

A decorative graphic on the left side of the slide, consisting of a network of white lines and circles on a blue gradient background, resembling a circuit board or a tree structure.

DIGITAL DESIGN

LAB14 VERILOG SUMMARY

WANGW6@SUSTECH.EDU.CN

LAB14 PART2

- Verilog
 - Don'ts in Verilog
 - Summary
 - Synthesizable vs Non-Synthesizable

MEMORY IN VERILOG

- Memory can be seen as a set of registers with the same bit width.

Modeling memory by building arrays of reg variables, and addressing each unit of the array by array index

- Definition:

```
reg [n-1:0] memory name [m-1:0]; // there are m units in memory, the size of each unit in the memory is n .
```

- Notes:

- A n-bit register can be assigned in an assignment statement, but **a full memory CAN NOT**.
- If you need to read and write a storage unit in memory, you must specify the **address** of the unit in memory.

To define a memory named **Mema** which has **5** memory units, each with a bit width of **3** bits.

```
reg [2:0] Mema [4:0];
```

To assign 3'b101 to Mema [1] unit in Mema

```
Mema [1]= 3'b101;
```

DON'TS IN VERILOG(1)

- **Non-Synthesizable Verilog** which is **NOT suggested** in your design
 - initial
 - Task, function
 - System task: \$display, \$monitor, \$strobe, \$finish
 - fork... join
 - **UserDefinedPrimitive**

DON'TS IN VERILOG(2)

- **Incomplete “if else” block or Incomplete “case” in combinational logical block** which are **NOT suggested** in your design.
 - a unexpected latch would be generated by EDA tool while finish the synthesis, the latch is not good for the combinational logic.

```
module updown_counter(D,CLK,CR,LD,UP,Q)
input [3:0]D;
input CLK,CR,LD,UP;
output reg [3:0] Q;

always @(posedge CLK )

if(!CR)
    Q=0;
else if(!LD)
    Q=D;
else if(UP)
    Q=Q+1;

endmodule
```

```
module decorder(cIn,data,addr);
input cIn;
input [1:0] addr;
output reg [3:0] data;

always @(cIn or addr )
begin
if(0==cIn)
    data=4'b0000;
else
    case(addr)
        2'b00:data=4'b1110;
        2'b01:data=4'b1101;
        2'b10:data=4'b1011;
    endcase
end
endmodule
```

DON'TS IN VERILOG(3)

- **NOT suggested**

- Embedded 'if-else'

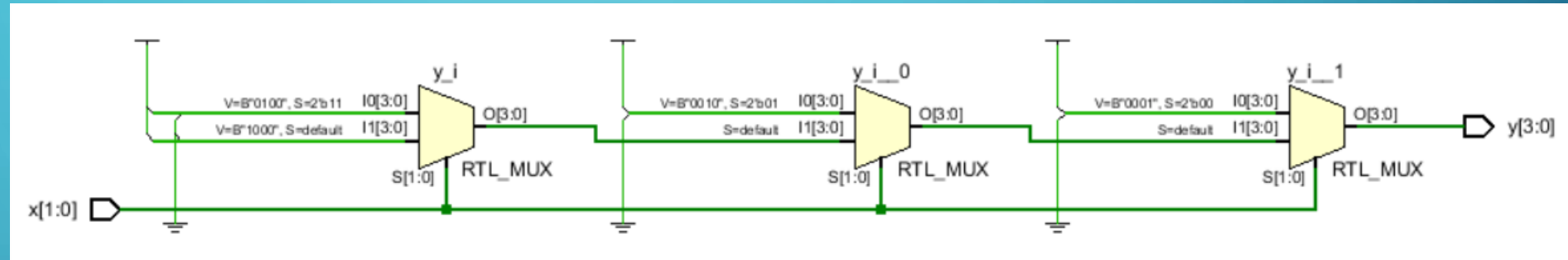
- **Suggested**

- Using an asynchronous reset to make your system go to initial state
- Using 'case' instead of embedded 'if-else' to avoid unwanted priority and longer delay

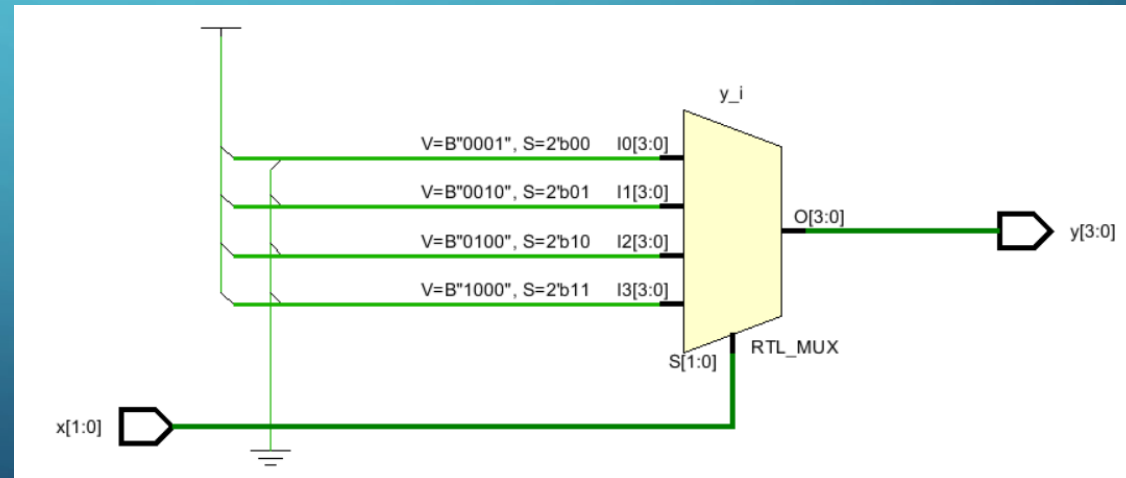
VERILOG (BE CAREFUL WITH EMBEDDED IF-ELSE)

- Embedded 'if-else' circuit brings priority and more latency compared to 'case'

```
always @*
  if( 2'b00 == x)
    y = 4'b0001;
  else if( 2'b01 == x)
    y = 4'b0010;
  else if( 2'b11 == x)
    y = 4'b0100;
  else
    y = 4'b1000;
```



```
always @*
  case(x)
    2'b00: y=4'b0001;
    2'b01: y=4'b0010;
    2'b10: y=4'b0100;
    2'b11: y=4'b1000;
  endcase
```



DON'TS IN VERILOG(4)

- **NOT suggested**

- Two different edge trigger for one always block
- (!!!) **a signal/port is assigned in more than one always block** (it won't report error while synthesized but its behavior maybe wrong after synthesize, pay special attention about the critical warning: **multiple driver**)
- Mix-use blocking assignment and non-blocking assignment in one always block.

VERILOG SUMMARY(1)

- **Design-Under-Test vs Test-Bench**

- **Structured design**

- (top module , instance modules)

- **Block**

- Combinational, Sequential

- **Statement**

- continuous assignment
- Unblock assignment vs block assignment
- If else, case, loop

- **Variable vs Constant**

- reg vs wire
- Splicing { , }
- Number system

VERILOG SUMMARY(2)

- **Non-Synthesizable Verilog** which is **NOT suggested** to use in your design
 - initial
 - Task, function
 - System task : \$display, \$monitor, \$strobe, \$finish
 - fork... join

VERILOG SUMMARY(3)

- **Suggested**

- Using an asynchronous reset to make your system go to initial state
- Using case instead of embedded 'if-else' to avoid unwanted priority and longer delay

- **NOT suggested**

- Embedded 'if-else'
- Two different edge trigger for one always block
- (!!!) **a signal/port is assigned in more than one always block** (it won't report error while synthesized but its behavior maybe wrong after synthesize)
- Mix-use blocking assignment and non-blocking assignment in one always block

DUT VS TESTBENCH

- DUT is a designed module with input and output ports
 - While do the design, non-synthesizable grammar means can't be convert to circuit, is NOT suggested!
 - DUT may be a top module using structured design which means the sub module is instanced and connected in the top module
- Testbench is used for test DUT with NO input and output ports
 - Instance the DUT, bind its ports with variable, set the states of variable which bind with inputs and check the states of variable which bind with outputs
 - Testbench is NOT part of Design, it only runs in FPGA/ASIC EDA, so the un-synthesizable grammar can be used in testbench

MODULE (STRUCTURED LEVEL VS TESTBENCH)

```
module multiplexer_153(out,c0,c1,c2,c3,a,b,g1n);
input c0,c1,c2,c3;
input a,b;
input g1n;
output reg [3:0] out;

always @(*)
if(1'b0==g1n)
    case({b,a})
        2'b00:out=4'b1110;
        2'b01:out=4'b1101;
        2'b10:out=4'b1011;
        2'b11:out=4'b0111;
    endcase
else
    out = 4'b1111;
endmodule
```

```
module multiplexer_153_2(out1,out2,c10,c11,c12,c13,a1,b1,g1n,
    c20,c21,c22,c23,a2,b2,g2n);
input c10,c11,c12,c13,a1,b1,g1n,c20,c21,c22,c23,a2,b2,g2n;
output out1,out2;

multiplexer_153 m1(
    .g1n(g1n),
    .a(a1),
    .b(b1),
    .c0(c10),
    .c1(c11),
    .c2(c12),
    .c3(c13),
    .out(out1)
);

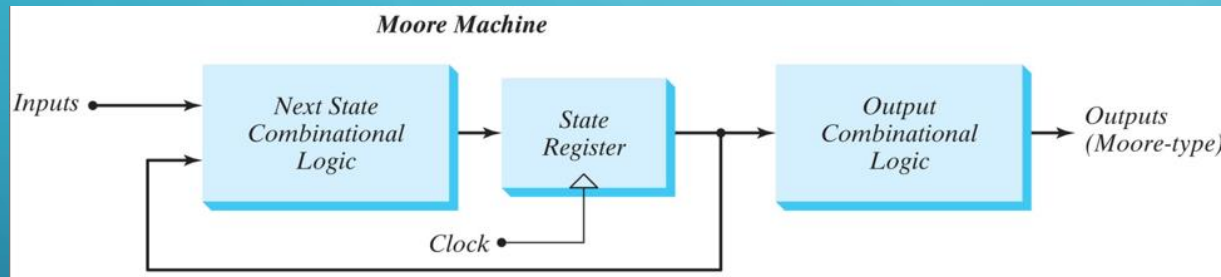
multiplexer_153 m2(
    .g1n(g2n),
    .a(a2),
    .b(b2),
    .c0(c20),
    .c1(c21),
    .c2(c22),
    .c3(c23),
    .out(out2)
);

endmodule
```

```
module lab3_df_sim( );
    reg simx,simy;
    wire simq1,simq2,simq3;
    lab3_df u_df(
        .x(simx), .y(simy), .q1(simq1), .q2(simq2), .q3(simq3) );

    initial
    begin
        simx=0;
        simy=0;
        #10
        simx=0;
        simy=1;
        #10
        simx=1;
        simy=0;
        #10
        simx=1;
        simy=1;
    end
endmodule
```

FSM AND VERILOG



```
`timescale 1ns / 1ps
///////////////////////////////////////////////////
module moore_2b(input clk,rst_n,x_in,output[1:0] state,next_state);
    reg [1:0] state,next_state;
    parameter S0=2'b00,S1=2'b01,S2=2'b10,S3=2'b11;
    always @(posedge clk,negedge rst_n) begin
        if(~rst_n)
            state <= S0;
        else
            state <= next_state;
    end
    always @(state,x_in) begin
        case(state)
            S0: if(x_in) next_state = S1; else next_state = S0;
            S1: if(x_in) next_state = S2; else next_state = S1;
            S2: if(x_in) next_state = S3; else next_state = S2;
            S3: if(x_in) next_state = S0; else next_state = S3;
        endcase
    end
end
endmodule
```


MODULE DESIGN

- Gate level
 - Implementation from the perspective of gate-level structure of the circuit, Using gates as components, connecting pins of gates
 - using logical and bitwise operators or original primitive(not , or , and , xor , xnor ..)
- Data streams
 - Implementation from the perspective of data processing and flow
 - Using continuous assignment, pay attention to the correlation between signals, the difference between logical and bitwise operators
- Behavior level
 - Implementation from the perspective of the Behavior of Circuits
 - Implemented in the always statement block
 - The variable which is assigned in the always block Must be Reg type.

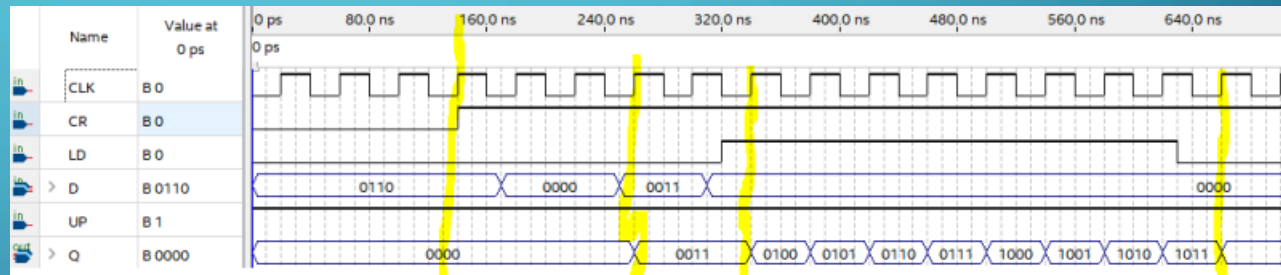
IF – ELSE IN BEHAVIOR MODELING

'if else' block can represent the priority between signals

From the overall structure, from top to bottom, priority decreases in turn

```
module updown_counter(D,CLK,CR,LD,UP,Q)
input [3:0]D;
input CLK,CR,LD,UP;
output reg [3:0] Q;

always @(posedge CLK )
if(!CR)
Q=0;
else if(!LD)
Q=D;
else if(UP)
Q=Q+1;
else
Q=Q-1;
endmodule
```



NOTIC:

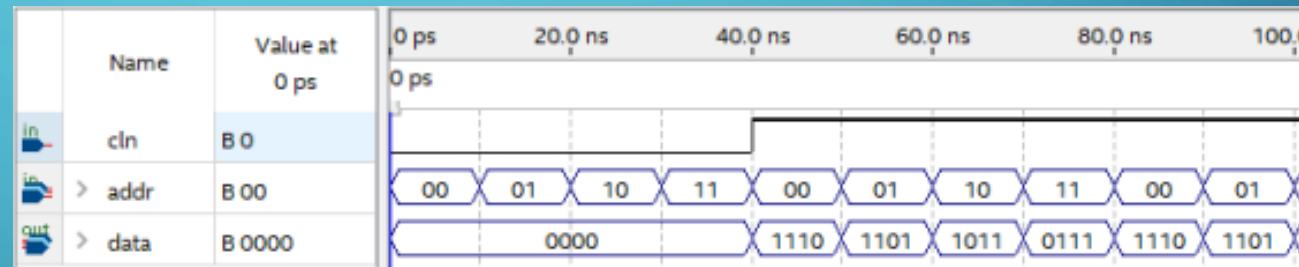
- 1) If there is no 'else' branch in the statement, latches will be generated while doing the synthesis.
- 2) Nested 'if-else' is NOT suggested, 'case' is suggested as an alternative.

CASE IN BEHAVIOR MODELING

case	0	1	x	z
0	1	0	0	0
1	0	1	0	0
x	0	0	1	0
z	0	0	0	1

```
module decoder(cIn,data,addr);
input cIn;
input [1:0] addr;
output reg [3:0] data;

always @(cIn or addr )
begin
if(0==cIn)
data=4'b0000;
else
case(addr)
2'b00:data=4'b1110;
2'b01:data=4'b1101;
2'b10:data=4'b1011;
2'b11:data=4'b0111;
endcase
end
endmodule
```



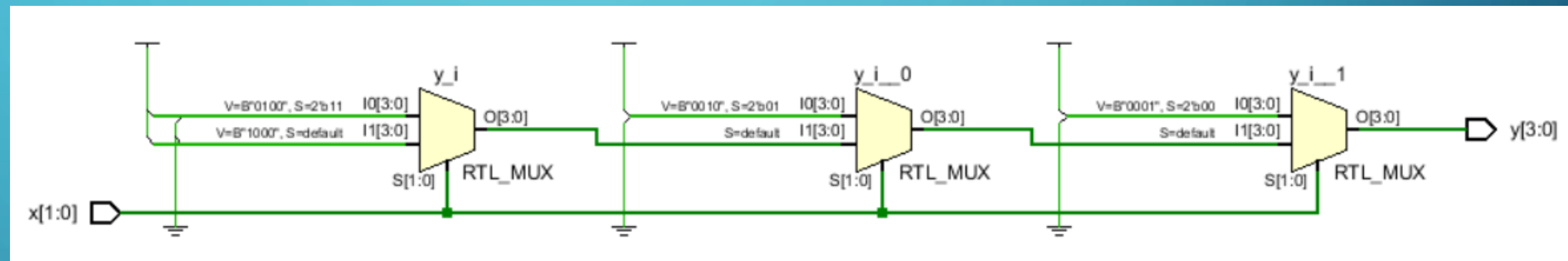
NOTIC:

Without defining default branches and NOT all situations is cleared under the “case”, latches will be generated while doing the synthesis.

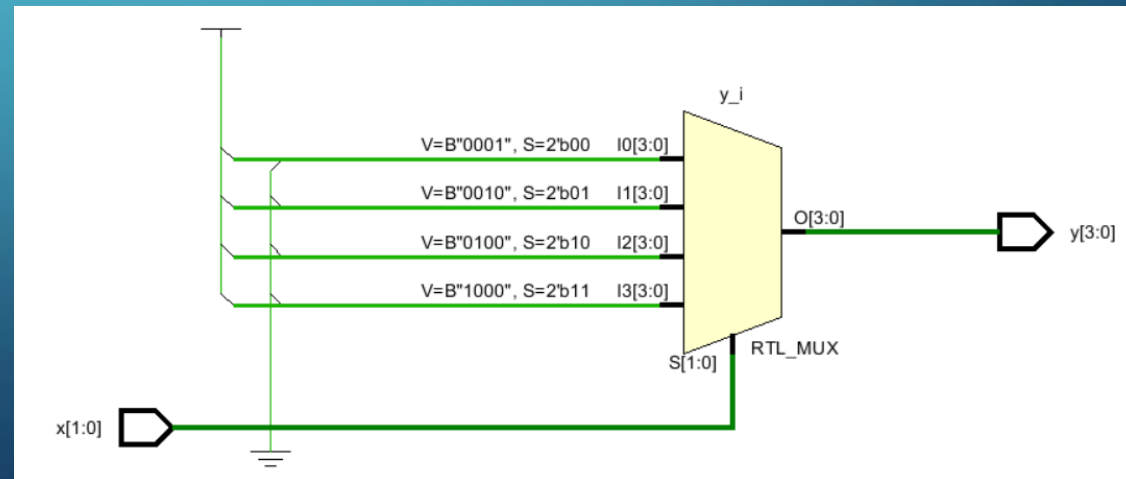
VERILOG (BE CAREFUL WITH EMBEDDED IF-ELSE)

- Embedded 'if-else' circuit brings priority and more latency compared to 'case'

```
always @*  
  if( 2'b00 == x)  
    y = 4'b0001;  
  else if( 2'b01 == x)  
    y = 4'b0010;  
  else if( 2'b11 == x)  
    y = 4'b0100;  
  else  
    y = 4'b1000;
```



```
always @*  
  case(x)  
    2'b00: y=4'b0001;  
    2'b01: y=4'b0010;  
    2'b10: y=4'b0100;  
    2'b11: y=4'b1000;  
  endcase
```



LOOP IN BEHAVIOR MODELING

- Loop is NOT used much in design, for its comprehensiveness is not very good.
- Loop is most often used in testbench to specify signal behavior:

```
repeat(12)
  begin
    Cin = 1'b0;
    Cin = #40000 1'b1;
    # 40000;
  end
```

STATEMENT

- Assignment

- Continuous assign (MUST to a wire variable)

- `assign A = 1'b0 ;` `//A MUST be defined as a wire`

- Block assign(used in initial or always block, MUST to a reg variable, usually in combinational block)

- `always @ *`

- `A = 1'b0;` `//A MUST be defined as a reg`

- `initial`

- `A = 1'b0;` `//A MUST be defined as a reg`

- Un-block assign (used in initial or always block, MUST to a reg variable , usually in sequential block)

- `always @(posedge clk)`

- `A <= 1'b0 ;` `//A MUST be defined as a reg`

CONSTANT(1)

- Expression
 - `<bit width>'<numerical system expression><number in the numerical system >`
 - numerical system expression
 - B / b : Binary
 - O / o : Octal
 - D / d : decimal
 - H / h : hexadecimal
 - `'<numerical system expression><number in the numerical system >`
 - The default value of bit width is based on the machine-system(at least 32 bit)
 - `<number>` : default in decimal
 - The default value of bit width is based on the machine-system(at least 32 bit)

CONSTANT(2)

- x(uncertain state) and z (High resistivity state)
 - The default value of a wire variable is Z before its assignment
 - The default value of a reg variable is X before its assignment
- negative value
 - Minus sign must be ahead of bit-width
 - -4'd3 (is ok) while 4'd-3 is illegal
- underline
 - Can be used between number but can NOT be in the bit width and numerical system expression
 - 8'b0011_1010 (is ok) while 8'_b_0011_1010(is illegal)

CONSTANT(3)

- Parameter (symbolic constants)
 - Used for improve the Readability and maintainability
 - Declare an identifier on a constant
 - Parameter `p1=expression1,p2=expression2,..;`

VARIABLE (1)

- Variable
 - Changeable while process
- Wire
 - Net
 - Can 't store info, must be driven (such as continuous assignment)
 - The input and output port of module is wire by default
 - Can NOT be the type of left-hand side of assignment in initial or always block

```
wire a;  
wire [7:0] b;  
wire [4:1] c,d;
```

VARIABLE (2)

- Reg
 - **MUST** be the type of left-hand side of assignment in initial or always block
 - The default initial value of reg is an indefinite value X. Reg data can be assigned positive values and negative values.
 - The value stored in register are raw binary data, it depends how you interpret the value.
 - By default, it's an unsigned value, that is, a positive value.
 - For example, a 4-bit register contains value 1011 stands for a decimal 11.
 - If you assign a -1 to the register, it is automatically represented using 2's complements, i.e. 1111.

WIRE VS REG

```
module sub_wr();  
  input reg in1,in2;  
  output out1;  
  output out2;  
endmodule
```

Error: Port in1 is not defined
Error: Non-net port in1 cannot be of mode input
Error: Port in2 is not defined
Error: Non-net port in2 cannot be of mode input

```
module sub_wr(in1,in2,out1,out2);  
  input in1,in2;  
  output out1;  
  output reg out2;  
  
  assign in1 = 1'b1;  
  
  initial begin  
    in2 = 1'b1;  
  end  
endmodule
```

Error: procedural assignment to a non-register in2 is not permitted, left-hand side should be reg/integer/time/genvar

```
23 module test_wire_reg(  
24   );  
25   wire i1,i2;  
26   reg o1,o2;  
27   sub_wr s1(i1,i2,o1,o2);  
28 endmodule  
29  
30 module sub_wr(in1,in2,out1,out2);  
31   input in1,in2;  
32   output out1;  
33   output reg out2;  
34 endmodule
```

[Synth 8-685] variable 'o1' should not be used in output port connection [test_wire_reg.v:27]

MEMORY

- Memory can be seen as a set of registers with the same bit width.

Modeling memory by building arrays of reg variables, and addressing each unit of the array by array index

- Definition:

```
reg [n-1:0] memory name [m-1:0]; // there are m unit in memory, the size of each unit in the memory is n .
```

- Notes:

- A n-bit register can be assigned in an assignment statement, but a full memory CAN NOT.
- If you need to read and write a storage unit in memory, you must specify the address of the unit in memory.

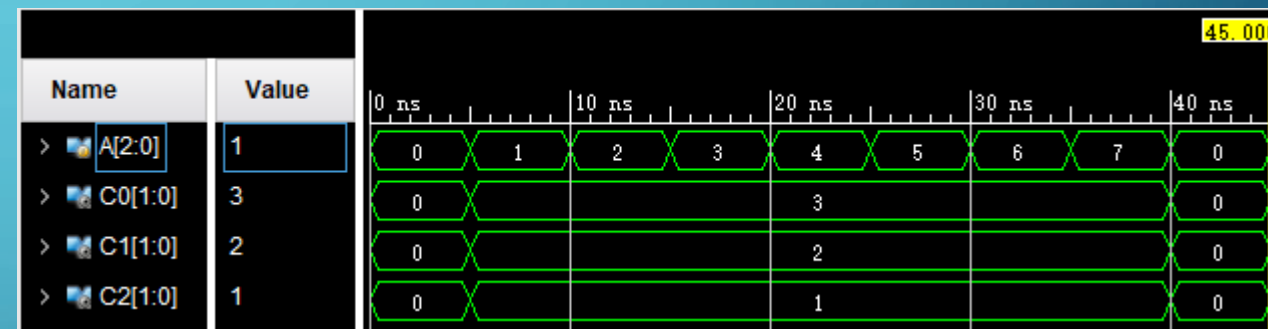
```
reg [2:0] Mema [4:0]; // define a memory named Mema which has 5 memory units, each with a bit width of 3 bits.
```

```
Mema [1]= 3'b101; // assign 3'b101 to Mema [1] unit in Mema
```

MEMORY (DEMO)

```
module test(  
    A, C0, C1, C2  
);  
    input [2:0] A;  
    output [1:0] C0, C1, C2;  
    reg[1:0] B [2:0];  
    assign {C0, C1, C2} = {B[0], B[1], B[2]};  
    always @(A)  
    if(A)  
    begin  
        B[0] = 2'b11;  
        B[1] = 2'b10;  
        B[2] = 2'B01;  
    end  
    else  
    begin  
        B[0] = 2'b00;  
        B[1] = 2'b00;  
        B[2] = 2'B00;  
    end  
endmodule
```

```
module test(  
    A, C0, C1, C2  
);  
    input [2:0] A;  
    output [1:0] C0, C1, C2;  
    reg[1:0] B [2:0];  
    assign {C0, C1, C2} = {B[0], B[1], B[2]};  
    always @(A)  
    if(A)  
    begin  
        {B[0], B[1], B[2]} = 6'b011011;  
        /*B[0] = 2'b11;  
        B[1] = 2'b10;  
        B[2] = 2'B01;*/  
    end  
    else  
    begin  
        {B[0], B[1], B[2]} = 6'b0;  
        /*B[0] = 2'b00;  
        B[1] = 2'b00;  
        B[2] = 2'B00;*/  
    end  
endmodule
```



OPERATOR(1)

highest priority	! ~
	* / %
	+ -
	<< >>
	< <= > >=
	== != === !==
	&
	^ ^~
	&&
lowest priority	
	? :

- Bit splicing operator { }

multiple data or bits of data are separated by commas in order, then using braces to splice them as a whole.

Such as: { a, B[1:0], w, 2'b10 } // Equivalent to { a, B[1], B[0], w, 1'b1, 1'b0 }

- Repetition can be used to simplify expressions

{ 4 {w} } // Equivalent to { w, w, w, w }


{ b, {2 {x, y} } } // Equivalent to { b, x, y, x, y }

OPERATOR(2)

When numeric values are used for conditional judgment, non-zero values represent logical truth and zero values represent logical false.

```
module test_bool(A,C);
input [2:0]A;
output reg [2:0]C;

always @(A )
begin
if(A)
C=2'b11;
else
C=2'b00;
end
endmodule
```

		Value at		0 ps	20.0 ns	40.0 ns	60.0 ns	80.0 ns				
Name		0 ps		0 ps								
	> A	B 000		000	001	010	011	100	101	110	111	000
	> C	B 000		000	011					000		

```
module test_bool(A,C);
input [2:0]A;
output reg [2:0]C;

always @(A )
begin
if(A==1)
C=2'b11;
else
C=2'b00;
end
endmodule
```

		Value at		0 ps	20.0 ns	40.0 ns	60.0 ns	80.0 ns	100.0 ns			
Name		0 ps		0 ps								
IP	> A	B 000	0 ps									
	000		001	010	011	100	101	110	111	000	001	
OUT	> C	B 000	000	011			000				011	

TIPS ON PROJECT(1)

- Using button on the developing board, notice the shaking of button while it is pressed and released.
- Avoid assigning to a variable in several always block, or it would cause conflicts
- Notice on the sensitive list of always block :
 - Suggested: '*' is suggested in combinational logic
 - NOT suggested:
 - (posedge clk, negedge clk) // there is no corresponding component in FPGA
 - (posedge in1) is not suggested to find a posedge of an input signal

TIPS ON PROJECT(2-1)

Don't using posege or negedge of a normal signal except 'clock' or 'reset' in the sensitive list of the 'always' statement block .

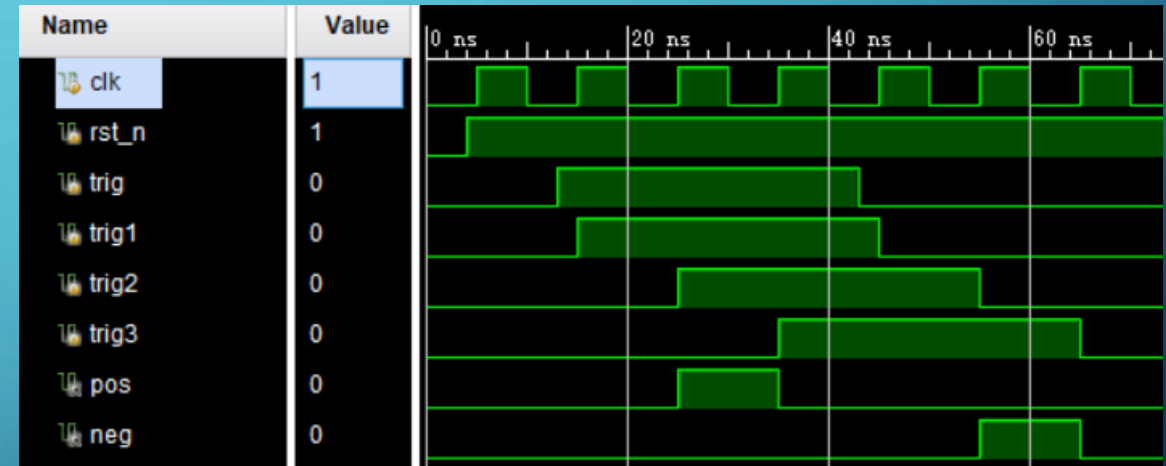
DON'Ts:

```
always @(posedge trig) begin
    x <= y;
end
```

Do Using following description instead

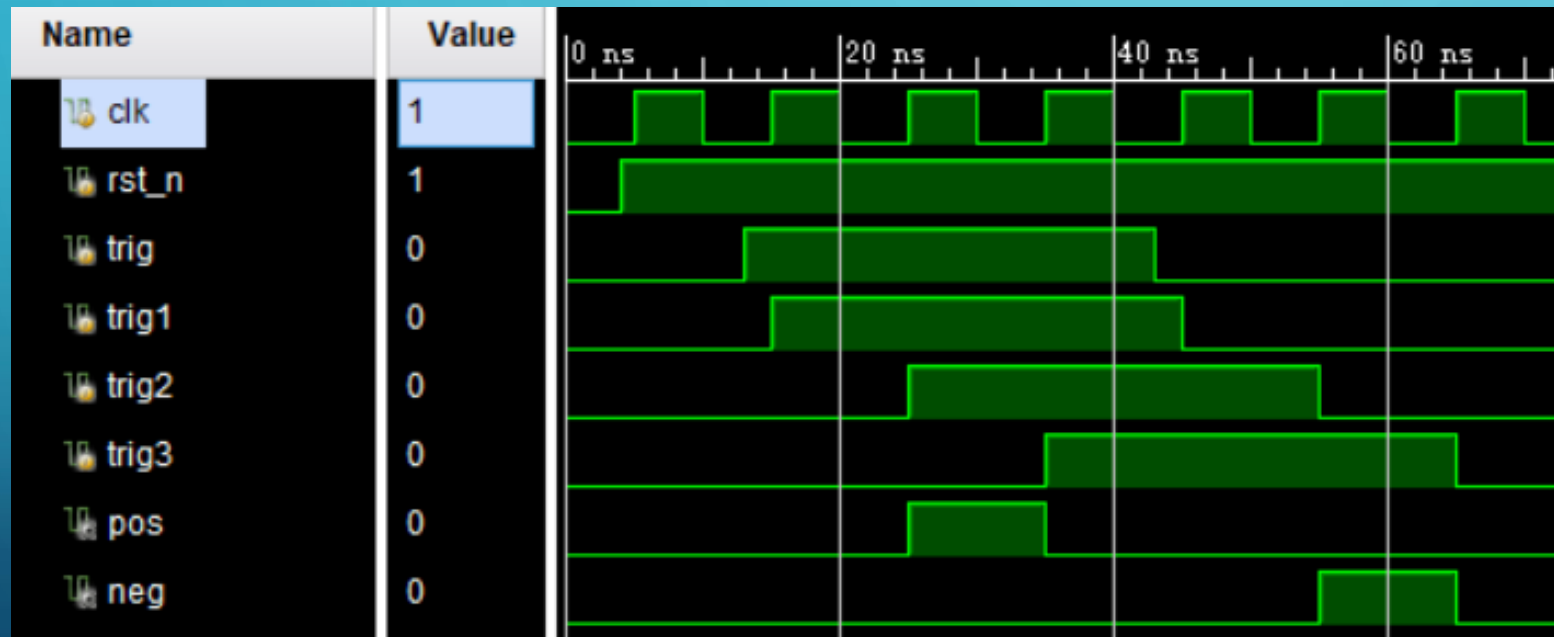
```
always @(pos) begin
    if(pos)
        x = y;
    else
        x = x;
end
```

//here pos is a new signal while it's 1'b1 means there is a posedge of trig



TIPS ON PROJECT(2)

To find the posedge or negedge of input signal 'trig'
Following method is suggested



Step1:

Sample the 'trig' at the rising edge(posedge) of clk to obtain 'trig1'

Step2:

Delay the 'trig1' by one clk cycle to obtain 'trig2'

Step3:

Delay the 'trig1' by one clk cycle to obtain 'trig3'

Step4:

a) After negating 'trig3', do logic and operation with 'trig2' to obtain 'pos'

b) After negating 'trig2', do logic and operation with 'trig3' to obtain 'neg'

TIPS ON VIVADO (ADD INTRAL SIGNAL TO WAVEFORM)

