# Digital Design with Verilog

Verilog - Synthesis

Lecture 20: Logic Synthesis with Verilog HDL – Part3

## Objectives

- Conditional Expressions
- Always constructs

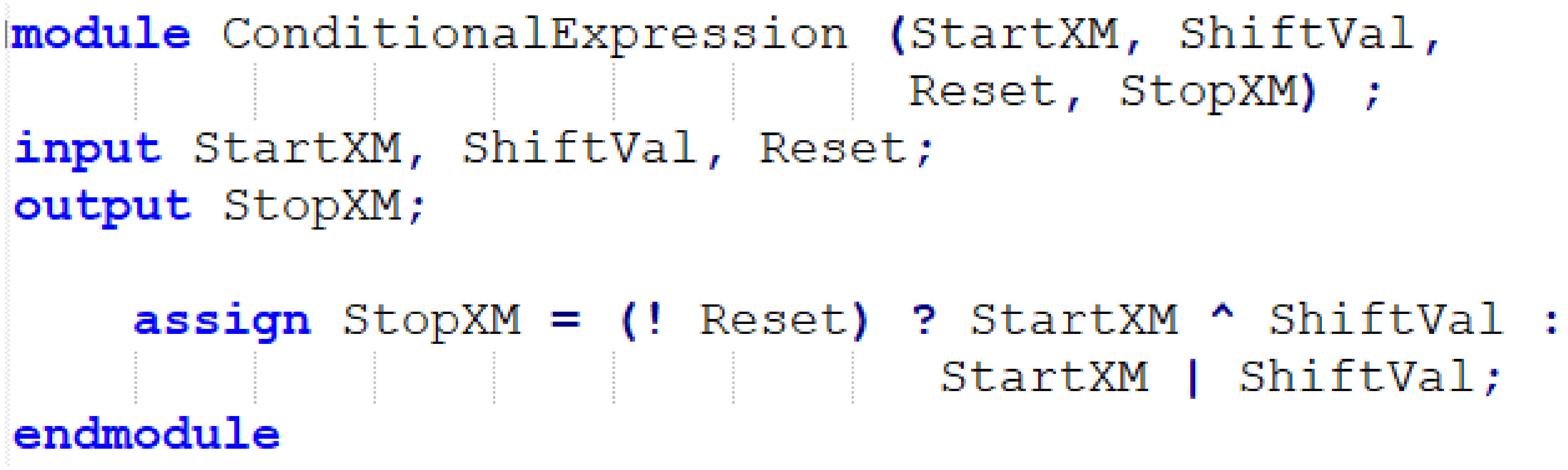
## Conditional Expression

- A conditional expression selects between two expressions according to the value of a condition.

*<condition>* ? *<expression1>* : *<expression2>*

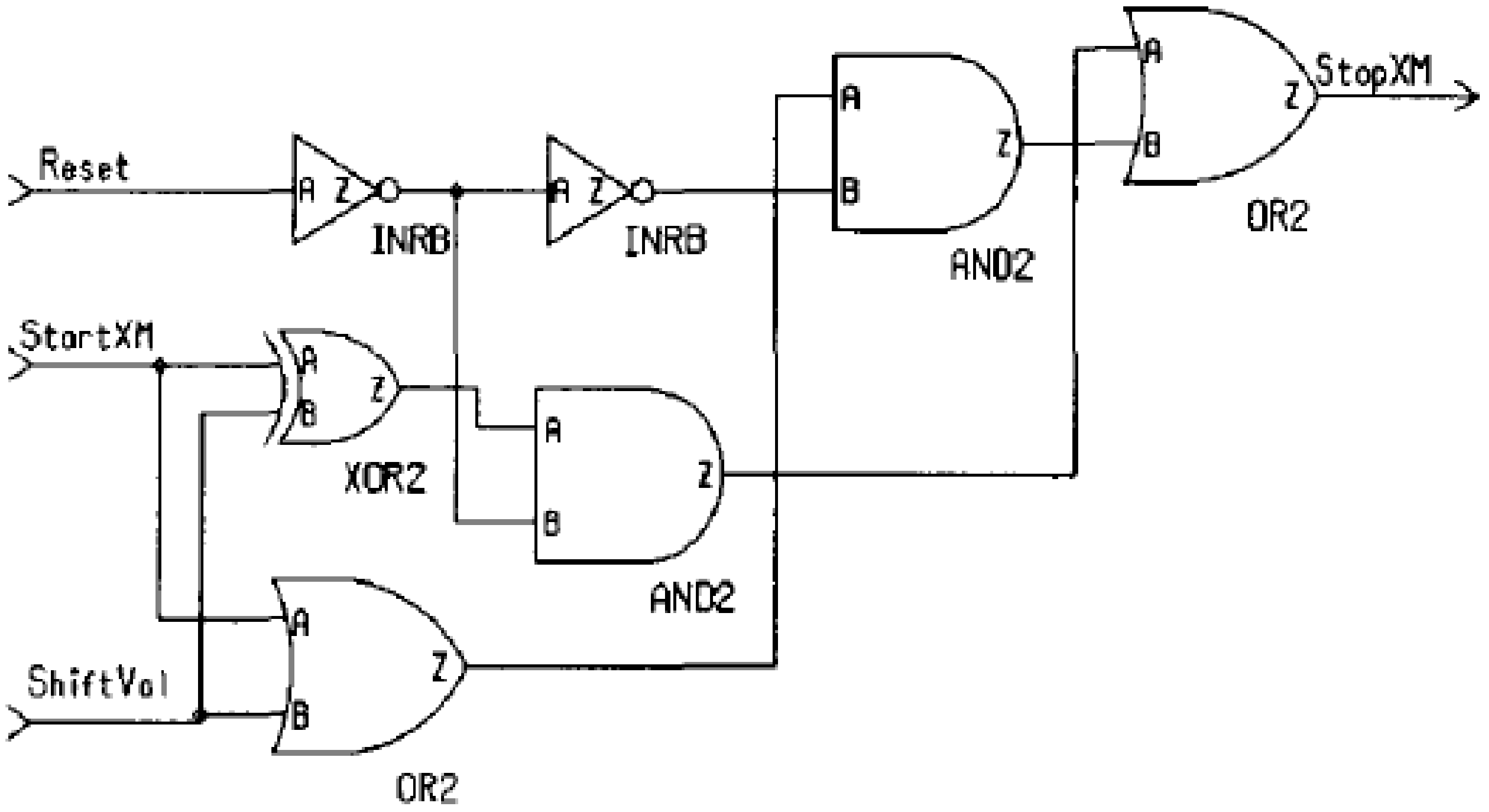
- If the condition is true, select the first expression, else select the second.

## Conditional Expression



```
module ConditionalExpression (StartXM, ShiftVal,
                              Reset, StopXM) ;
input StartXM, ShiftVal, Reset;
output StopXM;

    assign StopXM = (! Reset) ? StartXM ^ ShiftVal :
                               StartXM | ShiftVal;
endmodule
```

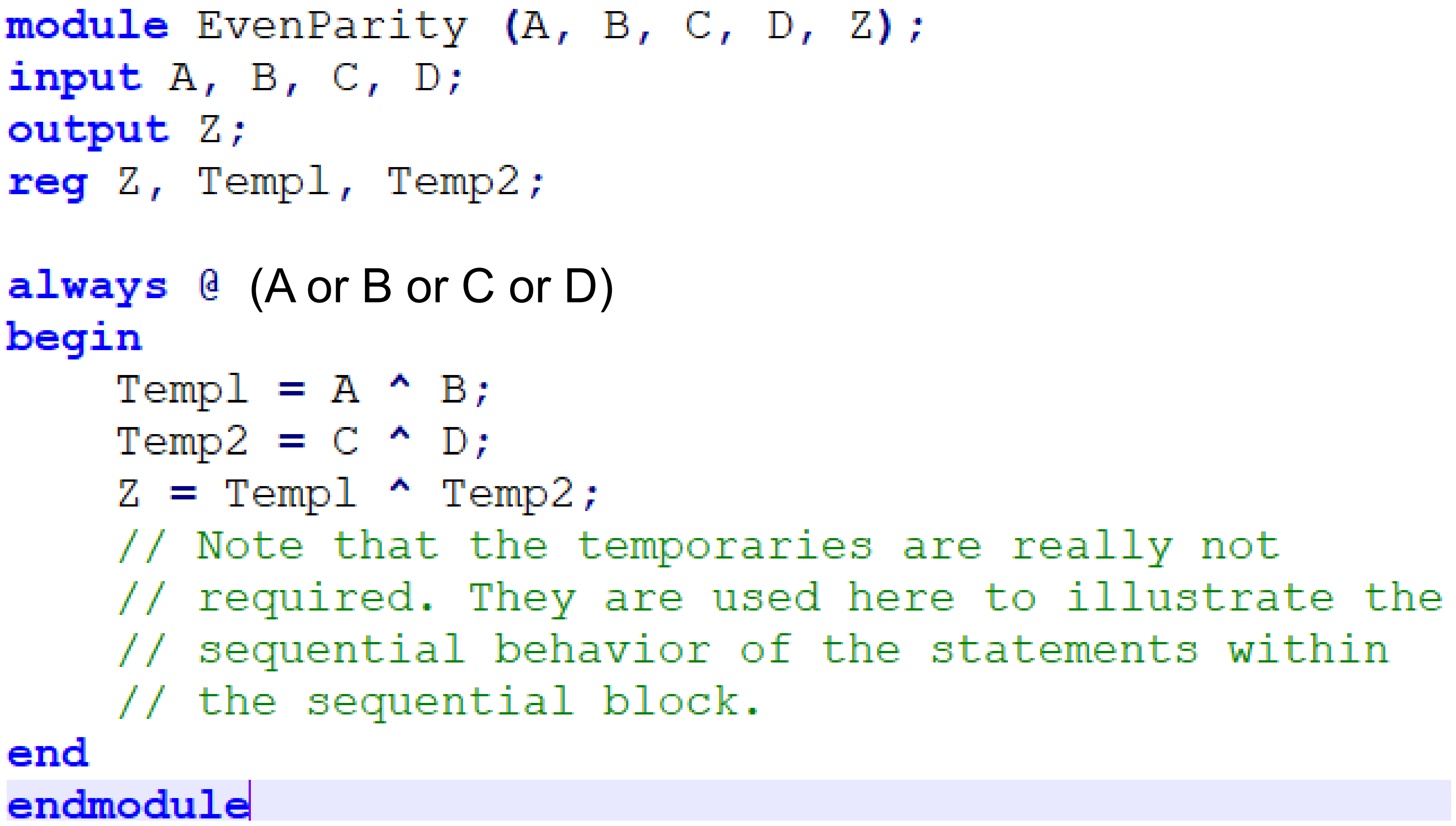
## Conditional Expression



Logic generated from a conditional expression

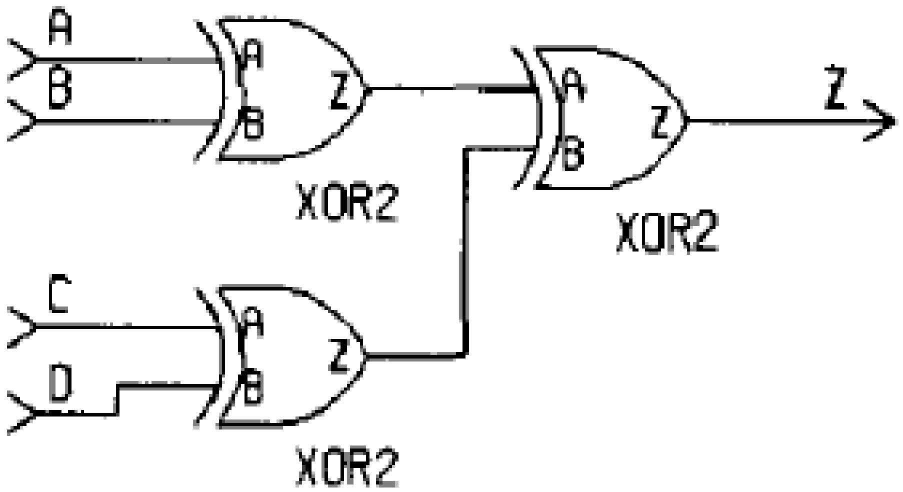
## Always Statement

- An always statement is used to model the procedural behavior of a circuit.



```
module EvenParity (A, B, C, D, Z);
input A, B, C, D;
output Z;
reg Z, Temp1, Temp2;

always @ (A or B or C or D)
begin
    Temp1 = A ^ B;
    Temp2 = C ^ D;
    Z = Temp1 ^ Temp2;
    // Note that the temporaries are really not
    // required. They are used here to illustrate the
    // sequential behavior of the statements within
    // the sequential block.
end
endmodule
```

## Always Statement

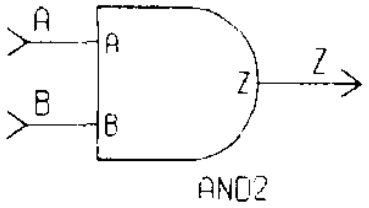


Procedural assignment statements.

## Always Statement

- All variables whose values are read in the always statement must appear in the event list.
- Otherwise, the functionality of the synthesized netlist may not match that of the design model.

```
module AndBehavior (Z, A, B);
input A, B;
output Z;
reg Z;
    always@(B)
        Z =A & B;
endmodule
```



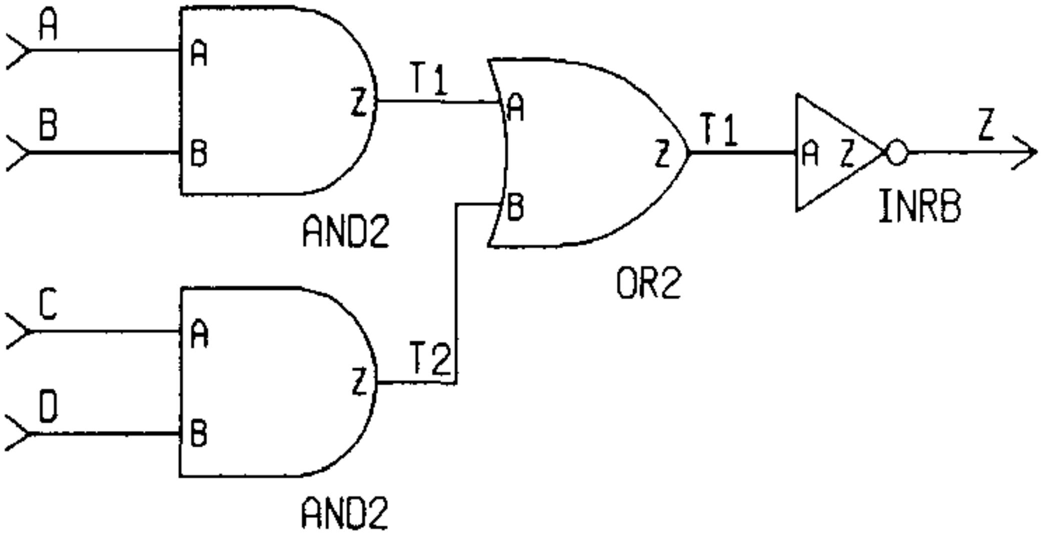
Incomplete event list.

## Always Statement

- A synthesis system usually would issue a warning about such missing variables in the event list.
- A good practice is to include all variables read in the always statement in the event list; this is true only when modeling combinational logic.
- When modeling sequential logic, a different kind of event list is required; this is described later.
- A variable declared within an always statement holds a temporary value and does not necessarily infer a unique wire in hardware as the following example shows.

## Always Statement

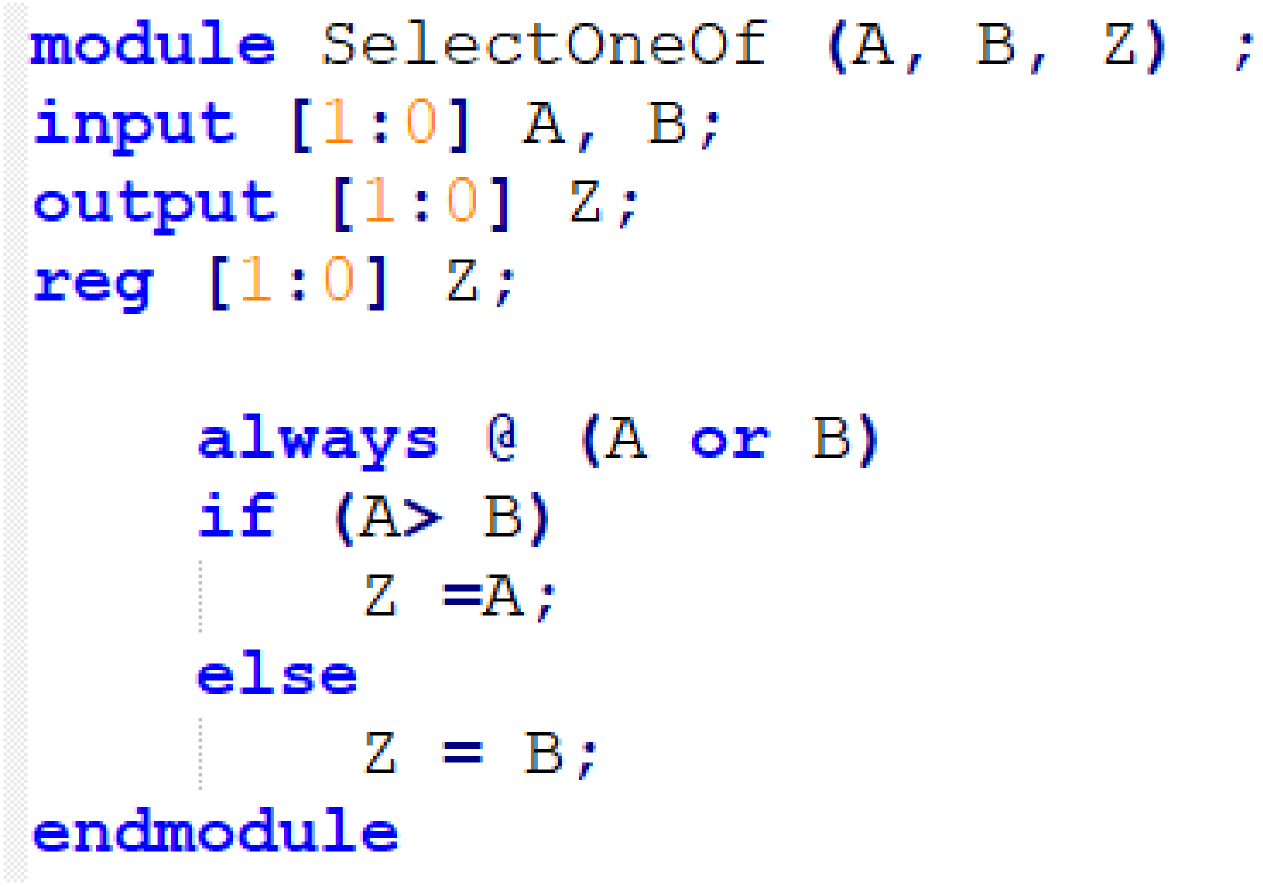
```
module VariablesAreTemporaries (A, B, C, D, Z);
input A, B, C, D;
output Z;
reg Z;

always @ (A or B or C or D)
begin: VAR_LABEL
    integer Tl, T2;
    Tl =A & B;
    T2 = C & D;
    Tl = Tl | T2;
    Z = ~Tl;
end
endmodule
```



One variable can represent many wires.

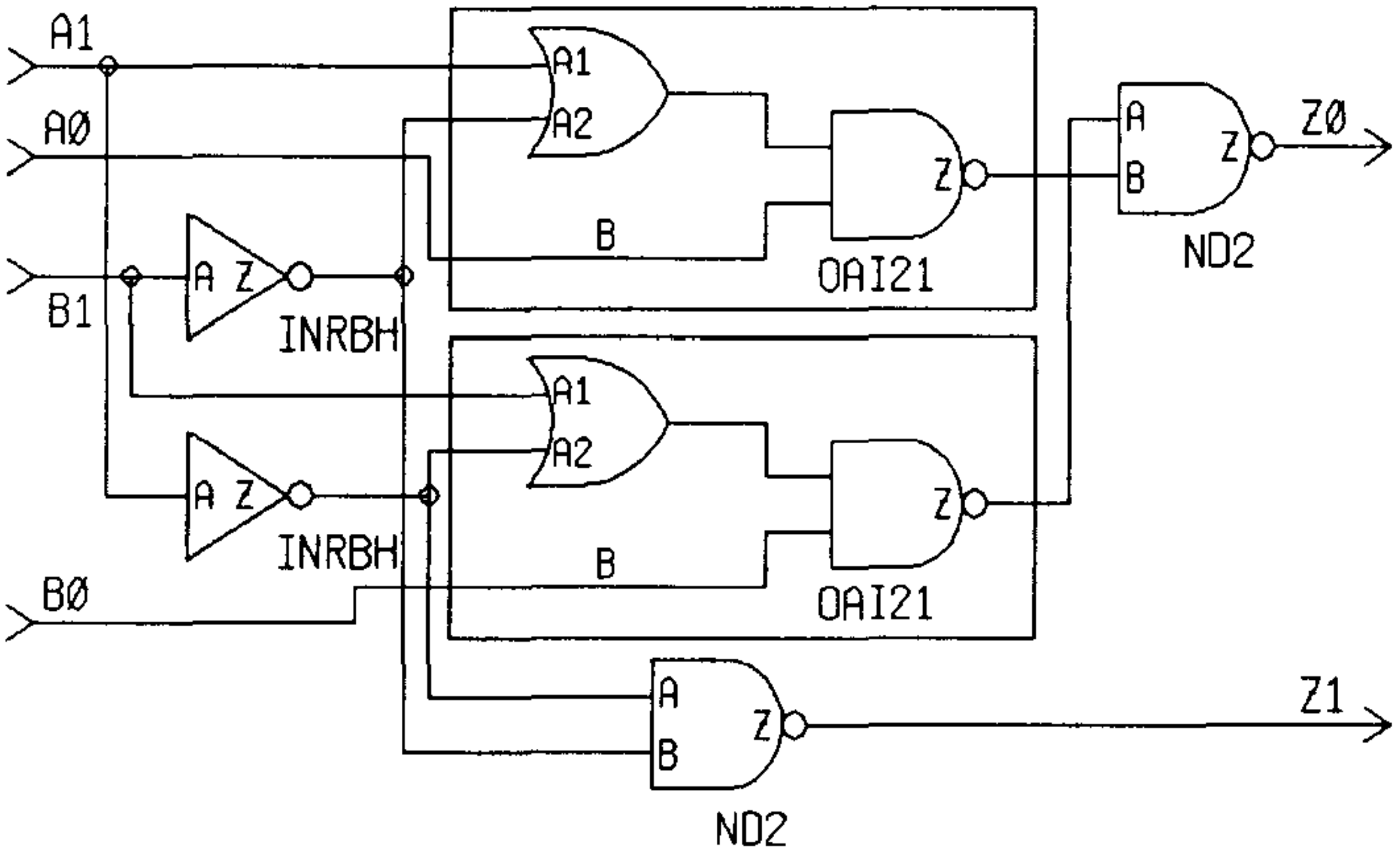
## If Statement

- An if statement represents logic that is conditionally controlled.



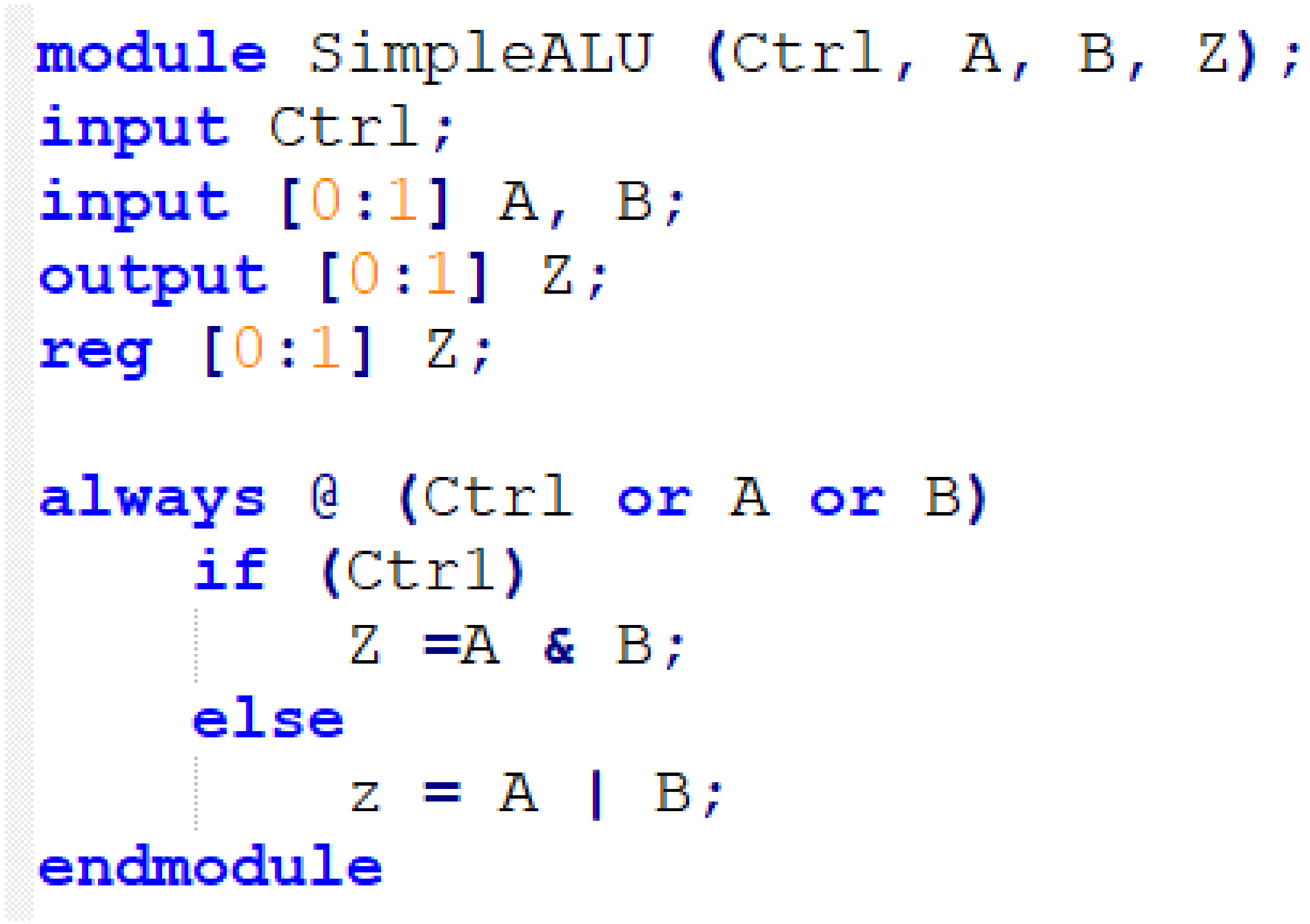
```
module SelectOneOf (A, B, Z) ;
input [1:0] A, B;
output [1:0] Z;
reg [1:0] Z;

    always @ (A or B)
    if (A> B)
        Z =A;
    else
        Z = B;
endmodule
```

## If Statement



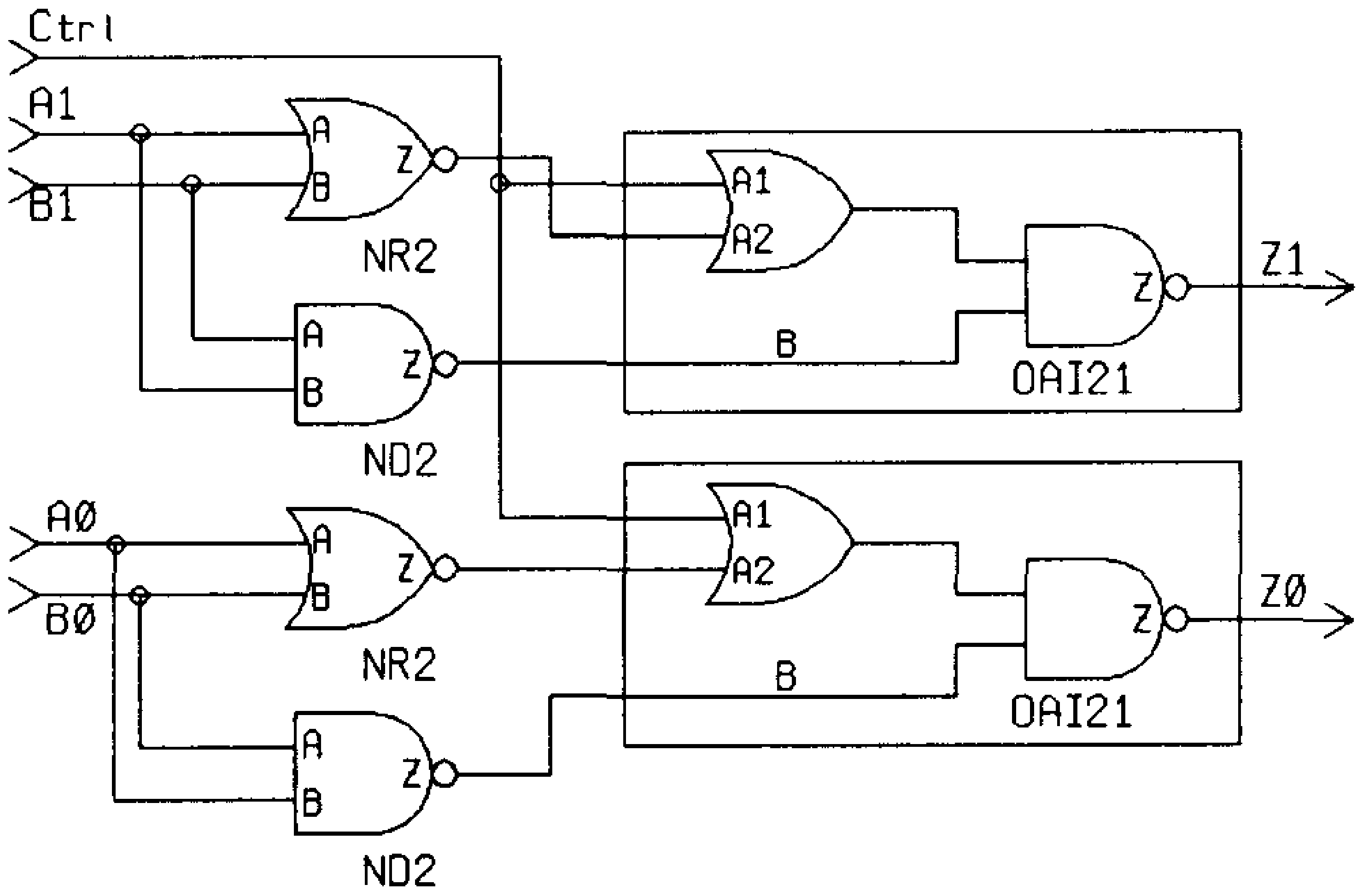
Logic derived from an if statement.

## If Statement



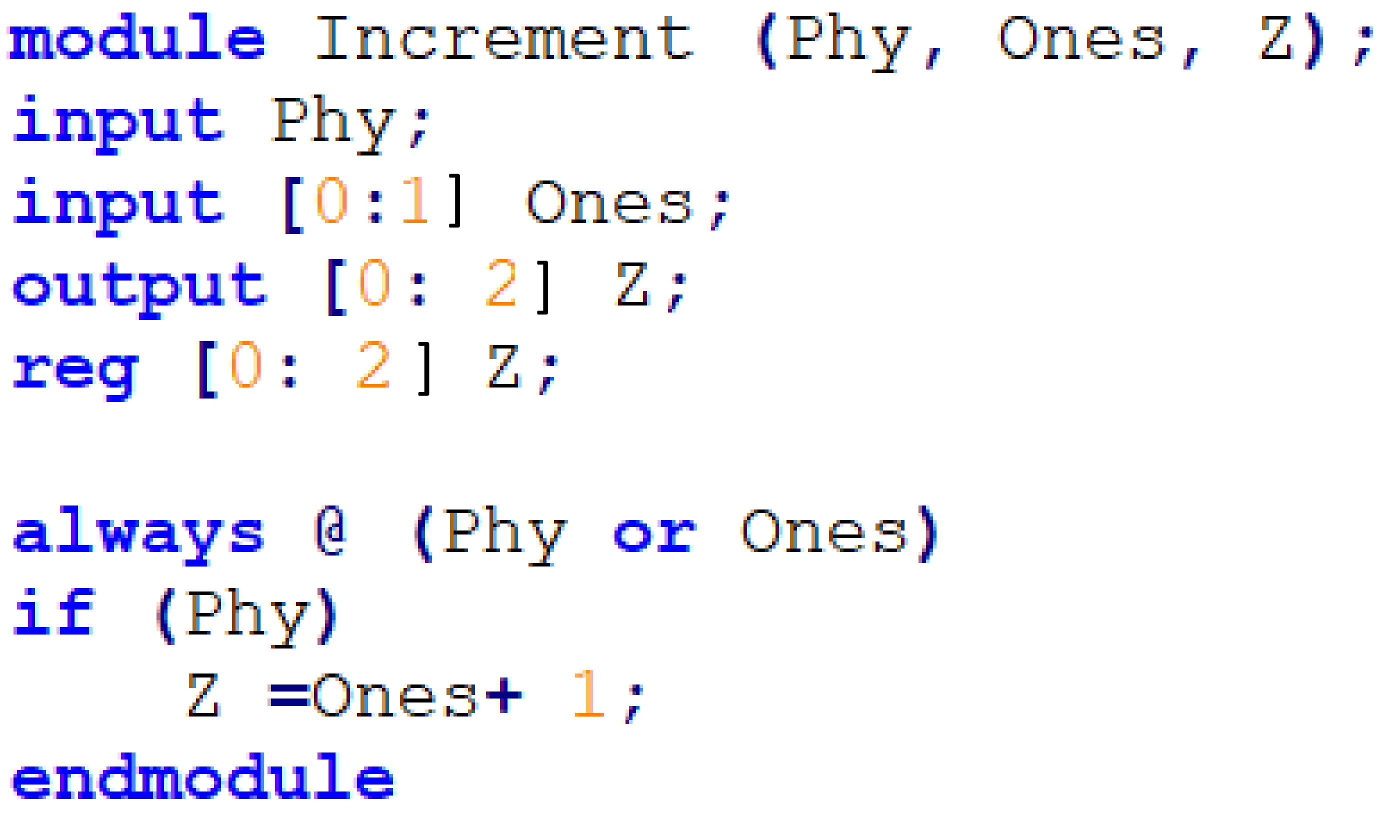
```
module SimpleALU (Ctrl, A, B, Z);
input Ctrl;
input [0:1] A, B;
output [0:1] Z;
reg [0:1] Z;

always @ (Ctrl or A or B)
    if (Ctrl)
        Z =A & B;
    else
        z = A | B;
endmodule
```

## If Statement



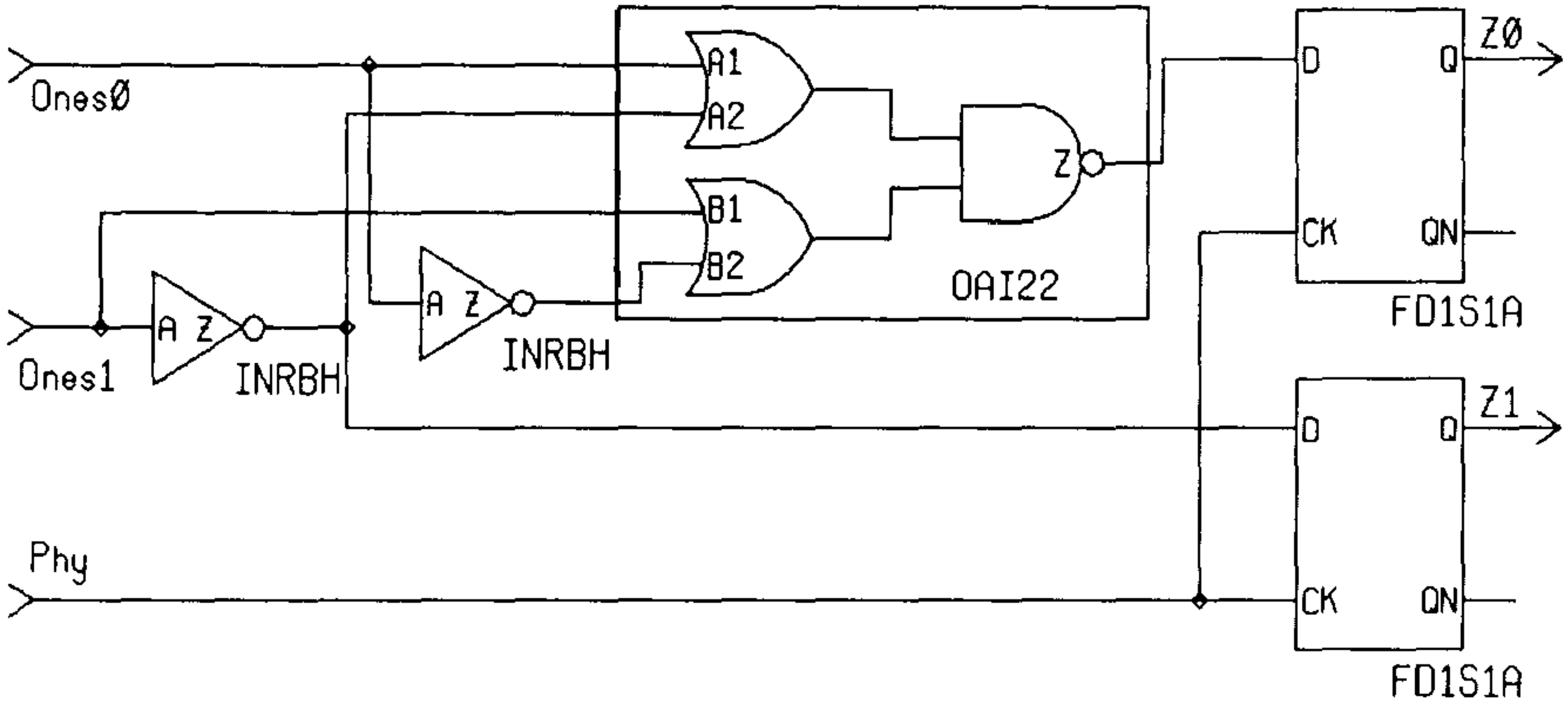
Conditional selection of operations

## Inferring Latches from If Statements



```
module Increment (Phy, Ones, Z);
input Phy;
input [0:1] Ones;
output [0: 2] Z;
reg [0: 2] Z;

always @ (Phy or Ones)
if (Phy)
    Z =Ones+ 1;
endmodule
```

## Inferring Latches from If Statements



A variable is synthesized as a latch.

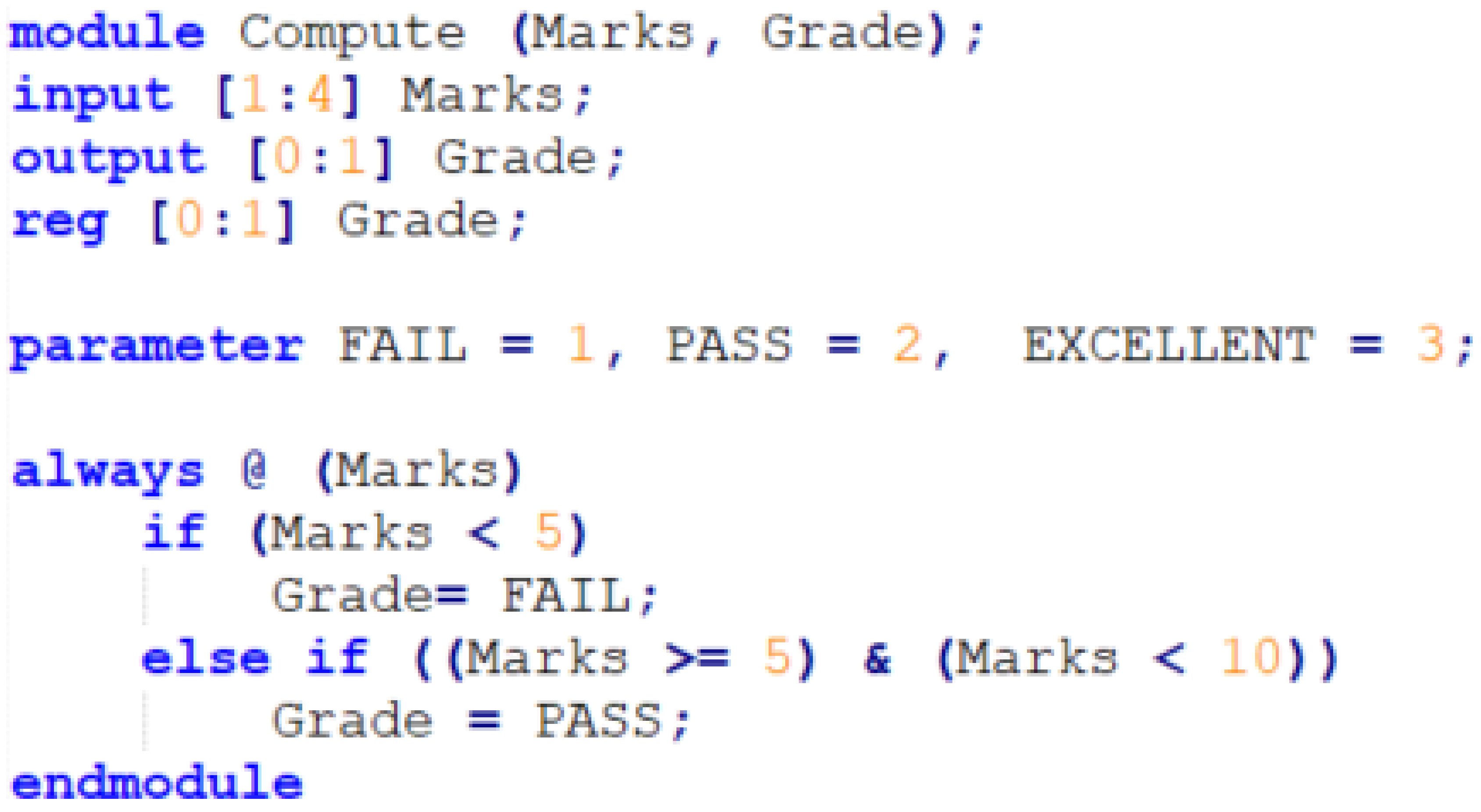
## Inferring Latches from If Statements

- The semantics of the always statement specifies that every time an event occurs on *Phy* or *Ones,* the if statement executes and the variable Z gets the value of *Ones* incremented by 1 if *Phy* is a 1.

- If *Phy* is a 0, the value in Z is retained; this is done using latches.

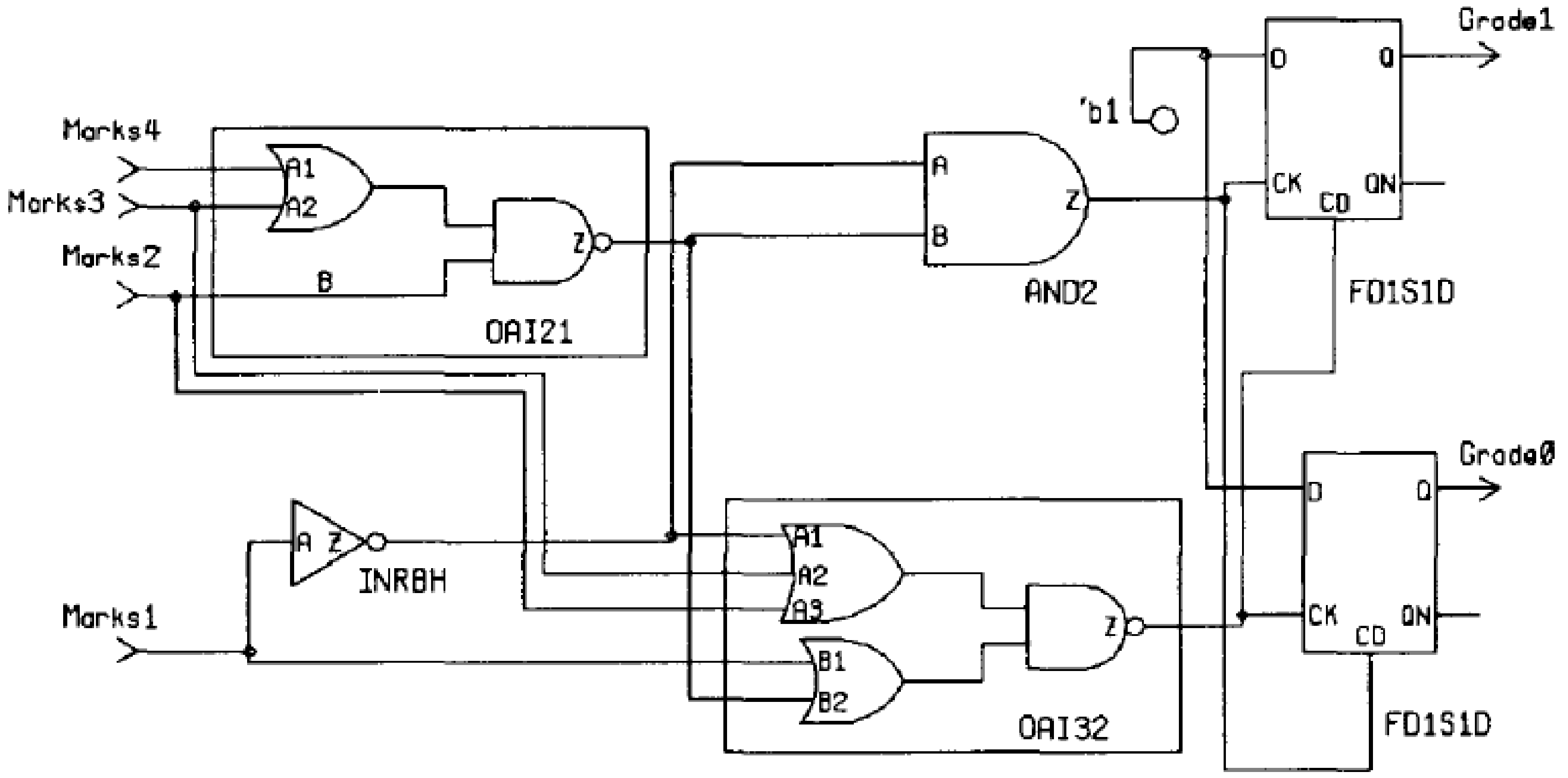
- A general rule for latch inferencing is that if a variable is not assigned in all possible executions of an always statement (for example, when a variable is not assigned in all branches of an if statement), then a latch is inferred.

## Inferring Latches from If Statements



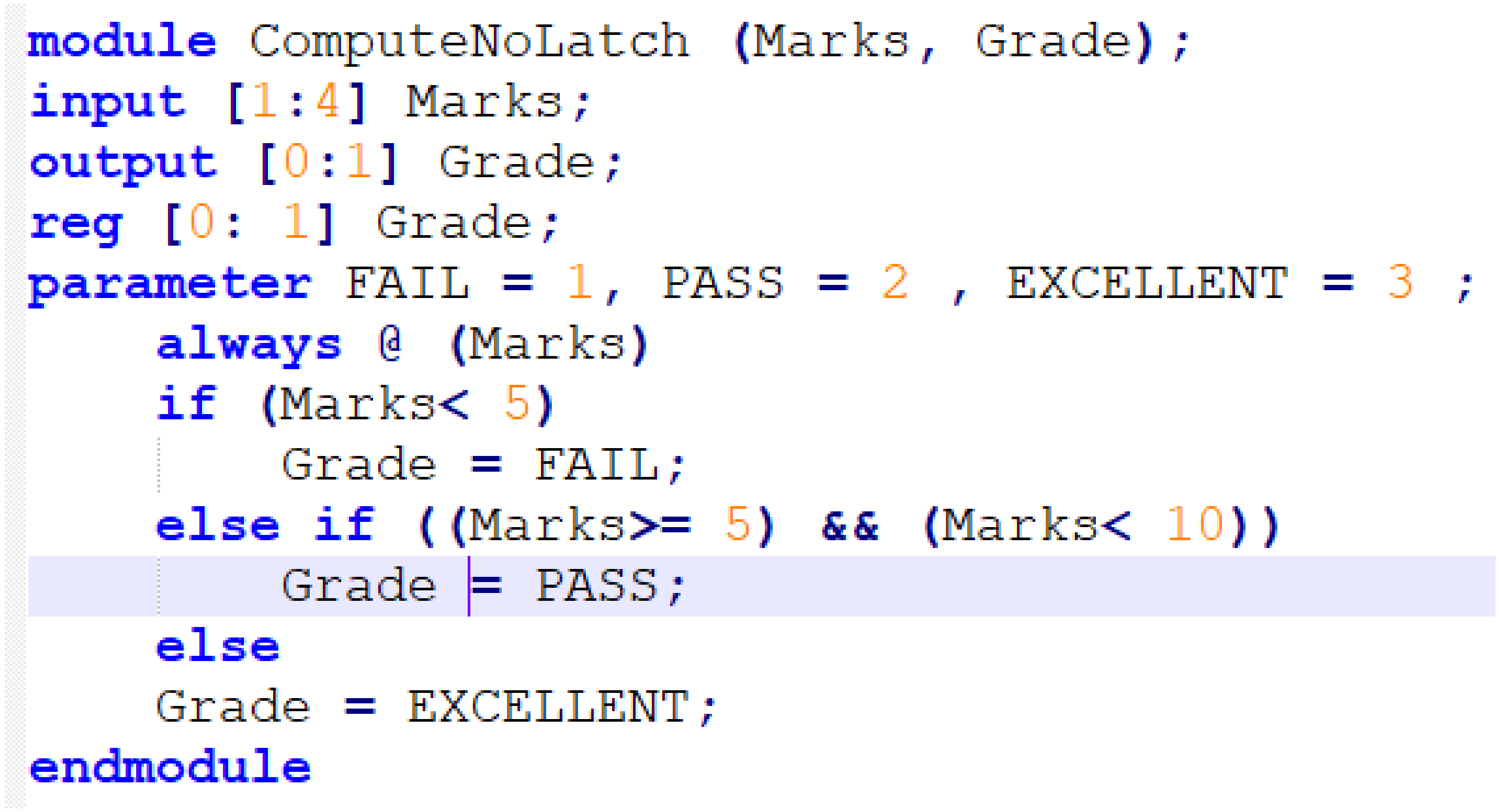
```
module Compute (Marks, Grade);
input [1:4] Marks;
output [0:1] Grade;
reg [0:1] Grade;

parameter FAIL = 1, PASS = 2,  EXCELLENT = 3;

always @ (Marks)
    if (Marks < 5)
        Grade= FAIL;
    else if ((Marks >= 5) & (Marks < 10))
        Grade = PASS;
endmodule
```

## Inferring Latches from If Statements



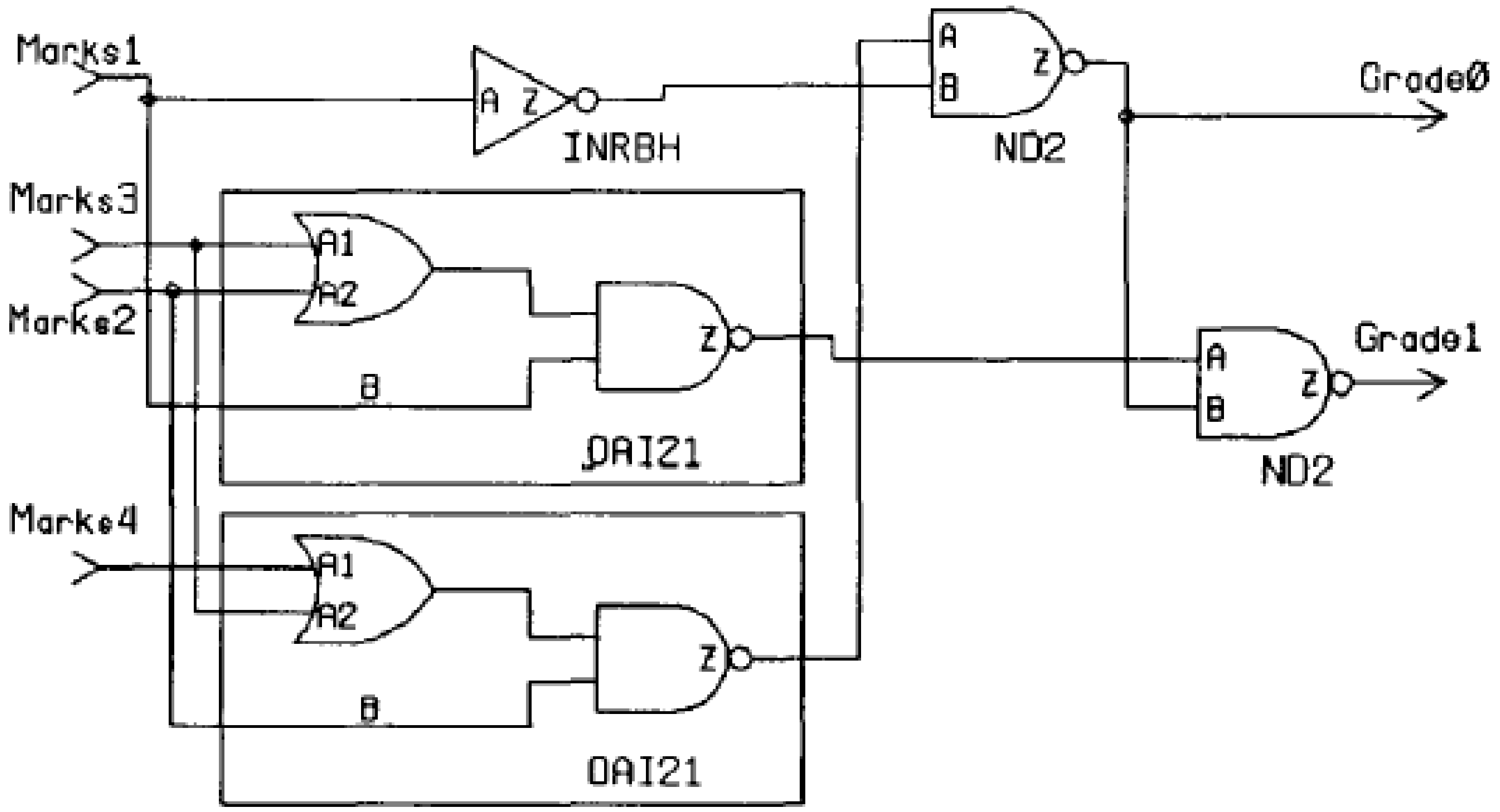
A variable is inferred as a latch.

## Inferring Latches from If Statements



```
module ComputeNoLatch (Marks, Grade);
input [1:4] Marks;
output [0:1] Grade;
reg [0: 1] Grade;
parameter FAIL = 1, PASS = 2 , EXCELLENT = 3 ;
    always @ (Marks)
    if (Marks< 5)
        Grade = FAIL;
    else if ((Marks>= 5) && (Marks< 10))
        Grade = PASS;
    else
    Grade = EXCELLENT;
endmodule
```

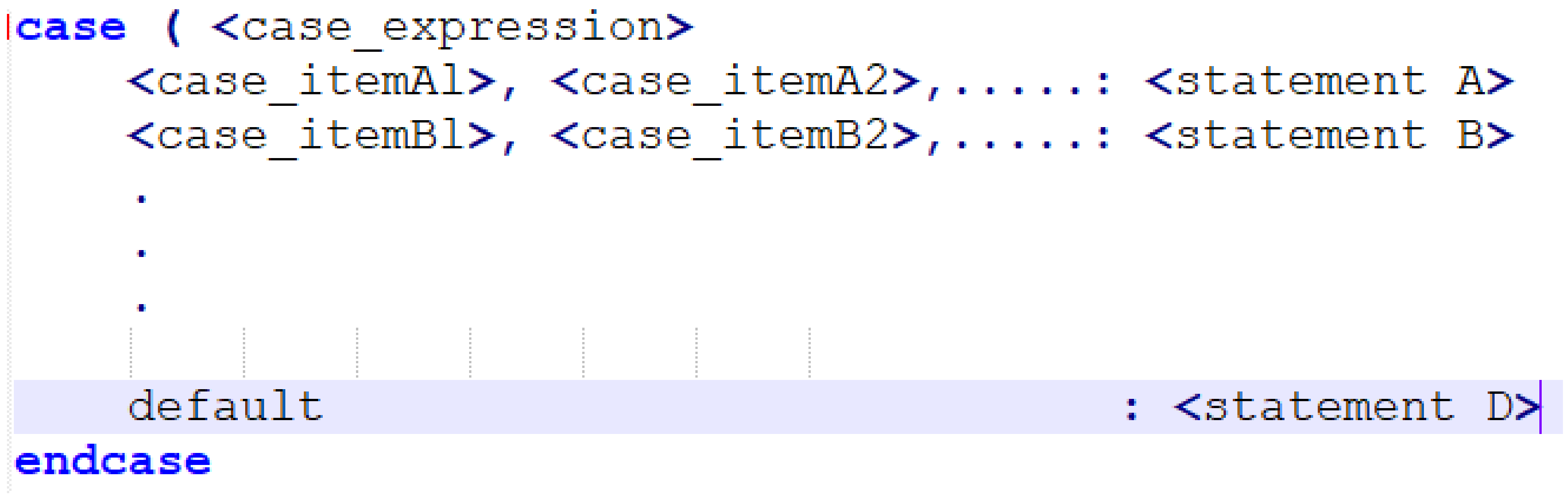
## Inferring Latches from If Statements



Variable GRADE is not a latch.

## Case Statement

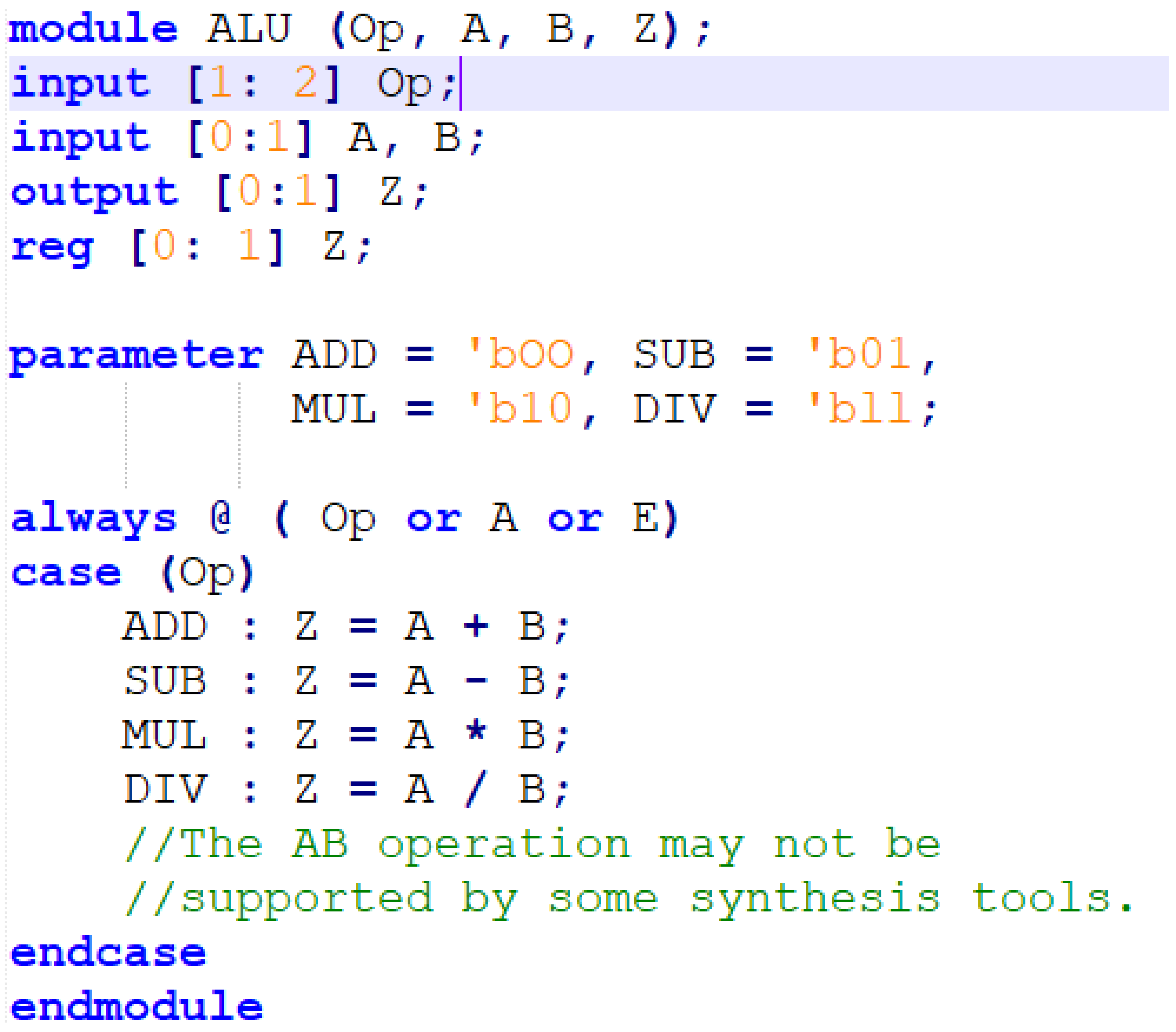
- A case statement is of the form:



```
case ( <case_expression>
    <case_itemAl>, <case_itemA2>,.....: <statement A>
    <case_itemBl>, <case_itemB2>,.....: <statement B>
    .
    .
    .

    default                            : <statement D>
endcase
```

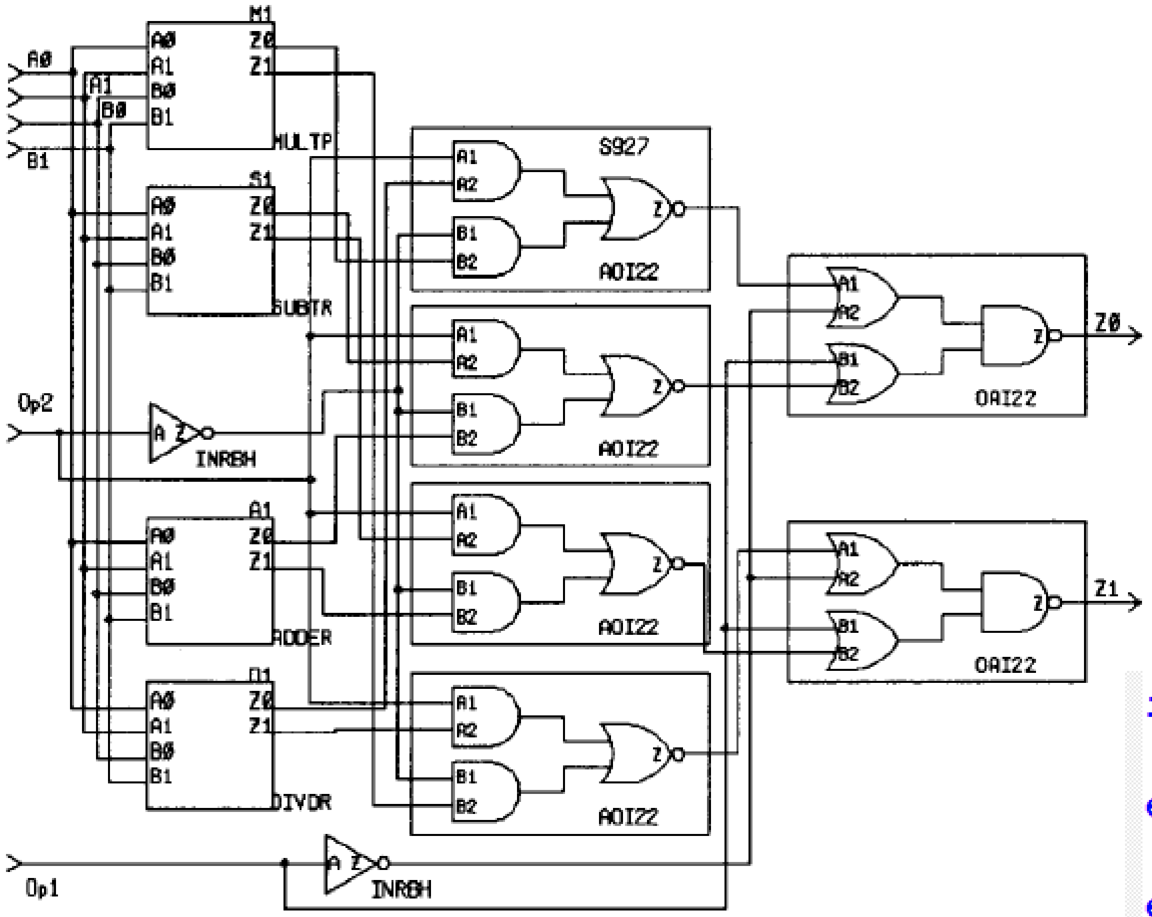
- The first branch that has a case item whose value matches the value of the case expression is selected.
- A case item may be a constant or a variable.

## Case Statement



```
module ALU (Op, A, B, Z);
input [1: 2] Op;
input [0:1] A, B;
output [0:1] Z;
reg [0: 1] Z;

parameter ADD = 'b00, SUB = 'b01,
          MUL = 'b10, DIV = 'b11;

always @ ( Op or A or E)
case (Op)
    ADD : Z = A + B;
    SUB : Z = A - B;
    MUL : Z = A * B;
    DIV : Z = A / B;
    //The AB operation may not be
    //supported by some synthesis tools.
endcase
endmodule
```

## Case Statement



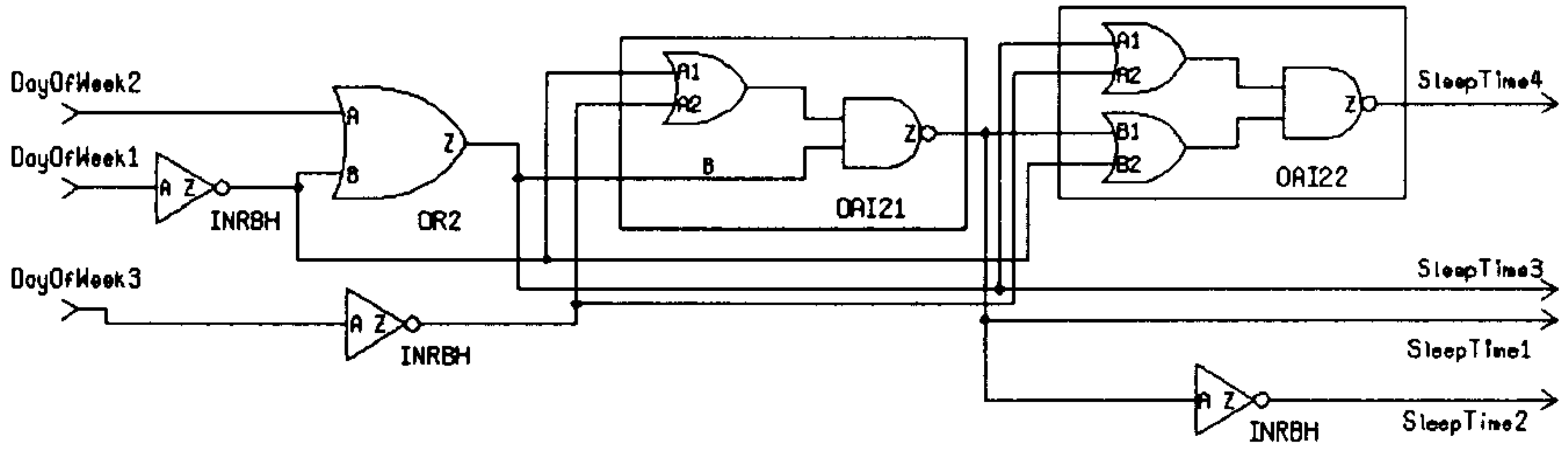
A 2-bit ALU.

```
if ( Op == ADD)
     Z =A + B;
else if ( Op == SUB)
     Z =A - B;
else if (Op == MUL)
     Z =A * B;
else if ( Op == DIV)
     Z =A / B;
```

## Case Statement

```
module CaseExample (DayOfWeek, SleepTime);
input [1:3] DayOfWeek;
output [1:4] SleepTime;
reg [1:4] SleepTime;
parameter MON = 0, TUE = 1 , WED = 2 ,
          THU = 3 , FRI = 4 ,
          SAT = 5, SUN = 6 ;
always@(DayOfWeek)
case (DayOfWeek)
    MON,
    TUE,
    WED,
    THU: SleepTime = 6;
    FRI: SleepTime = 8;
    SAT: SleepTime = 9;
    SUN: SleepTime = 7;
    default: SleepTime = 10; // Enjoy!
    //The default covers the case when DayOfWeek
    // has value 7.
endcase
endmodule
```

## Case Statement

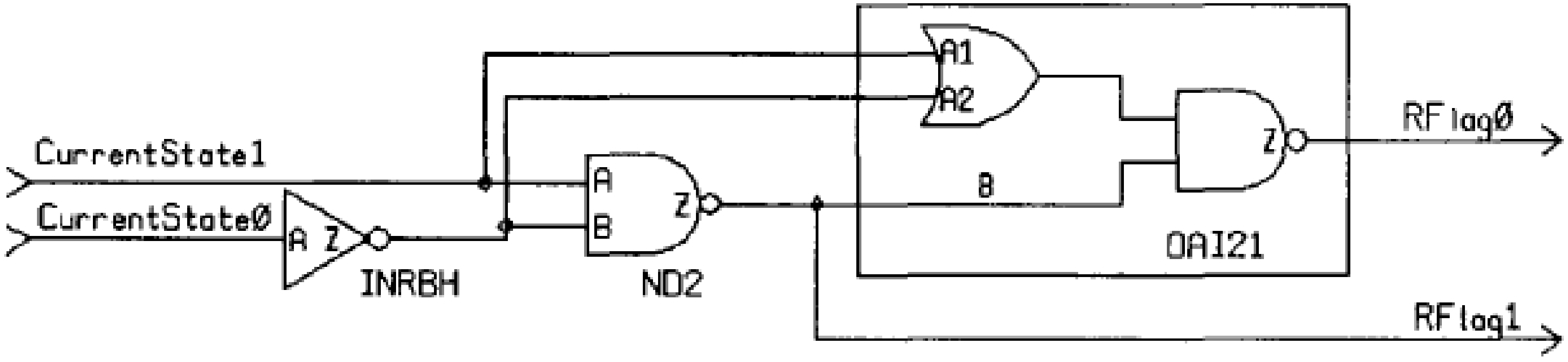


A case statement example.

## Case Statement

```
module SelectAndAssign (CurrentState, RFlag);
input [0:1] CurrentState;
output [0:1] RFlag;
reg [0:1] RFlag;
parameter RESET= 2'b01, APPLY= 2'b11, WAITS = 2'b10,
DONTCARE = 2'b00
always @ (CurrentState)
case (CurrentState)
RESET: RFlag = WAITS;
APPLY: RFlag = RESET;
WAITS: RFlag = APPLY;
default : RFlag = DONTCARE;
endcase
endmodule
```

## Case Statement



Logic generated from a case statement

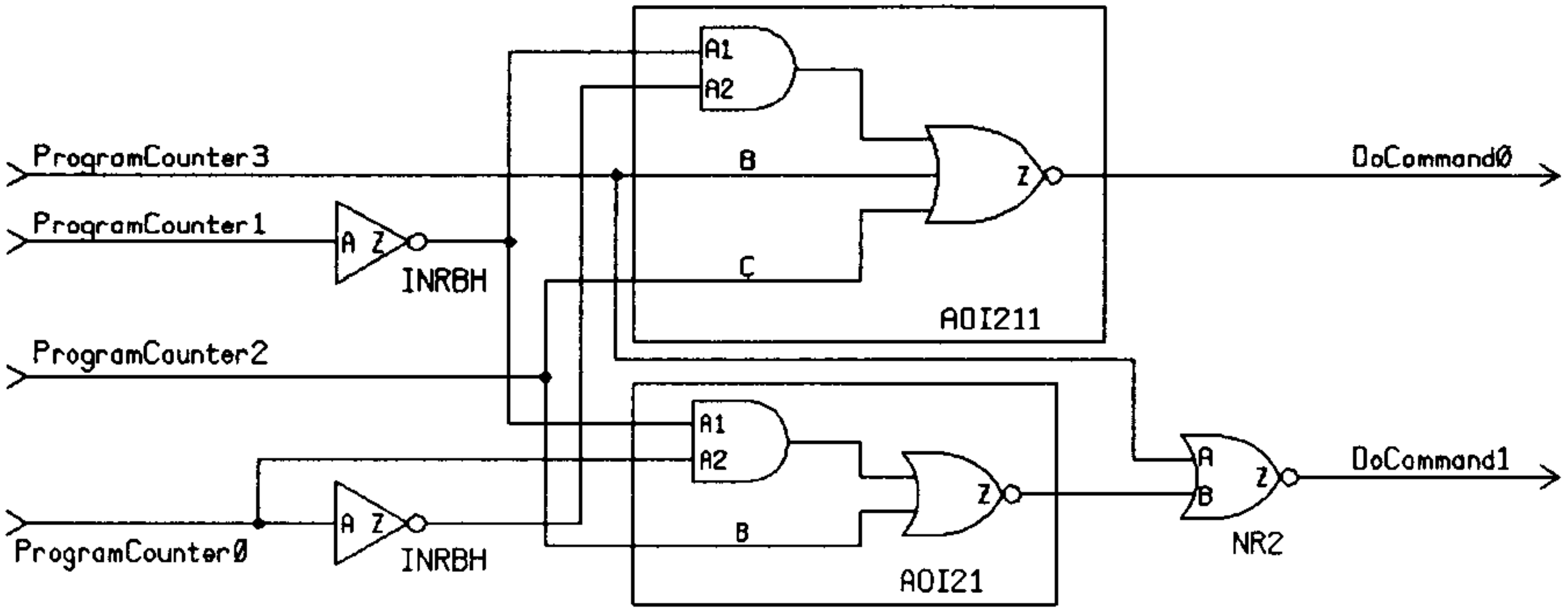
## Casez Statement

- In a casez statement, the value z is considered as a don't-care when it appears in a case item expression.
- The ? character can also be used alternatively for the character z.
- Values z and x are not allowed in a case expression.
- Additionally, value x cannot appear in a case item expression.

## Casez Statement

```
module CasezExample (ProgramCounter, DoComrnand);
input [0:3] ProgramCounter;
output [0:1] DoCommand;
reg [ 0 : 1] DoCommand;

always @ (ProgramCounter)
    casez (ProgramCounter)
    4'b???1 : DoCommand = 0 ;
    4'b??10 : DoCommand = 1 ;
    4'b?100 : DoCommand = 2 ;
    4'b1000 : DoCommand = 3 ;
default : DoCommand = 0 ;
endcase
endmodule
```

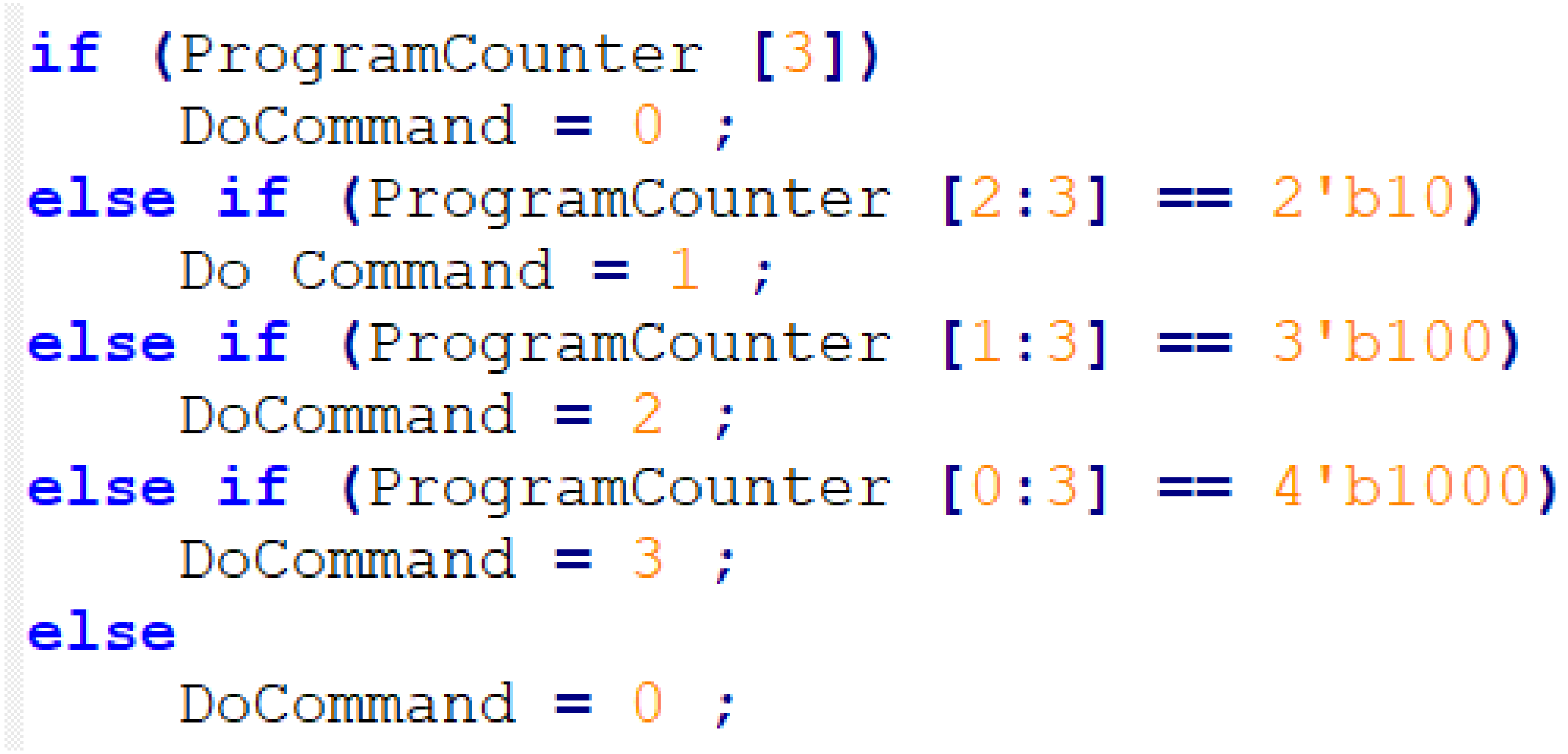
## Casez Statement



Casez statement example.

## Casez Statement

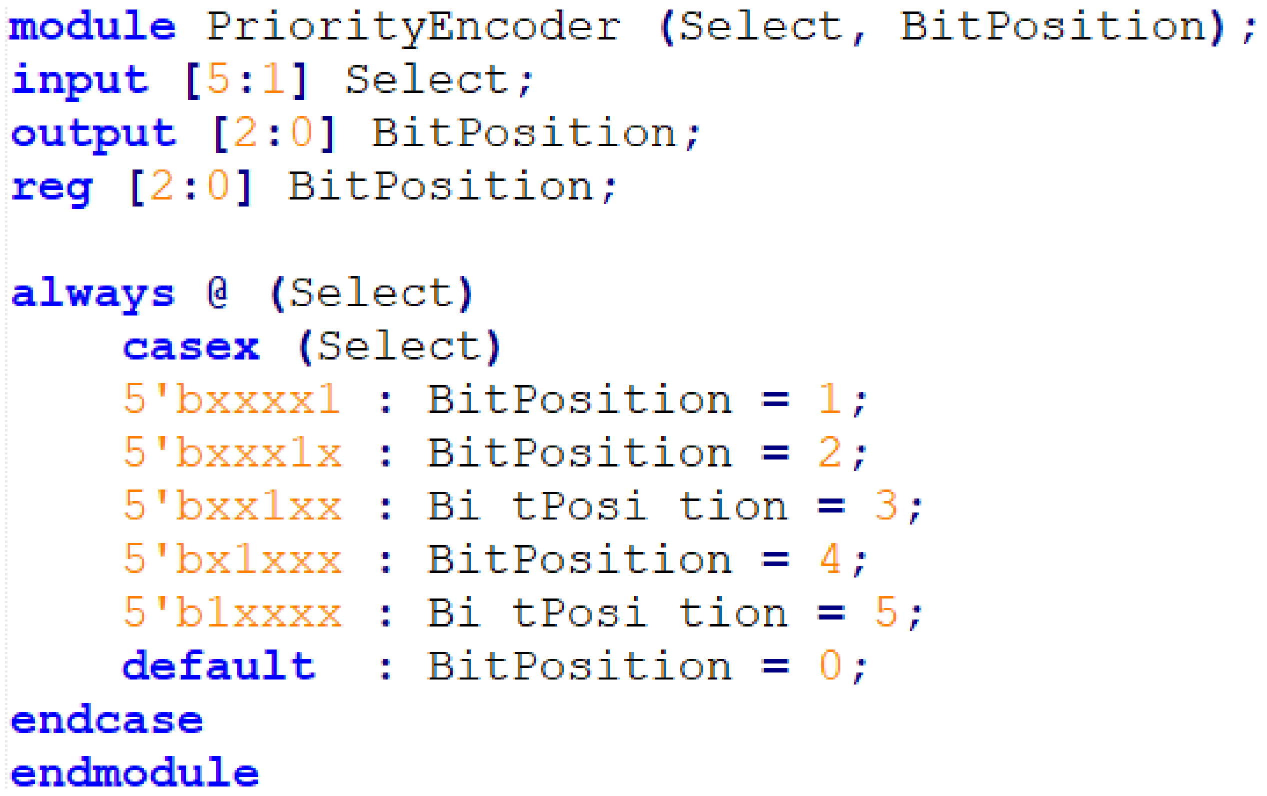
- The casez statement is equivalent to the following if statement (note that the ? character in a case item denotes a don't-care value).



```
if (ProgramCounter [3])
    DoCommand = 0 ;
else if (ProgramCounter [2:3] == 2'b10)
    Do Command = 1 ;
else if (ProgramCounter [1:3] == 3'b100)
    DoCommand = 2 ;
else if (ProgramCounter [0:3] == 4'b1000)
    DoCommand = 3 ;
else
    DoCommand = 0 ;
```

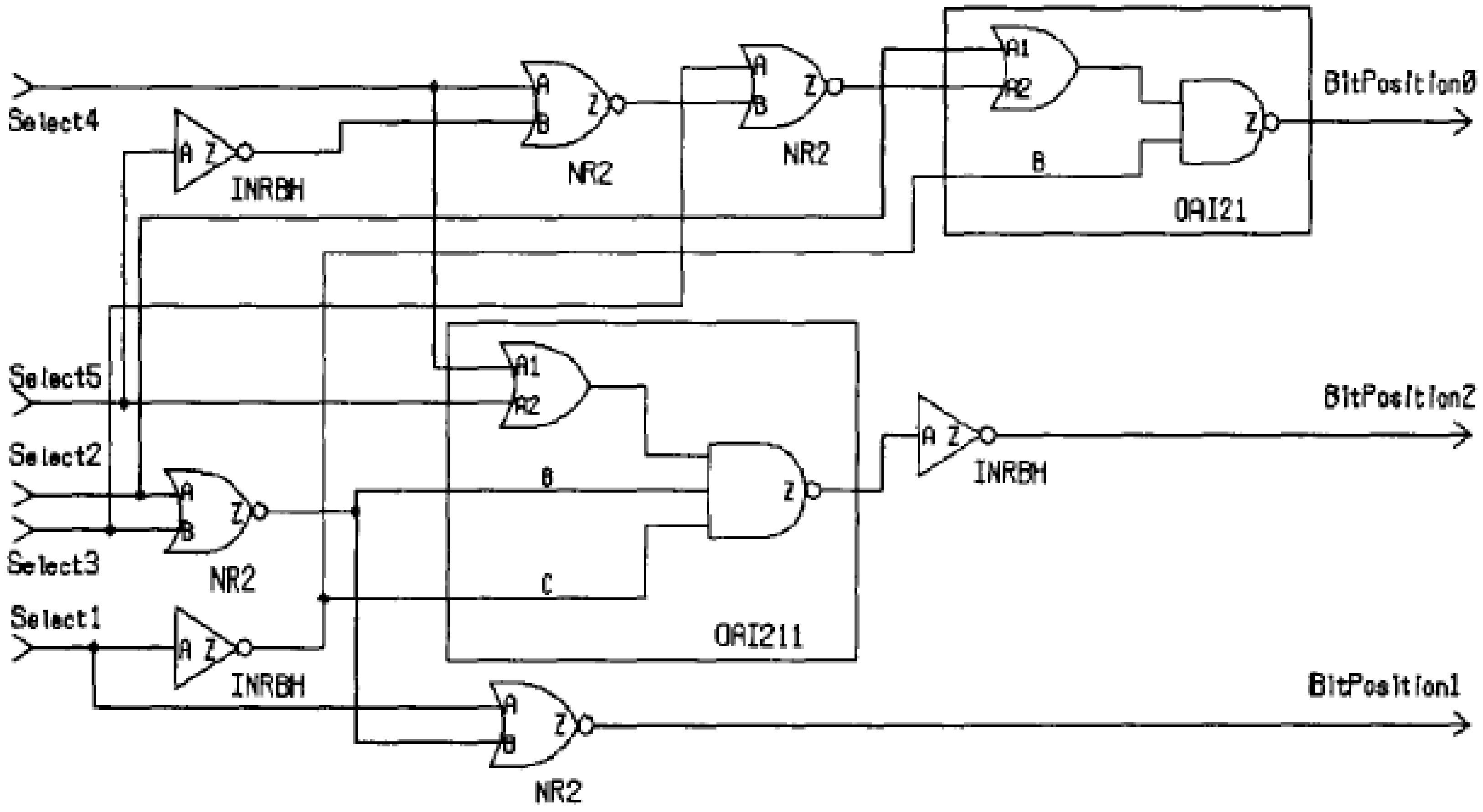
## Casex Statement

- In a casex statement, the values x and z in a case item expression are considered as don't-care values. These values, for synthesis purposes, cannot appear as part of the case expression.



```
module PriorityEncoder (Select, BitPosition);
input [5:1] Select;
output [2:0] BitPosition;
reg [2:0] BitPosition;

always @ (Select)
    casex (Select)
    5'bxxxx1 : BitPosition = 1;
    5'bxxx1x : BitPosition = 2;
    5'bxx1xx : Bi tPosi tion = 3;
    5'bx1xxx : BitPosition = 4;
    5'b1xxxx : Bi tPosi tion = 5;
    default  : BitPosition = 0;
endcase
endmodule
```

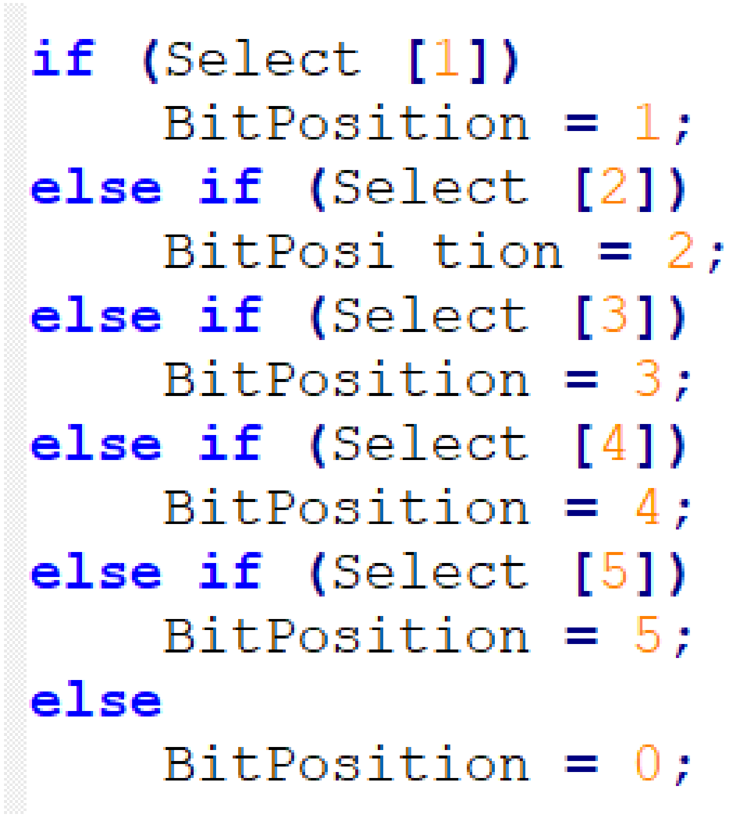
## Casex Statement



A priority encoder using casex statement

## Casex Statement

- The semantics of this casex statement can best be expressed by its equivalent if statement.



```
if (Select [1])
    BitPosition = 1;
else if (Select [2])
    BitPosi tion = 2;
else if (Select [3])
    BitPosition = 3;
else if (Select [4])
    BitPosition = 4;
else if (Select [5])
    BitPosition = 5;
else
    BitPosition = 0;
```

## References

- Chapter 2, Verilog HDL Synthesis  by J. Bhaskar

## Thank you