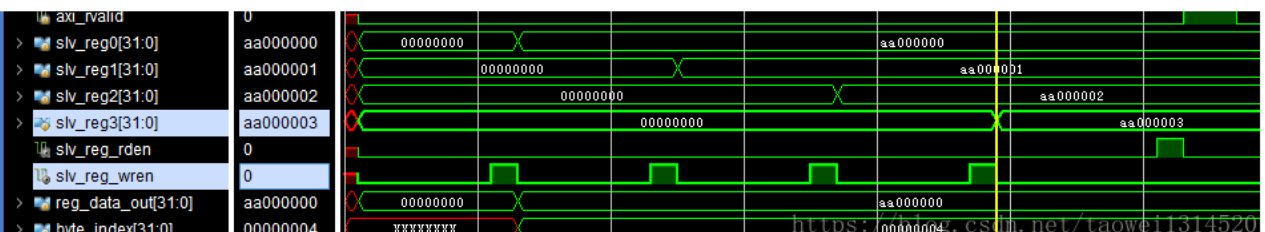
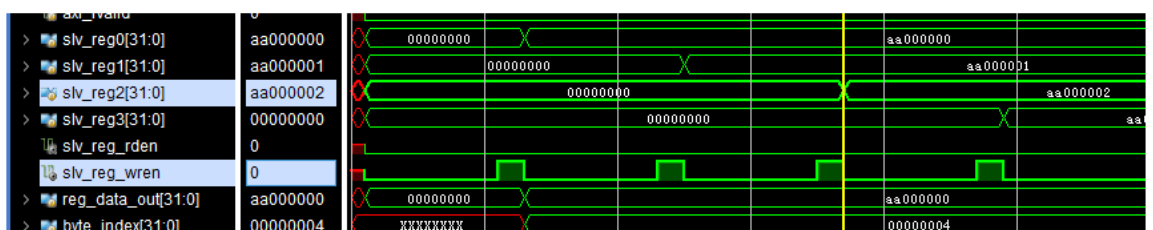
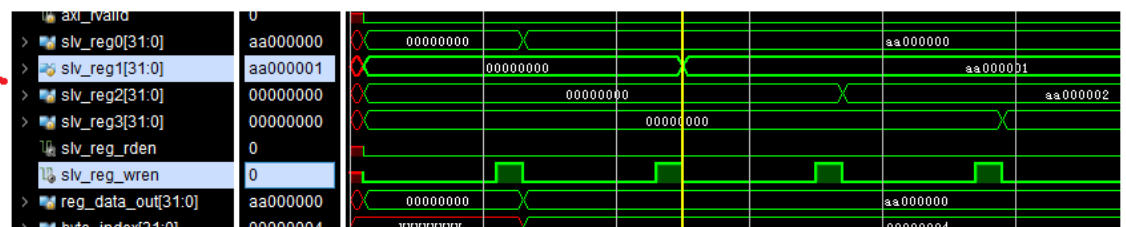
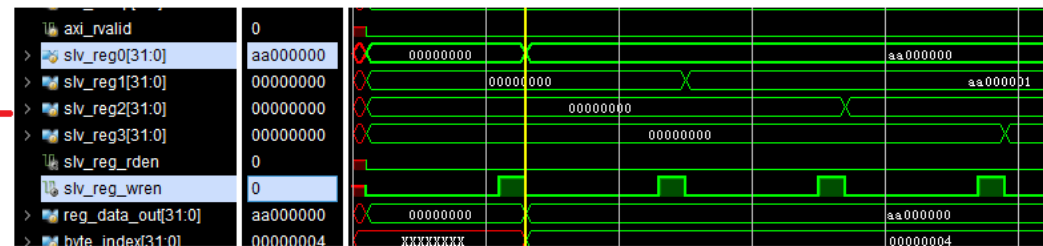
AWREADY信号：当这个  
信号为高电平时代表从机  
已经采集到来自主机  
的地址

WREADY信号：当这个信  
号为高电平时代表从机  
已经采集到来自主机的数据

从机采集到主机的地址和数据从机内部开始进行执行写

assign slv\_reg\_wren = axi\_wready && S\_AXI\_WVALID && axi\_awready && S\_AXI\_AWVALID;(从机开始写数据的使能条件)

```
begin
case ( axi_awaddr[ADDR_LSB+OPT_MEM_ADDR_BITS:ADDR_LSB] )
2'h0:
for ( byte_index = 0; byte_index <= (C_S_AXI_DATA_WIDTH/8)-1; byte_index = byte_index+1 )
if ( S_AXI_WSTRB[byte_index] == 1 ) begin
// Respective byte enables are asserted as per write strobes
// Slave register 0
slv_reg0[(byte_index*8) +: 8] <= S_AXI_WDATA[(byte_index*8) +: 8];
end
2'h1:
for ( byte_index = 0; byte_index <= (C_S_AXI_DATA_WIDTH/8)-1; byte_index = byte_index+1 )
if ( S_AXI_WSTRB[byte_index] == 1 ) begin
// Respective byte enables are asserted as per write strobes
// Slave register 1
slv_reg1[(byte_index*8) +: 8] <= S_AXI_WDATA[(byte_index*8) +: 8];
end
2'h2:
for ( byte_index = 0; byte_index <= (C_S_AXI_DATA_WIDTH/8)-1; byte_index = byte_index+1 )
if ( S_AXI_WSTRB[byte_index] == 1 ) begin
// Respective byte enables are asserted as per write strobes
// Slave register 2
slv_reg2[(byte_index*8) +: 8] <= S_AXI_WDATA[(byte_index*8) +: 8];
end
2'h3:
for ( byte_index = 0; byte_index <= (C_S_AXI_DATA_WIDTH/8)-1; byte_index = byte_index+1 )
if ( S_AXI_WSTRB[byte_index] == 1 ) begin
// Respective byte enables are asserted as per write strobes
// Slave register 3
slv_reg3[(byte_index*8) +: 8] <= S_AXI_WDATA[(byte_index*8) +: 8];
end
end
```



从机写完成后：

BREADY信号：主机收到  
从机的写完成信号，发  
送一个反馈信号告诉从机  
我已经收到写完成信  
号

BVALID信号：从机完成写数据完  
成后向主机发送一个高电平代表  
从机已经完成写操作

上面介绍的是一个master对slave写的全部过程

这里介绍的是master对slave读

master

slave



举报

ARADDR信号：主机向从机发送所要读的地址

ARVALID信号：当这个信号为高时代表从机可以采集主机的地址

RDATA信号：主机从从机那里读取到的数据

axi\_BREADY

axi\_ARADDR[31:0]

axi\_ARPROT[2:0]

axi\_ARVALID

axi\_ARREADY

axi\_RDATA[31:0]

axi\_RRESP[1:0]

axi\_RVALID

axi\_RREADY

C\_AXI\_ADDR...DTH[31:0]

C\_AXI\_DAT...DTH[31:0]

0

40000000

1

0

00000000

0

0

0

00000020

00000020

40000000

40000004

40000008

4000000c

40000010

1

0

aa000000

aa000001

aa000002

aa000003

0

00000020

00000020

ARREADY

当这个信号为高电平时代表从机已经采集到主机的地址

https://blog.csdn.net/taoweil1314520

assign slv\_reg\_rden = axi\_arready & S\_AXI\_ARVALID & ~axi\_rvalid;(进行读时使能条件)

```
assign slv_reg_rden = axi_arready & S_AXI_ARVALID & ~axi_rvalid;
always @(*)
begin
    // Address decoding for reading registers
    case ( axi_araddr[ADDR_LSB+OPI_MEM_ADDR_BITS:ADDR_LSB] )
        2'h0 : reg_data_out <= slv_reg0;
        2'h1 : reg_data_out <= slv_reg1;
        2'h2 : reg_data_out <= slv_reg2;
        2'h3 : reg_data_out <= slv_reg3;
        default : reg_data_out <= 0;
    endcase
end

// Output register or memory read data
always @( posedge S_AXI_ACLK )
begin
    if ( S_AXI_ARESEIN == 1'b0 )
    begin
        axi_rdata <= 0;
    end
    else
    begin
        // When there is a valid read address (S_AXI_ARVALID) with
        // acceptance of read address by the slave (axi_arready),
        // output the read data
        if (slv_reg_rden)
        begin
            axi_rdata <= reg_data_out;    // register read data
        end
    end
end
```

axi\_rvalid

slv\_reg0[31:0]

slv\_reg1[31:0]

slv\_reg2[31:0]

slv\_reg3[31:0]

slv\_reg\_rden

slv\_reg\_wren

reg\_data\_out[31:0]

byte\_index[31:0]

aw\_en

0

aa000000

aa000001

aa000002

aa000003

1

0

aa000000

00000004

1

aa000000

aa000001

aa000002

aa000003

00000004

axi\_rvalid

slv\_reg0[31:0]

slv\_reg1[31:0]

slv\_reg2[31:0]

slv\_reg3[31:0]

slv\_reg\_rden

slv\_reg\_wren

reg\_data\_out[31:0]

byte\_index[31:0]

aw\_en

0

aa000000

aa000001

aa000002

aa000003

1

0

aa000002

00000004

1

aa000000

aa000001

aa000002

aa000003

00000004

axi\_rresp[1:0]

axi\_rvalid

slv\_reg0[31:0]

slv\_reg1[31:0]

slv\_reg2[31:0]

slv\_reg3[31:0]

slv\_reg\_rden

slv\_reg\_wren

reg\_data\_out[31:0]

byte\_index[31:0]

0

0

aa000000

aa000001

aa000002

aa000003

1

0

aa000003

00000004

0

aa000000

aa000001

aa000002

aa000003

00000004

主机读取完成后：

RVALID信号：当主机读取数据完成后会发生一个高电平告诉从机数据读取完成

RREADY信号：从机收到主机的读取完成信号，会反馈一个高电平告诉主机我已经收到你读取完成的信号

S\_AXI\_ARPROT[2:0]

S\_AXI\_ARVALID

S\_AXI\_ARREADY

S\_AXI\_RDATA[31:0]

S\_AXI\_RRESP[1:0]

S\_AXI\_RVALID

S\_AXI\_RREADY

axi\_awaddr[3:0]

axi\_awready

1

0

0

00000000

0

0

0

c

0

1

0

0

aa000000

aa000001

aa000002

aa000003

0

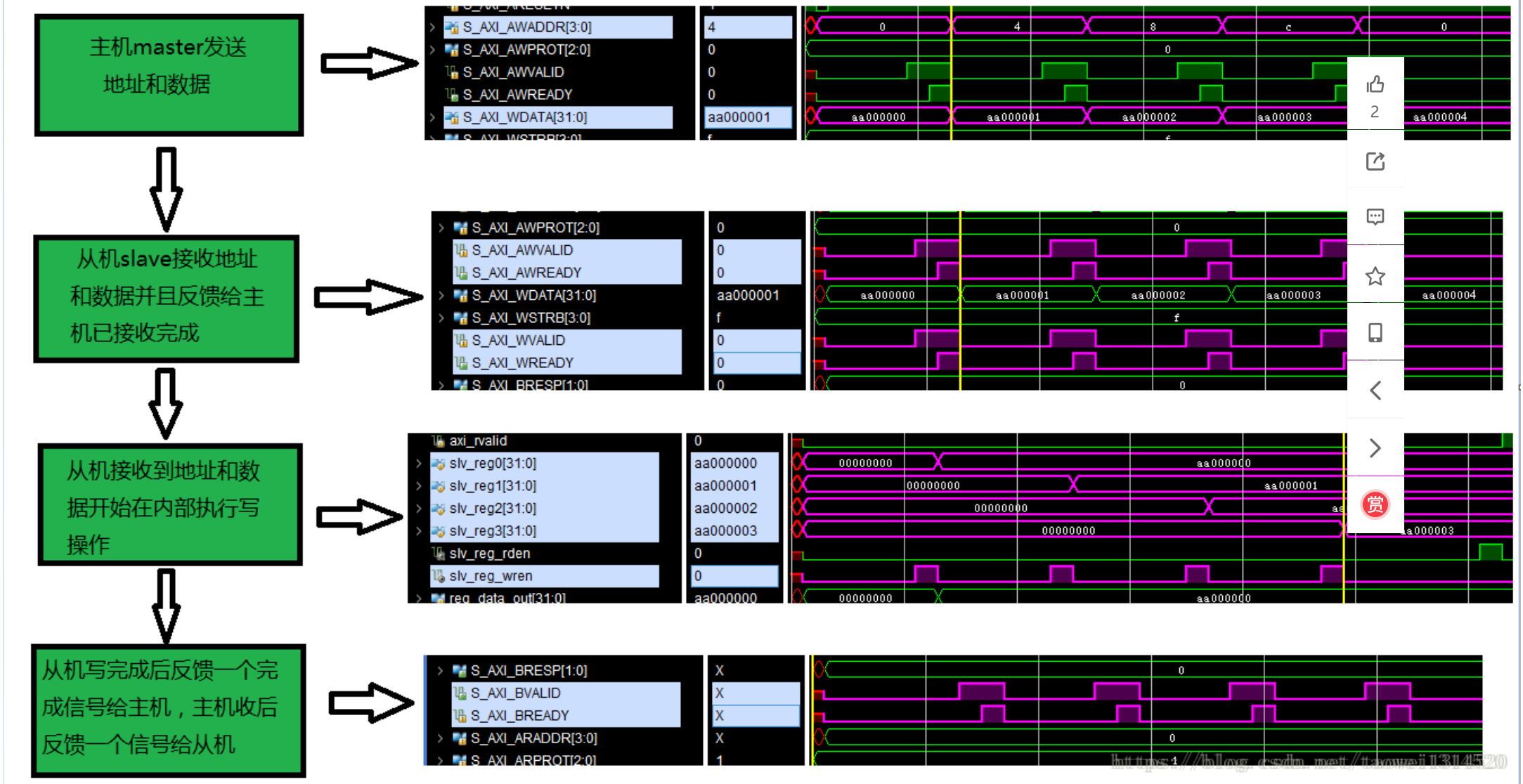
c

https://blog.csdn.net/taoweil1314520

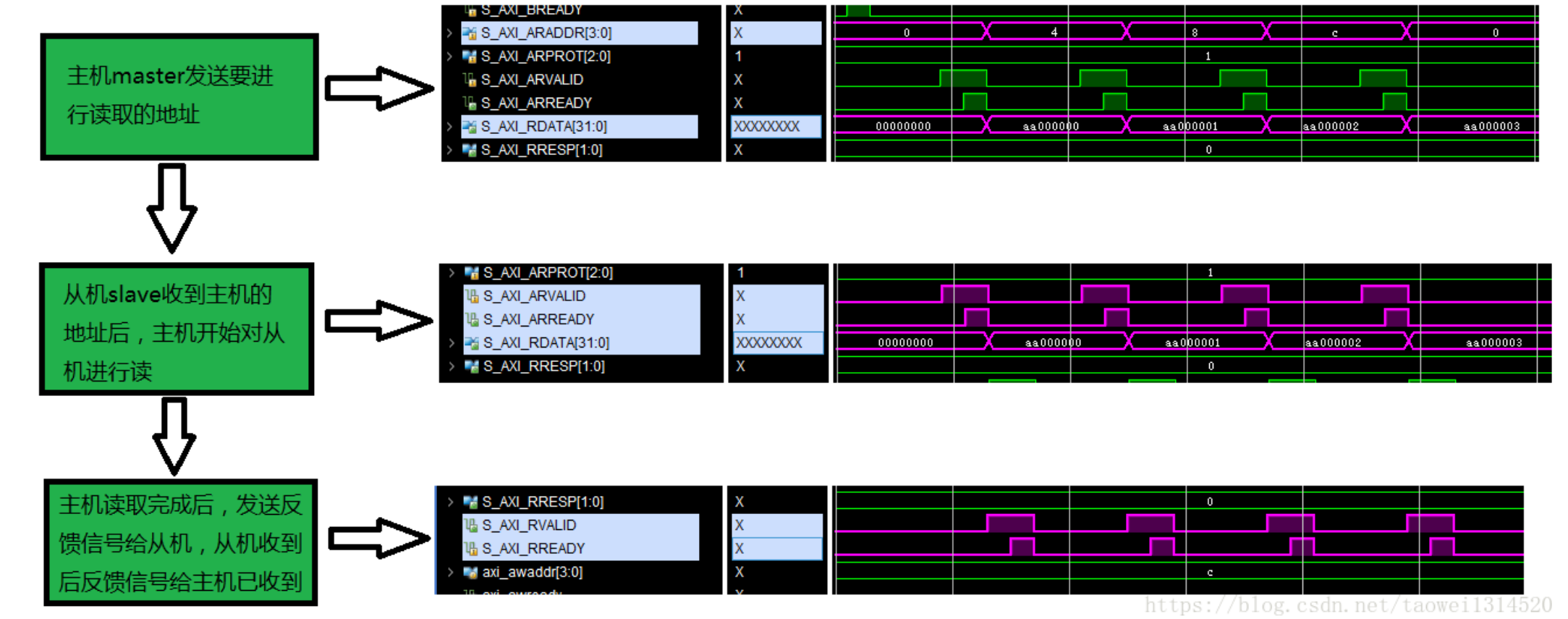
这里最后将这个读写过程总结一下

主机master进行写






主机master进行读



将这个当做笔记来进行记录以免以后自己忘记了

点赞 2 收藏 分享 ...




虚无缥缈vs威武

发布了37 篇原创文章 · 获赞 58 · 访问量 15万+

私信


关注



数据管理

发现称量样品及手工记录数据过程中可能发生的错误

广告 METTLER TOLEDO



想对作者说点什么

Xilinx AXI 验证 IP (VIP)作为AXI4-lite master 仿真验证AXI4-lite slave

AXI 验证 IP 简介AXI Stream VIP 可用于为支持定制 RTL 设计流程的 AXI 主设备及 AXI 从设备验证连接和基本功能...

博文 来自: 怀剑听雨

阅读数 1225

ZYNQ-7000 Vivado 自定义IP封装

软件版本: vivado 2018.01操作系统: cent os 6.0本文中主要介绍在vivado中如何使用系统工具封装我们自己的I...

博文 来自: mexican007的专栏

阅读数 951

举报

**FPGA项目——基于AXI4总线的RAM读写** 阅读数 1992

基于xilinx IP 核 Block Memory Gnerator V 8.3AXI总线读写协议简介在vivado上仿真实现IP核配置一共两种选择， ... 博文 来自： [wb3jdw58的博客](#)

**MYIR-ZYNQ7000系列-zturn教程(26)：自定义axi\_lite IP点亮LED灯** 阅读数 173

开发板环境：vivado 2017.4，开发板型号xc7z020clg400-1，这个工程主要功能是自定义一个axi\_lite IP然后在SDK... 博文 来自： [taowei1314520的...](#)

**AXI-Lite总线及其自定义IP核使用分析总结** 阅读数 96

ZYNQ的优势在于通过高效的接口总线组成了ARM+FPGA的架构。我认为两者是互为底层的，当进行算法验证... 博文 来自： [weixin\\_34041003...](#)

**AXI总线整理总结** 阅读数 1404

AXI总线一、Definition嵌入式系统是当今计算机工业发展的一个热点，随着超大规模集成电路的迅速发展，半导体工... 博文 来自： [tristan\\_tian的博客](#)

**AXI总线简介** 阅读数 625

0.绪论AXI是高级扩展接口，在AMBA3.0中提出，AMBA4.0将其修改升级为AXI4.0。AMBA4.0 包括AXI4.0、AXI4.0... 博文 来自： [weixin\\_33881753...](#)

**AXI\_03 AXI\_LITE\_SLAVE\_IP核设计与验证** 阅读数 820

由于该系列文章阅读有顺序性，所以请跳转至该系列文章第一篇从头开始阅读，并按照文章末尾指示按顺序阅读，否... 博文 来自： [比特电子工作室](#)

**深入 AXI4 总线 （三） 突发传输机制** 阅读数 2221

本文参考AMBA® AXI and ACE Protocol Specification 来写本系列我想深入探寻 AXI4 总线。不过事情总是这样， ... 博文 来自： [l471094842的博客](#)

**Vivado下产生AXI-Lite Ipcore及AXI-Lite源代码解析** 阅读数 7693

一. AXI-Lite接口Ipcore的生成Xilinx开发工具EDK和Vivado都可自动生成AXI-Lite、AXI-Stream主从模式接口。关... 博文 来自： [yang2011079080...](#)

**如何在vivado中使用AXI IP核搭建ZYNQ 7000平台(以spi IP核为例)** 阅读数 95

新建vivado工程打开vivado软件，我这里使用的是vivado2019.1，单击create project来创建一个新的工程。 单击n... 博文 来自： [橙的博客](#)



**懷劍聽雨**  
11篇文章  

关注


排名:千里之外



**gdboyi**  
5篇文章  

关注

排名:千里之外



**lishanshan2455**  
8篇文章  

关注

排名:千里之外

**ZedBoard学习手记（二） 开发自定义AXI总线外设IP核——以LED和开关为例** 阅读数 1966

想要发挥ZYNQ芯片的特长，让整个系统协同工作起来，就需要将PS与PL两部分结合在一起，在Cortex-A9核和FPG... 博文 来自： [ascend的专栏](#)

**zc702-自定义AXI-IP核实验** 阅读数 8042

对zc702的custom AXI-IP的实验 博文 来自： [huamingshen](#)

**Vivado中AXI IP核的创建和读写逻辑分析** 阅读数 2万+

Vivado中AXI IP核的创建和读写逻辑分析总述本文包含两部分内容：1）AXI接口简介；2）AXI IP核的创建流程及读... 博文 来自： [limoon1212的博客](#)

**FPGA\_AXI4总线** 阅读数 4304

一)AXI总线是什么？ AXI是ARM 1996年提出的微控制器总线家族AMBA中的一部分。AXI的第一个版本出现在AM... 博文 来自： [yake827的专栏](#)

**通过状态机来对axi\_lite总线操作的仿真测试** 阅读数 250

上一篇《通过状态机来对axi\_lite总线读写操作》中，分享网友的代码。本工程为VIVADO 2017.04版本，先自定义A... 博文 来自： [keilzc的博客](#)

**MYIR-ZYNQ7000系列-zturn教程(19)：对axi\_stream核进行仿真以及axi\_stream总线的初步讲解** 阅读数 1226

我这里一共调用了两个自定义的IP都是基于axi\_stream的IP核，一个是主机master一个是从机slave，然后将这两个... 博文 来自： [taowei1314520的...](#)

**Zynq 7000 自定义ip 的仿真** 阅读数 1054

本文是《zynq 7000 自定义ip实验》一文的继续，也是《AXI4 协议分析》一文的基础上写出来的，如果没看这2文， ... 博文 来自： [曾立文的博文](#)

**参考设计，实现简单的AXI-M接口的DMA功能** 阅读数 62

`timescale 1 ns / 1 ps module myip\_v3\_S01\_AXI # ( // Users to add parameters here // User paramete... 博文 来自： [nigulasitangguo...](#)

**PS通过AXI-lite读取PL端数据** 阅读数 1934


1，创建AXI-lite总线的IP，并加上自己的逻辑， 注意：一定要编译（保证ip无逻辑和功能错误）； 同一个reg不能... 博文 来自： [zhangduojia的博客](#)


**[ip核]AXI\_Quad\_SPI学习** 阅读数 3306


标准spi使用的引脚： 相关寄存器： 过程描述： SPI通信过程的参考资料： https://blog.csdn.net/bytxl/article/det... 博文 来自： [qwerty的博客](#)


**ZYNQ--从入门到起飞--AXI总线接口分析(LITE)** 阅读数 2045


分析逻辑模块C\_S\_AXI\_DATA\_WIDTH表示数据总线的位宽C\_S\_AXI\_ADDR\_WIDTH表示数据地址的位宽Users to ad... 博文 来自： [ZKERK的博客](#)


2



















举报

**ZYNQ的AXI\_Lite 总线详解** 阅读数 5081

https://www.cnblogs.com/milinker/p/6474706.html12.1前言ZYNQ拥有ARM+FPGA这个神奇的架构，那么ARM... 博文 来自: lyfwill的博客

**Zynq-创建包含AXI4\_lite总线控制器的主从机通信系统(2)** 阅读数 463

原文: http://www.eefocus.com/antaur/blog/17-08/423754\_f75f7.html0.引言在上一节中，为了验证AXI4-Lite... 博文 来自: ningjinghai11的博...

**AXI学习笔记-1** 阅读数 279

本文首发于个人博客1.AXI总线结构AXI总线由5个通道构成：通道名称通道功能数据流向read address读地址通道主... 博文 来自: 月见樽

**AXI总线的一些知识** 阅读数 7909

AXI-stream总线简介-LDD本节介绍的AXI是个什么东西呢，它其实不属于Zynq，不属于Xilinx，而是属于ARM。它... 博文 来自: GoUpToTheSky的...

**AXI总线概述** 阅读数 447

AXI（Advanced eXtensible Interface）是一种总线协议，该协议是ARM公司提出的AMBA3.0中最重要的部分，是... 博文 来自: bleauchat的博客

**axi\_bfm仿真模型** 12-28

axi\_bfm\_ug\_examples.tar 仿真模型 下载

**ZYNQ入门(一)-AXI总线** 阅读数 2617

ZYNQ\_AXI总线ZYNQ\_AXI总线Accelerator Coherency Port, AXI\_ACP (加速一致性接口)High Performance,AXI\_H... 博文 来自: 今日你学左米啊

**AXI 调试波形记录** 阅读数 2523

升级了Evernote，把密码忘了，暂时登录不了。现在这记录下吧。 驱动代码： int main() { init\_platform(); xil\_p... 博文 来自: RZJMPB的博客

**xilinx UART-lite AXI4接口testbench** 阅读数 7040

升级到vivado2015后，为了升级以及zynq系列FPGA MPSOC考虑，xilinx后续IP将都支持AXI接口，但UART的设计... 博文 来自: shichaog的专栏

**仿真video in to axi\_stream和axi\_stream to video out** 阅读数 2056

最近做h264的压缩和解压要用到这两个IP,于是联合起来仿真一下,碰到一些配置问题,仿真时间有点慢,搞得很郁闷,以... 博文 来自: 苍白的手漆黑的刀

**AXI接口简介** 阅读数 4486

此部分，有参考他人帖子的内容，加上自己的理解，感恩原作者 1、 AXI（Advanced eXtensible Interface）协议主... 博文 来自: weixin\_42639919...

**axi stream 仿真模型，可用modelsim仿真** 07-18

axi stream 仿真模型，可用modelsim仿真 下载

**Zedboard自定义AXI总线IP详解（多图）** 阅读数 4140

Zedboard上Axi总线的IP核挂载实验（个人观点，多谢指正） 博文 来自: edo\_full的专栏

**AXI\_DMA调试说明** 阅读数 1695

程序源码https://github.com/fpgadeveloper/microzed-axi-dma 程序说明： https://blog.csdn.net/weilxuext/ar... 博文 来自: GoUpToTheSky的...

**AXI-Lite总线及 AXI4总线master和slave源码对应分析** 阅读数 4300

参考我的下载页： https://download.csdn.net/my 博文 来自: yanxiaopan的博客

**"30年---我与赛灵思FPGA的故事”： ZYNQ-7000使用总结(6) ——AXI接口简述** 阅读数 1万+

由 allan 于 星期五, 06/27/2014 - 17:35 发表在前面的几个例子中，我们经常会看到AXI接口或是总线，那么AXI到... 博文 来自: 青蛙@嘎嘎

**FPGA中的除法运算及初识AXI总线** 阅读数 1832

PGA中的硬件逻辑与软件程序的区别，相信大家在做除法运算时会有深入体会。若其中一个操作数为常数，可通过简... 博文 来自: blog

**Vivado下创建基于AXI-Lite的用户IP核** 阅读数 1846

http://comm.chinaaet.com/adi/blogdetail/37170.htmlVivado下创建基于AXI-Lite的用户IP核本文是为一位网友... 博文 来自: kebu12345678的...

python json java mysql pycharm android linux json格式



3 获赞

虚无缥缈vs威武

TA的个人主页 >

原创

粉丝

获赞

评论

访问

37

195

58

164

15万+

等级:

博客 4

周排名: 3万+



举报

关注

私信

全新ARM Cortex-M7系統開發實作

嵌入式物聯網工程師養成，  
嵌入式單晶片 M7+ARM  
RTOS+WiFi+藍牙+LoRa物  
通訊應用

最新文章

quartus II 12.1 使用教程（7） vga显示测试

MYIR-ZYNQ7000系列-zturn教程(27): lwip测试

quartus II 12.1 使用教程（6） ROM 测试

quartus II 12.1 使用教程（5） eeprom 读写测试

quartus II 12.1 使用教程（4） uart 测试

分类专栏

	VIVADO 安装教程	1篇
	quartus II	5篇
	三态门详解	
	quartus II 12.1 使用...	1篇
	ZYNQ7000	27篇

归档

2019年12月	1篇
2019年9月	1篇
2019年8月	5篇
2019年7月	2篇
2019年4月	1篇
2019年3月	2篇
2019年1月	1篇
2018年11月	1篇

展开

热门文章

VIVADO 安装教程

阅读数 84216

三态门详解

阅读数 15398

quartus II 12.1 使用教程（1） 怎样调用 PLL 核

阅读数 7556

MYIR-ZYNQ7000系列-zturn教程(17): 用 axi\_uart发送数据

阅读数 4156

MYIR-ZYNQ7000系列-zturn教程(9): 将 bit文件固化到QSPI\_Flash

阅读数 4055



2



举报

VIVADO 安装教程

rq8866: 缺License的小伙伴 链接: https://pan.baidu.com/s/11mjkpyERdUH3q5C\_TpfQxQ ...

FT232H如何使用jtag接口

taowei1314520: [reply]qq\_42662835[/reply]我是直接对eeprom里写数据进去的，数据我已经 ...

FT232H如何使用jtag接口

taowei1314520: [reply]sssshhhhhhhh[/reply]这个vivado有这个usb驱动也需要安装一下，你 ...

FT232H如何使用jtag接口

sssshhhhhhhh: 你好，插上电脑以后显示 USB Serial Conventor （仅配置了USB和EEPROM这 ...

MYIR-ZYNQ7000系列-z...

kuyunge: SPI一次是通信一个字节码？



- QQ客服
- kefu@csdn.net
- 客服论坛
- 400-660-0108

工作时间 8:30-22:00

[关于我们](#) [招聘](#) [广告服务](#) [网站地图](#)

京ICP备19004658号 经营性网站备案信息

公安备案号 11010502030143

©1999-2020 北京创新乐知网络技术有限公司

网络110报警服务

北京互联网违法和不良信息举报中心

中国互联网举报中心 家长监护 版权申诉

2

举报