CT Verilog Series

Verilog Overview Part 5
Sequential Blocks

黃稚存

Chih-Tsun Huang

cthuang@cs.nthu.edu.tw

國立清華大學
資訊工程學系

07

聲明

- 本課程之內容 (包括但不限於教材、影片、圖片、檔案資料等)，僅供修課學生個人合理使用，非經授課教師同意，不得以任何形式轉載、重製、散布、公開播送、出版或發行本影片內容(例如將課程內容放置公開平台上，如 Facebook, Instagram, YouTube, Twitter, Google Drive, Dropbox 等等)。如有侵權行為，需自負法律責任。

Outline

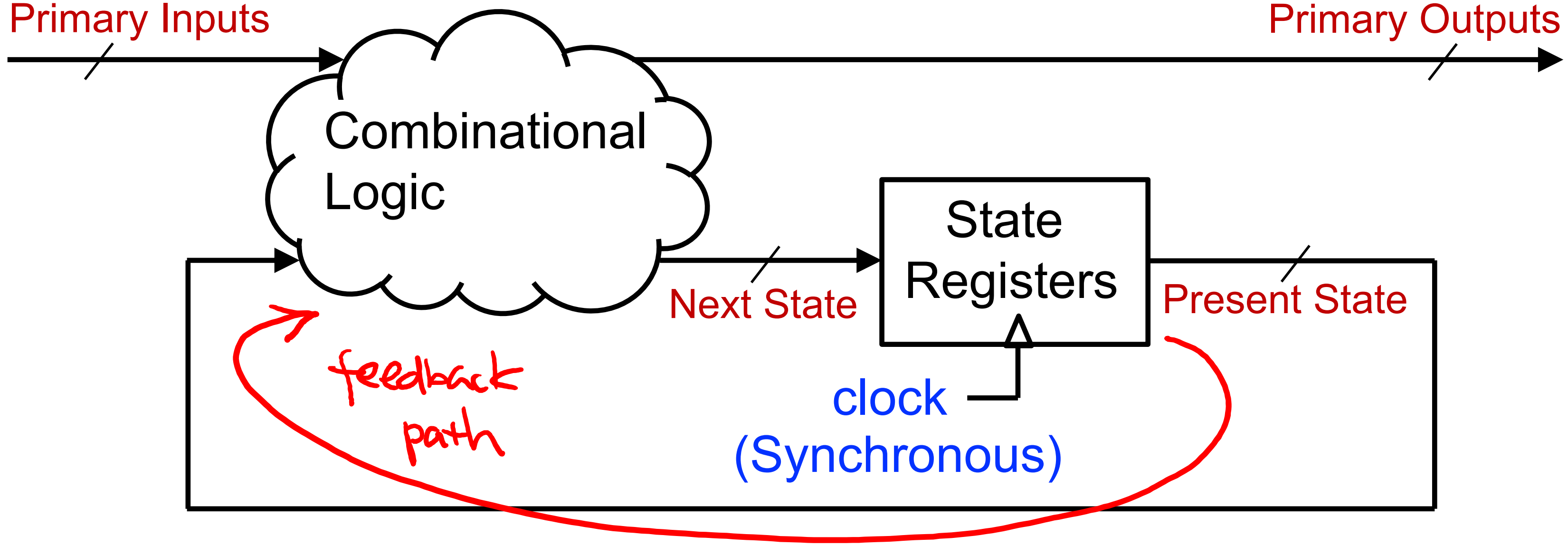
- Sequential Blocks
- Sequential Example:
  4-bit Sequential Counter
- Summary

- Part 4 Recap
  - Combinational Blocks
  - Combinational Examples
  - Module Instantiation
  - Hierarchical Design Style

Sequential Blocks

Sequential Circuits

- A sequential circuit consists of a combinational circuit to which storage elements (state registers) are connected to form a feedback *path*
  - Binary information stored in the memory elements at any given time defines the *state* of the sequential circuit.
  - (primary inputs, present state) ➔ (primary outputs, next state)
  - The behavior is specified by a *time sequence* of inputs and internal states.



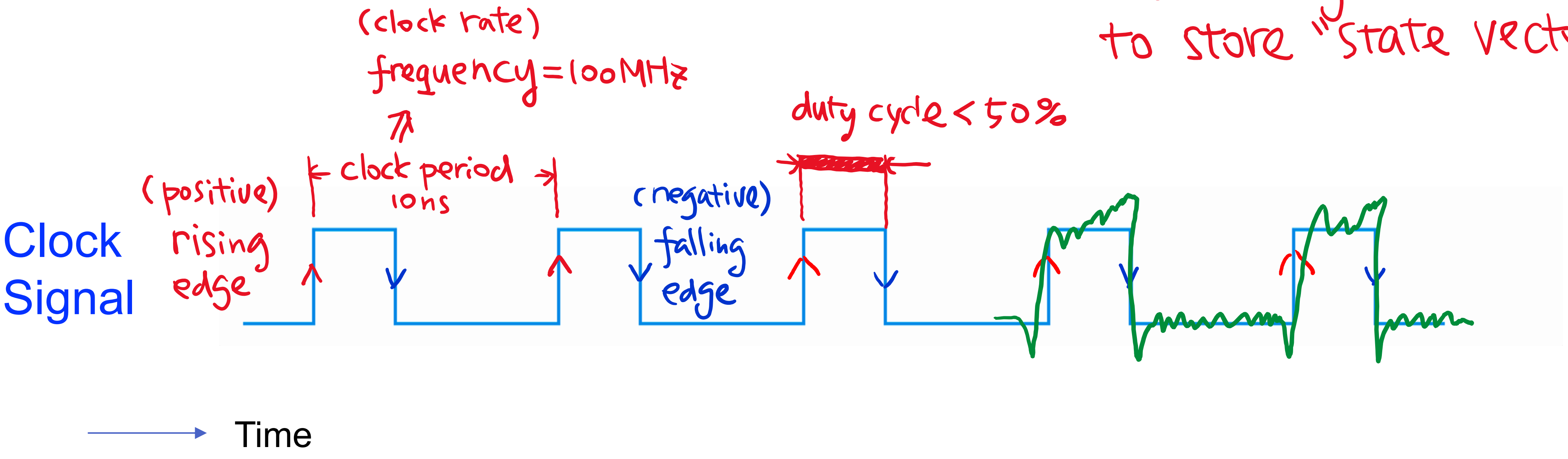
Synchronous vs. Asynchronous Sequential Circuits

- Synchronous
  - Behavior is defined from the input signals at *discrete instants* of time
  - Clocked sequential circuits
  - Most commonly used
  - No instability problems
- Asynchronous
  - Behavior depends on the value and change order of input signals at *any* instant of time
  - Can be viewed as combinational circuit with feedback
  - May be unstable at times

Synchronous Sequential Circuits

- Synchronization usually is achieved by a timing device: clock generator
  - Define the *discrete instants* of time
- Clock generator generates a periodic train of clock pulses distributed throughout the system to trigger the storage elements (flip-flops)

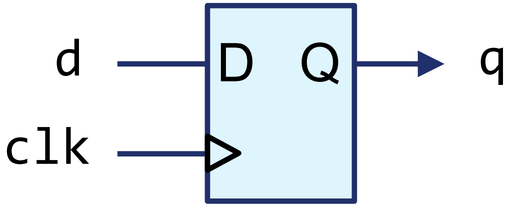
as state registers to store "state vector"



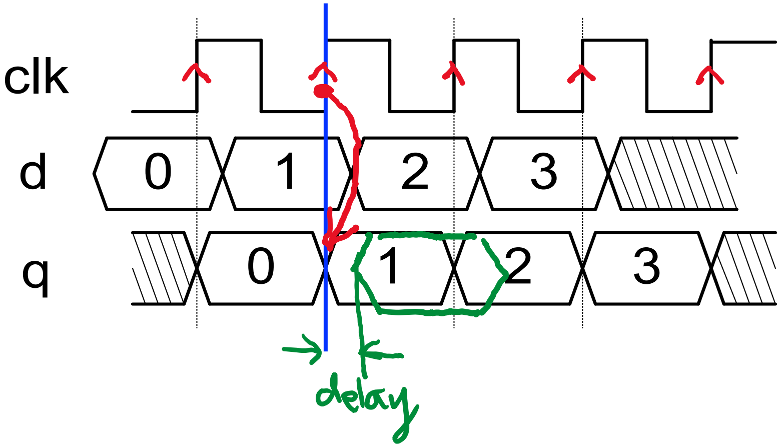
Sequential Assignment (1/4)

- The sensitivity list with edge-triggered condition

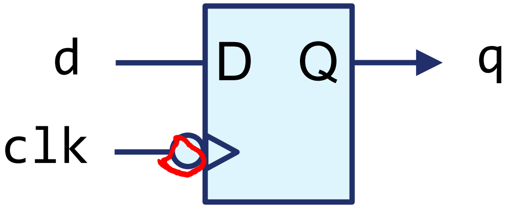
```
// positive edge-triggered DFF
reg [1:0] d, q;
always @(posedge clk) begin
  q <= d;
end
```

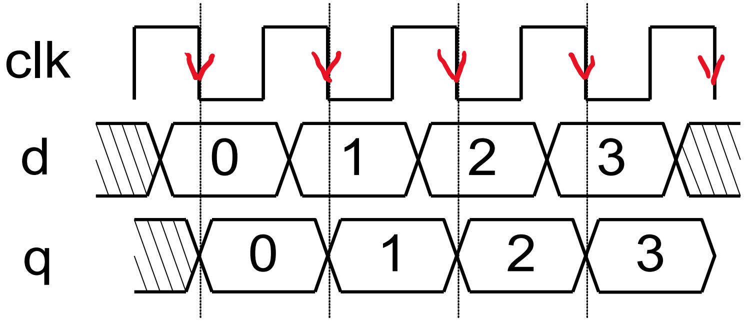


Zero-Delay Model



```
// negative edge-triggered DFF
reg [1:0] d, q;
always @(negedge clk) begin
  q <= d;
end
```





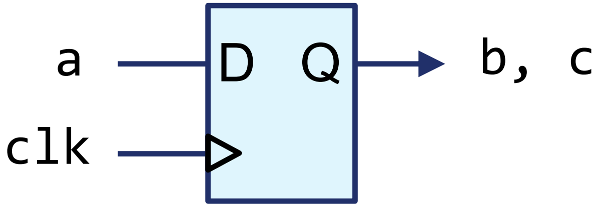
Sequential Assignment (2/4)

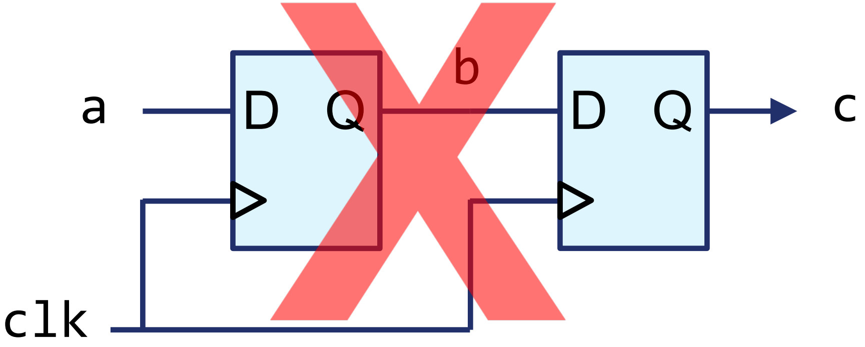
```
always @(posedge clk) begin
  c = b ;
  b = a ;
end
```

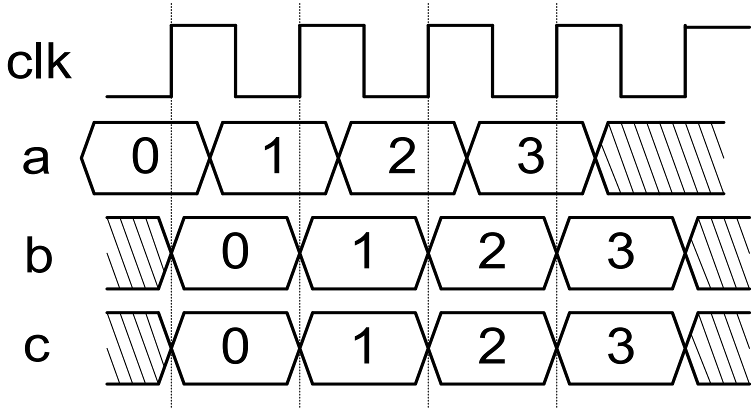
- Blocking assignment
  - Blocking assignments are evaluated serially within a block:

```
always @(posedge clk) begin
  b = a;
  c = b;
end
```

race condition





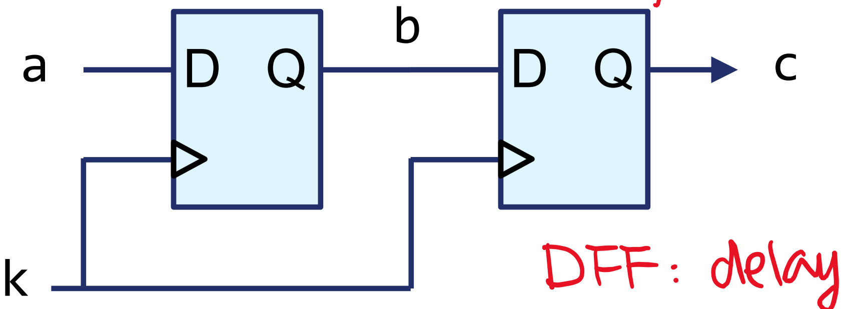


Sequential Assignment (3/4)

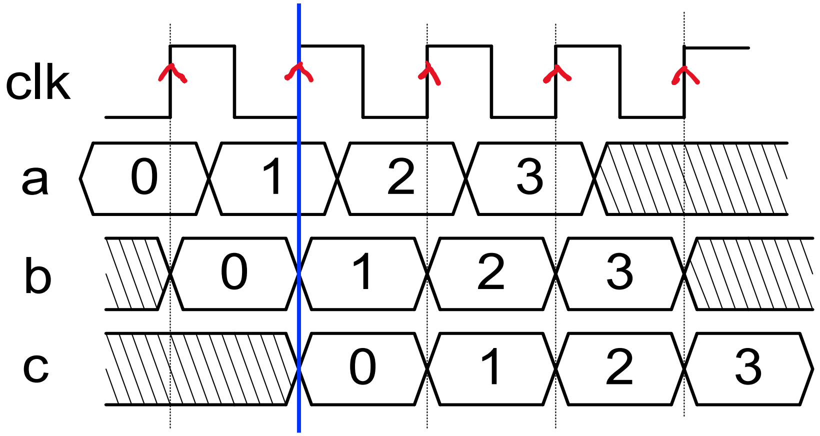
- Non-blocking assignment
  - Non-blocking assignments are evaluated in parallel within a block (hardware behavior)

```
always @(posedge clk) begin
   b <= a;
   c <= b;
end
```

```
tmpa=a;
tmpb=b;
 b=tmpa;
 c=tmpb;
```



DFF: delay element



Sequential Assignment (4/4)

- Flip-flop (register) with synchronous positive reset

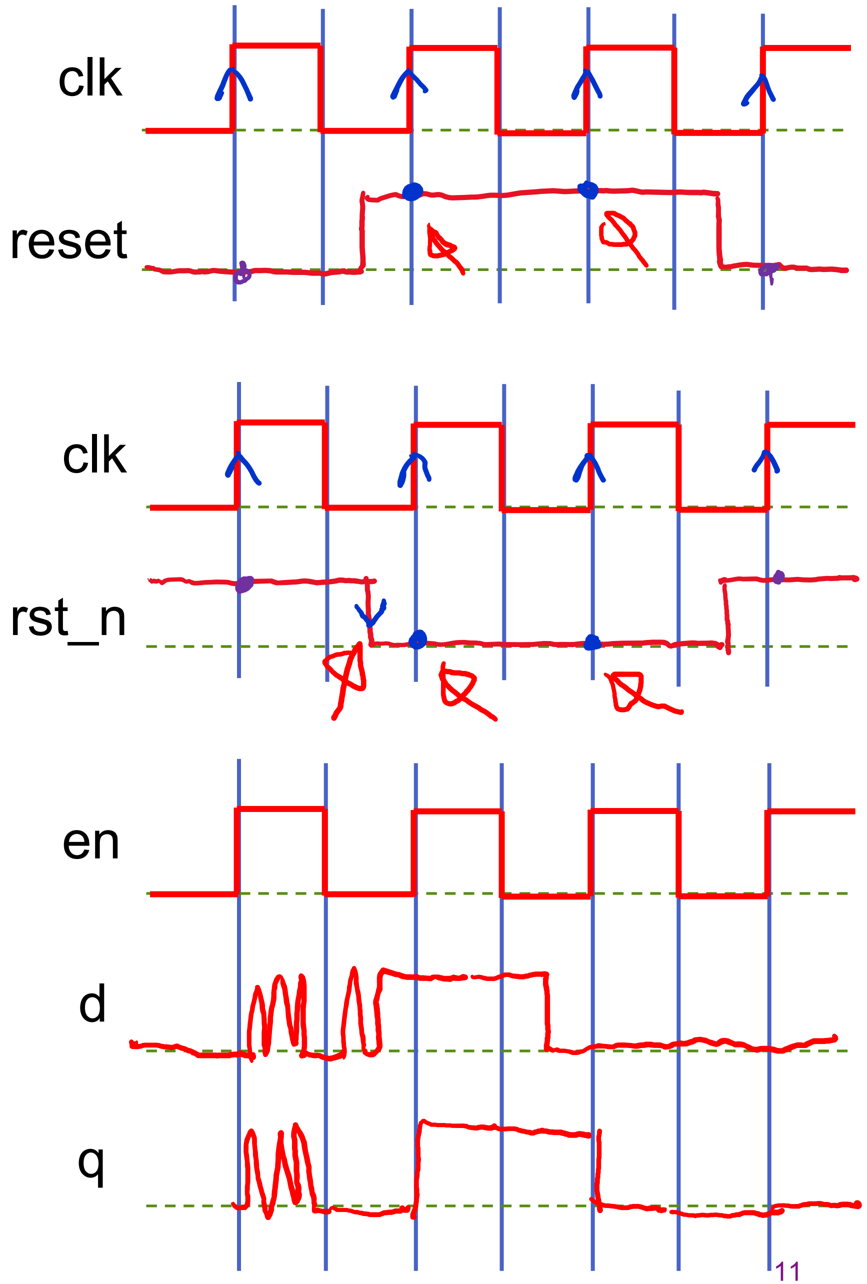
```
always @(posedge clk) begin
   if (reset)  q <= 0
   else  q <= d
end
```

- Flip-flop (register) with asynchronous negative reset

```
always @(posedge clk, negedge rst_n) begin
   if (!rst_n)  q <= 0
   else  q <= d
end
```

- Latch

```
always @(d or en) begin
   if (en)  q = d;
end
```



D Flip-Flop with Synchronous Set and Reset

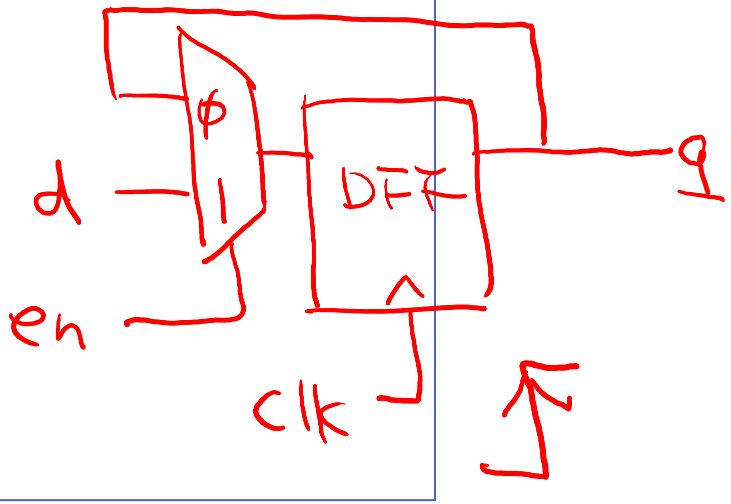
```
module flipflop (q, data_in, clk, set, rst_n);
  input    data_in, clk, set, rst_n;
  output   q;
  reg      q;

  always @ (posedge clk) begin
    if (rst_n == 1'b0)
      q <= 0;
    else if (set == 1'b1)
      q <= 1;
    else
      q <= data_in;
  end
endmodule
```

D Flip-Flop with Asynchronous Set and Reset

```
module flipflop (q, data_in, clk, set_n, rst);
  input   data_in, clk, set_n, rst;
  output  q;
  reg     q;
  always @ (posedge clk, posedge rst, negedge set_n)
    begin
      if (rst == 1'b1)
        q <= 0;
      else if (set_n == 1'b0)
        q <= 1;
      else
        q <= data_in;
    end
endmodule
```

Register with Enable (or Load) and Asynchronous Negative Reset

```
reg [7:0] q;
always @(posedge clk, negedge rst_n) begin
  if (rst_n == 1'b0) begin
    q <= 0;
  end else if (en == 1'b1) begin
    q <= d;
  end
end
```



Sequential Example:
4-bit Sequential Counter

4-bit Counter (1/3)

- Case 1 (putting all together) (counter1.v)

```
module counter (cnt, clk, rst_n);
  output [3:0] cnt;
  input  clk, rst_n;
  reg    [3:0] cnt;
  always @(posedge clk or negedge rst_n) begin
    if (rst_n == 1'b0) begin
      cnt <= 0;
    end else if (cnt == 4'b1111) begin
      cnt <= 0;
    end else begin
      cnt <= cnt + 1'b1;
    end
  end
endmodule
```

boundary condition

cnt

0000
0001
0010
⋮
1110
1111
1 0000

4-bit Counter (2/3)

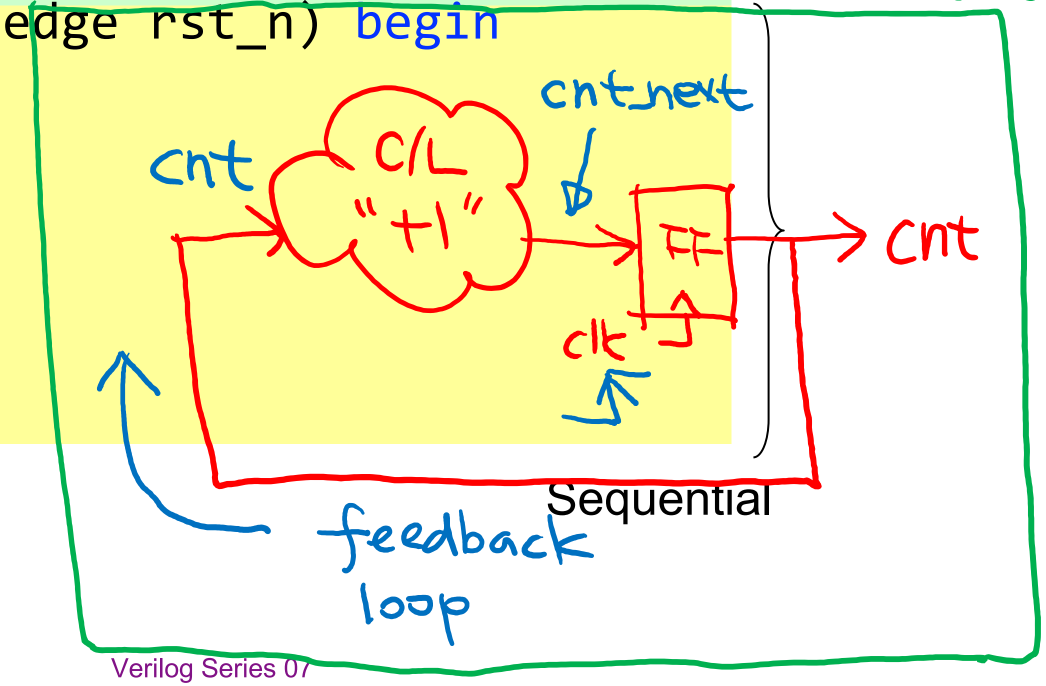
- Case 2 (separate combinational and sequential parts) (`counter2.v`)

```
module counter (cnt, clk, rst_n);
  output [3:0] cnt;
  input  clk, rst_n;
  reg    [3:0] cnt;
  wire   [3:0] cnt_next;
  assign cnt_next = (cnt == 4'b1111) ? 0 : cnt + 1'b1;
  always @(posedge clk or negedge rst_n) begin
    if (rst_n == 1'b0) begin
      cnt <= 0;
    end else begin
      cnt <= cnt_next;
    end
  end
endmodule
```

Combinational

always@(posedge clk, ...)

cnt<=cnt+1;



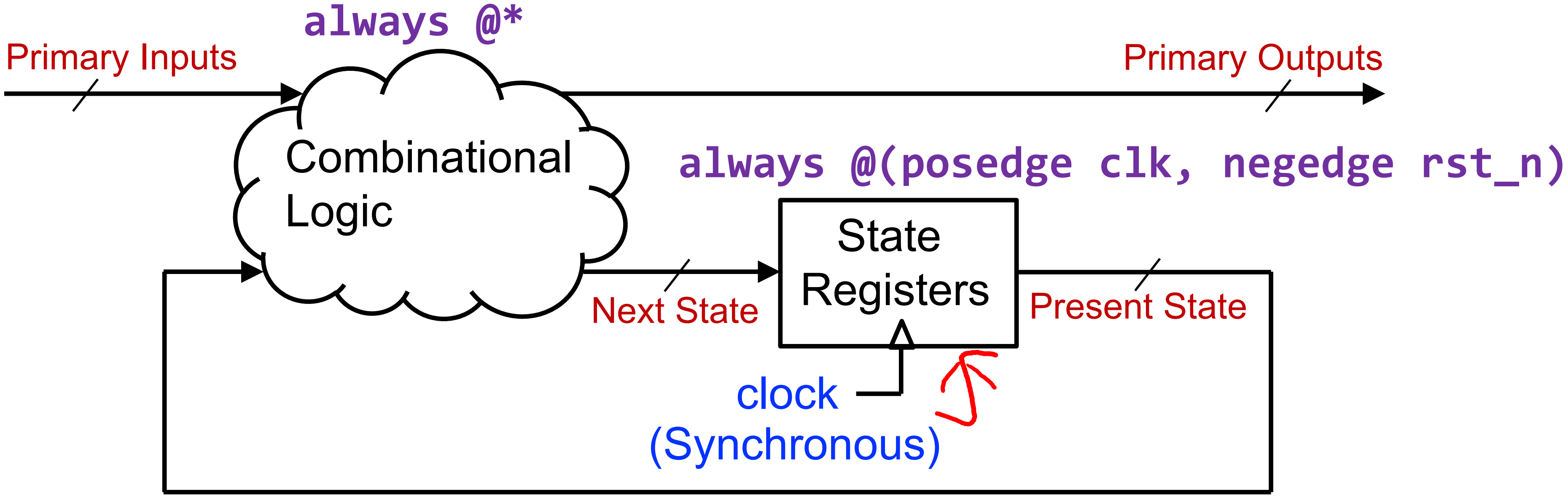
Sequential

4-bit Counter (3/3)

- Case 3 (the alternative combinational description) (`counter3.v`)

```
reg [3:0] cnt_next;
always @(*) begin
  if (cnt == 4'b1111) begin
    cnt_next = 0;
  end else begin
    cnt_next = cnt + 1'b1;
  end
end
```

Clocked Synchronous Sequential Circuits



Stimulus (Testbench)

- **Example** (counter_test.v)

```
`timescale 1ns/100ps
module stimulus;
  reg  clk, rst_n;
  wire [3:0] cnt;
  counter CNT1(cnt, clk, rst_n);
  always #10 clk = ~clk;
  always @(posedge clk) $display("time=%d cnt=%b", $time, cnt);
  initial begin
    clk = 0;
    rst_n = 1;
    #2  rst_n = 0;
    #20 rst_n = 1;
    #400;
    $finish;
  end
endmodule
```

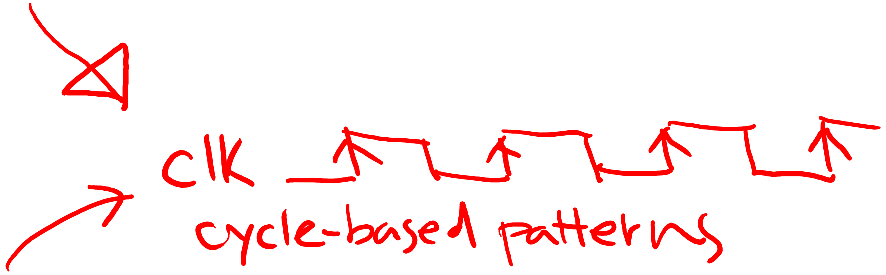
```
initial $monitor($time, "cnt=%b", cnt);
```

For graphical waveform

```
initial begin
  $fsdbDumpfile("stimulus.fsdb");
  $fsdbDumpvars;
end
```

Simulation Related Tasks

- To terminate simulation at a specific time
  - $finish;



- Text-based output
  - $display

```
always @(posedge clk)
    $display("time=%d clk=%b cnt=%d", $time, clk, cnt);
```

  - $monitor

```
initial
    $monitor($time, " clk=%b cnt=%d", clk, cnt);
```

  - What’s the difference?

Summary

Verilog Overview

- Part 1
  - Background
  - Digital Switches and Logic Gates
  - Tri-state Gates
  - Structural Modeling
  - Delay Model
- Part 2
  - Numbers
  - Data Types
  - Operators
  - Compiler Directives
  - Dataflow Modeling
- Part 3
  - Behavioral Modeling
  - Procedural Timing Controls
  - Review of Basic Module Structure
  - Test Stimulus
  - Tasks and Functions
- Part 4
  - Combinational Blocks
  - Combinational Examples
  - Module Instantiation
  - Hierarchical Design Style
- Part 5
  - Sequential Blocks
  - Sequential Example:
    4-bit Sequential Counter

Summary for Verilog HDL Modeling

- Design with block diagrams before the Verilog coding
- Verilog coding philosophy is not the same as C programming
- Every Verilog RTL construct has its own logic mapping (for synthesis)
  - Combinational blocks:
    - `assign`
    - `always @*`
  - Sequential blocks:
    - `always @(posedge clk)`
- We compose synthesizable Verilog RTL code for digital designs
- We compose testbenches to verify the designs
  - Testbenches are non-synthesizable