Introduction to Verilog

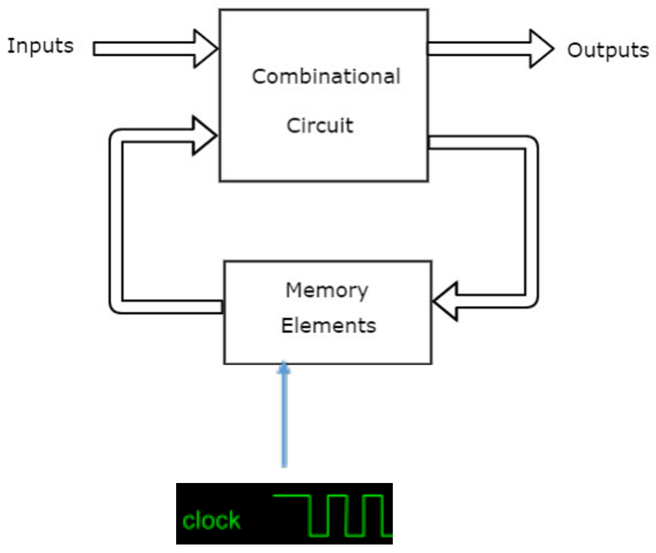
Mark

bhbruce.cs07g@nctu.edu.tw

Difference Between Hardware and Software

- Control synchronization
  - Combinational vs sequential
- Parallel data access
  - Parallel behavior

```
module Module_Name (
    input      xxx,
    output reg xxx
);

always @( posedge clk ) begin
    /* logic here */
end

endmodule
```



Data type

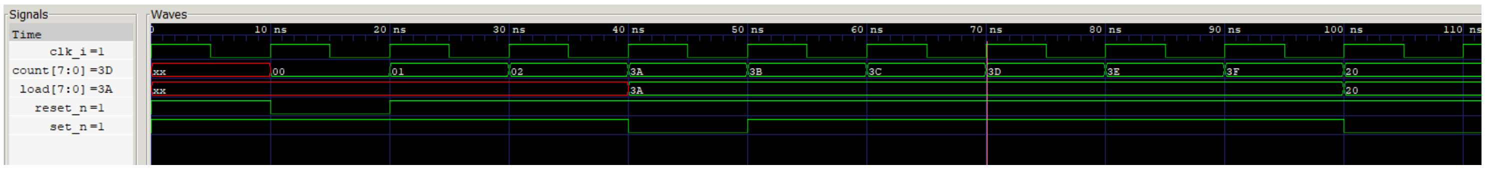
```
module example1 (
    input reset_n,
    input clk_i,
    input set_n,
    input        [7:0] load,
    output reg [7:0] count
);

always @( posedge clk_i ) begin
    if( !reset_n ) begin
        count <= 0;
    end
    else if ( !set_n ) begin
        count <= load;
    end
    else begin
        count <= count + 1;
    end
end

endmodule
```

Wire

1. "Status of the bus at the moment"
2. Cannot "store" over clock cycle
   1. Cannot be left value within clock triggered always block

Reg

3. Buffer along the circuit
4. In sequential circuit, this is often the pump of the circuit



Data type – Cont.

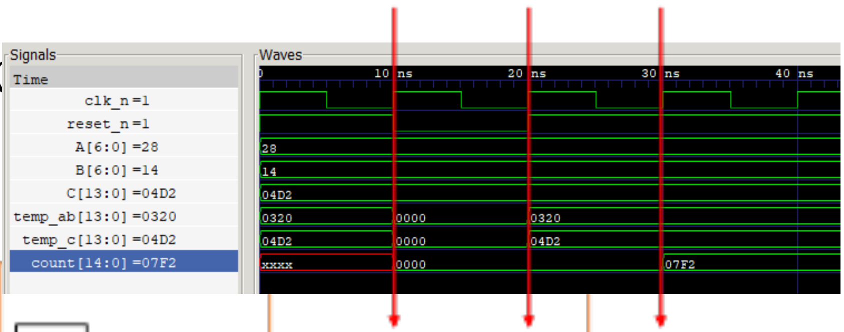
• parameter

```
module design_ip  ( addr,
                    wdata,
                    write,
                    sel,
                    rdata);

      parameter  BUS_WIDTH    = 32,
                 DATA_WIDTH   = 64,
                 FIFO_DEPTH   = 512;

      input addr;
      input wdata;
      input write;
      input sel;
      output rdata;

      wire  [BUS_WIDTH-1:0]  addr;
      wire  [DATA_WIDTH-1:0]  wdata;
      reg   [DATA_WIDTH-1:0]  rdata;

      reg [7:0] fifo [FIFO_DEPTH];

      // Design code goes here ...
endmodule
```

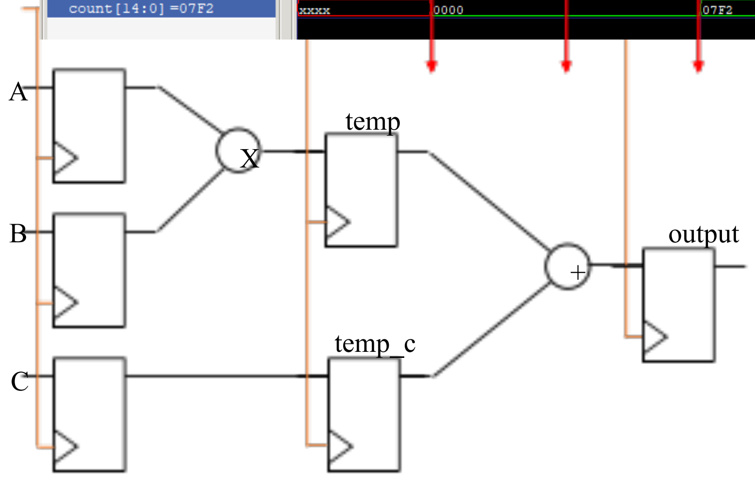
```
module tb;

   // Module instantiation override
   design_ip  #(BUS_WIDTH = 64, DATA_WIDTH = 128) d0 ( [port list]);

   // Use of defparam to override
   defparam d0.FIFO_DEPTH = 128;

endmodule
```

Blocking vs Non-Bl

```
module counter2 (
    input        reset_n,
    input        clk_n,
    input        [6:0]  A,
    input        [6:0]  B,
    input        [13:0] C,
    output reg   [14:0] count
);

reg [13:0] temp_ab;
reg [13:0] temp_c;

always @( posedge clk_n ) begin
    if( !reset_n ) begin
        count    <= 0;
        temp_ab  <= 0;
        temp_c   <= 0;
    end
    else begin
        temp_ab  <= A * B;
        temp_c   <= C;
        count    <= temp_ab + temp_c;
    end
end

endmodule
```

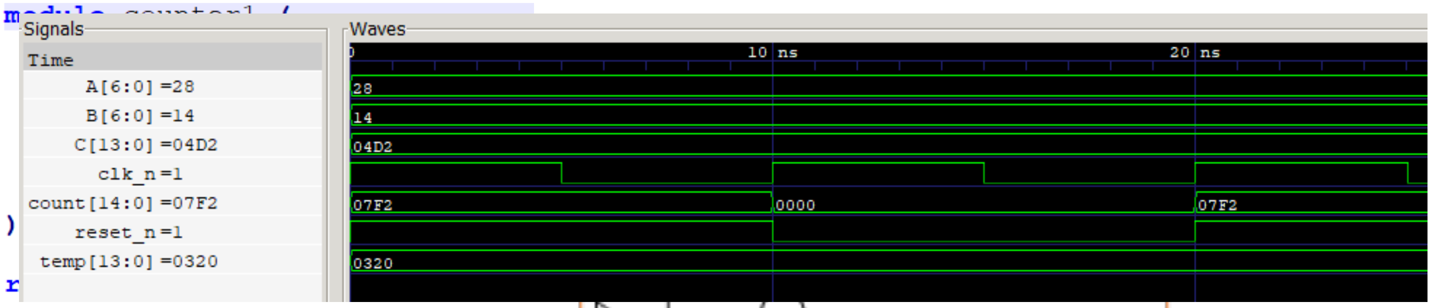


```
Signals                 Waves
Time                    0        10 ns         20 ns         30 ns         40 ns
clk_n=1
reset_n=1
A[6:0]=28               28
B[6:0]=14               14
C[13:0]=04D2            04D2
temp_ab[13:0]=0320      0320     0000          0320
temp_c[13:0]=04D2       04D2     0000          04D2
count[14:0]=07F2        xxxx     0000                        07F2
```



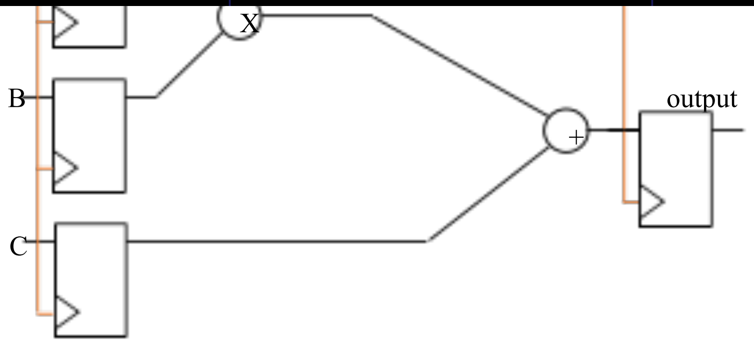
Blocking vs Non-Blocking

```
module counter1 (
)
r
```



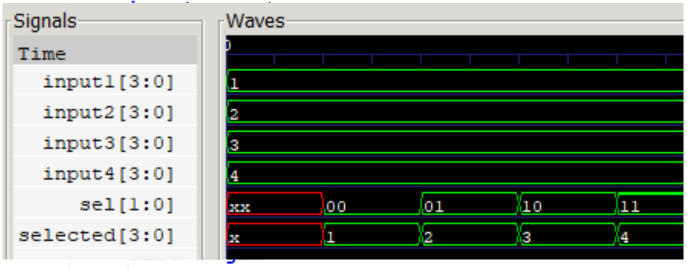
```
Signals            Waves
Time               0               10 ns               20 ns
A[6:0]=28          28
B[6:0]=14          14
C[13:0]=04D2       04D2
clk_n=1
count[14:0]=07F2   07F2            0000                07F2
reset_n=1
temp[13:0]=0320    0320
```

```
always @( posedge clk_n ) begin
    if( !reset_n ) begin
        count = 0;
    end
    else begin
        temp  = A * B;
        count = temp + C;
    end
end

endmodule
```



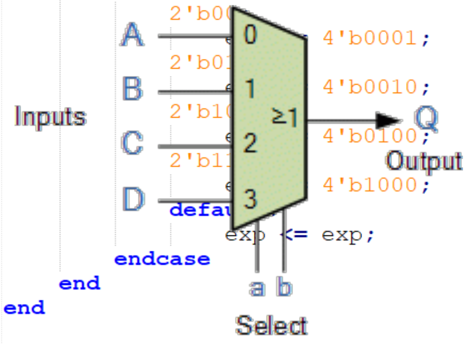
Decoder and Multiplexer

```
module exponential (
```

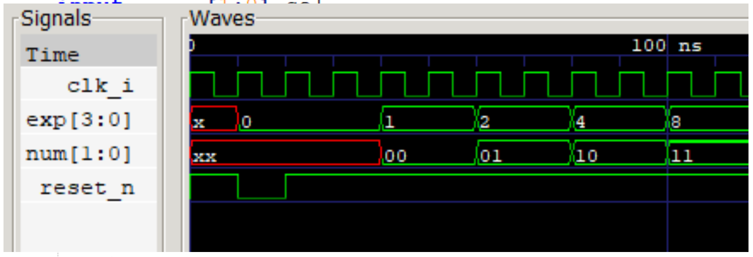


```
Signals       Waves
Time          0
input1[3:0]   1
input2[3:0]   2
input3[3:0]   3
input4[3:0]   4
sel[1:0]      xx  00  01  10  11
selected[3:0] x   1   2   3   4
```

```
        case( num )
          2'b00
                        4'b0001;
          2'b01
                        4'b0010;
          2'b10
                        4'b0100;
          2'b11
                        4'b1000;
          defau
                  exp <= exp;
        endcase
    end
end

endmodule
```



```
module multiplexer (
```



```
Signals     Waves
Time        0                                100 ns
clk_i
exp[3:0]    x  0        1     2     4     8
num[1:0]    xx          00    01    10    11
reset_n
```

```
                selected <= input1;
            2'b01:
                selected <= input2;
            2'b10:
                selected <= input3;
            2'b11:
                selected <= input4;
            default:
                selected <= selected;
        endcase
end

endmodule
```

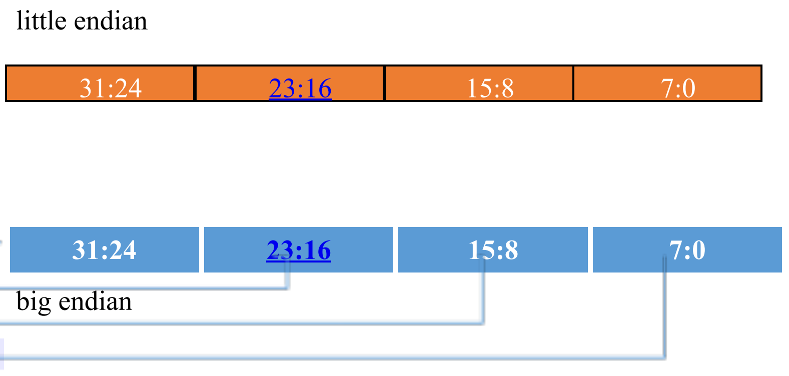
Avoiding latch in Verilog

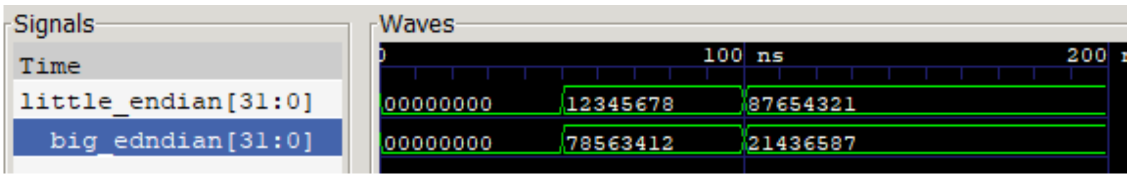
```
always @( a, b, c, ~~x, y, z~~ ) begin
  x = a;
  y = b;
  z = c;
End
always @( a, b ) begin
  if( a ) begin …..; end
  else if ( b ) begin ……; end
  else begin ….; end
end
```

- All inputs used must appear in trigger list
- Unless all conditions are listed, use default (case) or else (if else statement) to prevent unpredicted behavior

Partial bit operation

```
module concatenation(
    input  [31:0] little_endian,
    output  [31:0] big_edndian
    );

assign big_edndian =
    {little_endian[7:0],
     little_endian[15:8],
     little_endian[23:16],
     little_endian[31:24]};

endmodule
```



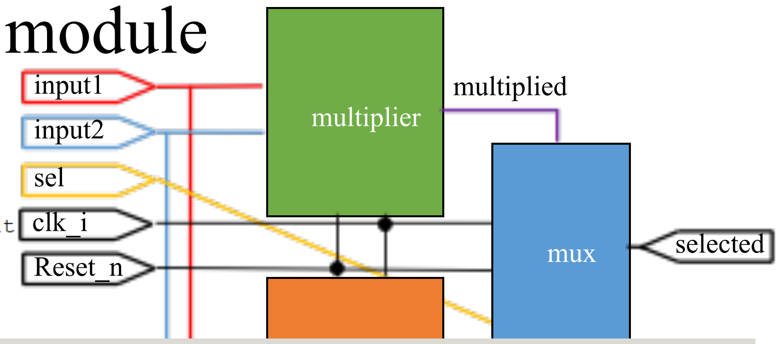


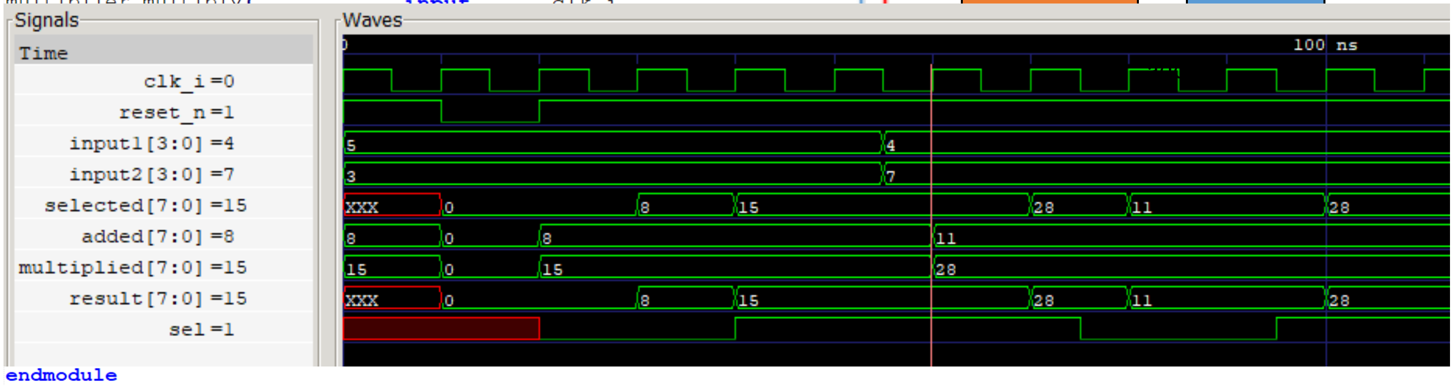
```
Time                    0        100 ns              200
little_endian[31:0]     00000000 12345678  87654321
big_edndian[31:0]       00000000 78563412  21436587
```

```
module add_or_multiply(
    input         reset_n,
    input         clk_i,
    input         sel,
    input   [3:0] input1,
    input   [3:0] input2,
    output  [7:0] selected
);

wire [7:0] added;
wire [7:0] multiplied;

adder add(
    .reset_n(reset_n),
    .clk_i(clk_i),
    .input1(input1),
    .input2(input2),
    .added(added)
);

multiplier multiply(
```

Nested module

```
module mux (
    input          reset_n,
    input          clk_i,
    input          sel,
    input  [7:0]   input1,
    input  [7:0]   input2,
    output reg [7:0] result
);

module adder(
    input          reset_n,
    input          clk_i,
```





```
Time                   0                                   100 ns
clk_i=0
reset_n=1
input1[3:0]=4          5                    4
input2[3:0]=7          3                    7
selected[7:0]=15       XXX  0      8   15          28   11      28
added[7:0]=8           8    0   8             11
multiplied[7:0]=15     15   0   15            28
result[7:0]=15         XXX  0      8   15          28   11      28
sel=1
```

```
endmodule
```