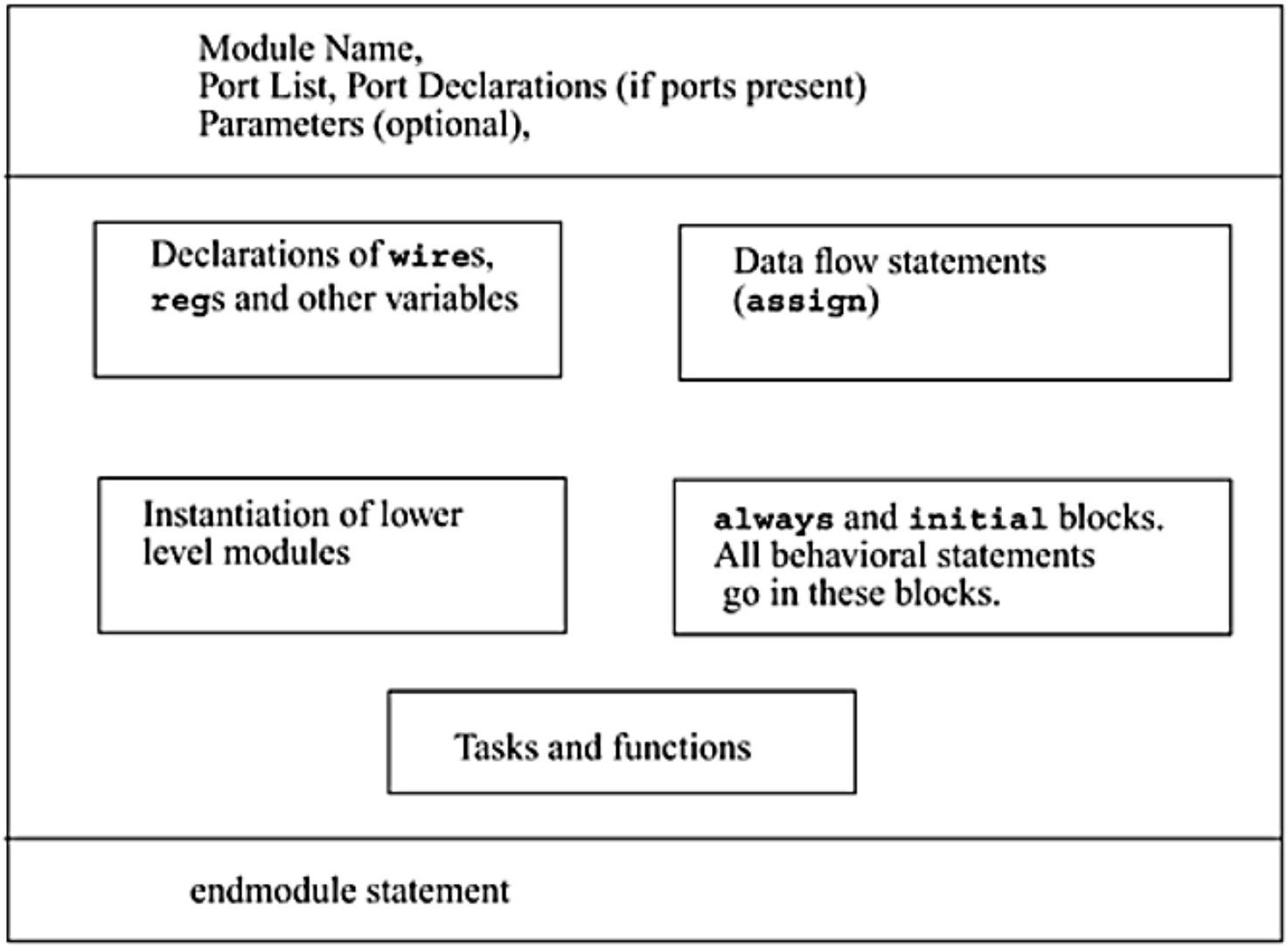
DATA FLOW MODELLING

Components of a Verilog Module



Introduction

- For small circuits, the gate-level modeling approach works very well because the number of gates is limited and the designer can instantiate and connect every gate individually.
- Also, gate-level modeling is very intuitive to a designer with a basic knowledge of digital logic design.
- However, in complex designs the number of gates is very large. Thus, designers can design more effectively if they concentrate on implementing the function at a level of abstraction higher than gate level.
- Dataflow modeling provides a powerful way to implement a design.
- Verilog allows a circuit to be designed in terms of the data flow between registers and how a design processes data rather than instantiation of individual gates.

- With gate densities on chips increasing rapidly, dataflow modeling has assumed great importance. No longer can companies devote engineering resources to handcrafting entire designs with gates.
- Currently, automated tools are used to create a gate-level circuit from a dataflow design description. This process is called **logic synthesis**.
- Dataflow modeling has become a popular design approach as logic synthesis tools have become sophisticated. This approach allows the designer to concentrate on optimizing the circuit in terms of data flow.
- For maximum flexibility in the design process, designers typically use a Verilog description style that combines the concepts of gate-level, data flow, and behavioral design.
- In the digital design community, the term RTL (Register Transfer Level) design is commonly used for a combination of dataflow modeling and behavioral modeling.

1. Continuous Assignments

- Continuous assignment is the most basic statement in dataflow modeling, used to drive a value onto a net.
- This assignment replaces gates in the description of the circuit and describes the circuit at a higher level of abstraction.
- A continuous assignment assigns a value to a net (it cannot be used to assign a value to a register).
- The assignment statement starts with the keyword assign. The syntax of an assign statement is as follows:

```
assign LHS_target = RHS_expression;
```

For example,

```
wire [3:0] Z, Preset, Clear;     // Net declaration.

assign Z = Preset & Clear;       // Continuous assignment
```

- The target of the continuous assignment is Z and the right-hand side expression is \"Preset & Clear\". Note the presence of the keyword assign in the continuous assignment.
- **When does a continuous assignment execute?** Whenever an event (an event is a change of value) occurs on an operand used in the right-hand side expression, the expression is evaluated and if the result value is different, it is then assigned to the left-hand side target.
- In the above example, if either Preset or Clear change, the entire right-hand side expression is evaluated. If this results in a change of value, then the resultant value is assigned to the net Z.

The target in a continuous assignment can be one of the following.

i. Scalar net

ii. Vector net

iii. Constant bit-select of a vector

iv. Constant part-select of a vector

v. Concatenation of any of the above

Here are more examples of continuous assignments.

```
assign BusErr = Parity | (One & OP);

assign Z = ~ (A | B) & (C | D) & (E | F);
```

The last continuous assignment executes whenever there is a change of value in A, B, C, D, E or F, in which case, the entire right-hand side expression is evaluated and the result is then assigned to the target Z

Example: **The target is a concatenation of a scalar net and a vector net.**

```
wire Cout, Cin;
wire [3:0] Sum, A, B;
. . .
assign {Cout, Sum} = A + B + Cin;
```

- Since A and B are 4-bits wide, the result of addition can produce a maximum of a 5-bit result. The left-hand side is specified to be five bits (one bit for Cout and 4 bits of Sum).
- The assignment therefore causes the rightmost four bits of the right-hand side expression result to be assigned to Sum and the fifth bit (the carry bit) to Cout

Example: Multiple assignments in one continuous statement

```
assign Mux = (S == 0) ? A : 'bz,
       Mux = (S == 1) ? B : 'bz,
       Mux = (S == 2) ? C : 'bz,
       Mux = (S == 3) ? D : 'bz;
```

This is a short form of writing the following four separate continuous assignments.

```
assign Mux = (S == 0) ? A : 'bz;
assign Mux = (S == 1) ? B : 'bz;
assign Mux = (S == 2) ? C : 'bz;
assign Mux = (S == 3) ? D : 'bz;
```

1-bit full-adder using the dataflow style

Sum = A XOR B XOR Cin

Cout = AB + B Cin + A Cin

```
module FA_Df (A, B, Cin, Sum, Cout);
  input A, B, Cin;
  output Sum, Cout;

  assign Sum = A ^ B ^ Cin;
  assign Cout = (A & Cin) | (B & Cin) | (A & B);
endmodule
```

4:1 MUX in data flow

Selection Lines		Output
S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

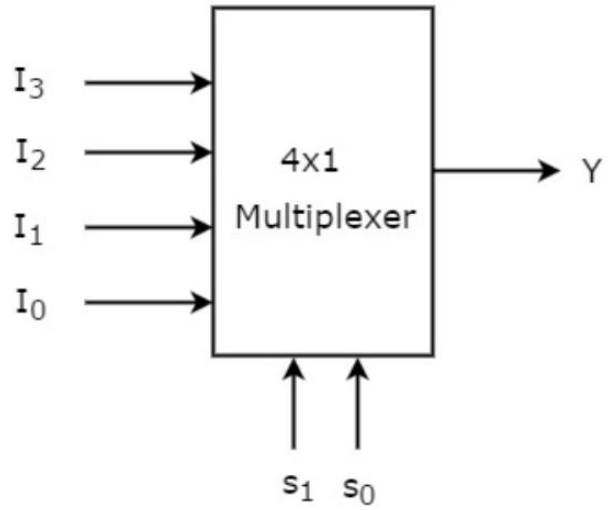
$$Y = S_1'S_0'I_0 + S_1'S_0I_1 + S_1S_0'I_2 + S_1S_0I_3$$

```
module mux4_to_1 (out, i0, i1, i2, i3, s1, s0);

// Port declarations from the I/O diagram
output out;
input i0, i1, i2, i3;
input s1, s0;

//Logic equation for out
assign out =     (~s1 & ~s0 & i0)|
                 (~s1 & s0 & i1) |
                 (s1 & ~s0 & i2) |
                 (s1 & s0 & i3) ;

endmodule
```



Net Declaration Assignment

- A continuous assignment can appear as part of a net declaration itself. Such an assignment is called a net declaration assignment.
- Here is an example.

```
wire [3:0] Sum = 4'b0;
wire Clear = 'b1;

wire A_GT_B = A > B, B_GT_A = B > A;
```

- A net declaration assignment declares the net along with a continuous assignment. It is a convenient form of declaring a net and then writing a continuous assignment

```
wire Clear;
assign Clear = 'b1;
```

is equivalent to the net declaration assignment:

```
wire Clear = 'b1;
```

- Multiple net declaration assignments on the same net are not allowed.
- If multiple assignments are necessary, continuous assignments must be used.

Delays

- If no delay is specified in a continuous assignment, as in the previous examples, the assignment of the right-hand side expression to the left-hand side target occurs with zero delay.
- A delay can be explicitly specified in a continuous assignment as shown in the following example.

```
assign #6 Ask = Quiet || Late;
```

- The delay specified, #6,is the delay between the right-hand side and the left-hand side.
- For example, if a change of value occurs on Late at time 5, then the expression on the right-hand side of the assignment is evaluated at time 5 and Ask will be assigned a new value at time 11 (= 5 + 6).

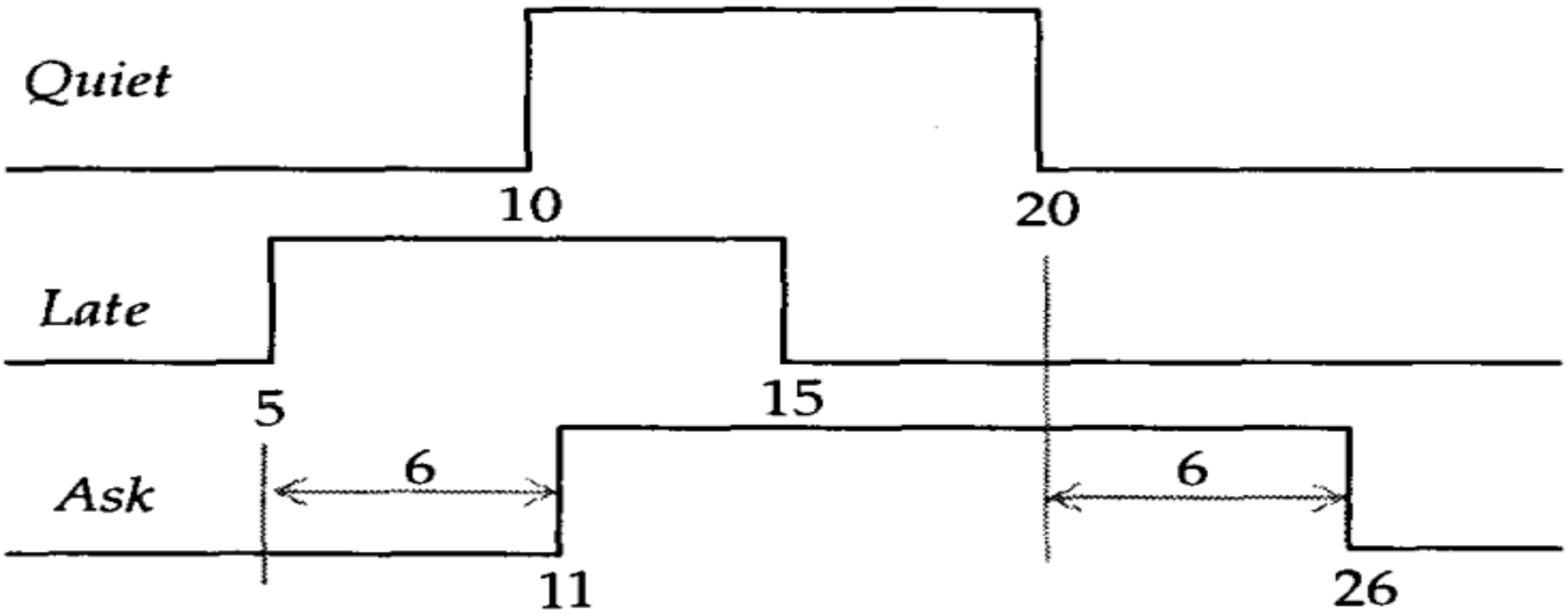


Fig 1: Delay in a continuous assignment.

What happens if the right-hand side changes before it had a chance to propagate to the left-hand side?

- In such a case, the latest value change is applied. Here is an example that shows this behavior

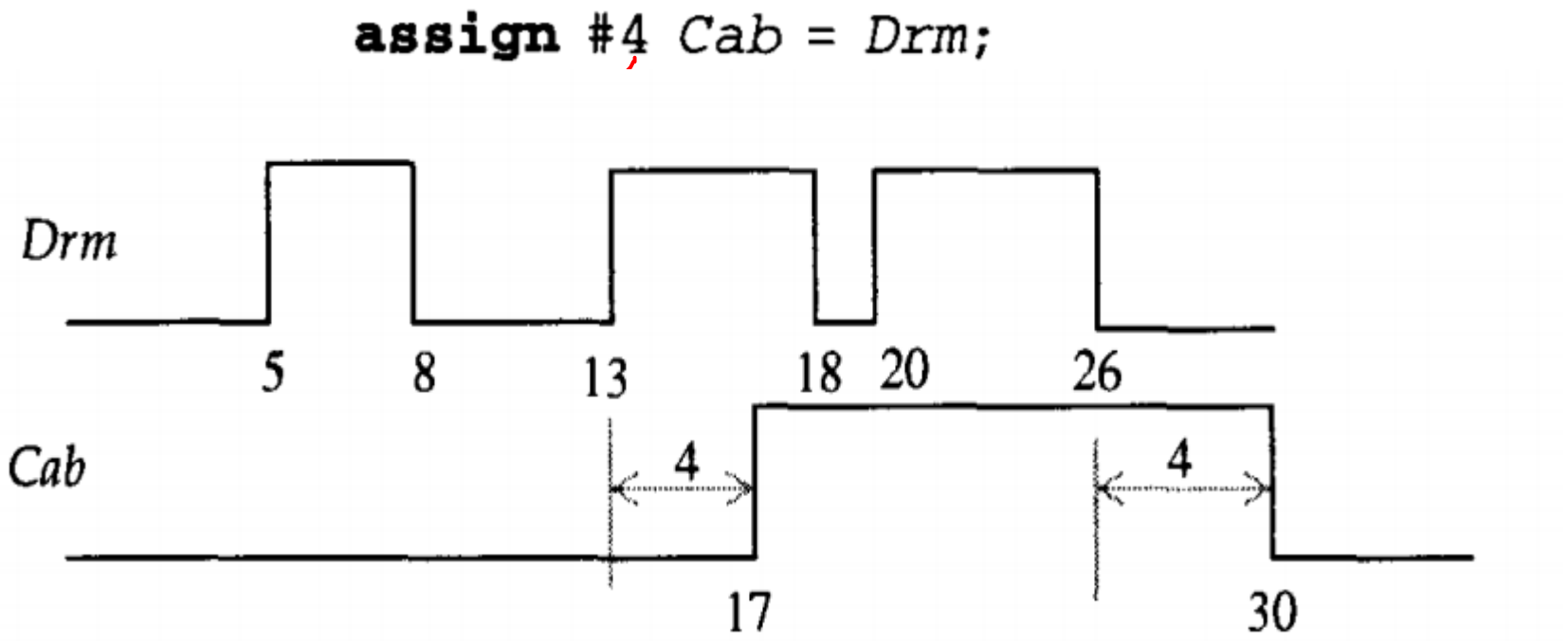


Fig 2: Values changing faster than delay interval.

- Figure 2 shows the effect. The changes on the right-hand side that occur within the delay interval are filtered out.
- For example, the rising edge on Drm at 5 gets scheduled to appear on Cab at 9, however since Drm goes back to 0 at 8, the scheduled value on Cab is deleted.
- Similarly, the pulse on Drm occurring between 18 and 20 gets filtered out. This corresponds to **the inertial delay**
- **In inertial delay** behavior, a value change on the right-hand side must hold steady for at least the delay period before it can propagate to the left-hand side; if the value on the right-hand side changes within the delay period, the former value does not propagate to the output.

- For each delay specification, up to three delay values can be specified.

i. Rise delay

ii. Fall delay

iii. Turn-off delay

Here is the syntax for specifying the three delays.

```
assign #( rise , fall , turn-off ) LHS_target = RHS_expression ;
```

EXAMPLE:

```
assign #4 Ask = Quiet || Late;          // One delay value.

assign #(4, 8) Ask = Quick;             // Two delay values.

assign #(4, 8, 6) Arb = & DataBus;      // Three delay values.

assign Bus = MemAddr[7:4];              // No delay value.
```

- In the first assignment statement, the rise delay, the fall delay and the turn-off delay and the transition to x delay are the same, which is 4.
- In the second statement, the rise delay is 4, the fall delay is 8 and the transition to x and z delay are the same, which is the minimum of 4 and 8, which is 4.
- In the third assignment, the rise delay is 4, the fall delay is 8 and the turn-off delay is 6; the transition to x delay is 4 (the minimum of 4, 8 and 6).
- In the last statement, all delays are zero.

- **What does a rise delay mean for a vector net target?**
  - If the right-hand side goes from a non-zero value to a zero vector, then fall delay is used.
  - If right-hand side value goes to z, then turn-off delay is used;
  - Else rise delay is used.

Net Delay

- A delay can also be specified in a net declaration, such as in the following declaration.

```
wire #5 Arb;
```

- This delay indicates the delay between a change of value of a driver for Arb and the net Arb itself. Consider the following continuous assignment to the net Arb.

```
assign #2 Arb = Bod & Cap;
```

- An event on Bod, say at time 10, causes the right-hand side expression to be evaluated. If the result is different, it is assigned to Arb after 2 time units, that is, at time 12.

- However since Arb has a net delay specified, the actual assignment to the net Arb occurs at time 17(=10+ 2 + 5). The waveforms in Figure 3 illustrates the different delays.

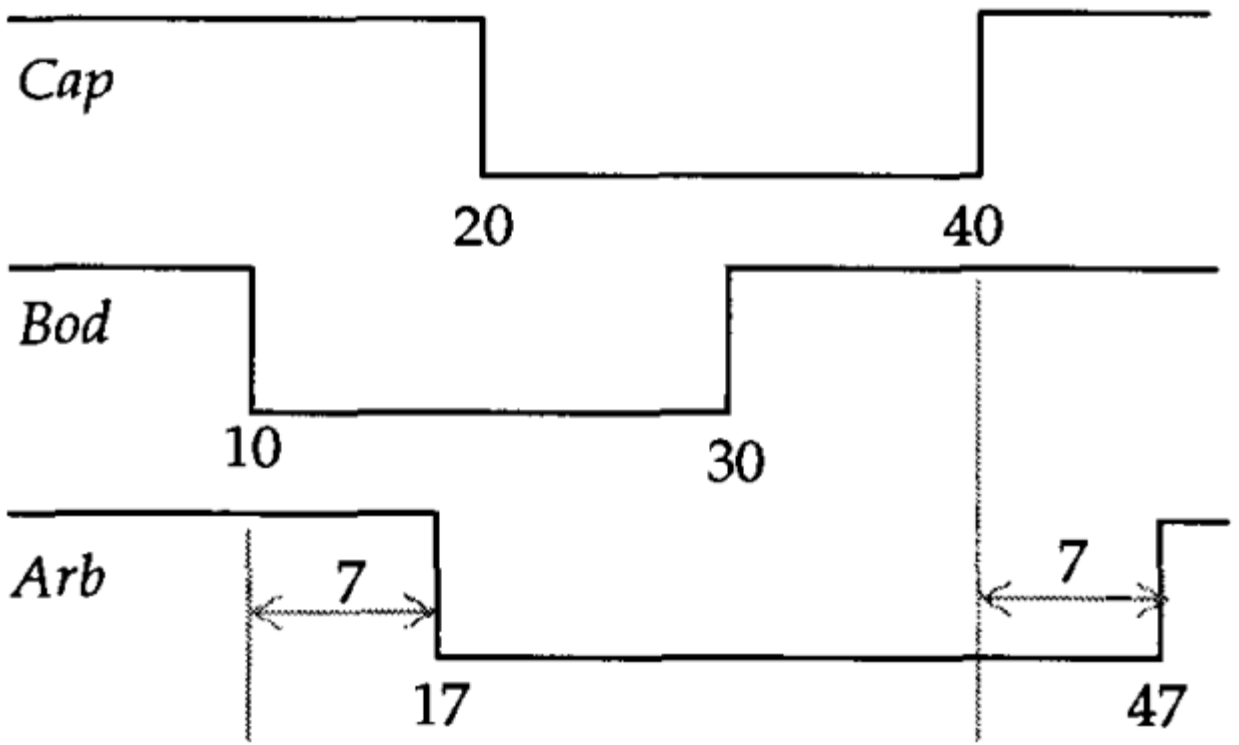


Fig: 3 Net delay with assignment delay.

- If a delay is present in a net declaration assignment, then the delay is not a net delay but an assignment delay. In the following net declaration assignment for A, 2 time units is the assignment delay, not the net delay.

```
wire #2 A = B - C;                    // Assignment delay.
```

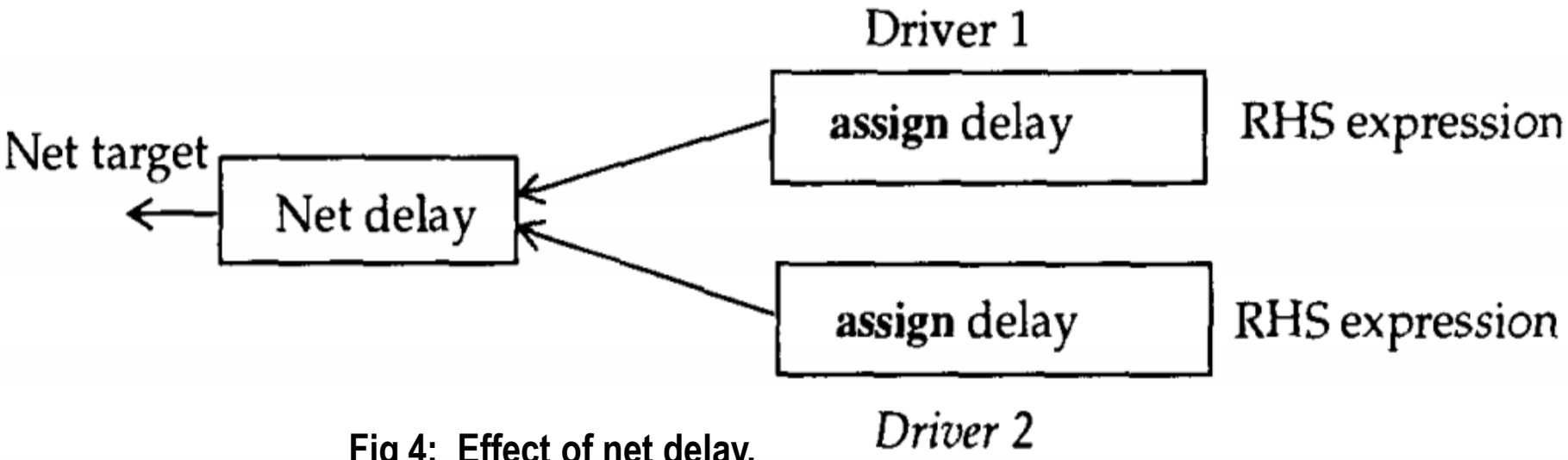
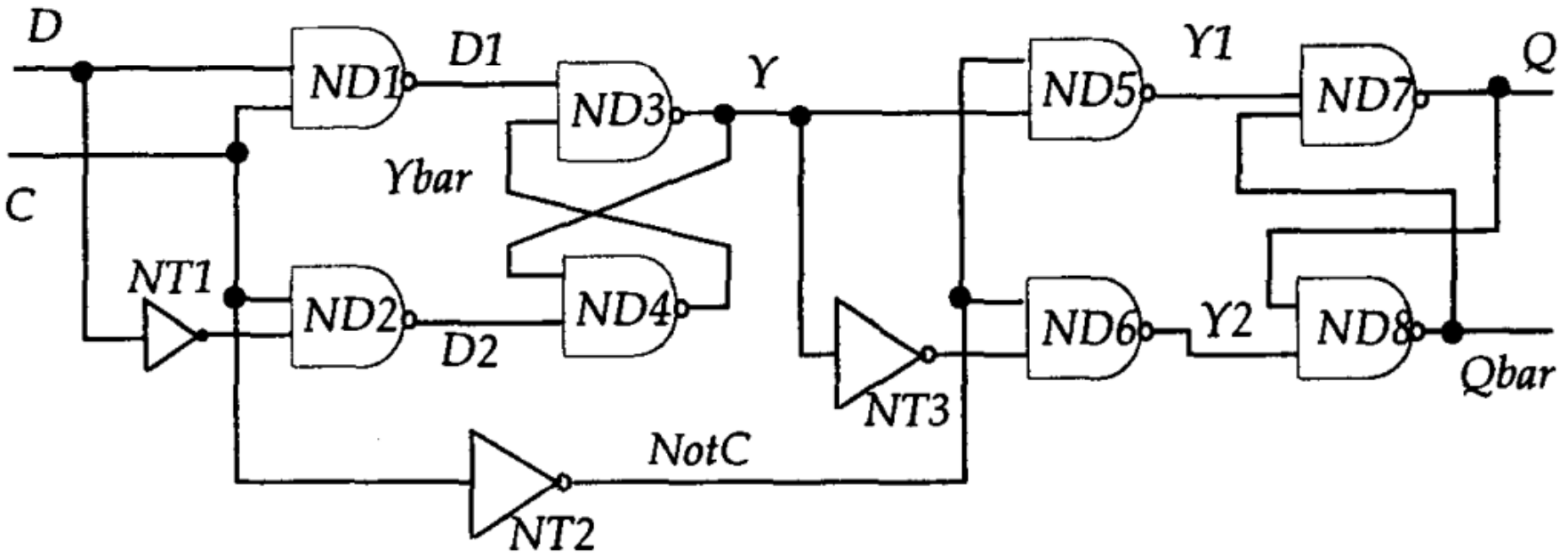


Fig 4: Effect of net delay.

Master-slave Flip-flop



```
module MSDFF_DF (D, C, Q, Qbar);
  input D, C;
  output Q, Qbar;
  wire NotC, NotD, NotY, Y, D1, D2, Ybar, Y1, Y2;

  assign NotD = ~ D;
  assign NotC = ~ C;
  assign NotY = ~ Y;

  assign D1 = ~ (D & C);
  assign D2 = ~ (C & NotD);
  assign Y = ~ (D1 & Ybar);
  assign Ybar = ~ (Y & D2);
  assign Y1 = ~ (Y & NotC);
  assign Y2 = ~ (NotY & NotC);
  assign Q = ~ (Qbar & Y1);
  assign Qbar = ~ (Y2 & Q);
endmodule
```

8 bit Magnitude Comparator

```
module MagnitudeComparator (A, B, AgtB, AeqB, AltB);
  parameter BUS = 8;
  parameter EQ_DELAY = 5, LT_DELAY = 8, GT_DELAY = 8;
  input [1 : BUS] A, B;
  output AgtB, AeqB, AltB;

  assign #EQ_DELAY   AeqB = A == B;
  assign #GT_DELAY   AgtB = A > B;
  assign #LT_DELAY   AltB = A < B;
endmodule
```

4-to-1 Multiplexer, Using Conditional Operators

```
module multiplexer4_to_1 (out, i0, i1, i2, i3, s1, s0);

// Port declarations from the I/O diagram
output out;
input i0, i1, i2, i3;
input s1, s0;

// Use nested conditional operator
assign out = s1 ? ( s0 ? i3 : i2) : (s0 ? i1 : i0) ;

endmodule
```

Selection Lines		Output
S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

THANK YOU