به نام خدا

عنوان

پروژه درس مدارهای واسط

استاد

دکتر فصحتی

نام و شماره دانشجویی

محمد علیزاده ۴۰۱۱۰۶۲۴۴

مهدی بهرامیان ۴۰۱۱۷۱۵۹۳

در این پروژه، هدف ما پیاده‌سازی اینترکانکشن‌های مذکور و مقایسه عملکرد آن‌ها با یکدیگر بود. به دلیل سختی محک زدن عملکرد و مقایسه آن میان دو اینترکانکشن متفاوت، این مقایسه با توجه به روش پیاده‌سازی انجام می‌گیرد، چرا که فرکانس برد با توجه به مشترک بودن ثابت است و تنها اختلاف در پیاده سازی اینترکانکشن‌هاست. در واقع اینترکانکشنی که چرخه‌های بیشتری نیاز داشته باشد، واضحا عملکرد بدتری دارد. در ادامه در هربخش پیاده سازی هرکدام از اینتکانکشن‌ها به همراه منابع و توضیح مختصر پروتکل قرار داده شده است.

۱) Avalon stream

بر طبق منبع mnl_avalon_spec-683091-667068.pdf و همچنین Avalon® ۵.۲ Streaming Interface Signal Roles، این پروتکل را پیاده سازی کردیم. ویژگی کلی این پروتکل به این صورت است که پیام‌ها صرفا به صورت یکطرفه ارسال می‌شوند. به منظور رفع این کاستی، ما دو بار این پروتکل را پیاده‌سازی کردیم که یکی از مستر به اسلیو و دیگری از اسلیو به مستر است. روند کلی ارسال داده نیز در آن به این فرم است که از طرف مستر، برای تنظیم مقادیر A و B، ابتدا یک مقدار فرستاده می‌شود که مشخص می‌کند می‌خواهیم A را تنظیم کنیم یا B را. این مقدار برای A 1 و برای B 2 است. در ادامه، داده به صورت ۸ بیت ۸ بیت از جایگاه پرارزش به کم‌ارزش ارسال می‌شود. در بخش اول، سیگنال استارت پکت و ولید روشن شده و پس از این بخش استارت پکت خاموش می‌شود ولی ولید تا زمان انتهای ارسال روشن می‌ماند. در انتها نیز یک سیگنال اند پکت ارسال می‌شود که با توجه به ثابت بودن طول بسته‌های ما عملا بلا استفاده است. کد زده شده برای این بخش در دو فایل avalon_st_slave_wrapper.sv و avalon_st_master_wrapper.sv قرار گرفته است. تست بنچ مربوطه نیز با نام avalon_st_tb.sv قرار داده شده است. در ادامه صرفا برخی از بخش‌های مهم را قرار داده و توضیح می‌دهیم:

```
;reg [7:0] send_state
;reg [7:0] receive_state
;assign out_clk = clk
always @(posedge clk, negedge _rst) begin
if (!_rst) begin
;send_state <= 0
;startofpacket_out <= 0
;endofpacket_out <= 0
;data_out <= 0
;valid_out <= 0
```

```
end
else begin
Sending A //
if (send_state == 0) begin
if (ready_in == 1) begin
;send_state <= 1
;startofpacket_out <= 1
;endofpacket_out <= 0
;data_out <= 1
;valid_out <= 1
end
end
else if (send_state == 1) begin
if (ready_in == 1) begin
;send_state <= 2
;startofpacket_out <= 0
;endofpacket_out <= 0
;data_out <= A[31:24]
;valid_out <= 1
end
else begin
;send_state <= 0
;startofpacket_out <= 0
;endofpacket_out <= 0
;data_out <= 0
;valid_out <= 0
end
end
else if (send_state == 2) begin
if (ready_in == 1) begin
;send_state <= 3
;startofpacket_out <= 0
;endofpacket_out <= 0
;data_out <= A[23:16]
;valid_out <= 1
end
else begin
;send_state <= 0
;startofpacket_out <= 0
;endofpacket_out <= 0
;data_out <= 0
;valid_out <= 0
end
end
else if (send_state == 3) begin
```

```
if (ready_in == 1) begin
;send_state <= 4
;startofpacket_out <= 0
;endofpacket_out <= 0
;data_out <= A[15:8]
;valid_out <= 1
end
else begin
;send_state <= 0
;startofpacket_out <= 0
;endofpacket_out <= 0
;data_out <= 0
;valid_out <= 0
end
end
else if (send_state == 4) begin
if (ready_in == 1) begin
;send_state <= 5
;startofpacket_out <= 0
;endofpacket_out <= 1
;data_out <= A[7:0]
;valid_out <= 1
end
else begin
;send_state <= 0
;startofpacket_out <= 0
;endofpacket_out <= 0
;data_out <= 0
;valid_out <= 0
end
end
else if (send_state == 5) begin
if (ready_in == 1) begin
;send_state <= 6
;startofpacket_out <= 0
;endofpacket_out <= 0
;data_out <= 0
;valid_out <= 0
end
else begin
;send_state <= 0
;startofpacket_out <= 0
;endofpacket_out <= 0
;data_out <= 0
;valid_out <= 0
```

```
end
end
```

همانطور که از بخش فوق پیداست، ارسال مقادیر به صورت استیت به استیت صورت می‌گیرد. این استیت در رجیستری با نام send_state نگهداری شده و در هر مرحله بروز می‌شود.

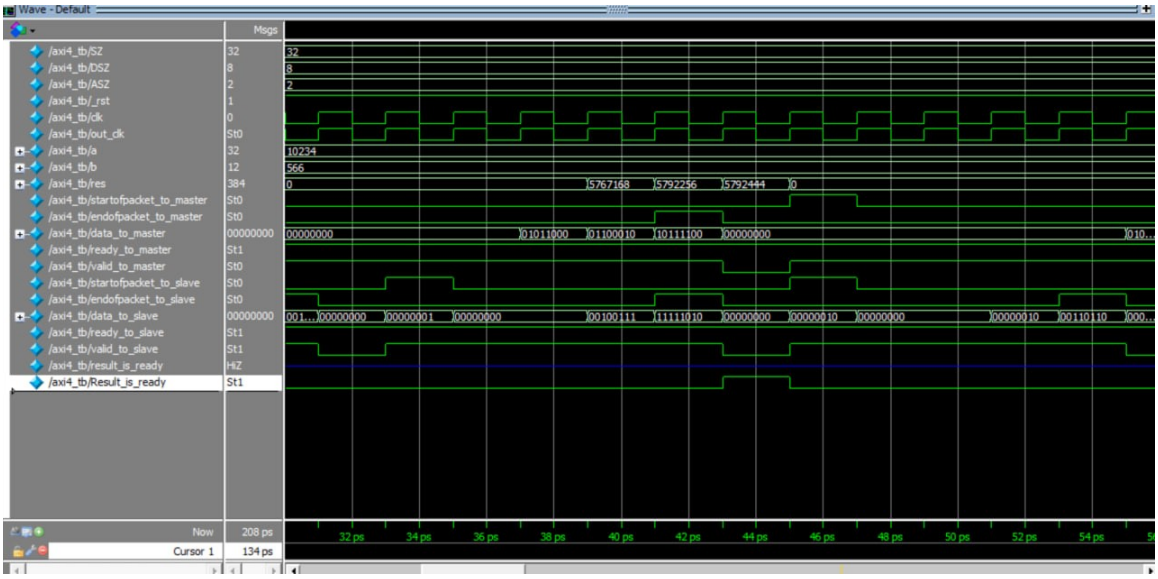
```
always @(posedge clk, negedge _rst) begin
if (!_rst) begin
;receive_state <= 0
;ready_res <= 0
;RES <= 0
;ready_out <= 1
end
else begin
Reading Result //
if (receive_state == 0) begin
if (valid_in == 1) begin
if (startofpacket_in == 1) begin
;receive_state <= 1
;ready_res <= 0
;RES[63:56] <= data_in
;ready_out <= 1
end
end
else begin
;receive_state <= 0
;ready_res <= 0
;RES <= 0
;ready_out <= 1
end
end
if (receive_state == 1) begin
if (valid_in == 1) begin
;receive_state <= 2
;ready_res <= 0
;RES[55:48] <= data_in
;ready_out <= 1
end
else begin
;receive_state <= 0
;ready_res <= 0
;RES <= 0
;ready_out <= 1
```

```
end
end
if (receive_state == 2) begin
if (valid_in == 1) begin
;receive_state <= 3
;ready_res <= 0
;RES[47:40] <= data_in
;ready_out <= 1
end
else begin
;receive_state <= 0
;ready_res <= 0
;RES <= 0
;ready_out <= 1
end
end
if (receive_state == 3) begin
if (valid_in == 1) begin
;receive_state <= 4
;ready_res <= 0
;RES[39:32] <= data_in
;ready_out <= 1
end
else begin
;receive_state <= 0
;ready_res <= 0
;RES <= 0
;ready_out <= 1
end
end
if (receive_state == 4) begin
if (valid_in == 1) begin
;receive_state <= 5
;ready_res <= 0
;RES[31:24] <= data_in
;ready_out <= 1
end
else begin
;receive_state <= 0
;ready_res <= 0
;RES <= 0
;ready_out <= 1
end
end
if (receive_state == 5) begin
```

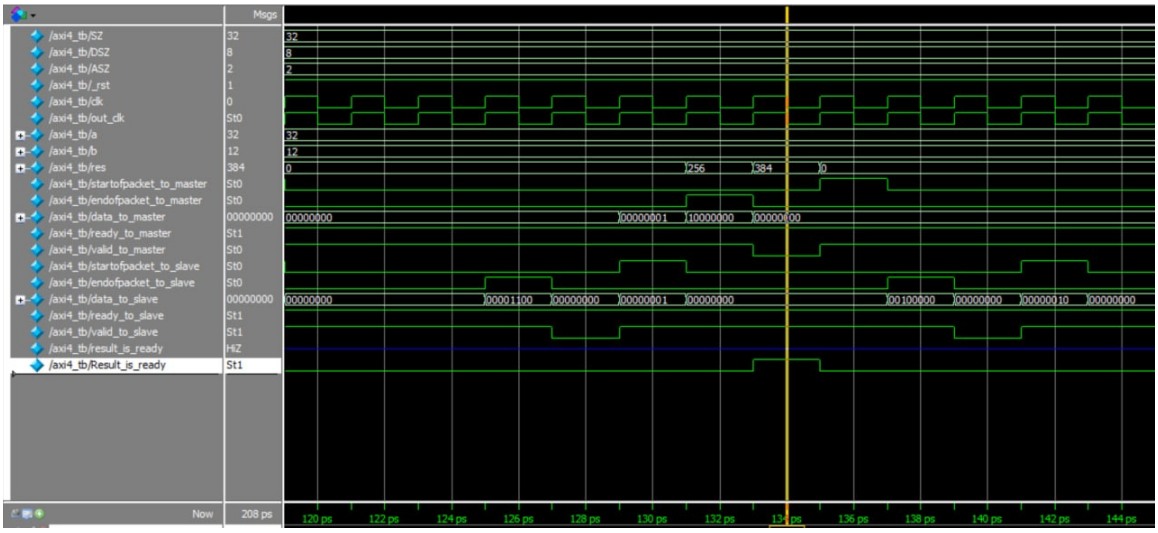
```
if (valid_in == 1) begin
;receive_state <= 6
;ready_res <= 0
;RES[23:16] <= data_in
;ready_out <= 1
end
else begin
;receive_state <= 0
;ready_res <= 0
;RES <= 0
;ready_out <= 1
end
end
if (receive_state == 6) begin
if (valid_in == 1) begin
;receive_state <= 7
;ready_res <= 0
;RES[15:8] <= data_in
;ready_out <= 1
end
else begin
;receive_state <= 0
;ready_res <= 0
;RES <= 0
;ready_out <= 1
end
end
if (receive_state == 7) begin
if (valid_in == 1) begin
;receive_state <= 0
;ready_res <= 1
;RES[7:0] <= data_in
;ready_out <= 1
end
else begin
;receive_state <= 0
;ready_res <= 0
;RES <= 0
;ready_out <= 1
end
end
end
end
```

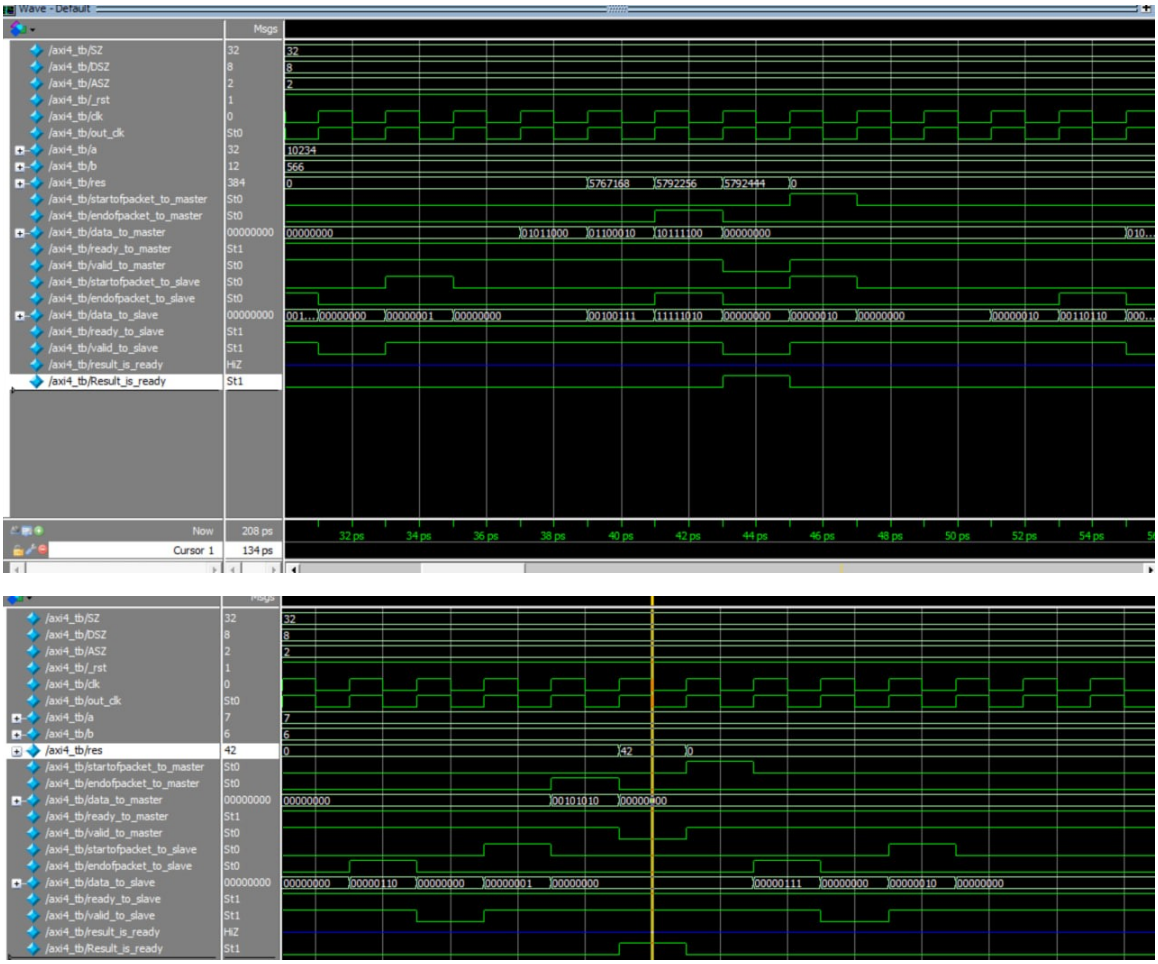
خواندن داده نیز به صورت بالا هندل شده است که به خوبی نمایانگر ساختار هندشیک گونه این اینترکانکشن است. تنها نکته حائز اهمیت این است که این کدها متعلق به مستر بودند. همانطور که گفتیم این پروتکل رویکرد تک جهته‌ای دارد. لذا پیاده سازی صورت گرفته در اسلیو عینا به همین شکل است تا امکان ارسال اسلیو به مستر داده را فراهم سازد.

مطابق شکل زیر، این پروتکل در زمان ۴۳ اولین ریزالت خود را محاسبه می‌کند. با توجه به اینکه از زمان ۸ از حالت ریست خارج شده و در حال پردازش است، متوجه می‌شویم زمان مورد نیاز بری انجام این محاسبه در بدترین حالت ۳۵ واحد طول کشید.



در ادامه چند نمونه عملیات ضرب صورت گرفته به کمک این اینترکانکشن را بررسی می‌کنیم.





۲) Avalon memory-mapped

این پروتکل بشدت شبیه پروتکل قبلی است با این تفاوت که دیگر یکطرفه نبوده و خودش ذاتا ویژگی‌های مورد نیاز ما را هندل کرده است. برای این پروتکل از دو داک Avalon® Memory Mapped Interface Signal Roles ۳.۲. و mnl_avalon_spec-683091-667068.pdf استفاده شده است. روند کلی این پروتکل نیز اینگونه است که داده با توجه به سیگنال عملیات خورده شده برروی باس متناسب قرار می‌گیرد. برای مثال برای خواندن داده، آدرس روی باس آدرس قرار میگیرد و پس از فعال شدن سیگنال read، مقدار خوانده شده در باس read قرار داده می‌شود. برای نوشتن نیز، داده در حال نوشته شدن برروی باس نوشتن قرار گرفته و پس از فعال شدن سیگنال نوشتن، داده نوشته می‌شود.

پیاده سازی مستر و اسلیو مربوطه در فایل avalon_mm_slave_wrapper.sv قرار دارد. تست بنچ مربوط به این فایل نیز با نام avalon_mm_tb.sv قرار گرفته است.

محتویات این فایل به شکل زیر است که در آن هر مقدار به تعدادی خانه حافظه پنداشته شـده و ماننـد خانه‌های حافظه با آن‌ها برخورد می‌شود.

```
)# module avalon_mm_slave_wrapper
parameter int SZ = 32
) (
,input _rst
,input clk
input [3:0]addr, // A1 A0 B1 B0 C3 C2 C1 C0
,input read
,input write
,input [15:0]write_data
output reg [15:0]read_data
;(

;reg [SZ-1:0] areg
;reg [SZ-1:0] breg

;wire [2*SZ-1:0] res

;wire ready, start


)# mult
SZ(SZ).
) main_module (
,a(areg).
,b(breg).
,res(res).
,start(start).
,rst(_rst)_.
,clk(clk).
ready(ready).
;(

always @(posedge clk, negedge _rst) begin
if (!_rst) begin
;areg <= 0
;breg <= 0
;read_data <= 0
end
```

```
else begin
if (read) begin
if (addr == 4) begin
;read_data <= res[15:0]
end
else if (addr == 5) begin
;read_data <= res[31:16]
end
else if (addr == 6) begin
;read_data <= res[47:32]
end
else if (addr == 7) begin
;read_data <= res[63:48]
end
else begin
;read_data <= 0
end
end
else if (write) begin
if (addr == 0) begin
;areg[15:0] <= write_data
end
else if (addr == 1) begin
;areg[31:16] <= write_data
end
if (addr == 2) begin
;breg[15:0] <= write_data
end
else if (addr == 3) begin
;breg[31:16] <= write_data
end
end
end
end
endmodule

)# module avalon_mm_master_wrapper
parameter int SZ = 32
) (
,input _rst
,input clk
,input [31:0]A
,input [31:0]B
,input [15:0]read_data
,output out_clk
```

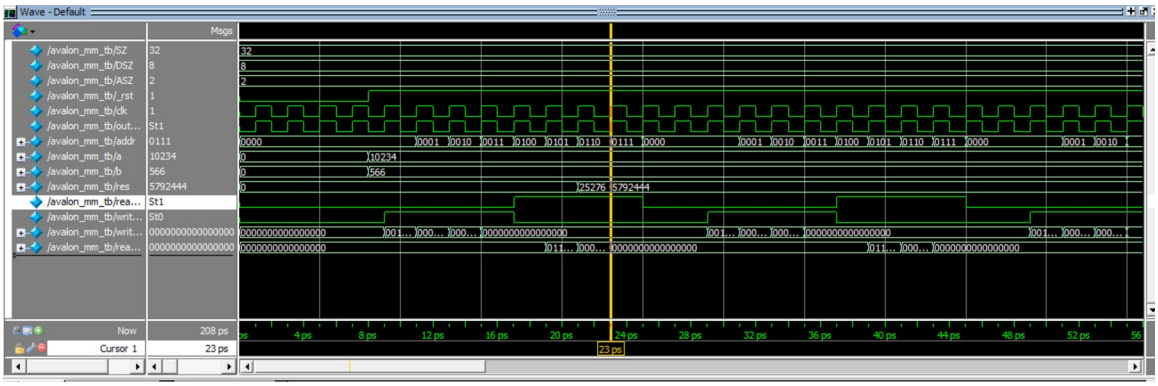
```
,output reg [63:0]res
output reg [3:0]addr, // A1 A0 B1 B0 C3 C2 C1 C0
,output reg read
,output reg write
output reg [15:0]write_data
;(

;assign out_clk = clk

;reg [7:0] state

always @(posedge clk, negedge _rst) begin
if (!_rst) begin
;state <= 0
;addr <= 0
;read <= 0
;write <= 0
;write_data <= 0
;res <= 0
end
else begin
if (state == 0) begin
;state <= 1
;addr <= 0
;read <= 0
;write <= 1
;write_data <= A[15:0]
end
else if (state == 1) begin
;state <= 2
;addr <= 1
;read <= 0
;write <= 1
;write_data <= A[31:16]
end
else if (state == 2) begin
;state <= 3
;addr <= 2
;read <= 0
;write <= 1
;write_data <= B[15:0]
end
else if (state == 3) begin
;state <= 4
;addr <= 3
```

```
;read <= 0
;write <= 1
;write_data <= B[31:16]
end
else if (state == 4) begin
;state <= 5
;addr <= 4
;read <= 1
;write <= 0
end
else if (state == 5) begin
;state <= 6
;addr <= 5
;read <= 1
;write <= 0
end
else if (state == 6) begin
;state <= 7
;addr <= 6
;read <= 1
;write <= 0
;res[15:0] <= read_data
end
else if (state == 7) begin
;state <= 8
;addr <= 7
;read <= 1
;write <= 0
;res[31:16] <= read_data
end
else if (state == 8) begin
;state <= 9
;addr <= 0
;read <= 0
;write <= 0
;res[47:32] <= read_data
end
else if (state == 9) begin
;state <= 0
;addr <= 0
;read <= 0
;write <= 0
;res[63:48] <= read_data
end
end
```

```
end
endmodule
```

همانطور که ملاحظه می‌شود، طرز برخورد مانند یک حافظه word-addressable با طول کلمه ۱۶ بیت است.

خروجی ویوفرم مربوط به این اینترکانشکن را نیز ببینیم:



همانطور که ملاحظه می‌شود، $display($time, " | master | ", "res : ", res) در این پروتکل در زمان ۲۳ اولین خروجی خود را می‌دهد، که با توجه به شروع از ۸، یعنی عملا ۱۵ واحد زمانی تا اولین خروجی خود زمان نیاز داشت.

۳) AXI4

به طور کلی پروتوکل های خانواده AXI از تجمیع تعدادی فرایند HandShaking تشکیل شده است. AXI4 به طور خاص از ۵ کانال استفاده میکند که آنهارا تحت عناوین AW (آدرس نوشتن) W(نوشتن) B (پاسخ نوشتن) AR(آدرس خواندن) R(خواندن) یاد میکند. برای پیاده‌سازی این پروتوکل نیز باید تحت ۲ ماژول، یکی برای master و دیگری برای slave استفاده کنیم.

```
else begin
    if (awready & awvalid) begin
        awaddr_reg <= awaddr;
        wpos <= awaddr * (SZ / DSZ);     ■ WIDTHTRUNC: Operator ASSIGNDLY expects 8
        awready <= 0;  // end addr read
        wready <= 1;  // begin data write
    end
    if (wready & wvalid) begin  // writing
        $display($time, " | slave | ", "rcv wdata : ", wdata, " ind : %1d", wpos);
        inregs[wpos] <= wdata;  // write to selected byte of input     ■ WIDTHTRUNC:
        wpos <= wpos + 1;
    end
    if (wready & wvalid & wlast) begin
        wready <= 0;  // end data write
        bresp  <= 1;  // begin write resp
        bvalid <= 1;
    end
    if (bready & bvalid) begin
        bvalid  <= 0;  // end write resp
        awready <= 1;  // begin addr read
    end
```

پیاده‌سازی مکانیزم HandShaking برای دریافت داده از ارباب.

```
if (arready & arvalid & mulready) begin
    $display($time, " | slave | ", "snd rdata : ", outregs[0], " ind : %1d", 0);
    araddr_reg <= araddr;
    arready <= 0;  // end addr read
    rdata <= outregs[0];  // begin data read
    rpos <= 0 + 1;
    rvalid <= 1;
    rresp <= 1;
    rlast <= 0;
end
if (rready & rvalid) begin
    $display($time, " | slave | ", "snd rdata : ", outregs[rpos], " ind : %1d", rpos);
    rdata <= outregs[rpos];     ■ WIDTHTRUNC: Bit extraction of array[7:0] requires 3 bi
    rpos  <= rpos + 1;
    rlast <= rpos == (2 * SZ / DSZ - 1);     ■ WIDTHEXPAND: Operator EQ expects 32 bits
end
if (rready & rvalid & rlast) begin
    rvalid  <= 0;  // end data read
    rdata   <= 0;
    rlast   <= 0;
    rresp   <= 0;
    arready <= 1;  // begin addr read
end
```

پیاده‌سازی مکانیزم HandShaking برای فرستادن داده به ارباب.

```
if (awvalid & awready) begin
    $display($time, " | master | ", "snd wdata : ", outregs[awaddr*4], " ind : %1d",
             awaddr * 4);
    awvalid <= 0;
    wdata   <= outregs[awaddr * 4];
    wpos    <= awaddr * 4 + 1;
    wvalid  <= 1;
    wlast   <= 0;
end
if (wvalid & wready) begin
    $display($time, " | master | ", "snd wdata : ", outregs[wpos], " ind : %1d", wpos);
    wdata <= outregs[wpos];     ■ WIDTHTRUNC: Bit extraction of array[7:0] requires 3 bit i
    wpos  <= wpos + 1;
    wlast <= ((wpos & 3) == 3);
end
if (wvalid & wready & wlast) begin
    wvalid <= 0;
    wlast  <= 0;
    wdata  <= 0;
    bready <= 1;
end
if (bready & bvalid) begin
    $display($time, " | master | ", "bresp : ", bresp);
    bready  <= 0;
    awvalid <= 1;
    awaddr  <= (awaddr + 1) & 1;
end
```

مکانیزم HandShaking برای فرستادن ورودی ها به برده.

```
    if (arvalid & arready) begin
        $display($time, " | master | ", "res : ", res);
        arvalid <= 0;
        rpos    <= 0;
        rready  <= 1;
    end
    if (rready & rvalid) begin
        $display($time, " | master | ", "rcv rdata : ", rdata, " ind : %1d", rpos);
        inregs[rpos] <= rdata;     ■ WIDTHTRUNC: Bit extraction of array[7:0] requires
        rpos <= rpos + 1;
    end
    if (rready & rvalid & rlast) begin
        rready  <= 0;
        arvalid <= 1;
    end
```

مکانیزم HandShaking برای خواندن نتیجه از برده.

در نهایت نتیجه آزمون این پروتوکل به شکل زیر است :

```
37 | master | snd wdata :   0 ind : 2
37 | master | rcv rdata :   0 ind : 4
37 | slave | rcv wdata :  49 ind : 1
37 | slave | snd rdata :   0 ind : 5
39 | master | snd wdata :   0 ind : 3
39 | master | rcv rdata :   0 ind : 5
39 | slave | rcv wdata :   0 ind : 2
39 | slave | snd rdata :   0 ind : 6
41 | master | snd wdata :  29 ind : 4
41 | master | rcv rdata :   0 ind : 6
41 | slave | rcv wdata :   0 ind : 3
41 | slave | snd rdata :   0 ind : 7
43 | master | bresp : 1
43 | master | rcv rdata :   0 ind : 7
43 | slave | snd rdata :   x ind : 8
45 | master | snd wdata :  29 ind : 4
45 | master | res :           517665995
45 | slave | snd rdata : 203 ind : 0
47 | master | snd wdata : 161 ind : 5
47 | master | rcv rdata : 203 ind : 0
```

همانطور که مشاهده میکنید، کل تاخیر رفت و برگشت این پروتوکل به اندازه ۴۵ است کـه تقریبا نیمی برای فرستادن مقادیر و نیم دیگر برای خواندن نتیجه صرف شده است.

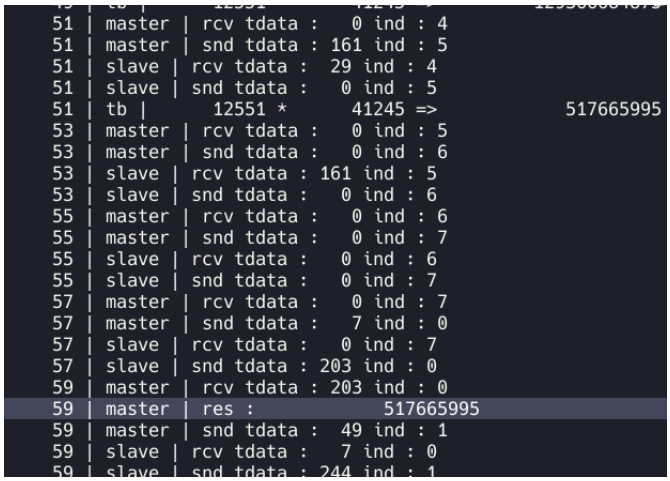
۴) AXI4 stream

این پروتوکل نیز بسیار شبیه AXI4 عادی است با این تفاوت که این پروتوکل دیگر دوطرفه نیست و تنها یک سویه است. علاوه بر این تنها یک کانال تحت عنوان T(ارسال) داریم کـه سـیگنال های اصـلی آن، Tdata, Tvalid, Tready و Tlast هستند. در اینجا با توجه به اینکه میخواهیم به ضرب کننده داده هارا فرستاده و نتیجه را بگیریم، به ۲ تا از این کانال‌ها نیاز داریم که یکی را از پردازنده بـه ماژول و دیگری را از ماژول به پردازنده در نظر میگیریم. حال باقی مراحل مشابه قبل است.

```
e begin
if (tready_to_slave & tvalid_to_slave) begin
    $display($time, " | slave | ", "rcv tdata : ", tdata_to_slave, " ind : %1d", wpos);
    inregs[wpos] <= tdata_to_slave;     ■ WIDTHTRUNC: Bit extraction of array[7:0] requ
    wpos <= tlast_to_slave ? 0 : wpos + 1;
end

if (tready_to_master & tvalid_to_master) begin
    $display($time, " | slave | ", "snd tdata : ", outregs[rpos], " ind : %1d", rpos);
    tdata_to_master <= outregs[rpos];     ■ WIDTHTRUNC: Bit extraction of array[7:0] re
    rpos <= (rpos + 1) & 7;
    tlast_to_master <= rpos == 7;
end
```

به طور کلی مکانیزم ارتباطی این پروتوکل به این شکل در ارباب و برده پیاده شده است. به طوری که به طور پیوسته هر طرف برای طرف دیگر مقادیر مورد نظر خود را میفرستد.

در نهایت نتیجه آزمون این پروتوکل نیز به شکل زیر است :

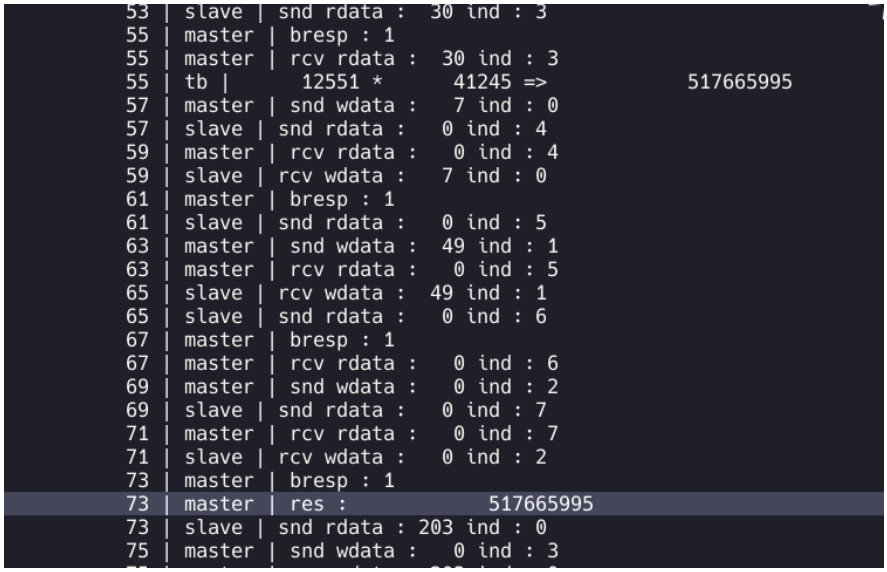


همانطور که مشاهده میکنید، این پروتوکل نیز در زمان ۵۹ به نتیجه رسیده است. البته مقداری زود تر نیز در زمان ۴۳ به نتیجه رسیده بود اما به دلایلی از جمله استفاده از ورودی ناقص برای محاسبه، به نتیجه اشتباه رسیده بود.

۵) AXI4 lite

این پروتوکل دقیقاً مانند AXI است تنها با این تفاوت که مدارات این پروتوکل بسیار ساده‌تر از AXI هستند و از روش burst پشتیبانی نمیکند. با توجه به این تغییر سیگنال های rlast و wlast را ندارد. با توجه به همین مساله، صرفاً باید تغییر جزئی به سیستم آدرس دهی خود دهیم و همه چیز به درستی کار خواهد کرد.

نتیجه آزمون این پروتوکل نیز به شکل زیر است :



همانطور که در نتیجه میبینید، این آزمون نیز در زمان ۷۳ به نتیجه رسیده که با توجه به نبـود قـابلیت burst پیشبینی میشد. به این ترتیب سربار و تاخیر فرستادن پیام ها بالا رفته است.