Digital Design with Verilog

Verilog - Synthesis

Lecture 23: Logic Synthesis with Verilog HDL – Part6

Objectives

- Clock Rules

- Flip-Flops with Asynchronous Preset/Clear

Clock Rule

- A synthesis system imposes a restriction that a variable cannot be assigned under the control of more than one clock.

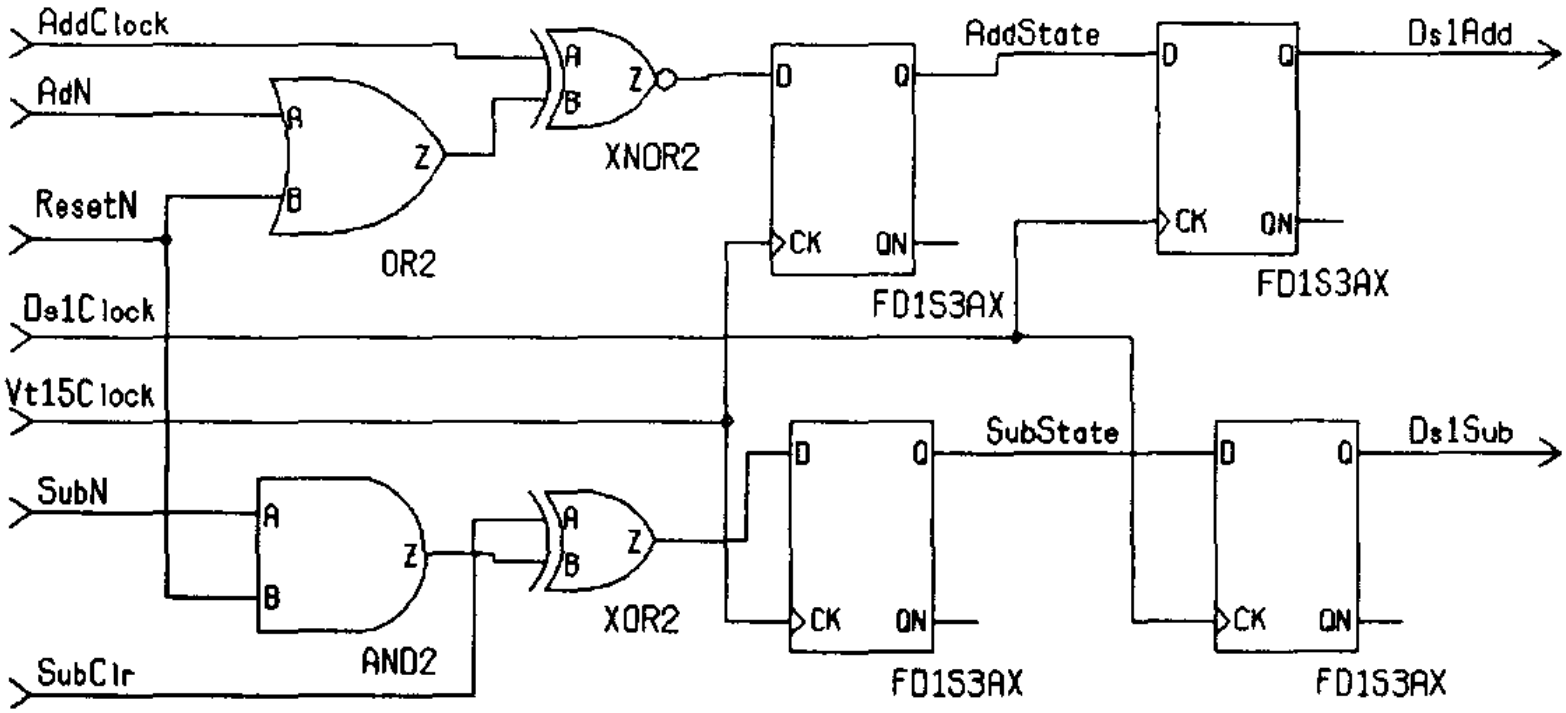
- However, there can be more than one clocks as the following multiple clock example suggests.

Multiple Clocks

```
module MultipleClocks (Vtl5Clock, AddClock, AdN,
ResetN, SubClr, SubN, DslClock, DslAdd, DslSub);
input Vtl5Clock, AddClock, AdN, ResetN, SubClr,
SubN, DslClock;
output DslAdd, DslSub;
reg DslAdd, DslSub;
reg AddState, SubState;

always@(posedge Vtl5Clock)
    begin
        AddState <= AddClock ^~ (AdN | ResetN);
        SubState <= SubClr ^ (SubN & ResetN);
    end
always @ (posedge DslClock)
    begin
        DslAdd <= AddState;
        DslSub <= SubState;
    end
endmodule
```

Multiple Clocks



Multiple clocks within an always statement.

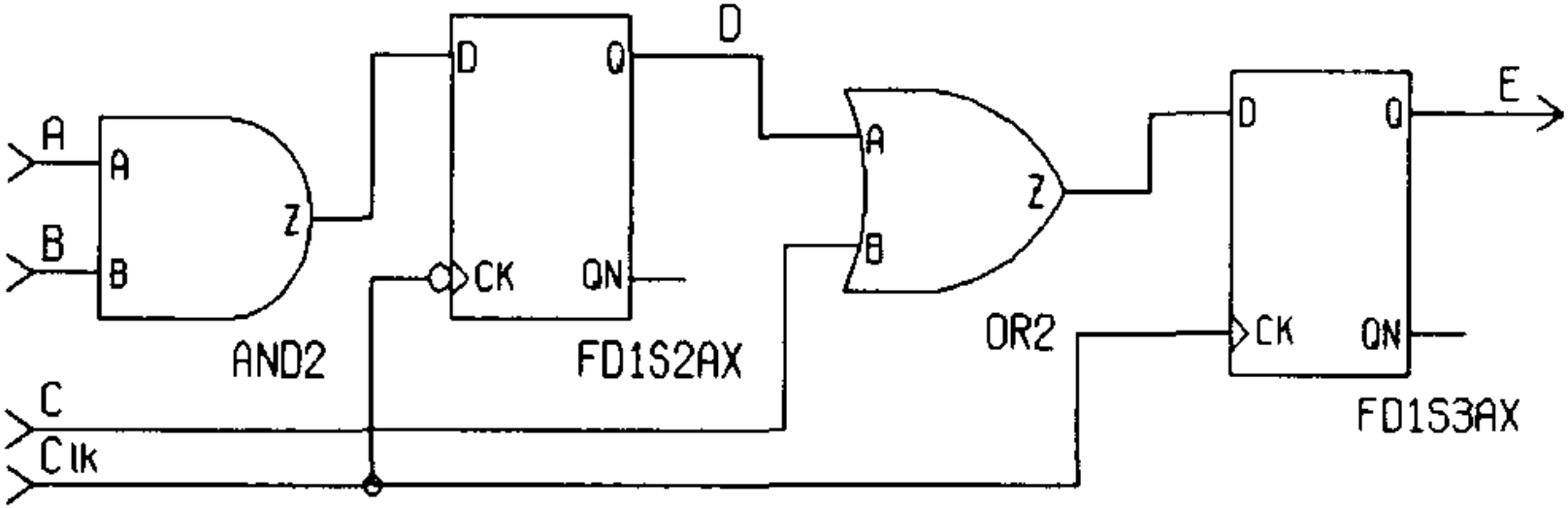
Multi-phase Clocks

- It is possible to have a single module with multiple clocked always statements in which different edges of the same clock are used.

```
module MultiPhaseClocks (Clk, A, B, C, E);
input Clk, A, B, C;
output E;
reg E, D;

always @ (posedge Clk)
    E <= D | C;
always @ (negedge Clk)
    D <=A & B;
endmodule
```

Multi-phase Clocks



Different edges of the same clock within a single module.

Flip-flops with Asynchronous Preset and Clear

- Inferring a flip-flop with Asynchronous Preset and Clear comes out of a combination of clocked and non-clocked events inside an always block's "sensitivity" list.

- There is one clocked event, say, "*posedge clk*" or "*negedge clk*"

Flip-flops with Asynchronous Preset and Clear

- The remaining events are “posedge A” or “negedge A”, where A can be any Boolean expression.
- The always loop looks like

```
always @ (posedge A or negedge B or negedge C ...
or posedge Clock)
if (A) // posedge A.
<statement>      // Asynchronous logic.
else if ( ! B)   // negedge B.
<statement>      // Asynchronous logic.
else if ( ! C)   // negedge C.
<statement>      // Asynchronous logic.
                 // Any number of else if's.
else             // posedge Clock implied.
<statement>      // Synchronous logic.
```

- Note that the conditions “*posedge A*” in sensitivity list appears in the corresponding “if” as “A” and similarly “*negedge B*” in sensitivity list appears as “!B” in the corresponding “if”.

Flip-flops with Asynchronous Preset and Clear

- Any variable that is assigned both in the synchronous and asynchronous part becomes a flip-flop with asynchronous preset/clear.

- The logic that drives it in the asynchronous part drives the asynchronous preset/clear port of the corresponding flip-flop, while the logic that drives it in the synchronous part is assigned to the D-input

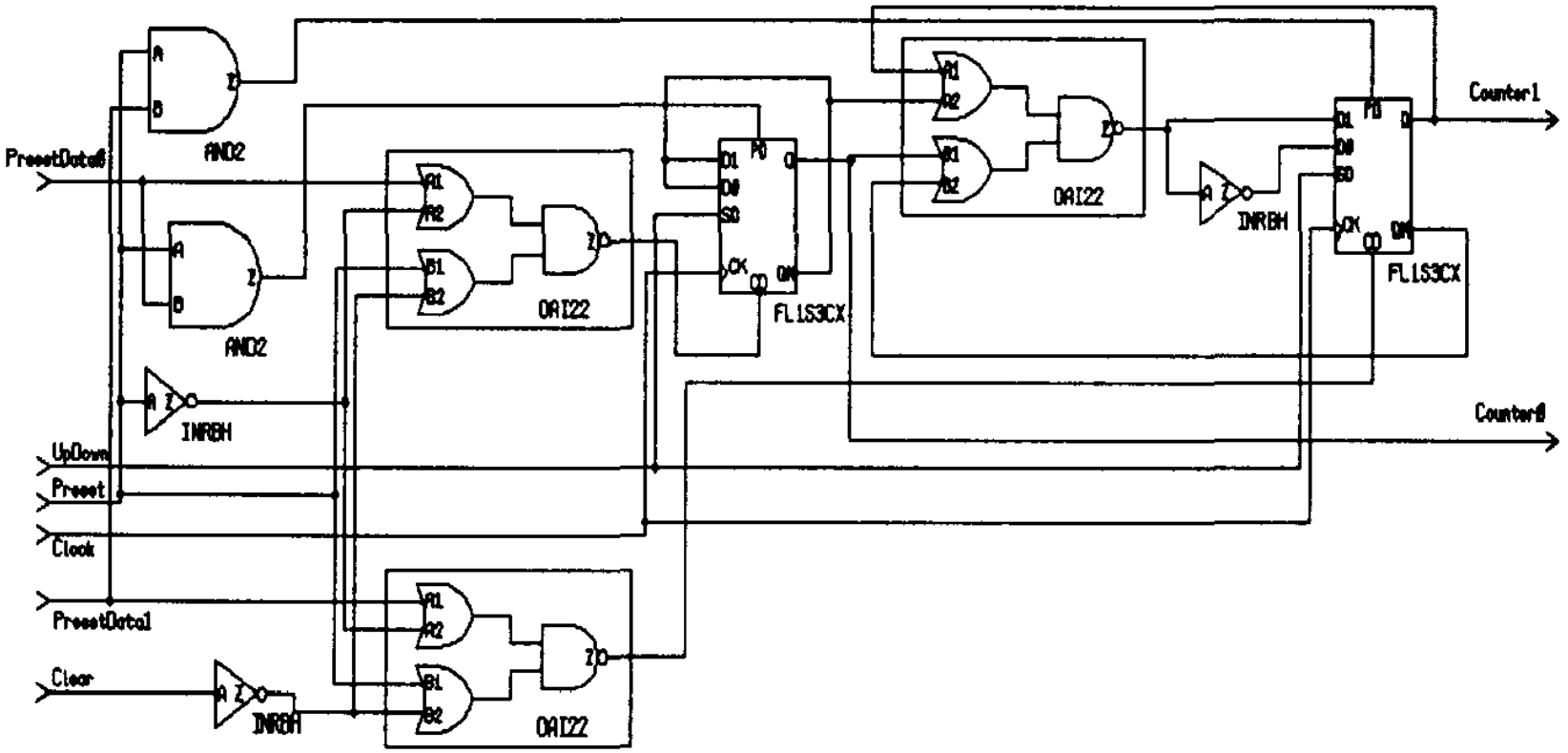
- The variable that forms the "*Clocked*" event, (the default case) drives the clock input of the flip-flop.

Flip-flops with Asynchronous Preset and Clear

```
module AsyncPreClrCounter (Clock, Preset, UpDown,
                           Clear, PresetData, Counter);
parameter NUM_BITS = 2 ;
input Clock, Preset, UpDown, Clear;
input [NUM_BIT5-1:0] PresetData;
output [NUM_BITS-1:0] Counter;
reg [NUM_BITS-1:0] Counter;

always@(posedge Preset or posedge Clear or
                                  posedge Clock)
    if (Preset)
        Counter<= PresetData;
    else if (Clear)
        Counter <= 0;
    else  // Implicit posedge Clock.
        begin  // Synchronous part
            if (UpDown)
                Counter <= Counter+ 1;
            else
                Counter <= Counter- 1;
        end
endmodule
```

Flip-flops with Asynchronous Preset and Clear



Flip-flops with asynchronous preset and clear

Caution

- When Preset is ‘1’and *Presetdata* changes, this will NOT be reflected in Verilog HDL but will be propagated to the flip-flops in the synthesized netlist.

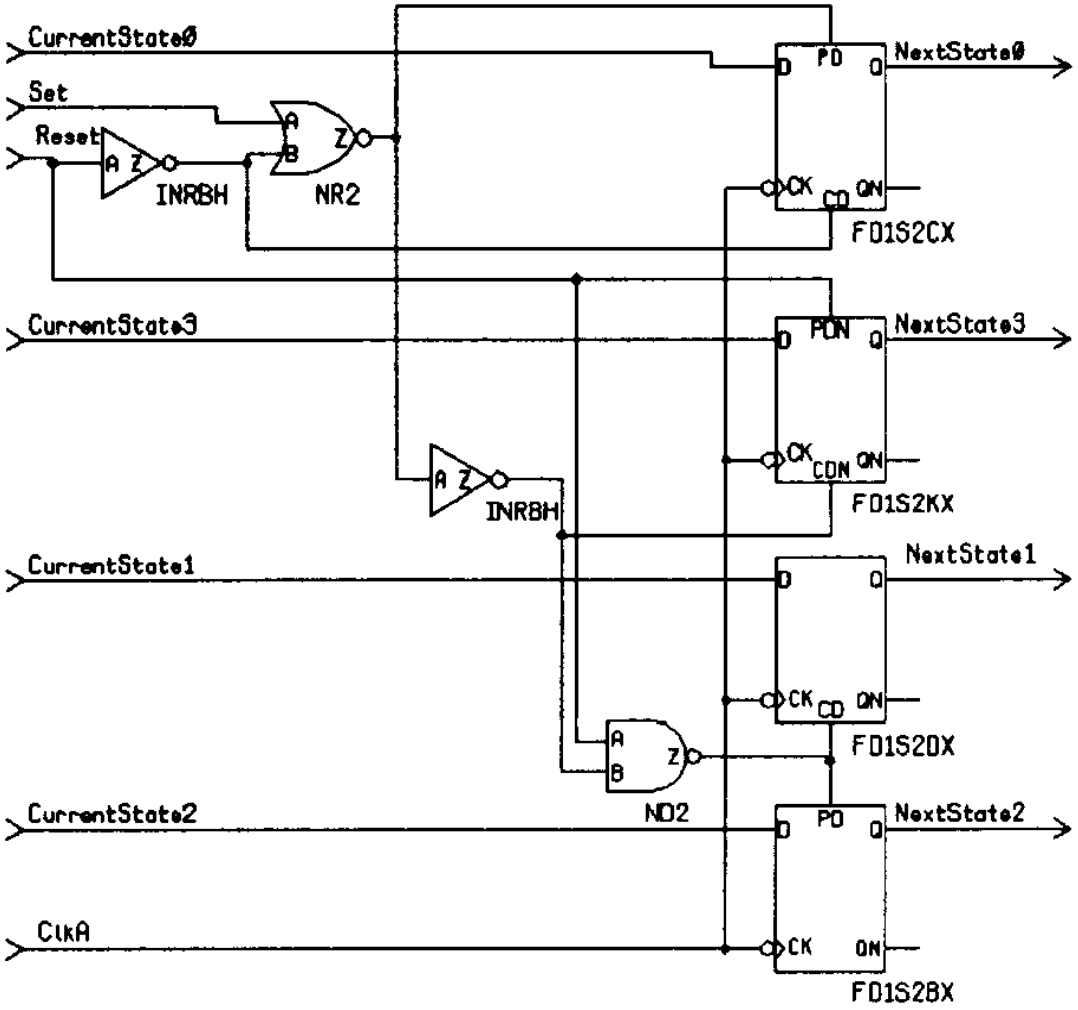
- Be careful while using Asynchronous Inputs to flip-flops for functional mismatch

Flip-Flops with Asynchronous Preset/Clear

```
module AsyncFlipFlop (ClkA, Reset, Set, CurrentState,
                                        NextState);
input ClkA, Reset, Set;
input [3:0] CurrentState;
output [3:0] NextState;
reg [3:0] NextState;

always @ (negedge Reset or negedge Set or negedge ClkA)
    if (! Reset)
        NextState <= 12;            // Stmt A
    else if (! Set)
        NextState <= 5;             // Stmt B
    else
        NextState <= CurrentState;  // Stmt C
endmodule
```

Flip-Flops with Asynchronous Preset/Clear



Flip-flops with asynchronous preset and clear

Some Interesting facts

- Second flip-flop has only ‘preset’ and third has only ‘clear’, while the others have both.

Flip-flops with Synchronous Preset/Clear

- Define the preset and clear logic inside a Clocked Always Statement.

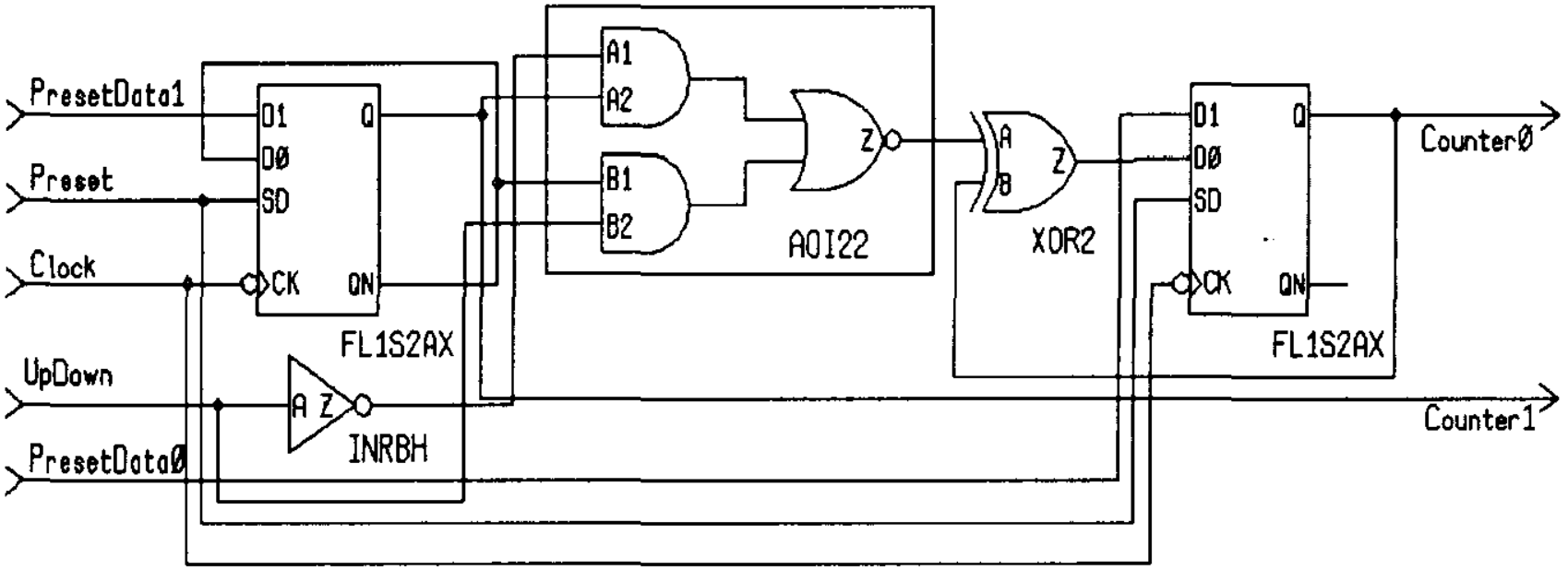
```
module SyncPresetCounter (Clock, Preset, UpDown,
                          PresetData, Counter);
parameter NBITS = 2 ;
input Clock, Preset, UpDown;
input [0:NBIT5-1] PresetData;
output [0:NBIT5-1] Counter;
reg [0:NBITS-1] Counter;

always @ (negedge Clock)
    if (Preset)
        Counter<= PresetData;
    else
    if (UpDown)
        Counter <= Counter + 1;
    else
        Counter<= Counter- 1;
endmodule
```

Two Approaches

- Direct the *PresetData* input to synchronous preset input of the synthesized flip-flops.
- Direct the *PresetData* input to the data input of the flip-flops.
- The following circuit is got using the latter option.

Flip-flops with Synchronous Preset/Clear



Synchronous preset clear synthesized as combinational logic.

Another Example

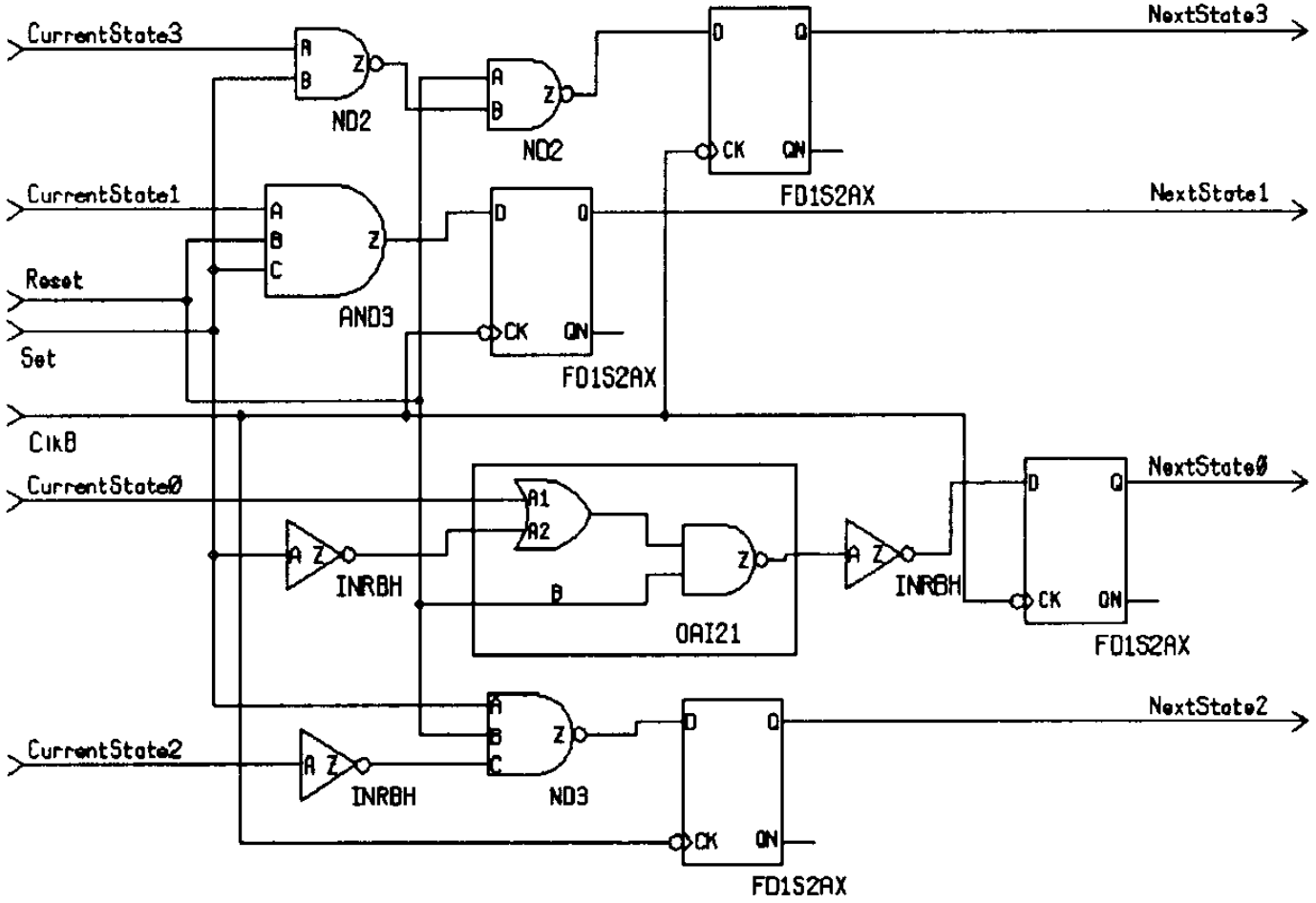
```
module SyncFlipFlop (ClkB, Reset, Set, CurrentState,
                     NextState);
input ClkB, Reset, Set;
input [3:0] CurrentState;
output [3:0] NextState;
reg [3:0] NextState;

always @ (negedge ClkB)
    if (! Reset)
        NextState <= 12;
    else if (! Set)
        NextState <= 5;
    else
    NextState <= CurrentState;
endmodule
```

The Values

- The value 12, 5, and CurrentState are to be multiplexed and directed to the D inputs.

Flip-flops with Synchronous Preset/Clear



Not a synchronous preset and clear flip-flop.

Thank you