Verilog Experiment 2 for SOC student

HowardYang

May 4, 2018

1 AMBA APB BUS INTERFACE

Deadline : Nov 10th 2016

In this section. You will see a popular bus interface called “APB” from ARM and learn how the bus works then try to use the bus with our uart module. You can use document “dw_apb_uart_db” as your design reference.

Must do:

a)!You must use a enhanced text editor,such as notepad++,vi,sublime,emacs,etc.Choose the one you like.But at least,the editor should has the keyword highlight.

b)!!!You must use clk/clock as the clock signal’s name, use resetn/reset_n/rst_n/rstn as the reset signal’s name,all reset should be active low in our lab.Besides,if a data signal is active low, you should add _n or n or _b or b at the end of the signal, _n is preferred

c)!!!!You should never use ”signal1”,”signal2” as your signal name

d)!!You must choose Modelsim/VCS as your verilog compiler,and use Modelsim/Verdi to watch simulation waveform, or you will not get help from others in our lab. We do only use Modelsim and Synopsys tools

e)!!!You must write a script to compile the source file and do simulation.

Should do:

a)The editor may also have auto complete and auto indent

b)Try a better font so you can distinguish 0 and O

1.1 APB

Read the pdf in the git project, learn how the apb bus works, then use “TimeGen” to draw a single apb write to addr 0x04 data 0xDEADBEEF,then a single apb read from 0x08. With knowledge of C Programming Language answer the question below:

```
typedef struct myStruct
{
  uint16_t head;
  uint32_t type;
  uint32_t data[15];
  uint8_t  checksum;
}myStruct,*pmyStruct;
int main()
{
        myStruct a;
        a.head = 0x1234;
```

```
        a.type = 0xdeadbeef;
    return 0;
}
```

let address of a = 0x1000, and we use a 32-bit cpu system

Q1: sizeof(a) = ?

Q2: int b = (int)&a.type - (int)&a.head; b = ?

Q3: if the system use little endian, write down the memory from 0x1000 to 0x1008 byte by byte: 0x1000 = ? *0x1001 = ? *0x1002 = ? ...

Q4: with "#pragma pack(1)" before the typedef, answer all 3 question above again.If we do a read operation b = a.type now, describe what will happen in cpu.

Now lets get back to verilog & soc, we will see what happens in the uart module.

Define some register according to our own uart module, at least include "control reg" , "data tx reg" and "data rx reg". for every register, give it an address offset and tell the definition of the register, for example, you can set control reg at offset 0x4 and bit0 of the control reg means "enable" while bit1 of the control reg means "verify on" bit2 "odd even verify".set data tx reg and data rx reg both at offset 0x8 so that write to this address means trigger a shoot procedure while read from this address means read back of the uart value.Another register is interrupt reg, which is "W1C(write 1 clear)". Q5: write a verilog module, it accepts the apb signal and translate them to our register.Note: you can add or reduce signals in the list.

```
module myuart_reg
(
  input pclk;
  input presetn;
  input psel;
  input pwrite;
  input penable;
  input [5:0] paddr;
  input [31:0] pwdata;
  output [31:0] prdata;

  output interrupt;

  output shoot;
  output [7:0] datatx;
  input busyn;
  input [7:0] datarx
)
```

There is no need of make datatx and datarx register to 32bit wide.

Q6: Translate the "TimeGen" wave to testbench and connect the myuart_reg module with our uart module. In testbench generate proper transform and make a uart tx procedure.

Note: you can set clk divisor to a small number to fast your simulation.

Q7: Consider how does the fifo increase your system performance.write it down.