Computer Organization and Architecture Course Design

A Parallel Output Controller

04018512

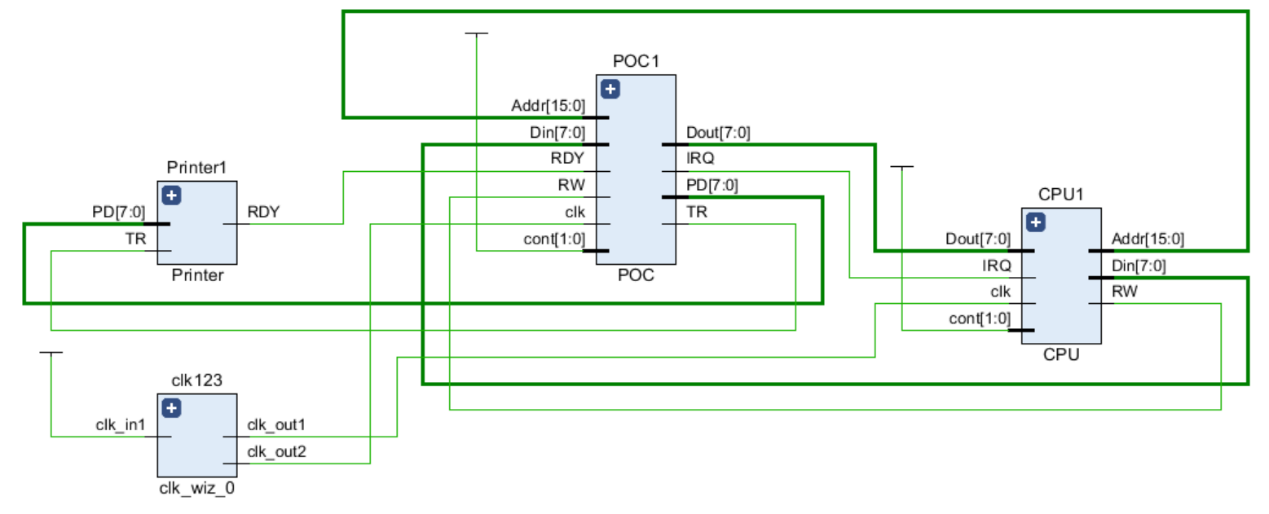
Shang Chen

1. Purpose

The purpose of this project is to design and simulate a parallel output controller (POC) which acts an interface between system bus and printer.

1. Tasks

POC has the ability conduct programmed IO and interrupt-driven IO.

1. Overall connections
2. Design description of input

Clk:100Mhz clock

Clk\_CPU:500Mhz clock of CPU

Clk\_POC:50Mhz clock of POC

Dout:Data transmitted from POC

Din:Data transmitted from CPU

Addr:address select signal from CPU

RW:control signal of three-state buffer Data

IRQ:interrupt request signal

RDY:ready signal of printer

TR:transfer request from POC

PD:parallel data from POC

Cont:control signal, select programmed IO or interrupt-driven IO

1. Simulation results

Programmed IO:

图形用户界面

中度可信度描述已自动生成

Initialization

BR=Data SR7=0

Handshaking with printer

SR7=0

SR7=1

4000ns:cont comes 01 from 00, CPU conduct programmed IO, SR7 is currently 1

4005ns:BR=Data, SR=0, prepare for the handshaking

4025ns:POC generates a pulse on TR, RDY=0, buffer gets Data 0x12

4125ns:RDY returns 1

4605ns:handshaking is finished, SR7=1

interrupt-driven IO：

Initialization

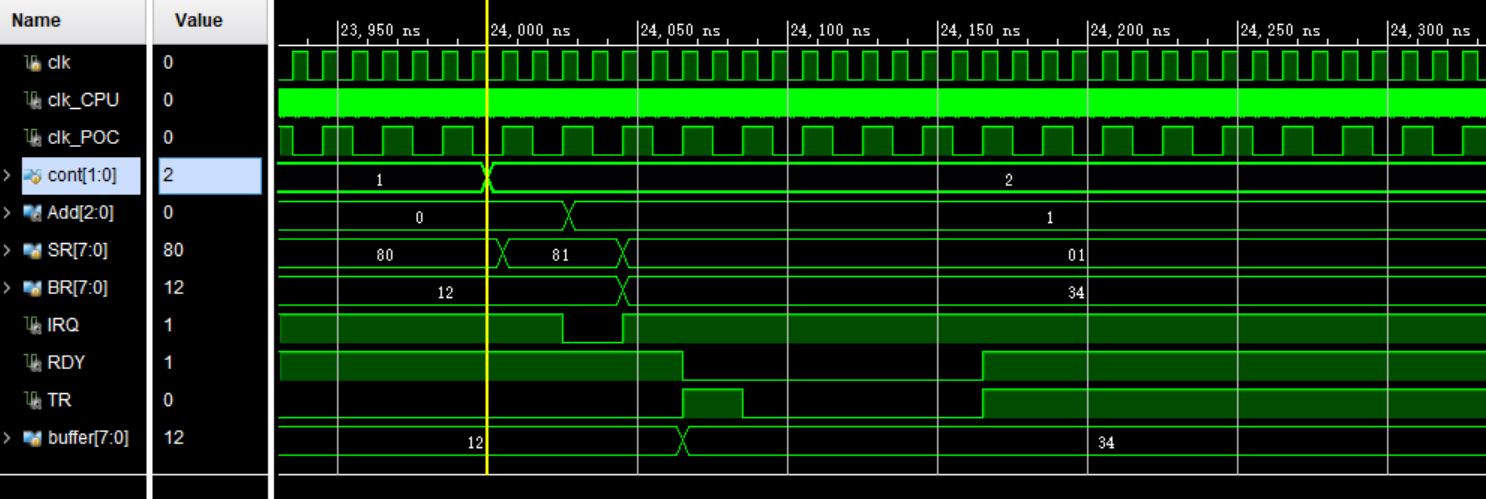
IRQ=0

BR=Data SR7=0

Handshaking with printer

SR7=1 SR0=1

SR7=1



24000ns:cont=10, CPU conduct interrupt-driven IO

24005ns:SR0=1

24025ns:POC generate IRQ signal, IRQ=0

24045ns:CPU finds that SR=0x81, BR=Data, SR7=0

24065ns:handshaking begins. POC generates TR pulse, RDY=0, buffer=Data

24165ns:after waiting for a few time, RDY=1

1. Conclusions and discussions

As is shown in the above, the simulation is in line with expectations and I will go further for some design details.

1. clock

We all know that CPU is device with fast speed while POC with slow speed, so I choose to generate a 100Mhz clock signal and use the IP core clock wizard to generate a 500Mhz clock for CPU and a 50Mhz clock for POC.

I have considered whether to generate the POC clock from CPU or place all the clock outside the two components. After carefully observe modern motherboards, it is found that clock of CPU, pcie devices, memory and others are all provided by a particular clock circuit where crystal oscillator generates basic impulse and clock generator IC multiply the basic signal with required times.

1. Data bus

In the slides, data bus is the input and output of both CPU and POC. However, data bus has only one line, therefore data there is a bidirectional bus. Bidirectional bus, actually, can be considered as tristate buffer. In this POC example, RW can act as the control signal of this tristate gate. In contrast to common type input or output, Data has the special type inout.

1. Appendix

top.v

`timescale 1ns / 1ps

module top;

reg clk;

wire clk\_CPU;

wire clk\_POC;

wire [15:0] Addr;

wire[7:0] Din,Dout;

wire RW;

wire IRQ;

reg [1:0] cont;

wire RDY;

wire TR;

wire [7:0] PD;

initial

begin

clk=0;cont=2'b00;

forever #5 clk=~clk;

end

initial

begin

#4000 cont=2'b01;

#20000 cont=2'b10;

end

clk\_wiz\_0 clk123

(

// Clock out ports

.clk\_out1(clk\_CPU),

.clk\_out2(clk\_POC),

// Clock in ports

.clk\_in1(clk)

);

CPU CPU1

(

.Addr(Addr),

.Din(Din),

.Dout(Dout),

.RW(RW),

.clk(clk\_CPU),

.IRQ(IRQ),

.cont(cont)

);

POC POC1

(

.Addr(Addr),

.Din(Din),

.Dout(Dout),

.RW(RW),

.clk(clk\_POC),

.IRQ(IRQ),

.cont(cont),

.RDY(RDY),

.TR(TR),

.PD(PD)

);

Printer Printer1

(

.RDY(RDY),

.TR(TR),

.PD(PD)

);

endmodule

CPU.v

`timescale 1ns / 1ps

module CPU(Addr, Din, Dout, RW, clk, IRQ, cont);

output Addr;

//inout Data;

output Din;

input Dout;

output RW;

input clk;

input IRQ;

input cont;

wire [15:0] Addr;

wire [7:0] Din;

reg [7:0] din=8'h12;

wire [7:0] Dout;

wire [7:0] Data;

wire clk;

reg RW=0;

wire [1:0] cont;

wire IRQ;

reg [15:0] Add=16'h0000;

assign Din=din;

always@(posedge clk)

begin

if(cont==2'b01)

begin

RW=0;

end

end

always@(posedge clk)

begin

if(IRQ==0)

begin

Add=16'h0001;

din=8'h34;

end

end

assign Addr=Add;

endmodule

POC.v

`timescale 1ns / 1ps

module POC(Addr, Din, Dout, RW, clk, IRQ, cont, RDY, TR, PD);

input [15:0]Addr;

//inout Data;

input Din;

output Dout;

input RW;

input clk;

output IRQ;

input cont;

input RDY;

output TR;

output PD;

wire [7:0] Din;

wire [7:0] Dout;

//wire [7:0] Data;

wire clk;

wire RW;

wire [1:0] cont;

wire[2:0] Add=Addr[2:0];

reg [7:0] SR=8'b10000000;

reg [7:0] BR=8'b00000000;

wire RDY;

wire TR;reg tr=0;

wire [7:0] PD=BR;

wire IRQ;reg irq=1;

assign IRQ=irq;

always@(posedge clk)

begin

if(RDY==1&&BR==8'h12&&SR[7]==0&&SR[0]==0)

begin

tr=1;

#20 tr=0;

end

end

always@(posedge clk)

begin

if(RDY==1&&BR==8'h34&&SR[7]==0&&SR[0]==1)

begin

tr=1;

#20 tr=0;

end

end

always@(posedge clk)

begin

if(cont==2'b01)

begin

if(SR[7]==1&&BR==8'h00)

begin

BR=Din;

SR[7]=0;

end

if(SR[7]==0&&cont==2'b01)

begin

end

end

end

always@(posedge clk)

begin

if(SR[0]==1&&SR[7]==1)

begin

irq=0;

end

end

always@(posedge clk)

begin

if(SR[0]==1&&SR[7]==0)

begin

end

end

always@(posedge clk)

begin

if(Add==3'b001)

begin

BR=Din;

SR[7]=0;

end

end

always@(posedge clk)

begin

if(cont==2'b01) SR[0]=0;

if(cont==2'b10&&BR==8'h12)

begin

SR[0]=1;

SR[7]=1;

end

end

always@(posedge clk)

begin

if(IRQ==0) SR[7]=0;

end

always@(posedge clk)

begin

if(Din==8'h34&&RDY==1)

begin

irq=1;

end

end

always@(posedge clk)

begin

if(cont==2'b01&&TR==1)

begin

#1000 SR[7]=1;

end

end

always@(posedge clk)

begin

if(BR==8'h12)

begin

#600 SR[7]=1;

end

end

endmodule

Printer.v

`timescale 1ns / 1ps

module Printer(PD, TR, RDY);

input PD;

input TR;

output RDY;

wire TR;

wire RDY;reg rdy=1;

reg [7:0] buffer=8'b00000000;

wire [7:0] PD;

assign RDY=rdy;

always@(\*)

begin

if(TR==1)

begin

rdy=0;

buffer=PD;

#500 rdy=1;

end

end

always@(\*)

begin

if(buffer==8'h34)

begin

#100 rdy=1;

end

end

always@(\*)

begin

if(buffer==8'h12)

begin

#100 rdy=1;

end

end

endmodule

Dinout.v

`timescale 1ns / 1ps

module Dinout(din,z,clk,dout,dinout);

input [7:0] din;

input z;

input clk;

output [7:0] dout;

inout [7:0] dinout;

reg [7:0] dout;

reg [7:0] din\_reg;

assign dinout = (!z)?din\_reg:8'bz;

always @(posedge clk)

begin

if(!z)

din\_reg=din;

else

dout=dinout;

end

endmodule