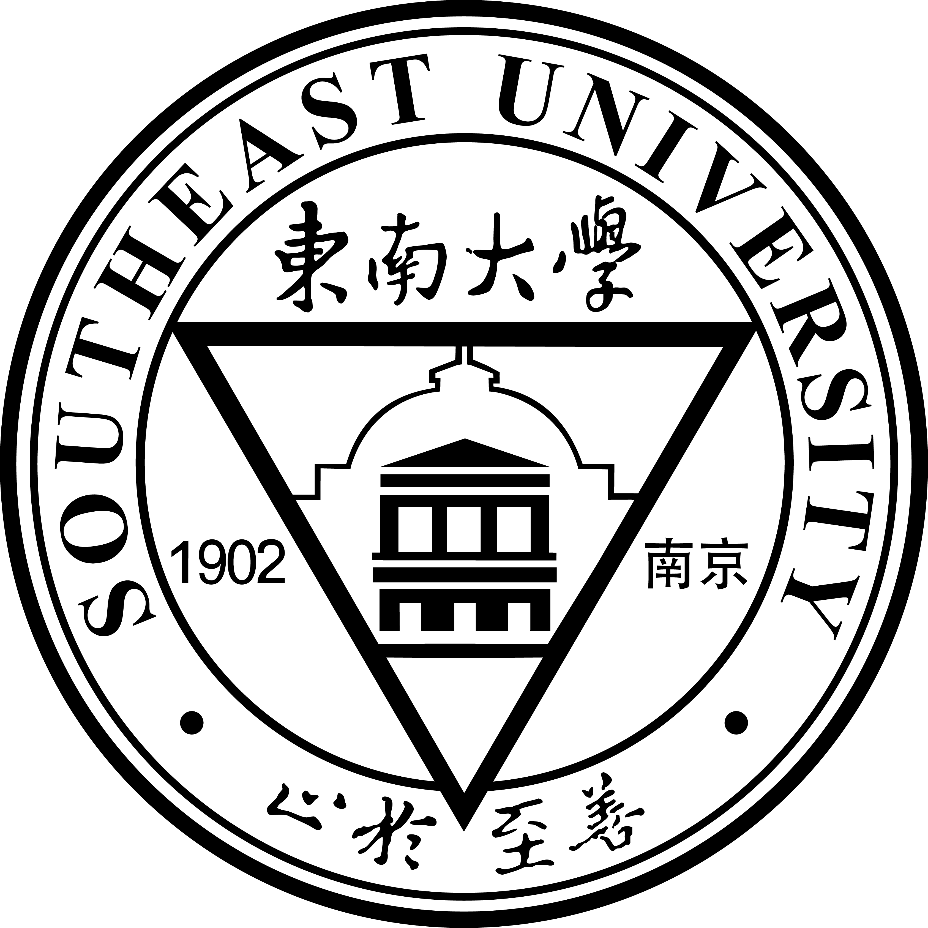
**Computer Organization and Architecture**



**School of Information Science and Engineering**

**Southeast University**

**March 2019**

A parallel output controller (POC)

Group members :

04016613 Wencheng Qiu

04016640 Jialong Xue

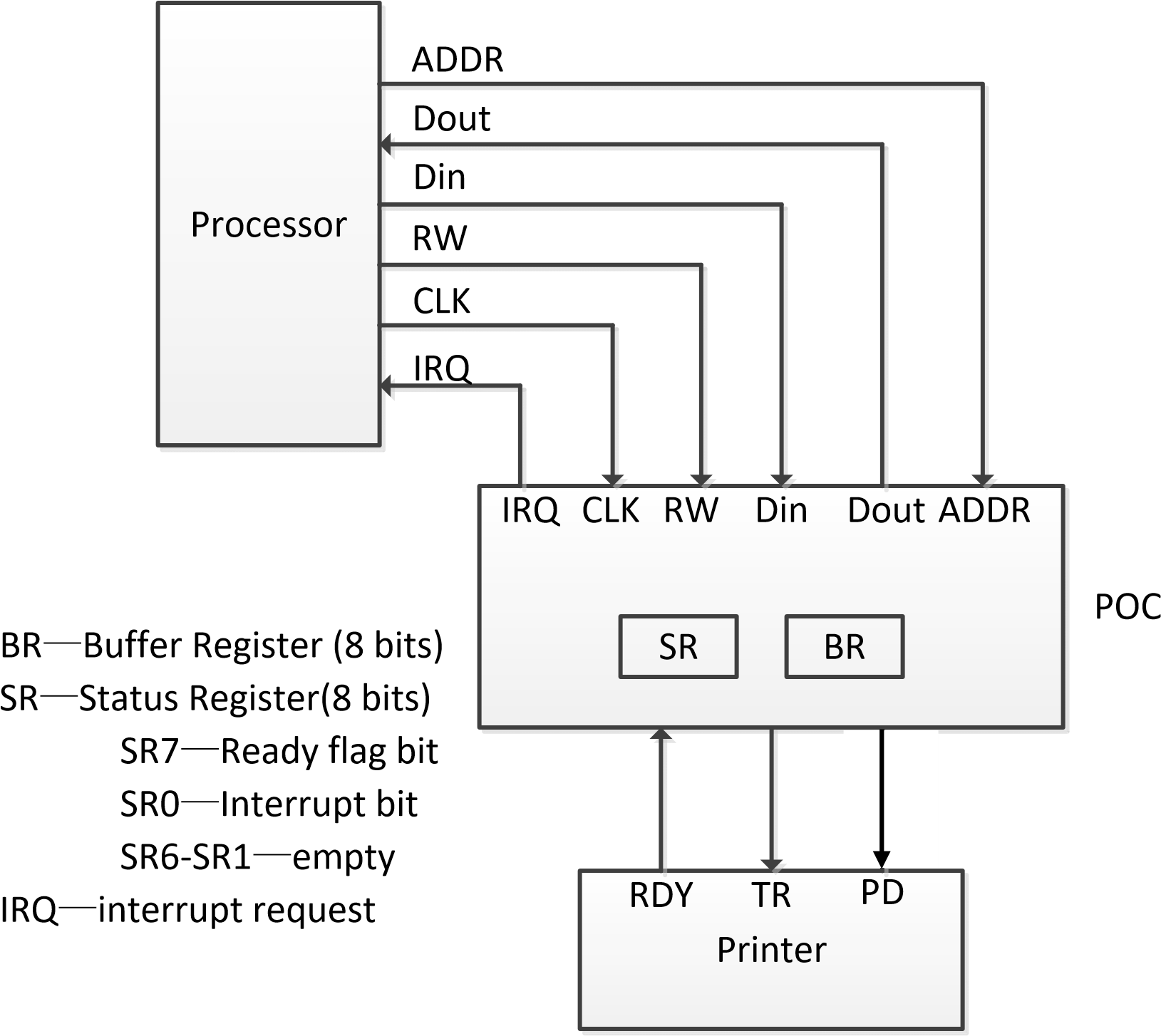
# 1.Purpose

The purpose of this project is to design and simulate a parallel output controller (**POC**) which acts an interface between system bus and printer. The Xilinx Vivado is recommended and provided for simulation. Please refer to **William Stallings** “**Computer Organization and Architecture, Designing for Performance**”**.**

# 2.Tasks

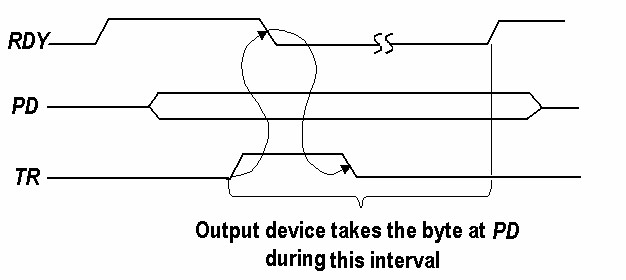
## 2.1 Main tasks

**POC** is one of the most common **I/O** modules, namely the parallel output controller. It plays the role of an interface between the computer system bus and the peripheral (such as a printer or other output devices).



**Figure 1** Printer Connection

**Figure 1** shows the connecting of a printer to the system bus through the **POC**. The communication between **POC** and the printer is controlled by a “handshake” protocol illustrated in **Figure 2**.



**Figure 2** The handshake timing diagram between POC and the printer

The handshaking process is described as follows**:** When the printer is ready to receive a character, it holds **RDY**=**1**.The **POC** then hold a character at **PD** (parallel data) port and generate a one-cycle pulse at the port **TR** (transfer request). When detecting the effective TR signal, the printer will change **RDY** to **0**, take the character at **PD** and hold the **RDY** at **0** until the character has been printed (e.g. 5 or 10ms), then set **RDY**=**1** again when it is ready to receive the next character. (Suppose the printer has only a one character “buffer” register, so that each character must be printed before the next character is sent).

The buffer register **BR** is used to temporarily hold a character sent from the processor, which character will be transferred to the printer later. The status register **SR** is used for two control functions**:** **SR7** serves as a ready flag to indicate POC is ready or not to receive a new character from the processor, and **SR0** is used to enable the interrupt requests sent by **POC.** In interrupt mode, If **SR0=1**, then **POC** will send an interrupt request signal to processor when it is ready to receive a character (i.e., when **SR7=1**).

If **SR0=0**, then **POC** will not interrupt. The other bits of **SR** are not used and empty.

The transfer of a character to POC via the system bus proceeds as follows.

***In polling mode, SR0 is always 0.***

The processor selects SR by accessing the relative address, then reads SR register, if SR7=1, the processor selects BR and writes a character into BR, then processor clears SR7 to indicate that the new character has been written into BR and not printed yet. When POC detects that SR7 is set to 0, POC then proceeds to start the handshaking operations with the printer. After sending character to printer, POC sets the SR7 to 1, which indicates POC is ready to receive another character from the processor. The transfer cycle can now repeat. During the handshaking operations between POC and printer, the processor continues to fetch and execute instructions. If it happens to read SR, it will find SR7=0 and hence will not attempt to send another character to the POC.

***In interrupt mode, SR0 is always 1.***

After sending character to printer, POC sets the SR7 to 1, since SR0=1, the interrupt request signal (IRQ) is set to 0, which indicate an effective interrupt signal to the processor. When the processor detects the effective IRQ signal, the processor directly selects BR and writes a character into BR, and then the processor sets the SR7 to 0, which indicates that the new character has been written into BR and not printed yet. When POC detects that SR7 is set to 0, POC then proceeds to start the handshaking operations with the printer. After sending character to printer, POC sets the SR7 to 1, which indicates POC is ready to receive another character from the processor. The transfer cycle can now repeat. During the handshaking operations between POC and printer, the processor does not try to access POC until it receives the interrupt request signal.

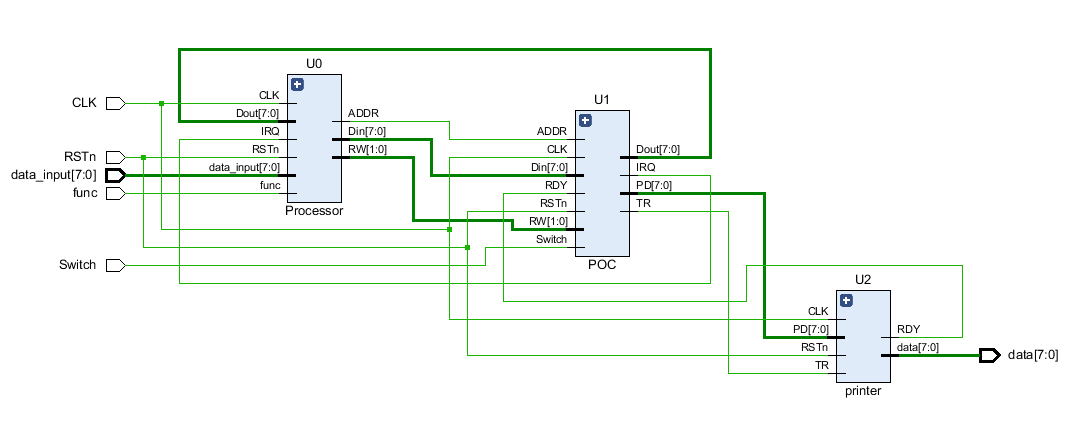
## 2.2 Other design requirements

1. The POC module must support both the polling mode and interrupt mode, users can switch one between them according to their needs.

2. For effective simulation verification, it is recommended to design a Processor module and simulate it jointly with the POC.

3. The printer needs to be designed separately to match the POC verification.

# 3.Top module form

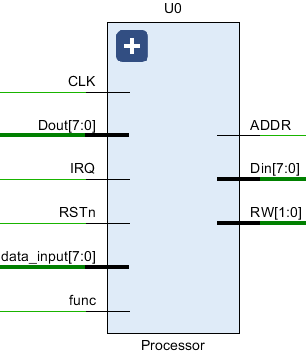


**Figure 3** The Top Schematic of the simulated printer and POC

There are three modules in the figure: the processor, the Printer and the POC. According to the hints in the direction, we know that in order to simulate the POC by computer, a simulated printer should be written.

# 4. Design of Each Module

## 4.1 Processor



**Figure 4** The Schematic Symbol of CPU

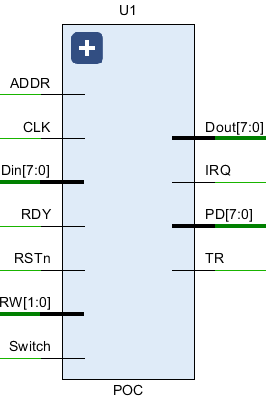
### Module input

|  |  |
| --- | --- |
| Pin | Function |
| CLK | Clock |
| Dout[7:0] | Data from POC to Processor. |
| IRQ | Interrupt Request, 0 valid. |
| RSTn | Reset, 0 Valid |
| Data\_input[7:0] | Data to print from other peripheral. |
| func | Whether there are data to be printed, 1 valid. |

### Module output

|  |  |
| --- | --- |
| Pin | Function |
| ADDR | Target Address Indicator. 0 – SR, 1 – BR. |
| Din[7:0] | Data from processor to POC |
| RW[1:0] | Read/Write Operation Indicator. 00 – do not read or write, 10 – Processor read data from POC, 11 – Processor write data to POC. |

## 4.2 POC



**Figure 5** The Schematic Symbol of POC

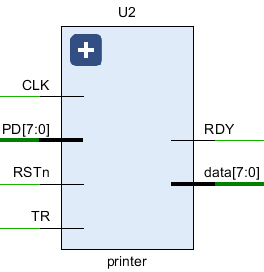
### Module input

|  |  |
| --- | --- |
| Pin | Function |
| ADDR | Target Address Indicator.0 – SR, 1 – BR. |
| CLK | Clock. |
| Din[7:0] | Data from processor to POC. |
| RDY | Peripheral Ready Indicator. |
| RSTn | Reset, 0 Valid. |
| RW[1:0] | Read/Write Operation Indicator. 00 – do not read or write, 10 – Processor read data from POC, 11 – Processor write data to POC. |
| Switch | Switch between pulling mode(0) and interrupt mode(1). |

### Module output

|  |  |
| --- | --- |
| Pin | Function |
| Dout[7:0] | Data from POC to Processor. |
| IRQ | Interrupt Request,0 valid. |
| PD[7:0] | Data Send to Printer. |
| TR | Transfer Request to Peripheral. |

## 4.3 Printer



**Figure 6** The Schematic Symbol of Printer

### Module input

|  |  |
| --- | --- |
| Pin | Function |
| CLK | Clock |
| PD[7:0] | Data Send From POC |
| RSTn | Reset, 0 Valid. |
| TR | Transfer Request From POC |

### Module Output

|  |  |
| --- | --- |
| Pin | Function |
| RDY | Printer Idle Indicator |
| data[7:0] | Show what the Printer is printing |

# 5.Simulation Results

## 5.0 Introduction

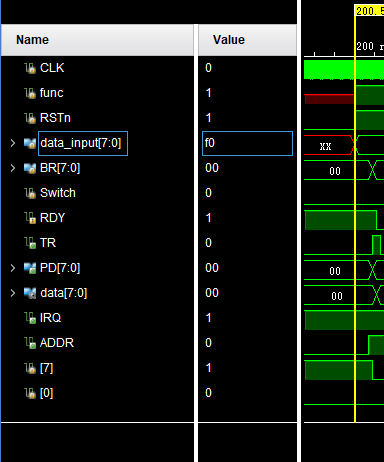
The clock is set to 1ns, at the beginning, RSTn is 0, the whole system is idle. At 200ns RSTn and func are both set to 1 and data\_input is 11110000, the printing process begins.



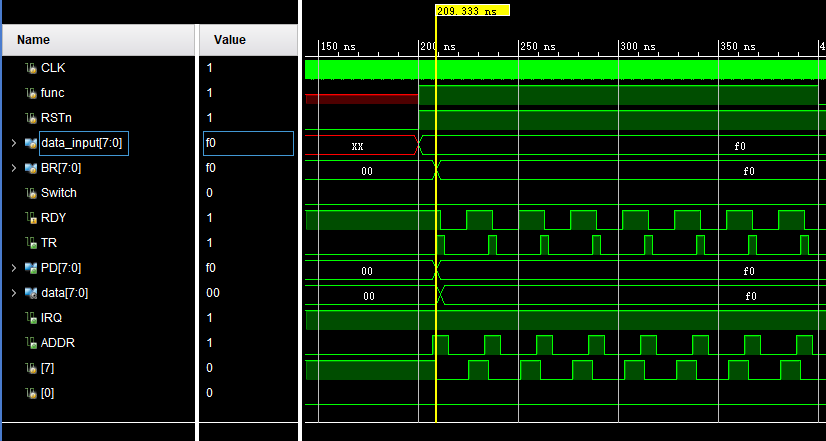
**Figure 7-0** The initial mode

## 5.1 Polling mode

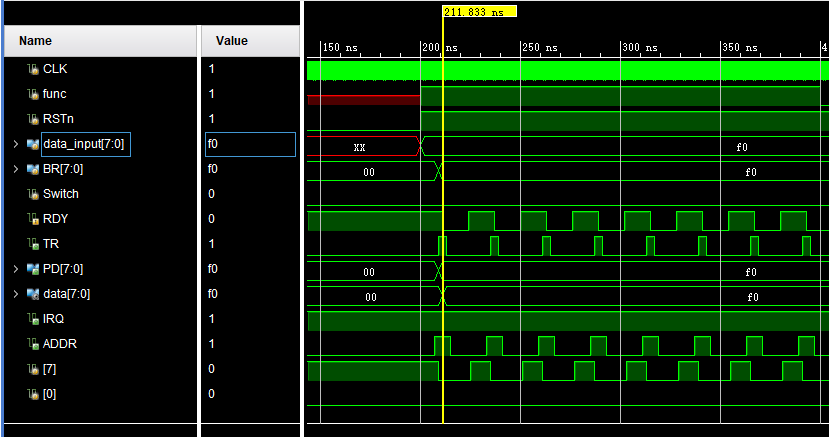
The Switch is 0 at the beginning, so the system works by default in the polling mode.



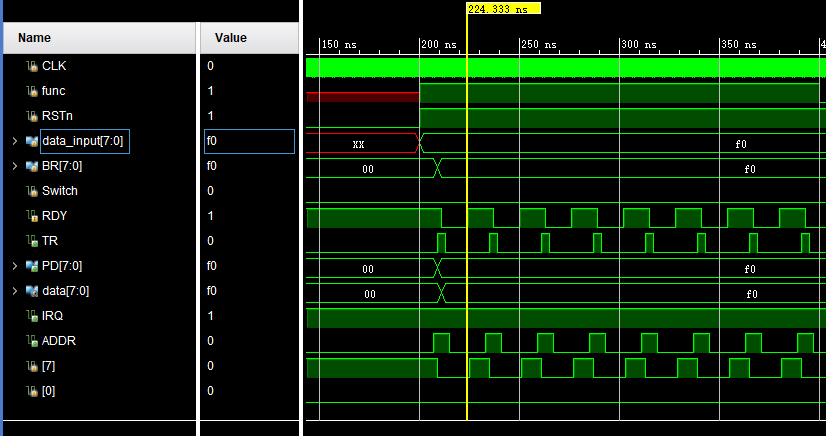
**Figure 7-1** : CPU Sets ADDR to be 0 to get POC Status(SR).



**Figure 7-2** : POC is idle, CPU sends new data to POC Buffer Register and refresh POC-SR. then POC puts data on PD, starting handshake with printer.

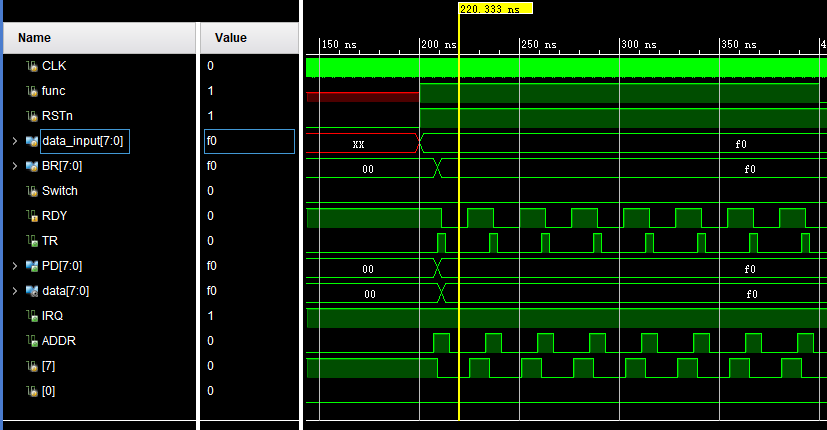


**Figure 7-3**: TR is set high, the printer gets the data, then start printing. RDY turn to 0.



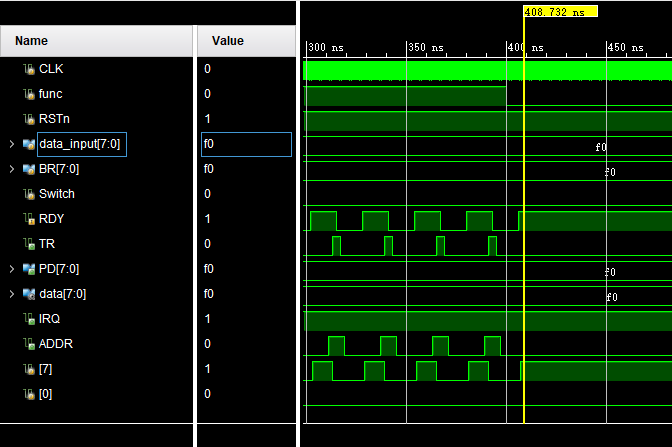
**Figure 7-4**: Printing process finished, RDY turn high to inform POC that printer

is idle. Then SR7 is set to 1.



**Figure 7-5**: When POC and printer is busy, CPU query will be denied, so that

CPU will not send new data to POC.

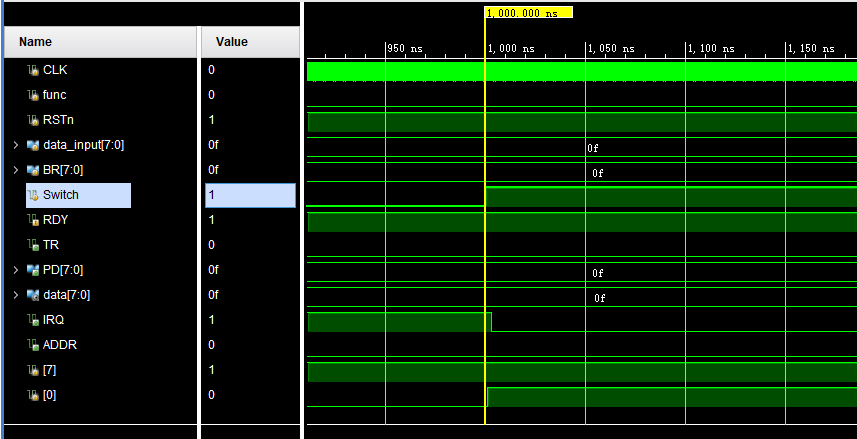


**Figure 7-6** When there is nothing to be printed, the func is set to 0 and the print stops.

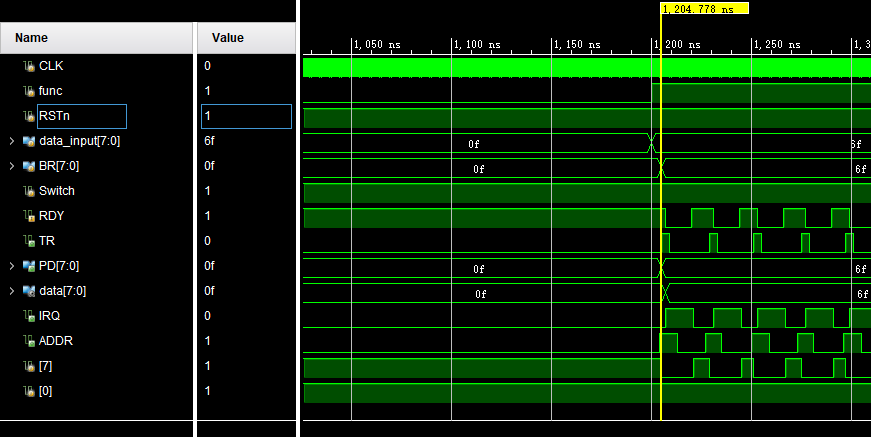


**Figure 7-7** New data needs to be printed and func is set to 1 again.

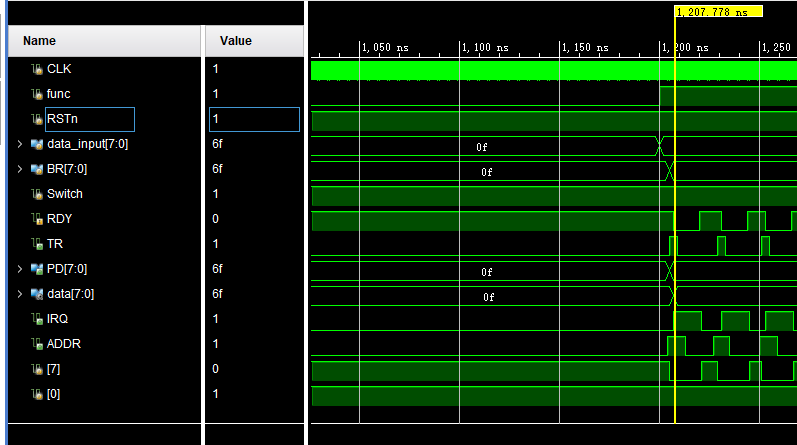
## 5.2 Interrupt mode



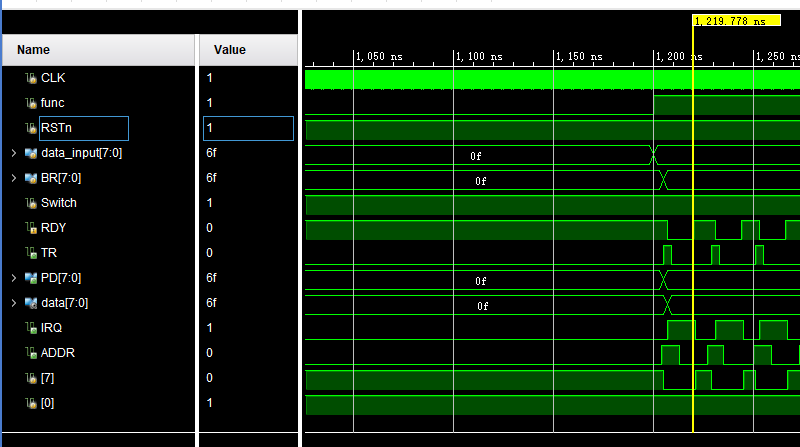
**Figure 8-1** Switch is set to 1 and POC turns to interrupt mode(SR0 is always 1).



**Figure 8-2** : Without access to POC-SR, CPU writes data to BR directly, then CPU refresh POC-SR to 0 .

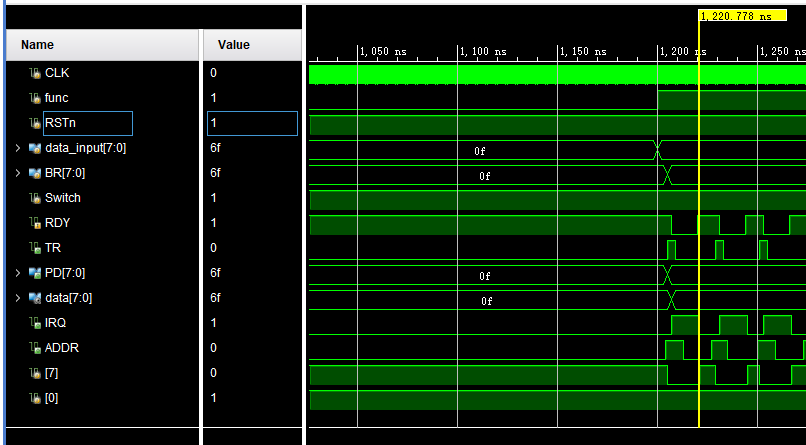


**Figure 8-3** : Handshake between POC and the printer. The printer starts printing.



**Figure 8-4** : Printing process finished, RDY turn high to inform POC that printer

is idle.



**Figure 8-5** : POC set IRQ valid to inform CPU that the printer is available.

# 6.Conclusions and Discussions

First of all, we would like to thank Teacher Zhang for the explanation. Through this experiment, we have successfully completed the design of the POC and realized the functions required by the title. It can switch between the interrupt mode and the polling mode. At the same time, in the process of design, we also found our shortcomings, in the second CPU design，we will learn from the experience and apply the knowledge of COA learned in the last semester to practice, strengthen our understanding of the whole computer architecture.

# 7.Appendix

## 7.1 Source codes

|  |
| --- |
| **module** Processor**(**  **input** func**,**  **input** CLK**,**  **input** RSTn**,**  **input** IRQ**,**//Interrupt request  **input** **[**7**:**0**]** data\_input**,**  **input** **[**7**:**0**]** Dout**,**//Data output to CPU  **output** **reg[**1**:**0**]** RW **=** 2'b00**,**//Control of read and write  **output** **reg[**7**:**0**]** Din **=** 8'b00000000**,**//Data input to POC  **output** **reg** ADDR **=** 0//Address  **);**  **always** **@** **(** **posedge** CLK **or** **negedge** RSTn **)**  **if(**RSTn**)**  **begin**  **if** **(**IRQ **==** 1'b0**)**  **begin**//Interrupt mode  **if(**func **==**1'b1 **)** // To enter something  **begin**  Din**[**7**]** **=** data\_input**[**7**];**  Din**[**6**]** **=** data\_input**[**6**];**  Din**[**5**]** **=** data\_input**[**5**];**  Din**[**4**]** **=** data\_input**[**4**];**  Din**[**3**]** **=** data\_input**[**3**];**  Din**[**2**]** **=** data\_input**[**2**];**  Din**[**1**]** **=** data\_input**[**1**];**  Din**[**0**]** **=** data\_input**[**0**];**  **#**3  ADDR **=** 1'b1**;**// Directly select BR  RW **=** 2'b11**;**  **#**8  RW **=** 2'b00**;**  **end**  **else** RW **=** 2'b00**;**  **end**  **else**  **begin**//Polling mode  **if(**func **==**1'b1 **)** // To enter something  **begin**  ADDR **=** 1'b0**;**// The CPU selects the SR register through the appropriate address to query the SR7 information.  RW **=** 2'b10**;**  **#**3  **if(**Dout**[**7**]==**1'b1**)**  **begin**  Din**[**7**]** **=** data\_input**[**7**];**  Din**[**6**]** **=** data\_input**[**6**];**  Din**[**5**]** **=** data\_input**[**5**];**  Din**[**4**]** **=** data\_input**[**4**];**  Din**[**3**]** **=** data\_input**[**3**];**  Din**[**2**]** **=** data\_input**[**2**];**  Din**[**1**]** **=** data\_input**[**1**];**  Din**[**0**]** **=** data\_input**[**0**];**  **#**3  ADDR **=** 1'b1**;**// Directly select BR  RW **=** 2'b11**;**  **#**8  RW **=** 2'b00**;**  **end**  **else** RW **=** 2'b00**;**  **end**  **end**  **end**  **else**  **begin**  RW **=** 2'b00**;**  Din **=** 8'b00000000**;**  ADDR **=** 0**;**  **end**  **endmodule**  **module** POC**(**  **input** Switch**,**  **input** CLK**,**//Clock  **input** RSTn**,**  **input** **[**1**:**0**]** RW**,**//Control of read and write  **input** **[**7**:**0**]** Din**,**//Data input to POC  **input** RDY**,**  **input** ADDR**,**//Address  **output** **reg** TR **=** 0**,**  **output** **reg[**7**:**0**]**PD **=** 8'b00000000**,**  **output** **reg** IRQ **=** 1**,**//Interrupt request  **output** **reg[**7**:**0**]** Dout **=** 8'b00000000//Data output to CPU  **);**  **reg[**1**:**0**]** flag **=** 2'b00**;**  **reg[**7**:**0**]** BR **=** 8'b00000000**;** //Buffer Register  **reg[**7**:**0**]** SR **=** 8'b10000000**;**//Status Register  //SR7--Ready flag bit  //SR0--interrupt bit  //SR6-SR1--empty  **always** **@** **(** **posedge** CLK **or** **negedge** RSTn **)**  **if(**RSTn**)**  **begin**  **if(** **(**SR**[**0**]** **==** 1'b1**)** **)**  **if(**SR**[**7**]==** 1'b1**)**  IRQ **=** 1'b0**;**  **else** IRQ **=** 1'b1**;**  **else** IRQ **=** 1'b1**;**  **end**  **else** IRQ **=** 1'b1**;**    **always** **@** **(** **posedge** CLK **or** **negedge** RSTn **)**  **if(**RSTn**)**  **begin**  **if(**Switch **==** 1'b0**)** SR**[**0**]** **=** 1'b0**;**  **else** **if(**Switch **==** 1'b1**)** SR**[**0**]** **=** 1'b1**;**  **if** **(**SR**[**0**]** **==** 1'b0**)** **begin**// Polling mode: SR0 is always 0  IRQ **=** 1'b1**;**  **if** **(**RW **==** 2'b11**)**//Read from CPU Din  **begin**  **case(**ADDR**)**  1'b1**:** // If SR7=1, the CPU selects the BR register and writes one byte of data to be printed to the BR.  **begin**  BR**[**7**]** **=** Din**[**7**];**  BR**[**6**]** **=** Din**[**6**];**  BR**[**5**]** **=** Din**[**5**];**  BR**[**4**]** **=** Din**[**4**];**  BR**[**3**]** **=** Din**[**3**];**  BR**[**2**]** **=** Din**[**2**];**  BR**[**1**]** **=** Din**[**1**];**  BR**[**0**]** **=** Din**[**0**];**  SR**[**7**]** **=** 0**;**// The CPU sets the SR7 register to 0 after completion.  **end**  **default:begin**  **end**  **endcase**  **end**  **else** **if(**RW **==** 2'b10**)**//Write to CPU Dout  **begin**  **case(**ADDR**)**  1'b0**:** // The CPU selects the SR register through the appropriate address to query the SR7 information.  **begin**  Dout**[**7**]** **=** SR**[**7**];**  **end**  **default:begin**  **end**  **endcase**  **end**  **else**//Do not read or write  **begin**  **end**  **if(!**SR**[**7**])**// Indicates that the CPU has written new data and has not yet been processed.  // Start the handshake operation with the peripheral (printer). After the operation is completed, the POC sets the SR7 register to 1, which is the "Ready" state.  **begin**  // After the POC completes the handshake with the CPU, it sends the data to the PD port.  PD**[**7**]** **=** BR**[**7**];**  PD**[**6**]** **=** BR**[**6**];**  PD**[**5**]** **=** BR**[**5**];**  PD**[**4**]** **=** BR**[**4**];**  PD**[**3**]** **=** BR**[**3**];**  PD**[**2**]** **=** BR**[**2**];**  PD**[**1**]** **=** BR**[**1**];**  PD**[**0**]** **=** BR**[**0**];**  **if** **(**RDY **==** 1'b1**)**  **begin**// POC detects the printer's RDY=1 and sends a pulse at TR  **if** **(**flag **==** 2'b00**)**  **begin**// Start transmitting data  flag **=** 2'b01**;**  **end**  **if** **(**flag **==** 2'b10**)**  **begin**  // Delayed for a while, after the print is complete, the printer sets RDY to 1 again.  flag **<=** 2'b00**;**  SR**[**7**]** **<=** 1'b1**;**  IRQ **<=** 1'b1**;**// Cause the IRQ signal to be pulled low to a low level of 0, that is, an interrupt request is issued.  **end**  **else** **begin**  TR **=** 1'b1**;**  **end**  **end**  **else**  **begin**  **if** **(**flag **==** 2'b01**)**  **begin**  flag **<=** 2'b10**;**  **end**  TR **=** 1'b0**;**// After detecting TR, set RDY to 0, receive PD data and send it to print.  **end**  **end**  **end**  **else** **if** **(**SR**[**0**]** **==** 1'b1**)**//Interrupt mode  **begin**  **if** **(**RW **==** 2'b11**)**//Read from CPU Din  **begin**  **case(**ADDR**)**  1'b1**:** // If SR7=1, the CPU selects the BR register and writes one byte of data to be printed to the BR.  **begin**  BR**[**7**]** **=** Din**[**7**];**  BR**[**6**]** **=** Din**[**6**];**  BR**[**5**]** **=** Din**[**5**];**  BR**[**4**]** **=** Din**[**4**];**  BR**[**3**]** **=** Din**[**3**];**  BR**[**2**]** **=** Din**[**2**];**  BR**[**1**]** **=** Din**[**1**];**  BR**[**0**]** **=** Din**[**0**];**  SR**[**7**]** **=** 0**;**// The CPU sets the SR7 register to 0 after completion.  **end**  **default:begin**  **end**  **endcase**  **end**  **else** **if(**RW **==** 2'b10**)**//Write to CPU Dout  **begin**  **case(**ADDR**)**  1'b0**:** // The CPU selects the SR register through the appropriate address to query the SR7 information.  **begin**  Dout**[**7**]** **=** SR**[**7**];**  **end**  **default:begin**  **end**  **endcase**  **end**  **else**// Do not read or write  **begin**  **end**  **if(!**SR**[**7**])**// Indicates that the CPU has written new data and has not yet been processed.  // Start the handshake operation with the peripheral (printer). After the operation is completed, the POC sets the SR7 register to 1, which is the "Ready" state.  **begin**  // After the POC completes the handshake with the CPU, it sends the data to the PD port.  PD**[**7**]** **=** BR**[**7**];**  PD**[**6**]** **=** BR**[**6**];**  PD**[**5**]** **=** BR**[**5**];**  PD**[**4**]** **=** BR**[**4**];**  PD**[**3**]** **=** BR**[**3**];**  PD**[**2**]** **=** BR**[**2**];**  PD**[**1**]** **=** BR**[**1**];**  PD**[**0**]** **=** BR**[**0**];**  **if** **(**RDY **==** 1'b1**)**  **begin**// POC detects the printer's RDY=1 and sends a pulse at TR  **if** **(**flag **==** 2'b00**)**  **begin**// Start transmitting data  flag **=** 2'b01**;**  **end**  **if** **(**flag **==** 2'b10**)**  **begin**  // Delayed for a while, after the print is complete, the printer sets RDY to 1 again.  flag **=** 2'b00**;**  SR**[**7**]** **=** 1'b1**;**  IRQ **=** 1'b0**;**// Cause the IRQ signal to be pulled low to a low level of 0, that is, an interrupt request is issued.  **end**  **else** **begin**  TR **=** 1'b1**;**  **end**  **end**  **else**  **begin**  **if** **(**flag **==** 2'b01**)**  **begin**  flag **<=** 2'b10**;**  **end**  TR **=** 1'b0**;**// After detecting TR, set RDY to 0, receive PD data and send it to print.  **end**  **end**  **end**  **end**  **else**  **begin**  flag **=** 2'b00**;**  TR **=** 0**;**  PD **=** 8'b00000000**;**  IRQ **=** 1**;**  Dout **=** 8'b00000000**;**  BR **=** 8'b00000000**;**  SR **=** 8'b10000000**;**  **end**  **endmodule**  **module** printer**(**  **input** CLK**,**//Clock  **input** RSTn**,**  **input** TR**,**  **input** **[**7**:**0**]**PD**,**  **output** **reg** RDY **=** 1'b1**,**  **output** **reg** **[**7**:**0**]**data **=** 8'b00000000  **);**  **reg[**1**:**0**]** flag\_printer **=** 2'b00**;**  **always** **@(posedge** CLK **or** **negedge** RSTn**)**  **if(**RSTn**)**  **begin**  **if** **(**TR **==** 1'b1**)**  **begin**  flag\_printer **=** 2'b01**;**// Receive PD data to print  RDY **=** 1'b0**;**  **end**  **if** **(**flag\_printer **==** 2'b01**)**// Receive PD data to print  **begin**  data**[**7**]** **=** PD**[**7**];**  data**[**6**]** **=** PD**[**6**];**  data**[**5**]** **=** PD**[**5**];**  data**[**4**]** **=** PD**[**4**];**  data**[**3**]** **=** PD**[**3**];**  data**[**2**]** **=** PD**[**2**];**  data**[**1**]** **=** PD**[**1**];**  data**[**0**]** **=** PD**[**0**];**  flag\_printer **<=** 2'b00**;**  RDY **=** **#**13 **(**1'b1**);**// Delay 13, after printing is complete, the printer sets RDY to 1, indicating that it is ready.  **end**  **end**  **else**  **begin**  flag\_printer **=** 2'b00**;**  RDY **=** **(**1'b1**);**  data **=** 8'b00000000**;**  **end**  **endmodule**    **module** top**(**  **input** CLK**,**  **input** func**,**  **input** RSTn**,**  **input** **[**7**:**0**]** data\_input**,**  **input** Switch**,**  **output** **[**7**:**0**]**data  **);**  **wire** IRQ**;**// Interrupt request  **wire** **[**7**:**0**]** Dout**;**  **wire** **[**1**:**0**]** RW**;**  **wire** **[**7**:**0**]** Din**;**  **wire** ADDR**;**  **wire** RDY**;**  **wire** TR**;**  **wire** **[**7**:**0**]** PD**;**  Processor U0  **(**  func**,**  CLK**,**  RSTn**,**  IRQ**,**//Interrupt request  data\_input**,**  Dout**,**//Data output to CPU  RW**,**//Control of read and write  Din**,**//Data input to POC  ADDR//Address  **);**  POC U1  **(**  Switch**,**  CLK**,**  RSTn**,**  RW**,**  Din**,**  RDY**,**  ADDR**,**  TR**,**  PD**,**  IRQ**,**  Dout  **);**  printer U2  **(**  CLK**,**  RSTn**,**  TR**,**  PD**,**  RDY**,**  data  **);**  **endmodule** |

## 7.2 Simulation codes

|  |
| --- |
| `timescale 1ns **/** 1ps  **module** test**;** // Declare the name of the test  **reg** CLK**;**  **reg** func**;**  **reg** RSTn**;** // Signal statement  **reg** **[**7**:**0**]**data\_input**;**  **reg** Switch**;**  **wire** **[**7**:**0**]**data**;**  // Instantiation of the following shift\_reg design  top dut**(**  **.**CLK **(**CLK**),**  **.**func **(**func**),**  **.**RSTn **(**RSTn**),**  **.**data\_input **(**data\_input**),**  **.**Switch **(**Switch**),**  **.**data**(**data**));**  // This process block sets the free running clock  **initial** **begin**  CLK **=** 0**;**  **forever** **#**1 CLK **=** **~**CLK**;**  **end**  **initial**  **begin**// This process block specifies the stimulus.  RSTn **=** 0**;**  Switch **=** 1'b0**;**//Polling mode  **#**200  RSTn **=** 1**;**  data\_input **=** 8'b11110000**;**  func **=** 1'b1**;**  **#**200  func **=** 1'b0**;**  **#**200  data\_input **=** 8'b00001111**;**  func **=** 1'b1**;**  **#**200  func **=** 1'b0**;**  **#**200  Switch **=** 1'b1**;**//Interrupt mode  **#**200  data\_input **=** 8'b01101111**;**  func **=** 1'b1**;**  **#**200  func **=** 1'b0**;**  **#**200  Switch **=** 1'b0**;**//Polling mode  **#**200  RSTn **=** 1**;**  data\_input **=** 8'b11110000**;**  func **=** 1'b1**;**  **#**200  func **=** 1'b0**;**  **#**200  data\_input **=** 8'b00001111**;**  func **=** 1'b1**;**  **#**200  func **=** 1'b0**;**  **#**200 $stop**;**  **end**  **endmodule** |