VERILOG HDL
(and C)

Some Reference Material

The following are suggested reading..

`http://engnet.anu.edu.au/DEcourses/engn3213/Documents/VERILOG/`

- **VerilogIntro.pdf** - Very simple introduction which emphasises the ground rules for good coding style
- **verilog-tutorial.pdf** - Another very simple introduction which introduces some other things but is no good on coding style
- **coding_and_synthesis_with_verilog.pdf** - explains ways to build some common circuits
- **VerilogTutorial.pdf** - a VERILOG reference with lots of useful information
- **cummings-nonblocking-snug99.pdf** - There are several other useful articles by Cummings eps. Finite State Machines in VERILOG.

Some Reference Material (ctd)

- **1996-CUG-presentation_nonblocking_assigns.pdf** - A discussion about blocking and non-blocking assignments but mostly for simulation. An interesting discussion about how to simulate inertial and transport delays in real hardware using VERILOG.
- **vlogref.pdf** - Cadence VERILOG compendium.. heaps of reference material
- **verilogSlides.pdf** - some of this lecture's slides
- **Beware of some inconsistencies and note those in the hardware blocks of MU0!**

How To Represent Hardware?

Hardware definition language provides a way of describing and documenting hardware so that

- You can read it again later
- Someone else can read and understand it
- It can be simulated and verified
- It may be synthesised into gates
- It can be shipped and you can make money

Ways to represent hardware

- Schematics
- Write a NETLIST
- Write primitive Boolean equations
- Use **Hardware Description Language**

ANU
THE AUSTRALIAN NATIONAL UNIVERSITY

What are Hardware Description Languages (HDLs)?

- Textual representation of a logic design but has various levels of **abstraction**
- **Not programming languages** **YOU MUST MAKE THE CONCEPTUAL LEAP**
- Similar development chain
  - Compiler → assembly code → binary machine code
  - Synthesis tool: HDL source → gate-level specification → hardware

Why use HDLs?

- Easier to make system abstractions (VIZ the **combinational** and **sequential logic** abstractions)
- Easy to write and edit
- Compact
- Don’t have to follow wires on a schematic
- Easy to analyse

- Why not to use an HDL
  - To avoid understanding the hardware
  - A schematic can be a complex work of art... **but not if you do a proper RTL design**

HDL HIstory

- 1970s: First HDLs
- Late 1970s: VHDL
  - VHDL = VHSIC HDL = Very High Speed Integrated Circuit HDL
  - VHDL inspired by programming languages of the day (Ada)
- 1980s:
  - Verilog first introduced
  - Verilog inspired by the C programming language
  - VHDL standardized

HDL HIstory

- 1990s:
  - Verilog standardized (Verilog-1995 standard)
- 2000s:
  - Continued evolution (Verilog-2001 standard)
- Both VHDL and Verilog evolving, still in use today
- ICARUS Verilog needs plenty of Verilog-2001 compliance - work in progress.

About Verilog...

- A surprisingly big language
  - lots of features for simulation and synthesis of hardware.
- **We are going to learn a focussed subset of VERILOG**
  - We will use it at a level appropriate for computer design
  - Focus on synthesisable constructs
  - Initially restrict some features just because they are not necessary
  - **If you haven’t seen it in lectures, ask me before you use it**

Why an HDL is not a Programming Language

- In a program, we start at the beginning (e.g. 'main'), and we proceed sequentially through the code as directed
- The program represents an algorithm, a step-by-step sequence of actions to solve some problem

```
for (i = 0; i<10; i=i+1) {
  if (newPattern == oldPattern[i]) match = i;
}
```

- Hardware is all active at once; there is no starting point

THE AUSTRALIAN NATIONAL UNIVERSITY

What do you really have to know to understand and use VERILOG?

- VERILOG can be for SYNTHESIS or for SIMULATION: you must learn to distinguish them
- SYNTHESISABLE VERILOG runs CONCURRENTLY in hardware (C.F. C language)
- VERILOG has some basic abstractions:

1. COMBINATIONAL and SEQUENTIAL
2. WIRES and REGS (not important but basic)

ANU THE AUSTRALIAN NATIONAL UNIVERSITY

What are the usual problems of VERILOG that you need to look out for?

- You are breaking with the COMBINATIONAL / SEQUENTIAL paradigm

- Your code is poorly synthesisable: Two ways:

1. You are using simulation-only constructs in your code for synthesis.

2. YOUR CODE IS FOR SYNTHESIS BUT TRANSLATES BADLY INTO HARDWARE - THE USUAL PROBLEM

WIRES and REGS

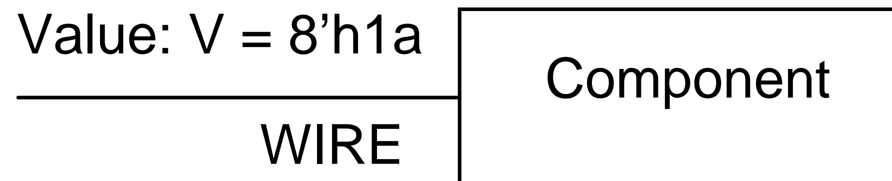
- Variables in VERILOG are either **WIRES** or **REGS**
- **Neither of these is like a variable in C**
- **WIRES**
  - A **WIRE** is a through-connection within or to a module. The input and output ports are **WIREs**. **ASSIGNS** assign to **WIRES**
- **REGS**
  - The combinational and sequential paradigms in terms of ALWAYS blocks imply a **persistance to the variables they assign to**
  - **Example: the variables on the left hand sign of statements inside ALWAYS blocks are REGS**
  - **REGS have persistance and store VALUES from one event to the next**
  - REGS are not registers in the electronic sense

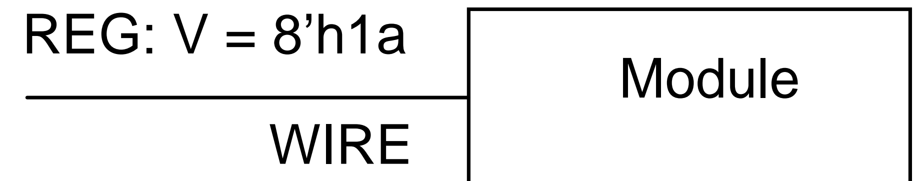
ANU THE AUSTRALIAN NATIONAL UNIVERSITY

WIRES and REGS

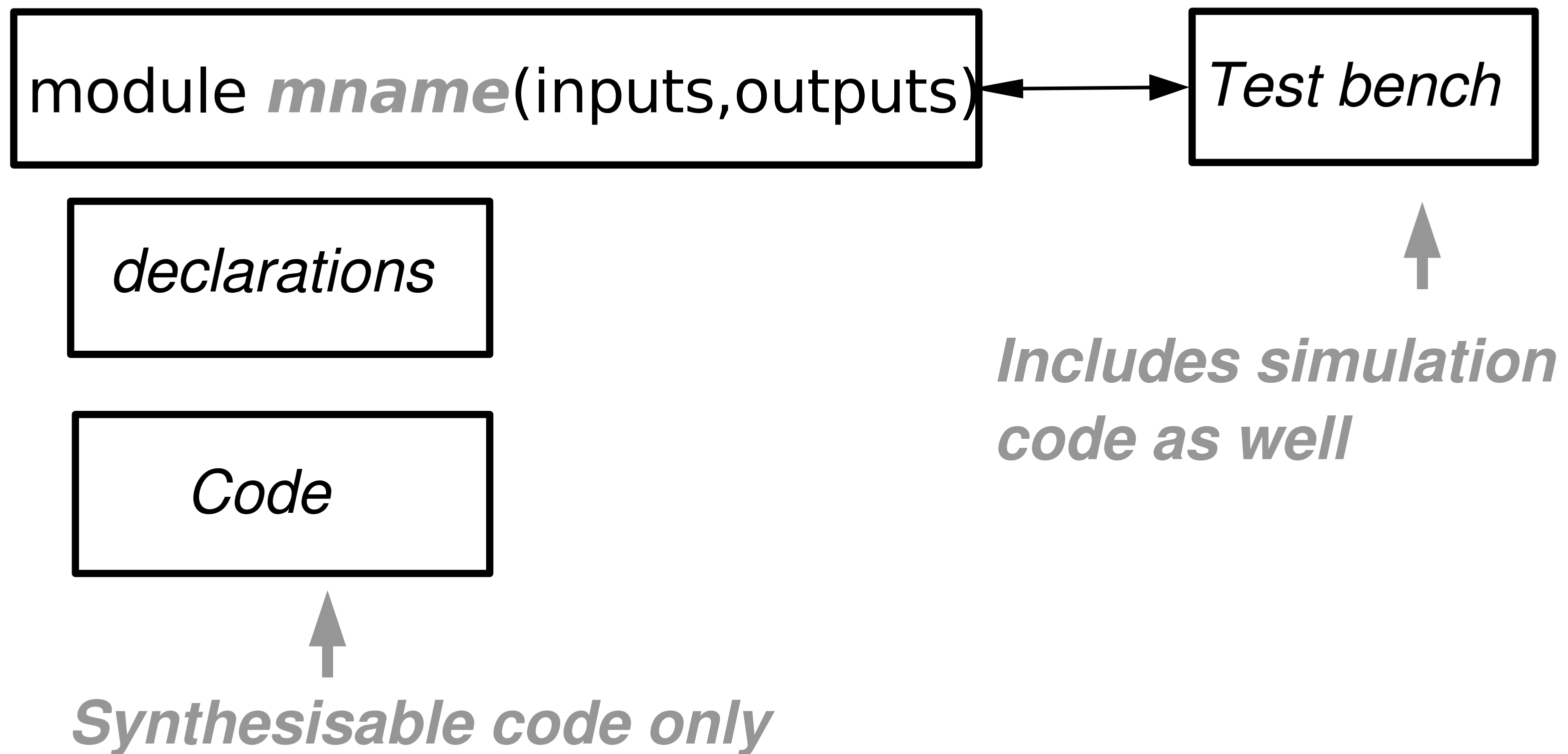
Circuit Schematic

VERILOG





Verilog Program Structure



ANU THE AUSTRALIAN NATIONAL UNIVERSITY

Modelling sequential and combinational circuits

```
/**********************************/
always @(posedge clk or A) begin

//Non-blocking IO
        X <= Y;
        Y<= A;
end
```

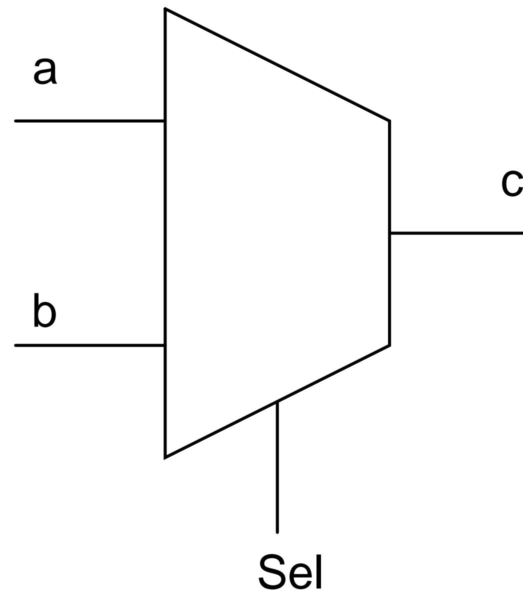
Sequential code
assignment at
the posedge clk

```
/**********************************/
always @(A or B) begin

//Blocking IO
        D = B;
        C = A;
end
```

Combinational code
All values settle,
clock period >> settling time

VERILOG by example: MULTIPLEXER (combinational)

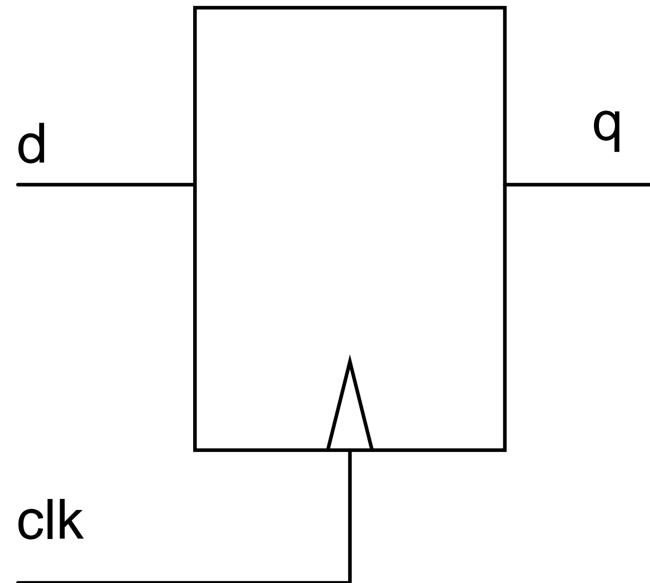
```
module mux(sel, a, b, c);
input a;
input b;
input sel;
output c;
reg c;
always@(a or b or sel)
   if (sel == 1'b1)
      c = a;
   else
      c = b;

//  c = sel ? a : b;

endmodule
```



ANU
THE AUSTRALIAN NATIONAL UNIVERSITY

VERILOG by example: D-type Flip Flop (sequential)

```
module dff(clk, d, q);
input clk;
input d;
output q;
reg q;
always@(posedge clk)
  q <= d;
endmodule
```



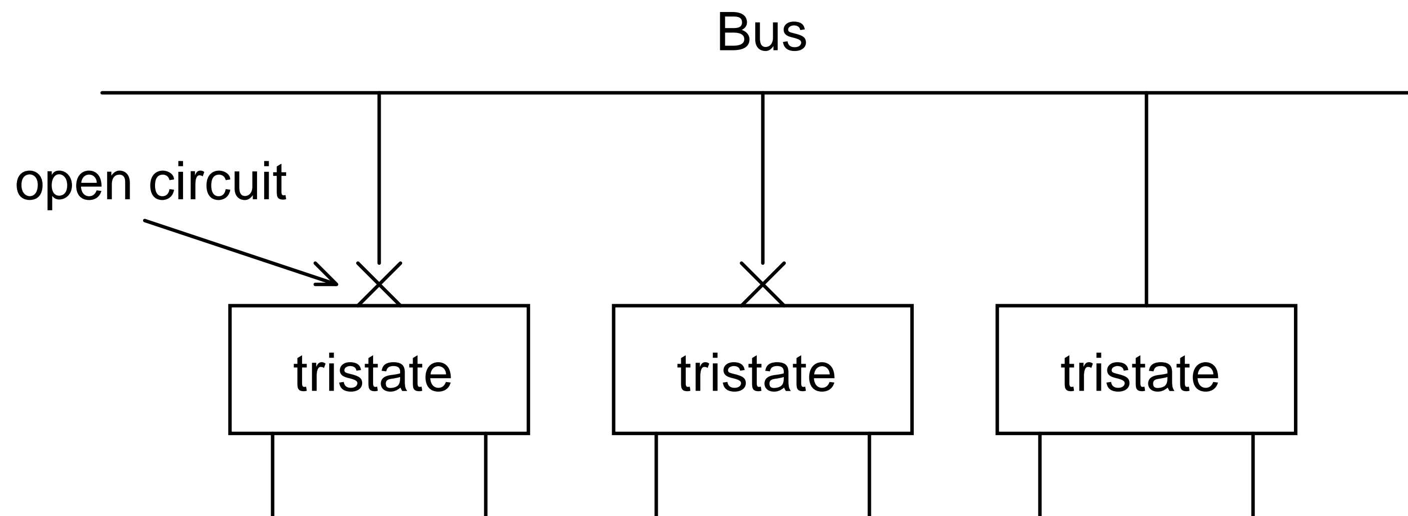
ANU THE AUSTRALIAN NATIONAL UNIVERSITY

ASSIGNS... an alternative construct for treating SIMPLE combinational circuits

```
module inv(a, y);
    input [3:0] a;
    output [3:0] y;
    assign y = ~a;
endmodule

module and8(a, y);
    input [7:0] a;
    output       y;
    assign y = &a;
endmodule;
```

Tristate Buffers

- Tristate buffers are multiplexers that have the possibility of producing an open circuit output



Tristate Buffers

```
module tristate(a, en, y);
    input [3:0] a;
    input       en;
    output [3:0] y;
    assign y = en ? a : 4'bz;
endmodule

module mux2(d0, d1, s, y);
    input [3:0] d0, d1;
    input       s;
    output [3:0] y;
    tristate t0(d0, ~s, y);
    tristate t1(d1, s, y);
endmodule
```

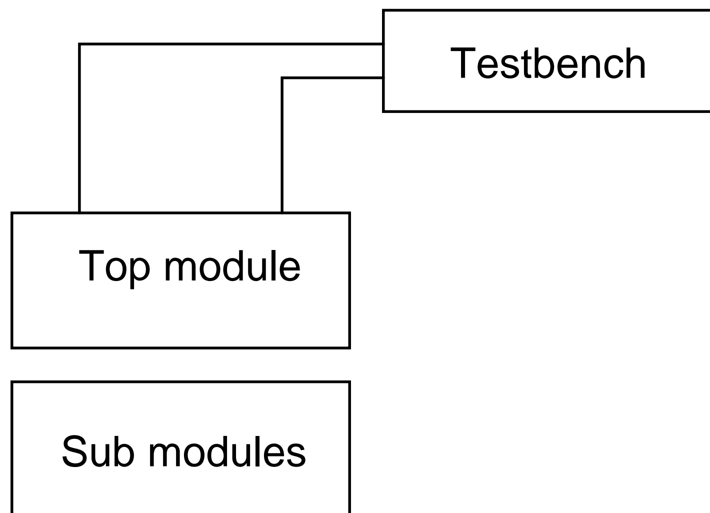
Combinational logic

```
// Using a reg
// -----------------------------
wire a,b;
reg c;
always @ (a or b)
   c = a & b;
// Using a wire
// -----------------------------
wire a,b,c;
assign c = a & b;
// using a built in primitive (without instance name)
// -----------------------------
reg a,b;
wire c;
and (c,a,b); // output is always first in the list
// using a built in primitive (with instance name)
// -----------------------------
reg a,b;
wire c;
and u1 (c,a,b); // output is always first in the list
```


ANU THE AUSTRALIAN NATIONAL UNIVERSITY

Program structure example: Testing an 8 bit adder

- For simulation the Verilog program structure consists of synthesisable code and a testbench.



ANU
THE AUSTRALIAN NATIONAL UNIVERSITY

Top module

```
module adder(Xw,Yw,Sw,Cout);

input [7:0] Xw;
input [7:0] Yw;

output [7:0] Sw;
output Cout;

wire [7:0] Sw;

full_adder fa0(Xw[0],Yw[0],Sw[0],0 ,C0);
full_adder fa1(Xw[1],Yw[1],Sw[1],C0,C1);
full_adder fa2(Xw[2],Yw[2],Sw[2],C1,C2);
full_adder fa3(Xw[3],Yw[3],Sw[3],C2,C3);
full_adder fa4(Xw[4],Yw[4],Sw[4],C3,C4);
full_adder fa5(Xw[5],Yw[5],Sw[5],C4,C5);
full_adder fa6(Xw[6],Yw[6],Sw[6],C5,C6);
full_adder fa7(Xw[7],Yw[7],Sw[7],C6,Cout);

endmodule
```

1 bit full adder

```
module full_adder(X,Y,S,Cin,Cout);

input X;
input Y;
input Cin;

output S;
output Cout;

assign S = (X^Y)^Cin;
assign Cout = ((X^Y)&Cin)|(X&Y);

endmodule
```

Testbenches: A test bench for the adder

```
module TB_adder; //No inputs or outputs for a testbench

reg [7:0] X;
reg [7:0] Y;

wire [7:0] S;
wire Cout;

adder ad(X,Y,S,Cout);

initial begin
        $display("\t\ttime\t X\t Y\t S\t Cout\n");
        #1 X = 8'hab;
        #1 Y = 8'hbb;
        #1$display("%d\t%b\t%b\t%b\t%b\t\n",$time,X,Y,S,Cout);
end

endmodule
```


ANU
THE AUSTRALIAN NATIONAL UNIVERSITY

Running the simulation

```
[ggb112@localhost adder]$ more cmp.sh


#!/bin/sh
iverilog full_adder.v adder.v  TB_adder.v
./a.out


[ggb112@localhost adder]$ sh cmp.sh


     time    X                Y             S                Cout

     3    10101011        10111011        01100110         1
```


THE AUSTRALIAN NATIONAL UNIVERSITY

Pitfalls of Treating Verilog like a Programming Language

- If you program sequentially, the synthesiser may add a lot of hardware to try to do what you say
- If you program in parallel (multiple ’always’ blocks), you can get non-deterministic execution - Which ’always’ happens first?

```
if(x == 1) out = 0;
if(y == 1) out = 1;// else out retains previous state? R-S latch!
```

- You don’t realize how much hardware you’re specifying **e.g.** $x = x + 1$ **can be a LOT of hardware**
- Slight changes may suddenly make your code ’blow up’. A chip that previously fit suddenly is too large or slow

Structural vs Behavioral HDL Constructs

- Structural constructs specify actual hardware structures
  - Low-level, direct correspondence to hardware
  - Primitive gates (e.g., and, or, not)
  - Hierarchical structures via modules
- Behavioural constructs specify an operation on bits
  - High-level, more abstract
  - Specified via equations, e.g.

```
out = (a&b)|c
```

- Not all behavioural constructs are synthesisable
  - We've already talked about the pitfalls of trying to 'program'
  - But even some combinational logic won't synthesise well
  - `out = a%b //modulo operation-what does this synthesise to?`
  - **The following are discouraged for synthesis:**
    $+ \ldots - \ldots * \ldots\ldots \% \ldots < \ldots <= \ldots < \ldots <= \ldots << \ldots >>$

ANU THE AUSTRALIAN NATIONAL UNIVERSITY

ENGN3213 Software

Applications Places System 6:30

Index of /DEcourses/engn3213/Software/WINDOWS - Mozilla Firefox

File Edit View Go Bookmarks Tools Help

http://engnet.anu.edu.au/DEcourses/engn3213/S(Go

Release Notes Fedora Project Fedora Weekly News Community Support Fedora Core 6

Index of /DEcourses/engn3213/Software/WINDOWS

Name	Last modified	Size	Description
Parent Directory		-	
GTKWAVE.zip	10-Mar-2009 22:40	8.9M	
MU0.zip	11-Mar-2009 08:49	1.1M	
MU0_README	12-Mar-2009 06:18	646	
MinGW-3.0.0-1.exe	06-		
PCCOMP.zip	06-		
RUNMU0.png	12-		
WebPACK_SFD_92i.zip	06-Mar-2009 13:45	1.6G	
iverilog-0.8.6_setup.exe	06-Mar-2009 13:42	1.3M	
xapp213.zip	06-		

Minimalist C-compiler for windows

C compiler for PICOBLAZE

PICOBLAZE

Apache/2.2.9 (Ubuntu) PHP/5.2.6-2ubuntu4.1 with Suhosin-Patch Server at engnet.anu.edu.au Port 80

Done

ggb112@localho... Index of /DEcour... lecture5_and_6_... RSPSE Webmail -...

Example: Counter

- 3 bit asynchronously resettable counter which counts 0, 1, 2, 3, 4, 5
- External reset to start the counter
- Issues - switch debouncing?

```
[ggb112@localhost counter]$ more cmp.sh
#!/bin/sh

iverilog counter_3bit.v TB_counter_3bit.v

./a.out

[ggb112@localhost counter]$ sh cmp.sh
VCD info: dumpfile C3B.vcd opened for output.
                 time     clk      rst      count
                    3     1        0         0000
                    7     1        0         0001
                   11     1        0         0010
                   15     1        0         0011
                   19     1        0         0100
                   23     1        0         0101
                   27     1        0         0000
                   31     1        0         0001
                   35     1        0         0010
                   39     1        0         0011
                   43     1        0         0100
                   47     1        0         0101
```

Example: Counter

```
module counter_3bit( clk, rst, count );

input clk;
input rst;

wire rst; //not necessary but true

output [3:0] count;

reg [3:0] count;   //count is assigned in ALWAYS block

// 3 bit asynchronously resettable
always @(posedge clk or posedge rst) begin
   if (rst)
      count <= 3'b0;
   else
      if (count == 3'b101)
         count <= 3'b0;
      else
         count <= count + 3'b001;
end

endmodule
```

Example: Counter

ANU THE AUSTRALIAN NATIONAL UNIVERSITY

```
module TB_counter_3bit;

reg clk;
reg rst;
wire [3:0] cnt;

counter_3bit c3b(clk, rst, cnt);

initial begin
clk = 0;
rst = 1;
#1rst = 0;
$display("\t\t time \t clk \t rst \t count");
forever #2 clk = ~clk;
end

initial begin
        $dumpfile("C3B.vcd");
        $dumpvars;
end

initial begin: stopat
#50; $finish;
end

always @(posedge clk) begin
$display("%d\t %b \t %b \t  %b",$time,clk, rst, cnt);
end
endmodule
```