

Verilog Coding Style

REF:

- Reuse Methodology Manual – For System-ON-A-Chip Design, Third Edition 2002
- CIC Training Manual – Logic Synthesis with Design Compiler, July, 2007
- Hsing-Chen, Lu, "ARES Lab 2008 Summer Training Course of Verilog Coding Style"
- Hsi-Pin, Ma, "LARC Lab Training Course of Design Concept and Coding Style"

2009.12.03

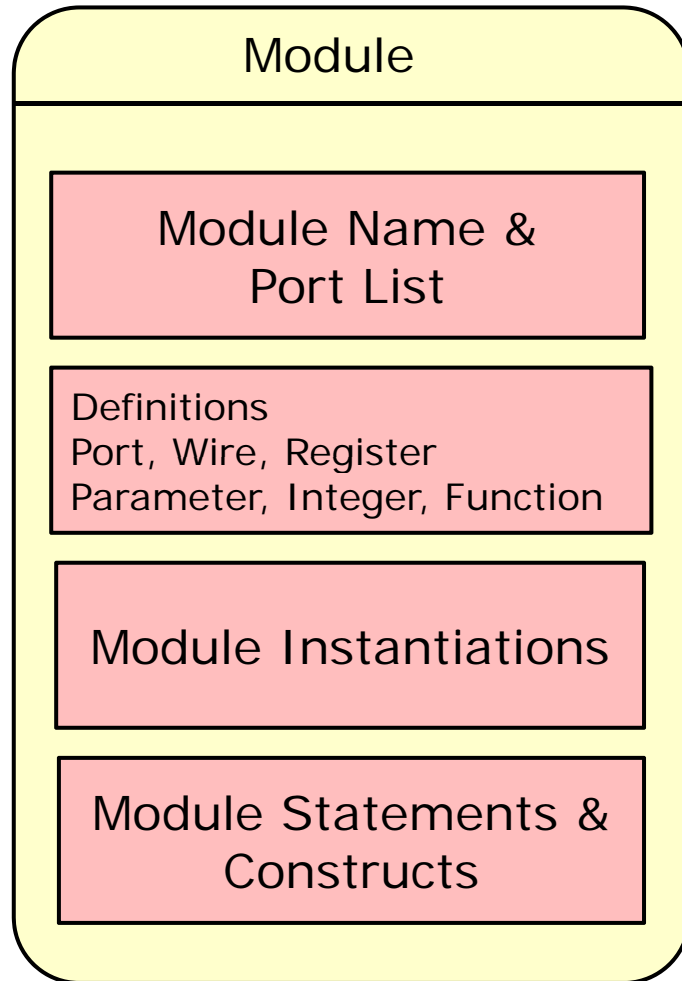
Outline

- ☐ Importance of Coding Style
- ☐ Basic Coding Practices
- ☐ Concept of Clocks and Reset
- ☐ Synthesizable Verilog
- ☐ Coding for Synthesis
- ☐ Tips for Verilog design

Importance of Coding Style

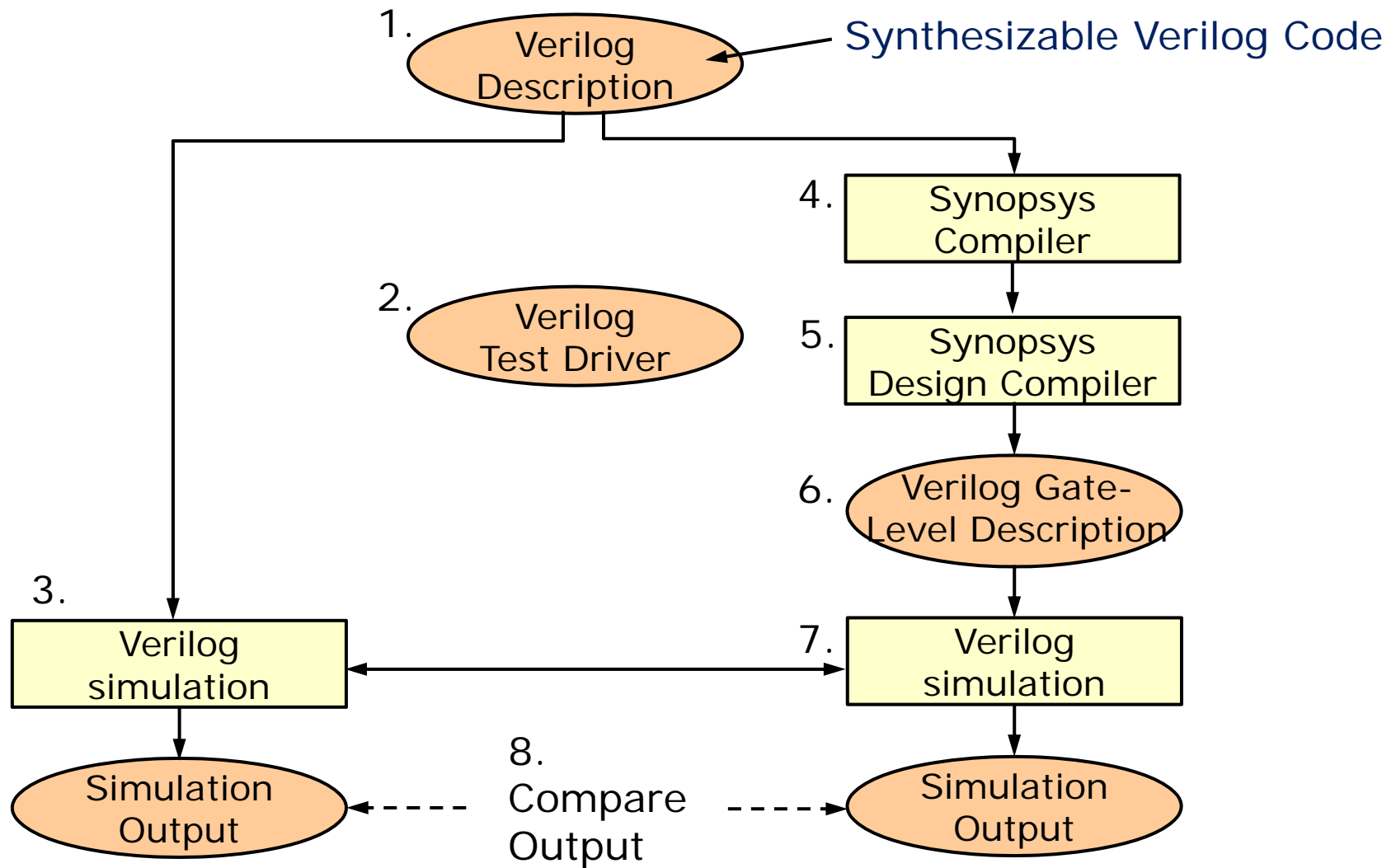
- ❑ Make sure your code is **Readable, Modifiable, and Reusable.**
- ❑ Good coding style helps to achieve the best compile times and synthesis results.
 - Testability
 - Performance
 - Simplification of static timing analysis
 - Gate-level circuit behavior that matches that of the original RTL code

Verilog Module



```
module test(a,b ,c,d,z,sum);  
  
    input    a,b;        //Inputs to nand gate  
    input [3:0] c,d;      //Bused Input  
    output   z;          //Output from nand gate  
    output [3:0] sum;     //Output from adder  
    wire     and_out;    //Output from and gate  
    reg [3:0] sum;       //Bused Output  
  
    And instance1 (a,b,and_out);  
    INV instance2 (and_out, z);  
  
    always@(c or d)  
    begin  
        sum= c+d;  
    end  
endmodule
```

Design Methodology



Synthesizable Verilog Code

- ❑ Synopsys DO NOT accept all kinds of Verilog and VHDL constructs
- ❑ Synopsys can only accept a subset of Verilog syntax and this subset is called “Synthesizable Verilog Code”

Basic Coding Practices

□ Simple and regular

- Use simple constructs and simple clocking schemes
 - Consistent coding style, consistent naming and state machines
 - Regular partitioning scheme
 - Easy to understand by comments and meaningful names
- No hard coded number

Basic Coding Practices (Cont')

□ Naming Conventions

- Use lowercase letters for all signal names, and port names, versus capital letters for names of constants and user-defined types.
- Use meaningful names
- For active low signals, end the signal name with an underscore followed by a lowercase character (e.g., rst_ or rst_n)
- Recommend using “bus[X:0]” for multi-bit signals.

Basic Coding Practices (Cont')

□ Include Headers in Source Files and Comments

```
//ARES Lab., EE Dept., NCU, Jhongli, TAIWAN 320
//http://ares.ncu.edu.tw/
//Project : SOFT-ERROR-MITIGATION BIST & DIAGNOSIS DATA COMPRESSION TECHNIQUES FOR HOV PROJECT
//Module : bist
//Adviser : Jin-Fu Li
//Author : Tsu-Wei Tseng, Chun-Hsien Wu
//E-mails : jfli@ee.ncu.edu.tw (Jin-Fu Li)
//          92521013@cc.ncu.edu.tw (Tsu-Wei Tseng)
//          93521039@cc.ncu.edu.tw (Chun-Hsien Wu)
//Date : 2007/08
//Abstract : Top module of the MBIST. This module consists of CTR, and Test Pattern Generator (TPG)

module bist(
    clk,
    rst,
    CSI,
    DO,
    hold,
    WEN_T,
    CS_T,
    OE_T,
    DI_T,
    ADDR_T,
    cmd_done,
    SYN,
    fail,
    test_done
);

//-----Parameter declarations-----
parameter INIT_ADR_NUM= 8'b00000000;
parameter FIN_ADR_NUM = 8'b11111111;
parameter WORD_LEN     = 8;
parameter ADR_LEN      = 8;
parameter ROW_ADR_LEN  = 4;
parameter COL_ADR_LEN  = 4;
parameter BIT_ADR_LEN  = 3;
parameter EXP_COUNT    = 5'b10100;

//-----IO declarations-----
//(BIST input control signals)
input      clk;
input      rst;
input      CSI;

//INITIAL ADDRESS OF THE ADDR COUNTER
//FINAL ADDRESS OF THE ADDR COUNTER
//WORD LENGTH
//ADDRESS LENGTH (ROW_ADR_LEN+COL_ADR_LEN)
//ROW ADDRESS LENGTH
//COLUMN ADDRESS LENGTH
//BIT ADDRESS LENGTH
//EXPORTATION COUNT (WORD_LEN+ADR_LEN+BIT_ADR_LEN+1)

//SYSTEM CLOCK
//MBIST RESET
//COMMAND SERIAL INPUT
```

Basic Coding Practices (Cont')

□ Indentation

```
//-----SERIAL READ COUNTER-----
always@(posedge clk or posedge rst)begin
  if(rst)begin
    ser_read_count <= 4'b0000;
  end
  else begin
    if(hold|self_hold)begin
      ser_read_count <= 4'b0000;
    end
    else begin
      if( (CS_T) || ((!CS_T)&&((addr_change)||(!WEN_T))) )begin
        ser_read_count <= ser_read_value;
      end
      else begin
        ser_read_count <= ser_read_count - 1'b1;
      end
    end
  end
end
```

Always use explicit mapping for ports and generics, using **named association** rather than positional association

□ Port Maps and Generic Maps

```
//_____ctr module: programmable_ctr _____
programmable_ctr programmable_ctr (
  .mar_or_xf(mar_or_xf),
  .rst_bist(rst_bist),
  .bsc(bsc),
  .bsi(bsi),
  .clk(clk),
  .test_done(test_done),
  .shift(shift),
  .fail(fail),
  .final_addr(final_addr),
  .final_data(final_data),
  .Comp(Comp),
  .data_type(data_type),
  .addr_type(addr_type),
  .a_count_shift(a_count_shift),
  .a_up_down(a_up_down),
  .a_left_right(a_left_right),
  .a_hold(a_hold),
  .d_hold(d_hold),
  .d_left_right(d_left_right),
  .CEN(CEN),
  .WEN(WEN),
  .OEN(OEN),
  .CEN_b(CEN_b),
  .WEN_b(WEN_b),
  .OEN_b(OEN_b)
);
```

Basic Coding Practices (Cont')

□ Use Functions or Tasks

- Instead of repeating the same sections of code

```
task ra;
begin
  WEB_T=1;//WEB=1:read
  EOP[w+2:w]=3'b011;//EOP[w]=1: a
  DI_T=CMD[w-1:0];
  FREE=CMD[w-1:0];
end
endtask

task rabar;
begin
  WEB_T=1;//WEB=1:read
  EOP[w+2:w]=3'b010;//EOP[w]=0: abar
  DI_T=~CMD[w-1:0];
  FREE=~CMD[w-1:0];
end
endtask

task wa;
begin
  WEB_T=0;//WEB=0:write
  EOP[w+2:w]=3'b001;
  DI_T=CMD[w-1:0];
  FREE=CMD[w-1:0];
end
endtask

task wabar;
begin
  WEB_T=0;//WEB=0:write
  EOP[w+2:w]=3'b000;
  DI_T=~CMD[w-1:0];//Maybe wrong
  FREE=~CMD[w-1:0];
end
endtask
```

```
begin
  case(CMD[w+3:w])
    4'b0000:begin // ra
      end_session=0;
      ra;
    end
    4'b0001:begin //wa'
      end_session=0;
      wabar;
    end
    4'b0010:begin //ra'
      end_session=0;
      rabar;
    end
    4'b0011:begin //wa
      end_session=0;
      wa;
    end
    4'b0100:begin //ra wa'
      end_session=1;
      case(session_state)
        4'b0000:ra;
        4'b0001:wabar;
        default:ra;
      endcase
    end
  end
```

Do **Not** Use Hard-Coded Number Values

□ Advantages using constants

- Constants are more intelligible as they associate a design intention with the value
- Constant values can be changed in one place



```
wire [7:0]      in;  
reg  [7:0]      out;
```

```
`define bus_size 8  
wire [bus_size -1:0] in;  
reg  [bus_size -1:0] out;
```

Specifying Constants

- ❑ Use constants in your design to substitute numbers to more meaningful names
- ❑ The use of constants helps make a design more readable and portable

```
parameter ZERO = 2'b00;
parameter A_AND_B = 2'b01;
parameter A_OR_B = 2'b10;
parameter ONE = 2'b11;
always@(OPCODE or A or B)
begin
    if (OPCODE == `ZERO)
        OP_OUT = 1'b0;
    else if (OPCODE == `A_AND_B)
        OP_OUT = A&B;
    else if (OPCODE == `A_OR_B)
        OP_OUT = A|B;
    else
        OP_OUT = 1'b1
end
```

Wire & Register

- Wire(wand, wor, tri)
 - Physical wires in a circuit
 - Cannot assign a value to a wire within a function or a begin...end block (i.e., always block)
 - A wire does not store its value
 - An undriven wire defaults to a value of Z(high impedance)
 - Input, Output, inout port declaration – wire data type (default)

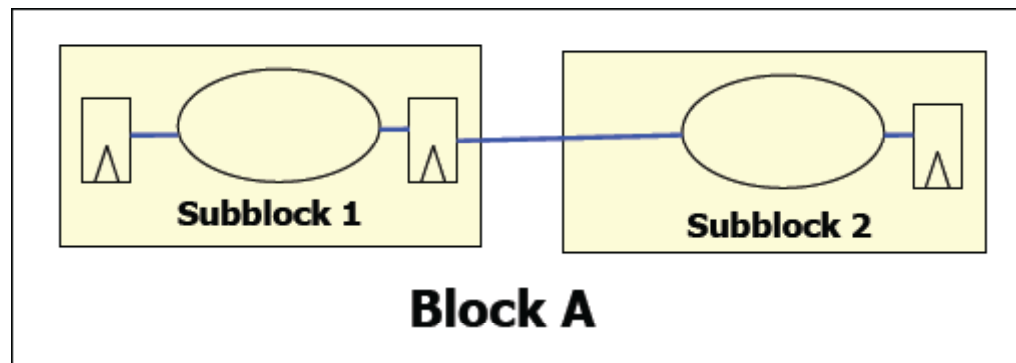
Wire & Register (Cont')

- ❑ reg: a variable in Verilog
- ❑ Use of “reg” data type is not exactly synthesized to a really register
- ❑ Compare to use of wire & reg
 - wire ➡ usually use “assign” and “assign” dose not appear in “always” block
 - reg ➡ only use “a=b”, always appear in “always” block

```
module test(a, b, c);  
  
    input    a,b;  
    output   c;  
    wire     c=a;  
    reg      c;  
  
    always@(a)begin  
        c=a;  
    end  
endmodule
```

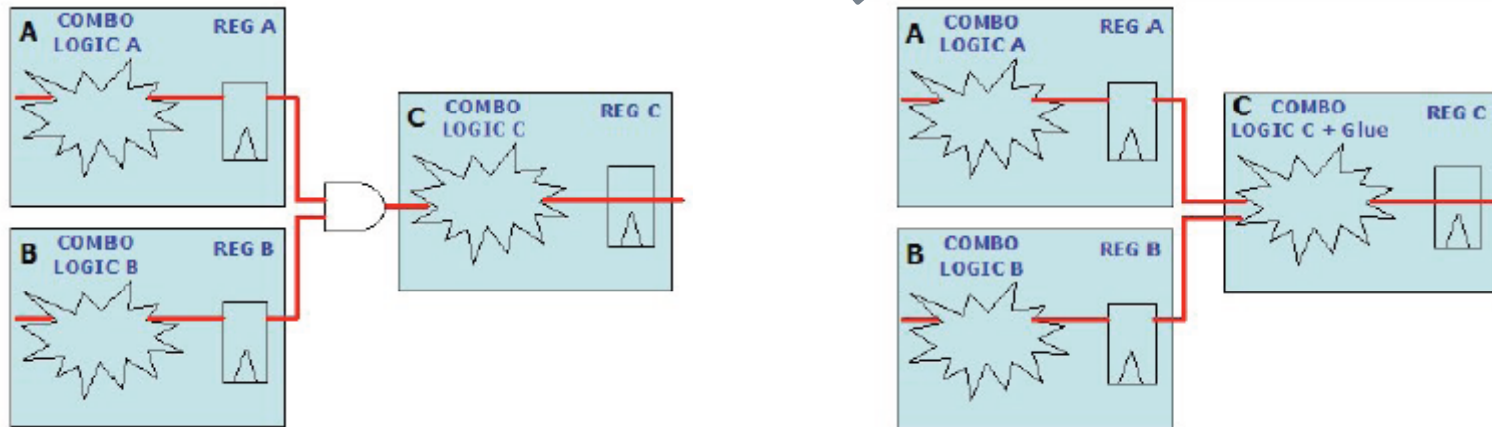
Register All Output

- ❑ For each subblock of a hierarchical macro design, register all output signals from the subblock



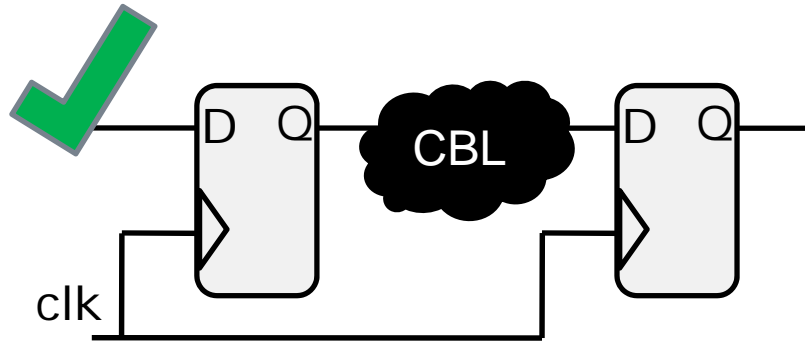
Eliminate Glue Logic at the Top Level

- ❑ Do **not** instantiate gate-level logic at the top level of the macro hierarchy

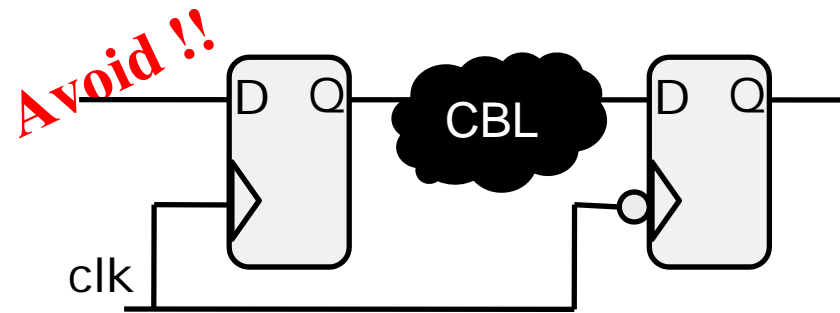


Concept of Clocks and Reset

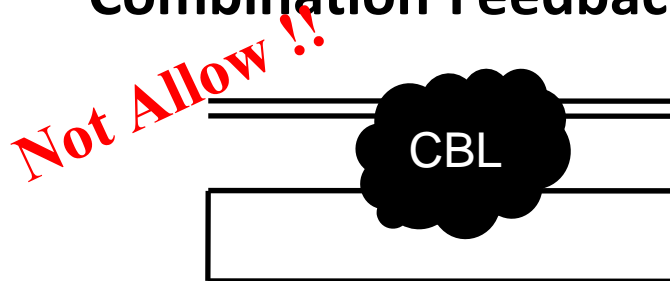
Synchronous



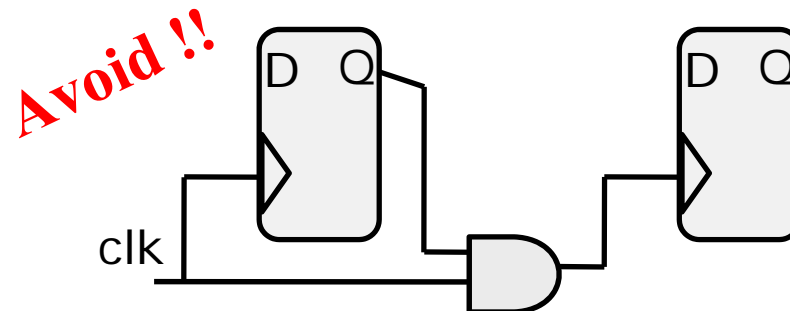
Mixed Clock Edges



Combination Feedback



Gated Clocks



Asynchronous and Synchronous Reset

□ //synchronous reset

```
always@(posedge clock)
    if (rst) begin
        .....
    end
```

□ //asynchronous reset

```
always@(posedge clock or negedge reset)
    if (!rst) begin
        .....
    end
```

Synthesizable Verilog

□ Verilog Basis


- parameter declarations
- wire, wand, wor declarations
- reg declarations
- input, output, inout
- continuous assignment
- module instructions
- gate instructions
- always blocks
- task statement
- function definitions
- for, while loop

□ Synthesizable Verilog primitives cells

- and, or, not, nand, nor, xor, xnor
- bufif0, bufif1, notif0, notif1

Synthesizable Verilog (Cont')

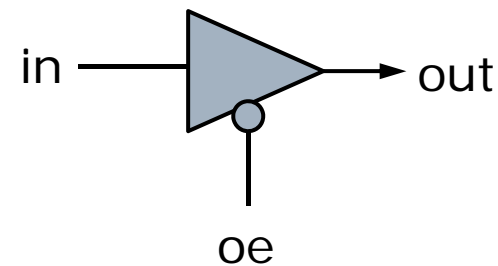
□ Operators

- Concatenation ({ }, {{{}} })
 - Unary reduction (!, ~, &, |, ^)
 - 2's complement arithmetic (+, -, *)
 - Logic shift (>>, <<)
 - Relational (>, <, >=, <=)
 - Equality (==, !=)
 - Binary bit-wise (&, |, ^, ~^)
 - Logical (&&, ||)
 - Conditional (?:)
- precedence
- 
- highest
- lowest

Compiler Unsupported

- delay
- initial
- repeat
- wait
- fork...join
- event
- deassign
- force
- release
- primitive –User defined primitive
- time
- triand, trior, tri1, tri0, trireg
- noms, pmos, cmos, rnmos, rpmos, rcmos
- pullup, pulldown
- rtran, tramif0, tranif1, rtranif0, rtranif1
- case identity and not identity operators
- division and modules operation
- ==, !=
- forever

Example: wire out=(!oe)?in:'hz
(replace "trior")



Coding for Synthesis

□ Combinational Blocks □ Sequential Blocks

```
always @ (d) begin
  case (d)
    2'b00: z = 1'b1;
    2'b01: z = 1'b0;
    default : z = 1'b0;
  endcase
end
```

```
always @ (a or x_temp)
begin
  if (a) begin
    x = x_temp + 1'b1;
  end
  else begin
    x = x_temp;
  end
end
```

```
always @ (posedge clk )
begin
  if (a) begin
    z <= 1'b1;
  end
  else begin
    z <= 1'b0;
  end
end
```

Coding for Synthesis (Cont')

❑ Avoid Combinational Feedback



```
always @ (a or x)begin
  if (a) begin
    x= x+1'b1;
  end
  else begin
    x= x;
  end
end
```



```
always @ (posedge clk) begin
  x_temp<=x;
end

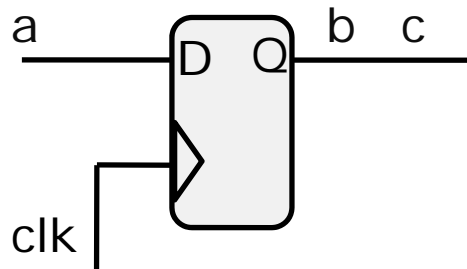
always @ (a or x_temp)begin
  if (a) begin
    x= x_temp+1'b1;
  end
  else begin
    x= x_temp;
  end
end
```


Coding for Synthesis

☐ Blocking Assignment ☐ Non-Blocking Assignment

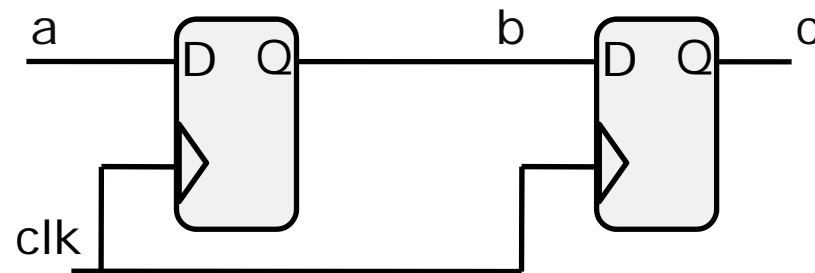
```
always @ (posedge clk)
begin
  b=a;
  c=b;
end
```

Just like "a=c;"



```
always @ (posedge clk)
begin
  b<=a;
  c<=b;
end
```

Just like "shift register"



Coding for Synthesis

❑ Avoid Latches

```
always @ (d) begin
  case (d)
    2'b00: z=1'b1;
    2'b01: z=1'b0;
    default : z=1'b0;
  endcase
end
```

```
always @ (d)begin
  if (a) begin
    .....
  end
  else begin
    .....
  end
end
```

```
always @ (d) begin
  x=1'b0;
  z=1'b0;
  case (d)
    2'b00: begin z=1'b1; x=1'b1; end
    2'b01: begin z=1'b0;      end
    default : begin z=1'b0;    end
  endcase
end
```

```
always @ (posedge clk )begin
  if (a) begin
    z<=1b1;
  end
  else begin
    z<=1'b0;
  end
end
```

Coding for Synthesis (Cont')

□ Sensitivity List

```
always @ (d) begin
  case (d)
    2'b00: z=1'b1;
    2'b01: z=1'b0;
    default : z=1'b0;
  endcase
end
```

```
always @ (a or b or c or d)begin
  if (a) begin
    .....
  end
  else begin
    if (b)begin
      z=c;
    end
    else begin
      z=d;
    end
  end
end
end
```

Coding for Synthesis (Cont')

- ❑ Syntax error for Verilog Simulation
 - Mixed edge-triggered and level-sensitive control in an always block

```
always@(addr or posedge clk)
begin
    ...
end
```

Coding for Synthesis (Cont')

- Key: The multiplexer is a faster circuit. If the priority-encoding structure is not required, we recommend using the case statement
- Using a conditional assignment to infer a Mux
 - `assign out = sel ? a : b ;`

Conditional Expressions

☐ If statement vs. Case statement

■ If statement

- ☐ Priority-encoded logic
- ☐ For speed critical path

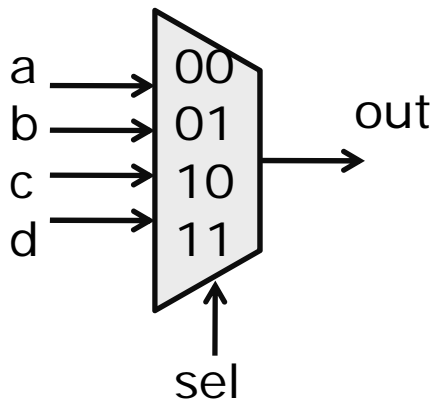
■ Case statement

- ☐ Balanced logic
- ☐ For complex decoding

Conditional Expressions (Cont')

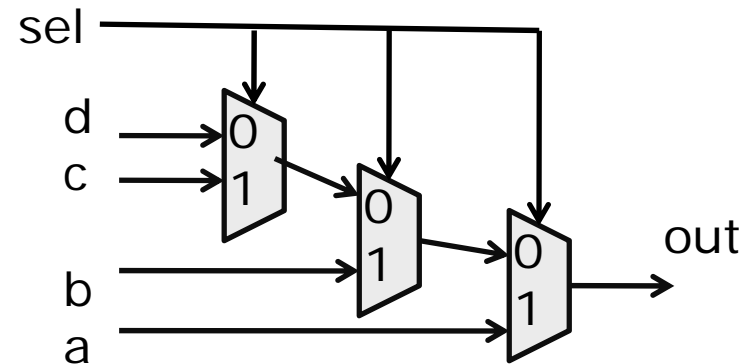
□ Case statements

```
always @ ( sel or a or b or c or d)begin
  case (sel)
    2'b00: out=a;
    2'b01: out=b;
    2'b10: out=c;
    2'b11: out=d;
  endcase
end
```



□ if – else statements

```
always @ ( sel or a or b or c or d)
begin
  if (sel==2'b00) out=a;
  else if (sel==2'b01) out=b;
  else if (sel==2'b10) out=c;
  else out=d;
end
```



FSM Coding Style

□ Explicit FSM design

```
always@(state or in)
begin
    case (state)
        S0:
            if (in) next_state = S1;
            else next_state = S0;
        S1:
            ...
    end

always@(posedge clk)
    if(~reset)
        state <= S0;
    else
        state <= next_state;
```


Non-Synthesizable Style

- ❑ Either non-synthesizable or incorrect after synthesis
- ❑ **initial** block is forbidden (non-synthesizable)
- ❑ Multiple assignments (multiple driving sources)

(non-synthesizable)

```
always@(src1 or src2)
    result = src1 + src2;
always@(inc1 or inc2 or offset)
    result = inc1 + inc2 + offset;
```

- ❑ Mixed blocking and non-blocking assignment

```
always@(src1 or src2 or inc)
begin
    ...
    des = src1 + src2;
    inc <= src1-4;
    ...
end
```

Tips for Verilog Design

- ☐ Resource Sharing
- ☐ Scalable Design
- ☐ Using () to describe complex circuits
- ☐ Timescale

Resource Sharing

- Operations can be shared if they lie in the same always block

```
Always @ (sel or a or b or c )  
begin  
    if (sel) z=a+b;  
    else    z=a+c;  
end
```

Scalable Design

```
parameter cb_size=8;
parameter data_size=64;
parameter address_size=13;

input clk;
input cen;
input wen;
input oen;
input [address_size-1:0] address;
input [data_size-1:0] data;
//
output [data_size-1:0] Q;
output ed;
output dec;
```

```
parameter size=8;
wire [3:0] a,b,c,d,e;
```

```
assign a=size+2;
```

```
assign b=a+1;
```

```
assign c=d+e;
```

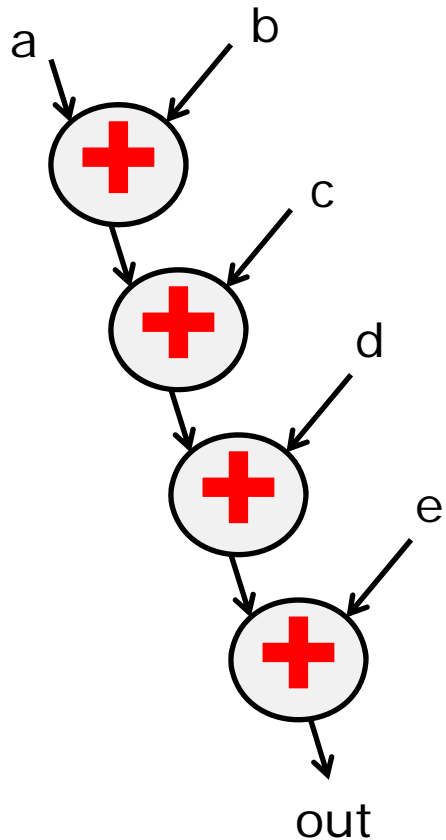
Constant
Increaser
Adder

Omit for Synthesis

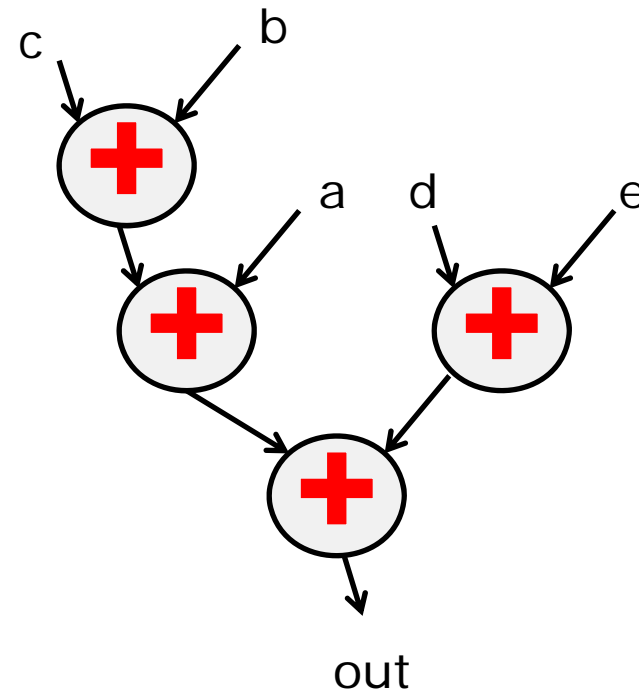
- ❑ Omit the Wait for XX ns Statement
 - Do not use “#XX;”
- ❑ Omit the ...After XX ns or Delay Statement
 - Do not use “assign #XX Q=0;”
- ❑ Omit Initial Values
 - Do not use “initial sum = 1'b0;”

Using () to describe complex circuits.

□ $\text{out} = a + b + c + d + e;$



□ $\text{out} = ((a + (b + c)) + (d + e));$



Timescale

- ❑ **`timescale**: which declares the time unit and precision.
 - ``timescale <time_unit> / <time_precision>`
 - e.g. : ``timescale 1s/1ps`, to advance 1 sec, the timewheel scans its queues 10^{12} times versus a ``timescale 1s/1ms`, where it only scans the queues 10^3 times.
- ❑ The `time_precision` must be at least as precise as the `time_unit`
- ❑ Keep precision as close in scale to the time units as is practical
- ❑ If not specified, the simulator may assign a default timescale unit
- ❑ The smallest precision of all the timescale directive determines the “simulation time unit” of the simulation.

Coding for Synthesis

- ☐ No initial in the RTL code
- ☐ FFs are preferred
- ☐ Avoid unnecessary latches
- ☐ Avoid combinational feedback
- ☐ For sequential blocks, use no-blocking statements
- ☐ For combinational blocks, use blocking statements
- ☐ Coding state machines
 - Two procedure blocks: one for the sequential and one for the combinational
 - Keep FSM logic and non-FSM logic in separate modules
 - Assign a default state

Artisan Memory Compiler

Overview

❑ Artisan SRAM Types:

Generator	Product Name	Executable
High-Speed/Density Single-Port SRAM	SRAM-SP	ra1sh
High-Speed/Density Dual-Port SRAM	SRAM-DP	ra2sh
High-Density Single-Port SRAM	SRAM-SP-HD	ra1shd
High-Density Dual-Port SRAM	SRAM-DP-HD	ra2shd
Low-Power Single-Port SRAM	SRAM-SP-LP	ra1shl

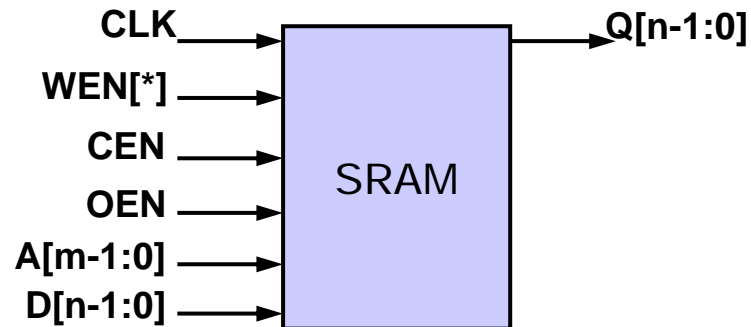
[REF: [Artisan User Manual](#)]

❑ Only **ra1shd** and **ra2sh** are supported in school

❑ Generated files:

- Memory Spec. (i.e. used for layout-replacement procedure in CIC flow)
- Memory Data Sheet
- Simulation models: Verilog Model & VHDL Model
- Memory Libraries for P&R: Synopsys Model & VCLEF Footprint
- Timing Files: TLF Model & PrimeTime Model

Pin Descriptions for Single-Port SRAM

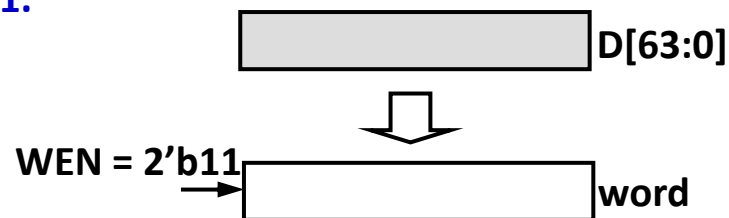


Name	Type	Description
Basic Pins		
CLK	Input	Clock
WEN[*]	Input	Write enable, active low. *If word-write mask is enabled, this becomes a bus
CEN	Input	Chip enable, active low
OEN	Input	Tri-state output enable
A[m-1:0]	Input	Address (A[0]=LSB)
D[n-1:0]	Input	Data inputs (D[0]=LSB)
Q[n-1:0]	Output	Data outputs (Q[0]=LSB)

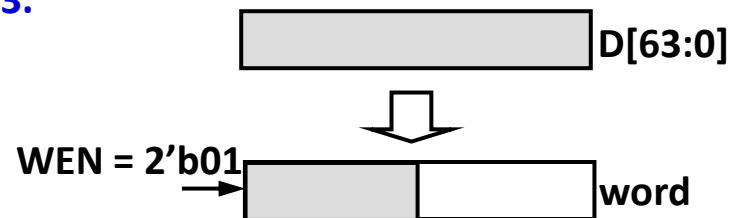
Example for Word-Write Mask

- Word Width: 64 bits
 - Word Partition Size: 32 bits
 - Mask Width = WEN Width = 2
 - WEN[1:0]
 - 11: No write
 - 10: Write to LSB part
 - 01: Write to MSB part
 - 00: Write to the whole word

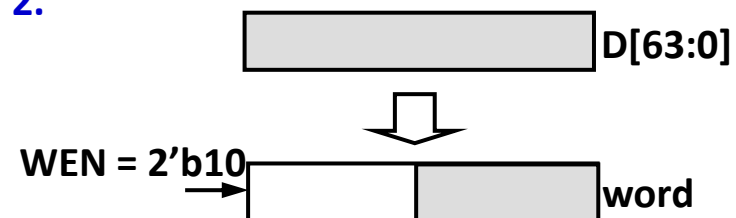
1.



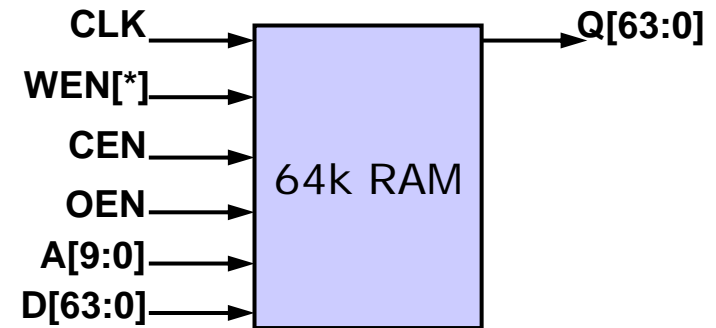
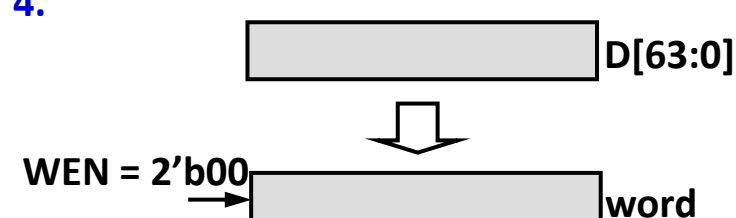
3.



2.

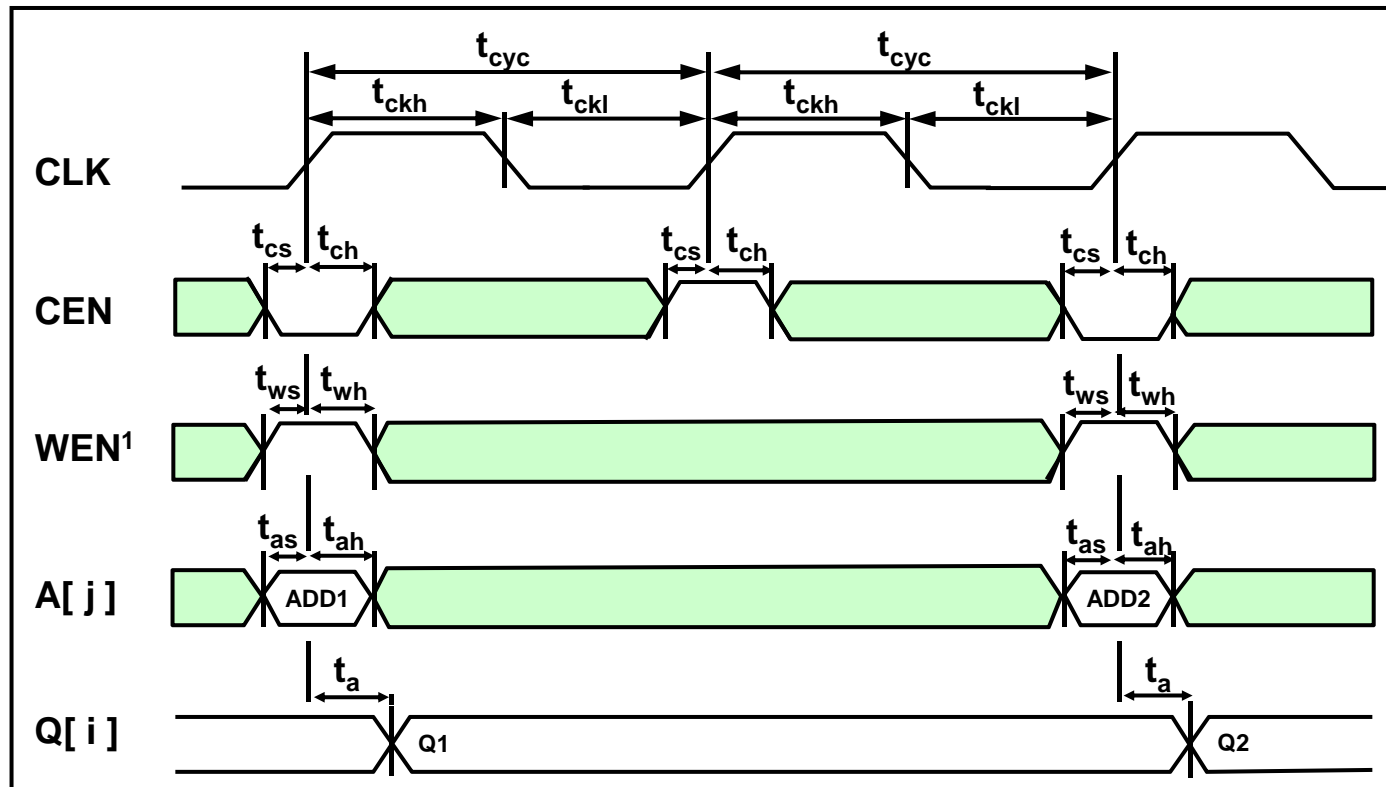


4.



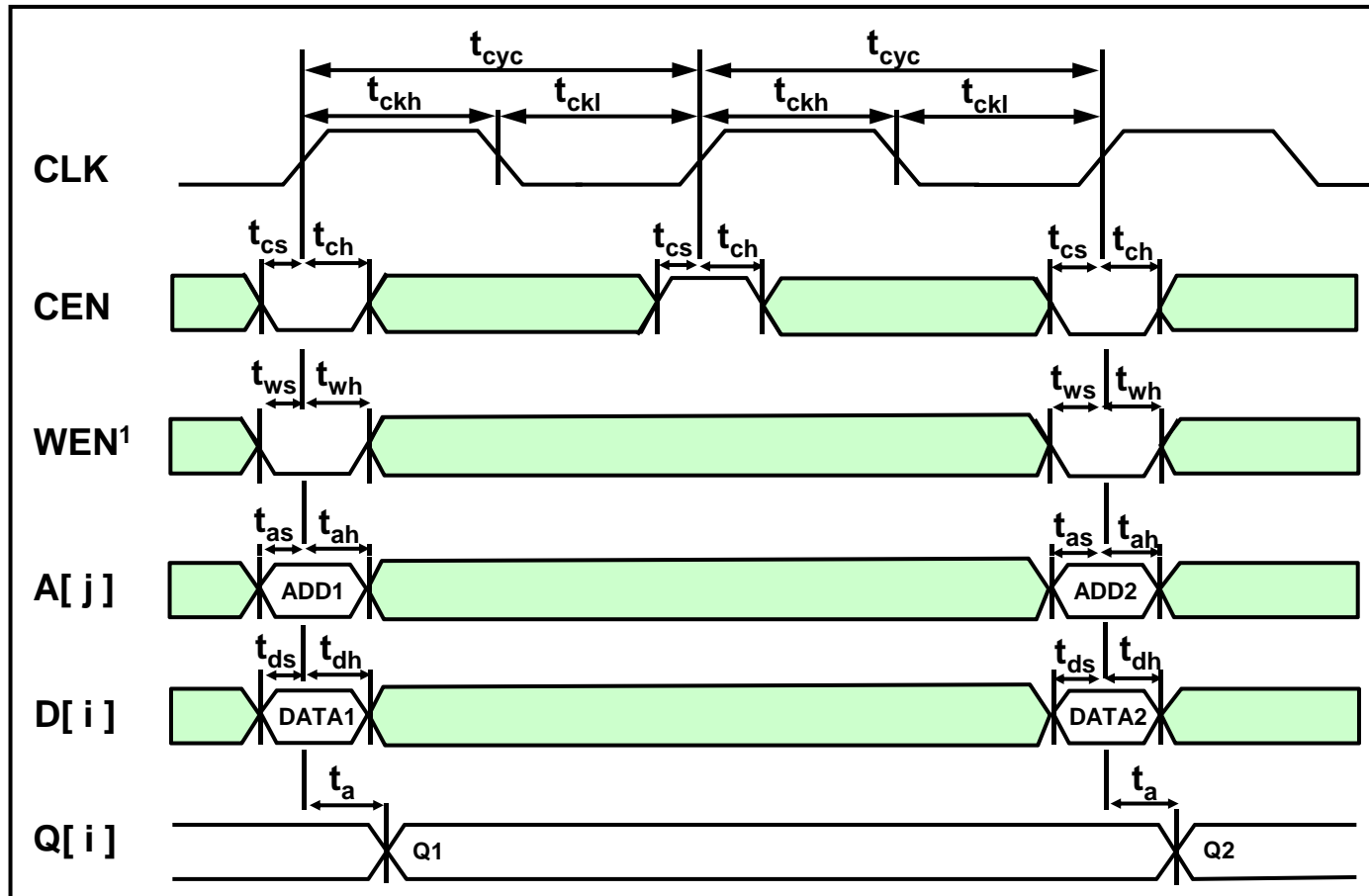
Waveforms for Single-Port SRAM

□ Read Cycle



Waveforms for Single-Port SRAM (Cont')

Write Cycle

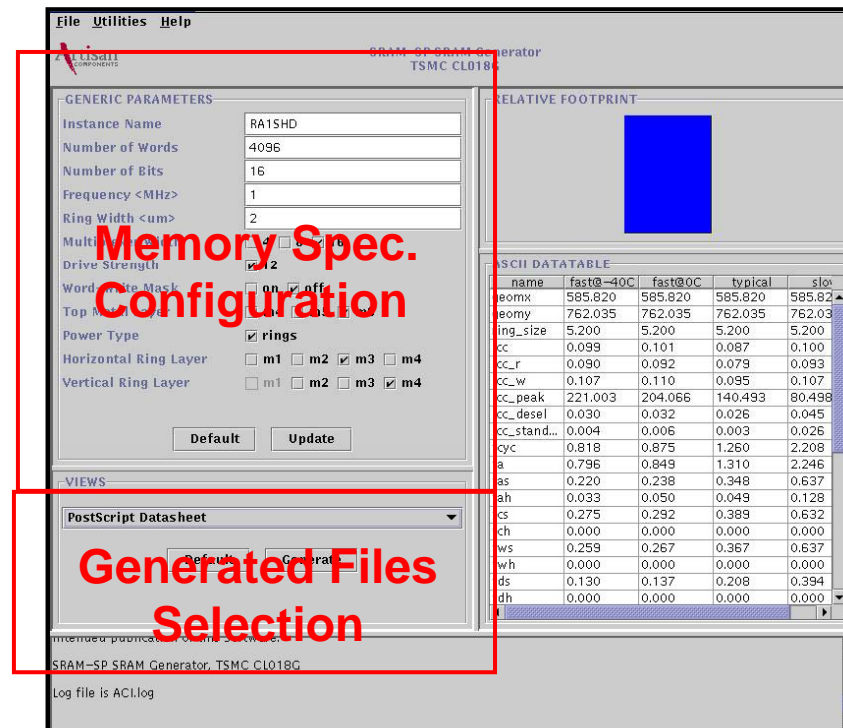


Getting Started

❑ `linux %> ssh -l "user name" cae18.ee.ncu.edu.tw` — Connect to Unix

(1-port RAM) `unix%> ~/cell_lib/CBDK018_TSMC_Artisan/CIC/Memory/ra1shd/bin/ra1shd`

(2-port RAM) `unix%> ~/cell_lib/CBDK018_TSMC_Artisan/CIC/Memory/ra2sh/bin/ra2sh`

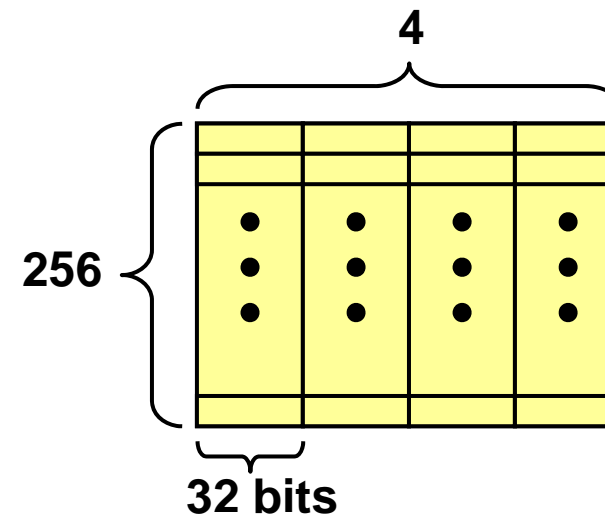
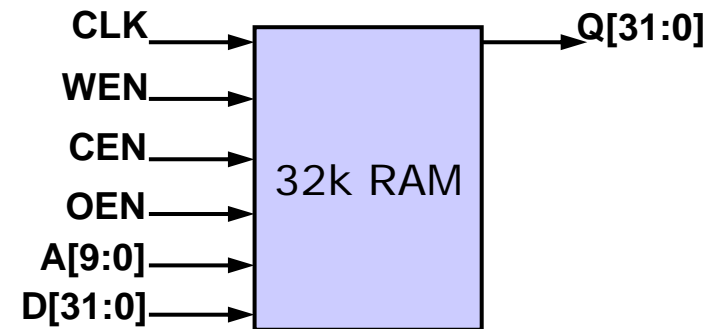


(GUI view of the Artisan)

Memory Spec Configuration (Example 1)

- ☐ Instance Name mem_32k
- ☐ Number of Words 1024
- ☐ Number of Bits 32
- ☐ Frequency <MHz> 100
- ☐ Ring Width <um> 2
- ☐ Multiplexer Width
 - ☒ 4 ☐ 8 ☐ 16
- ☐ Drive Strength
- ☐ Word-Write Mask
 - ☐ on ☒ off
- ☐ Top Metal Layer
 - ☐ m4 ☒ m5 ☐ m6
- ☐ Power Type
- ☐ Horizontal Ring Layer
 - ☐ m1 ☐ m2 ☒ m3 ☐ m4
- ☐ Vertical Ring Layer
 - ☐ m2 ☐ m3 ☒ m4

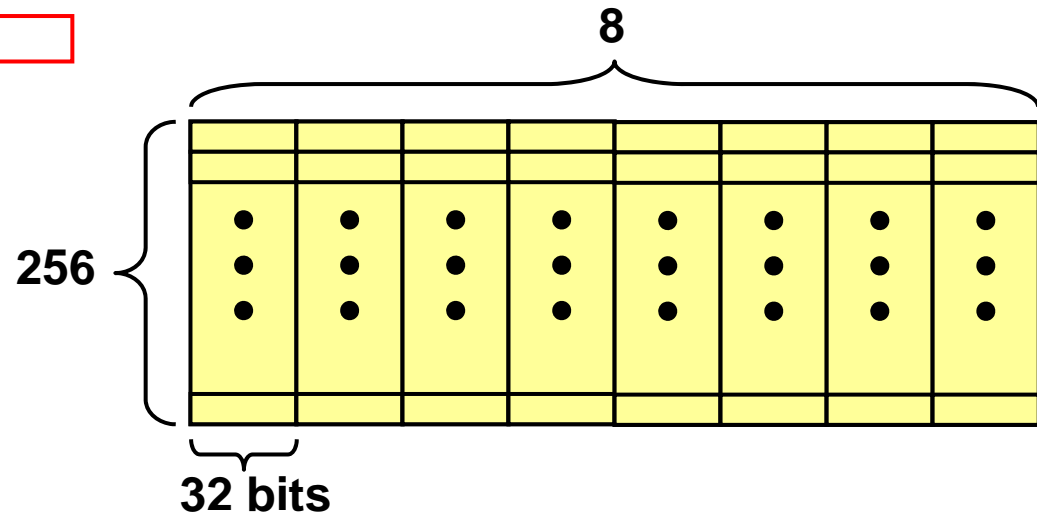
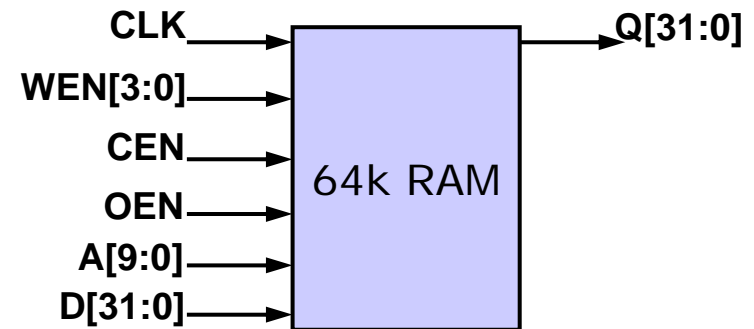
Ex: 32k RAM (no mask write)



Memory Spec Configuration (Example 2)

- ☐ Instance Name mem_64k
- ☐ Number of Words 2048
- ☐ Number of Bits 32
- ☐ Frequency <MHz> 100
- ☐ Ring Width <um> 2
- ☐ Multiplexer Width
 - ☒ 4 ☒ 8 ☐ 16
- ☐ Drive Strength
- ☐ Word-Write Mask
 - ☒ on ☐ off
 - ☐ Word Partition Size 8
- ☐ Top Metal Layer
 - ☐ m4 ☒ m5 ☐ m6
- ☐ Power Type
- ☐ Horizontal Ring Layer
 - ☐ m1 ☐ m2 ☒ m3 ☐ m4
- ☐ Vertical Ring Layer
 - ☐ m2 ☐ m3 ☒ m4

Ex: 64k RAM (with mask write)

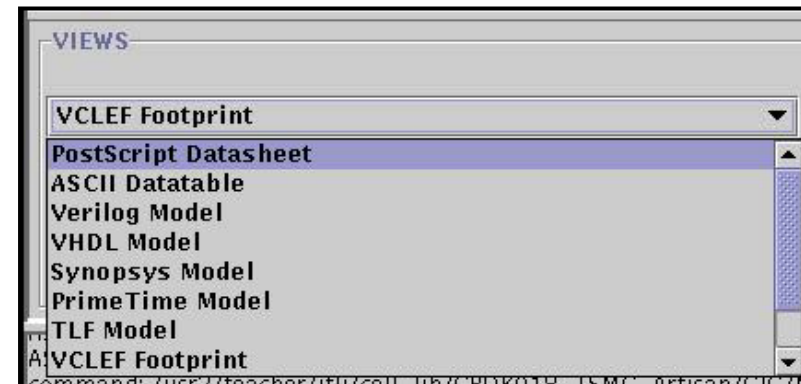


File Generation (Method 1)

☐ Pop-up window

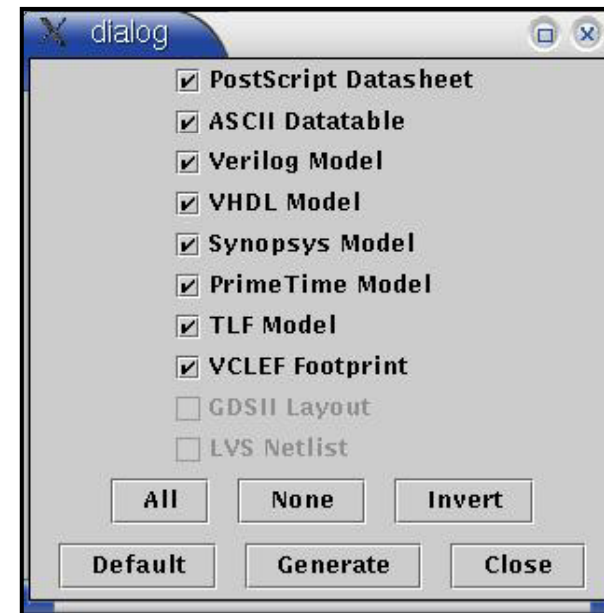
- PostScript Datasheet (.ps)
 - ☐ Convert to PDF file: *ps2pdf *.ps*
- ASCII Datatable (.dat)
- Verilog Model (.v)
- VHDL Model (.vhd)
- Synopsys Model (.lib)
 - ☐ The default library name is “USERLIB”
- PrimeTime Model
- TLF Model
- VCLEF Footprint (.vclef)

(File Selection)



File Generation (Method 2)

□ From the menu



□ Spec. Generation

- The memory spec. file will be used for the Layout Replacement procedure in the CIC server

