Computer Architecture
Tutorial 5

Sequential Logic Circuits: Implementation in Verilog HDL

Artem Burmyakov, Muhammad Fahim, Alexander Tormasov

October 1, 2020

INNOPOLIS UNIVERSITY

Content of the Tutorial

- Byte Order
  - Little endian
  - Big endian
- (Today's Topic) Sequential Logic
- HDL Implementation
  - SR-Latch
- State Diagram

Byte Order

- Some systems (e.g. Intel) use little endian byte order
  - Least significant byte of a multi-byte entity is stored at lowest memory address

The int 5 at address 1000:

1000	00000101
1001	00000000
1002	00000000
1003	00000000

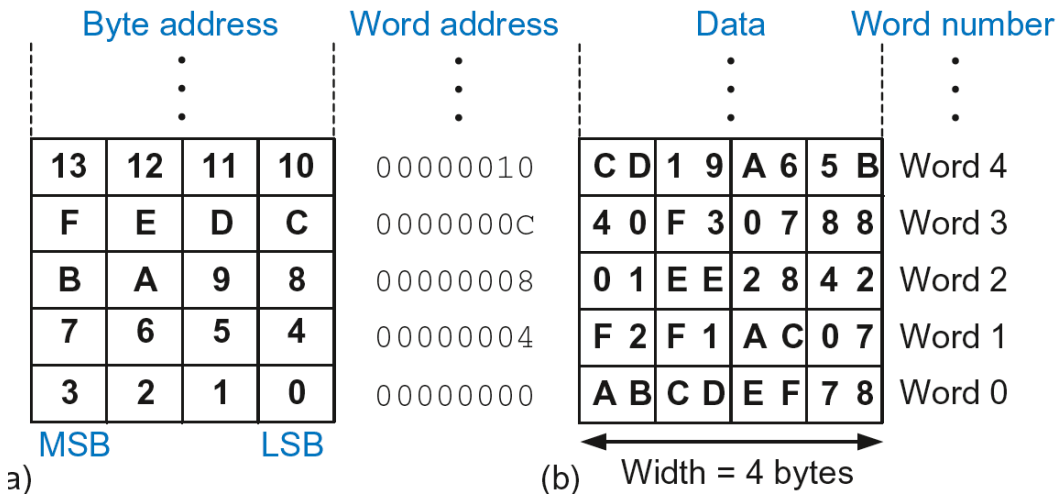
- Some systems (e.g. SPARC) use big endian byte order
  - Most significant byte of a multi-byte entity is stored at lowest memory address

The int 5 at address 1000:

1000	00000000
1001	00000000
1002	00000000
1003	00000101

Byte-Addressable Memory

- Each data **byte** has unique address
- 32-bit word = 4 bytes, so word address increments by 4

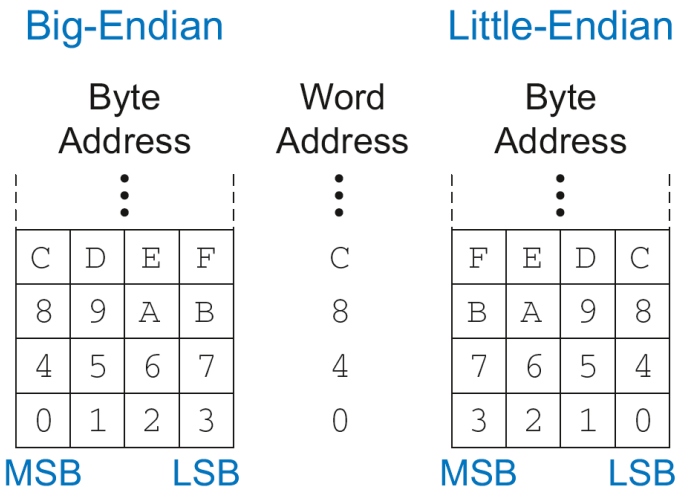


Big-Endian & Little-Endian Memory

- **How to number bytes within a word?**

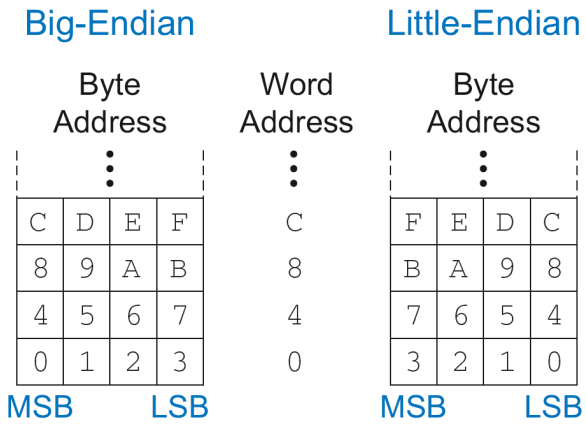
Big-Endian & Little-Endian Memory

- **How to number bytes within a word?**
  - **Little-endian:** byte numbers start at the **little** (least significant) end
  - **Big-endian:** byte numbers start at the **big** (most significant) end



Big-Endian & Little-Endian Memory

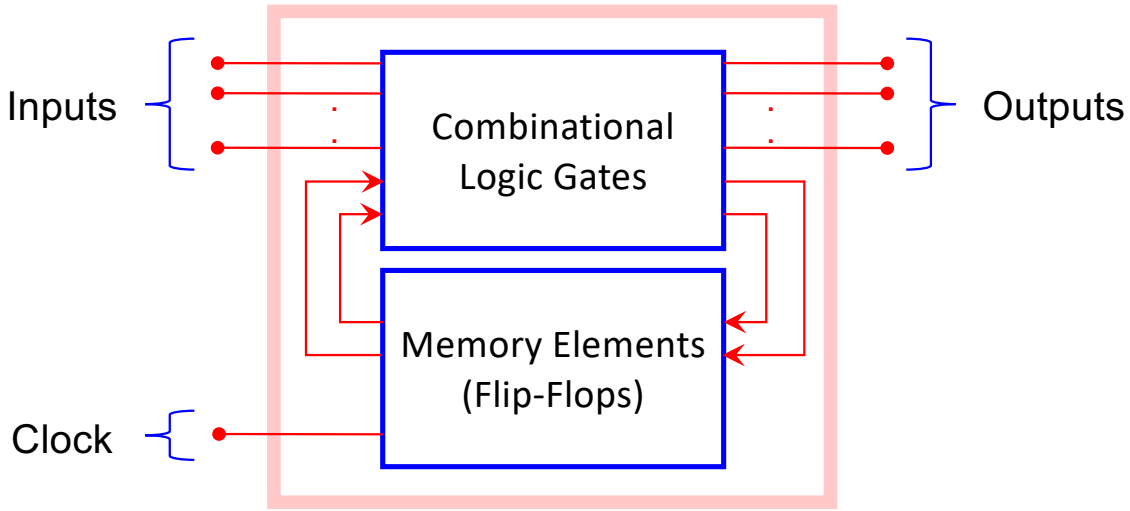
- **Jonathan Swift's *Gulliver's Travels*:** the Little-Endians broke their eggs on the little end of the egg and the Big-Endians broke their eggs on the big end
- **It doesn't really matter** which addressing type used – **except** when two systems **share data**



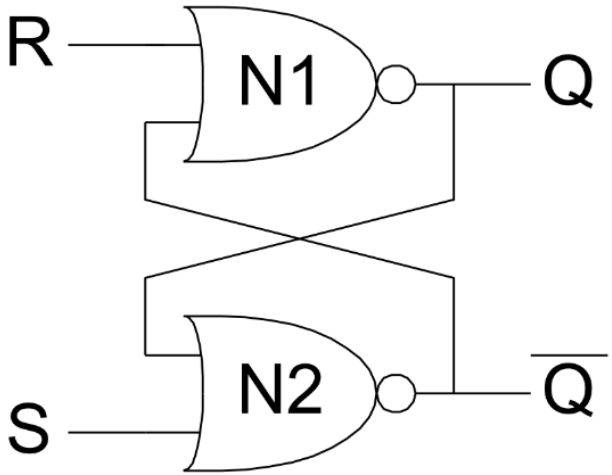
BIG ENDIAN - The way people always broke their eggs in the Lilliput land

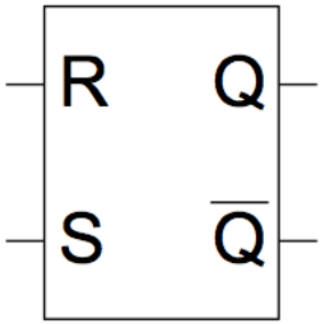
LITTLE ENDIAN - The way the king then ordered the people to break their eggs

Sequential Logic



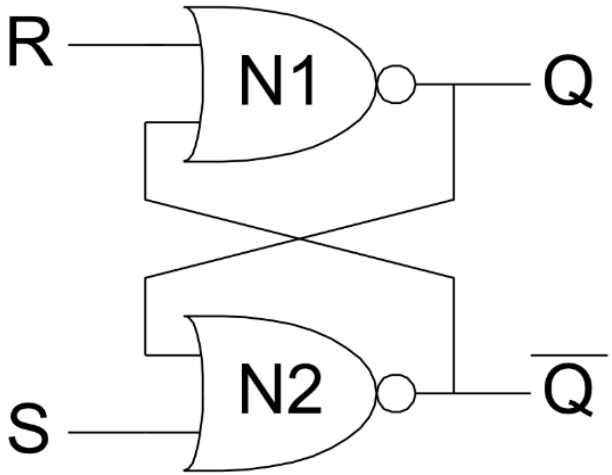
What kind of circuit?

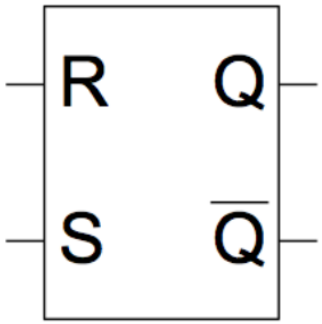




S	R	Q
0	0	
0	1	
1	0	
1	1	

What kind of circuit?

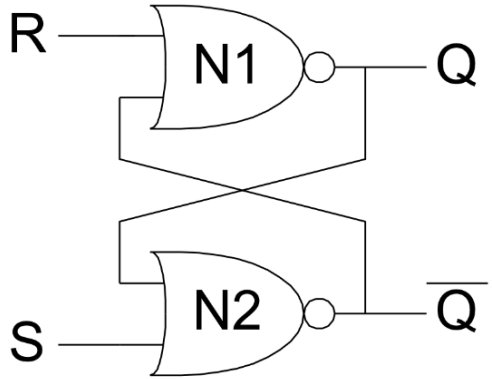




S	R	Q
0	0	hold
0	1	0
1	0	1
1	1	(Invalid)

SR Latch – Verilog HDL

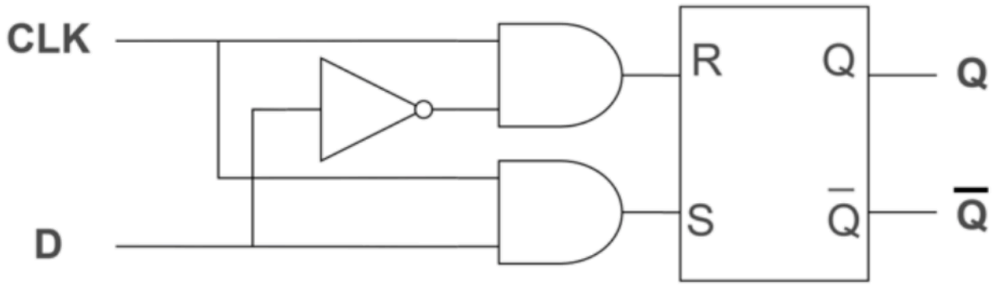
```
module sr_latch
 (
      input   s,
      input   r,
      output  q,
      output  q_n
 );
      assign q   = ~ ( r | q_n );
      assign q_n = ~ ( s | q   );

 endmodule
```



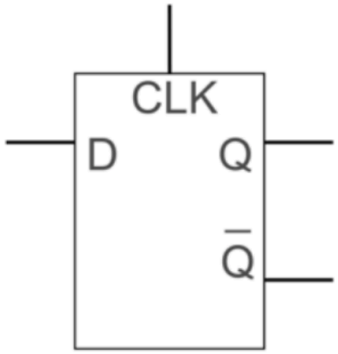
SR Latch – Test Bench

```
module testbench;

    reg  s, r;
    wire q, q_n;
    sr_latch sr_latch (s, r, q, q_n);
    initial $dumpvars;
    initial
    begin
       $monitor ("%0d s %b r %b q %b q_n %b", $time, s, r, q, q_n);
        # 10;    s = 0; r = 0;
        # 10;    s = 1; r = 0;
        # 10;    s = 0; r = 0;
        # 10;    s = 0; r = 1;
        # 10;    s = 0; r = 0;
        # 10;    s = 1; r = 1;
        # 10;    s = 0; r = 0;
        # 10;    s = 0; r = 0;
        # 10;
        $finish;
    end
endmodule
```

D Latch



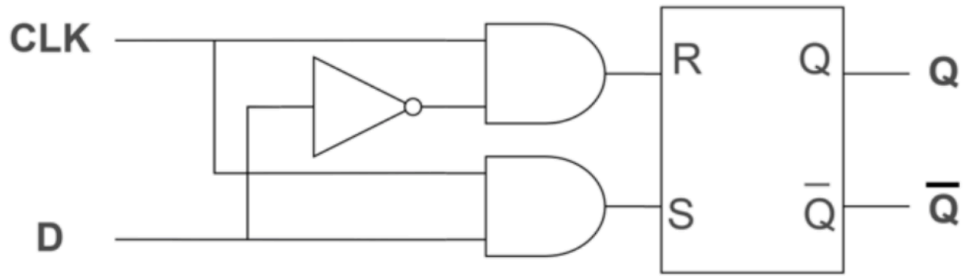
D-latch schematic



D-latch symbol

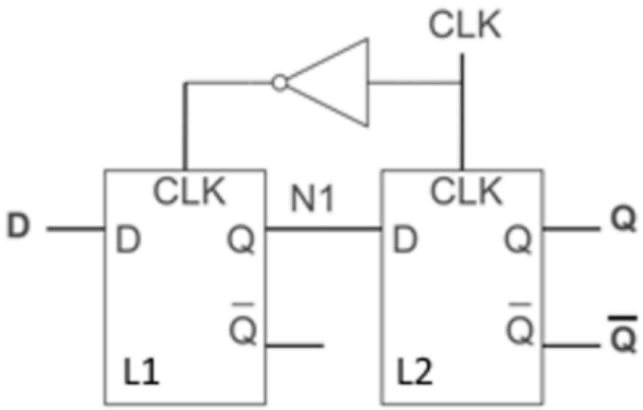
D Latch – Verilog HDL

```
module d_latch
 (
      input   clk,
      input   d,
      output  q,
      output  q_n
 );
      wire r = ~d & clk;
      wire s = d & clk;

      sr_latch sr_latch (s, r, q, q_n);
 endmodule
```

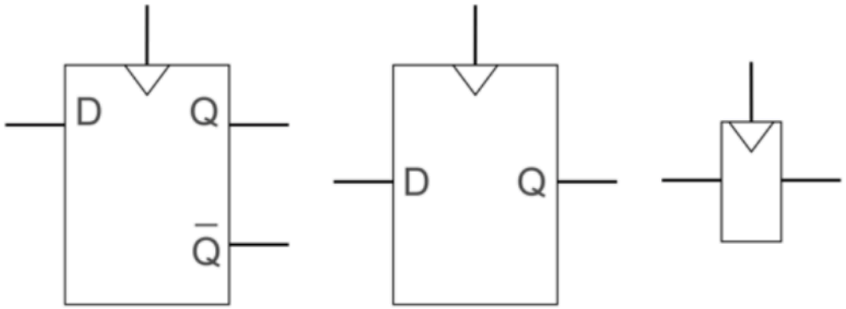


D-latch schematic

D Flip-flop schematic



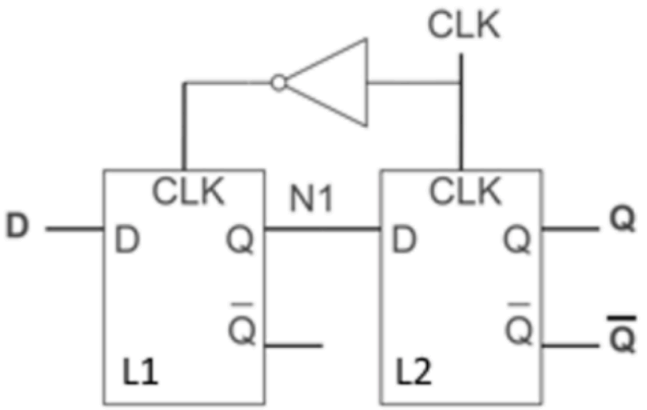
D flip-flop schematic



D flip-flop symbol

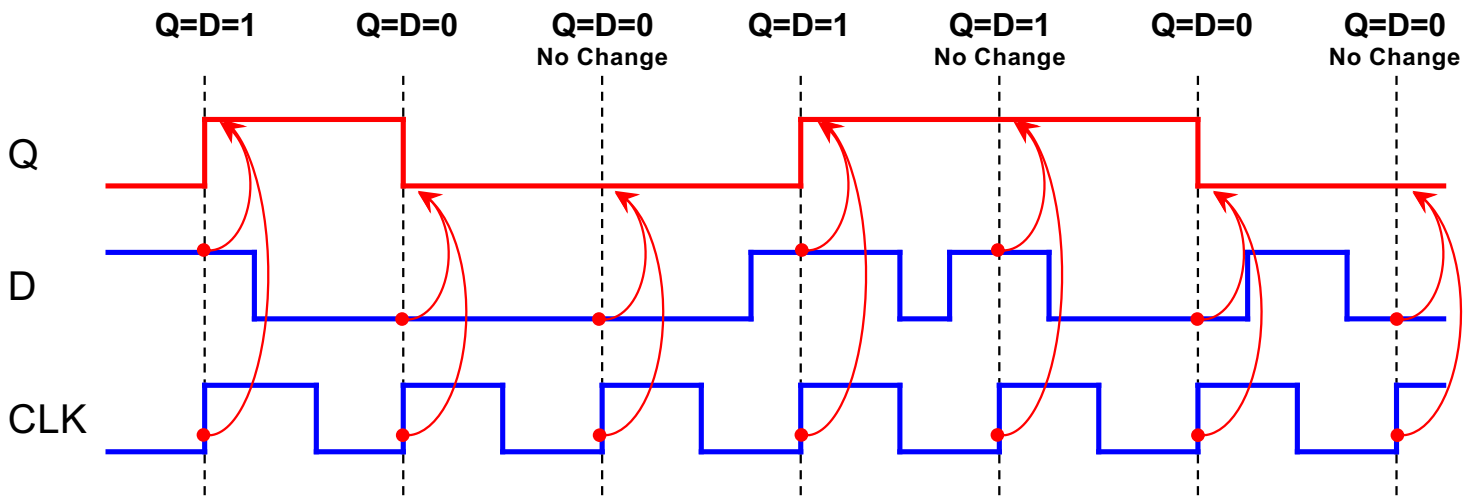
D Flip-flop – Verilog HDL

```
module d_flip_flop (
     input   clk,
     input   d,
     output  q,
     output  q_n
 );
     wire n1;

     d_latch master (
          .clk ( ~clk ),
          .d   (  d   ),
          .q   (  n1  )
     );

     d_latch slave (
          .clk ( clk ),
          .d   ( n1  ),
          .q   ( q   ),
          .q_n ( q_n )
     );
  endmodule
```



D flip-flop schematic

D Flip-Flop: Example Timing



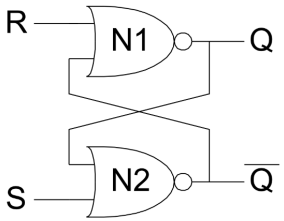
State Diagrams

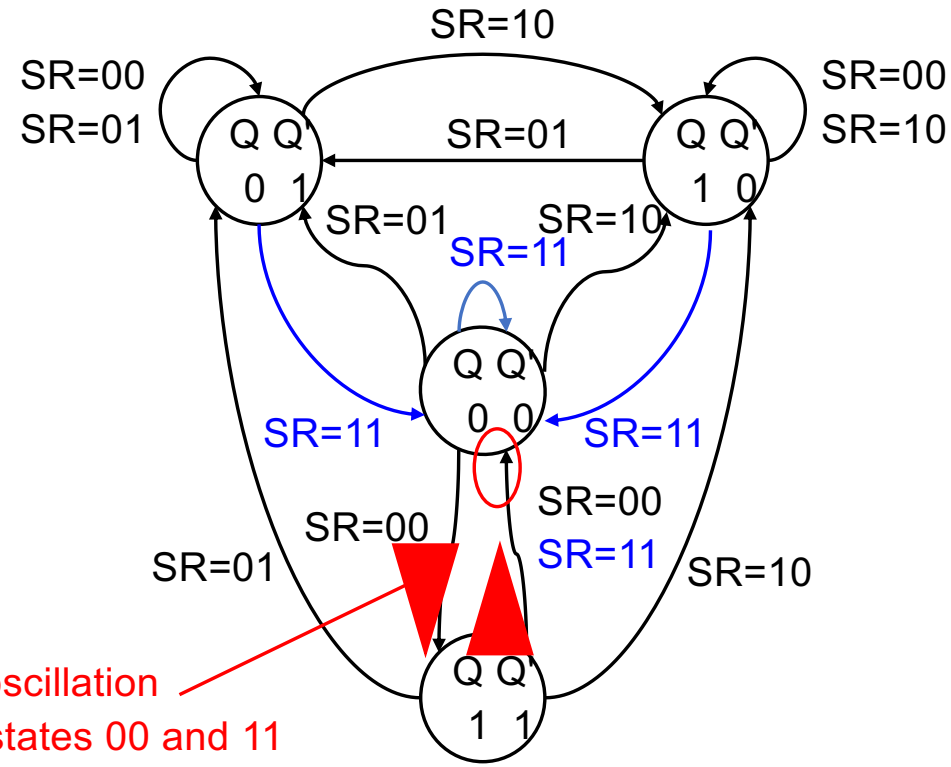
- How do we characterize logic circuits?
  - Combinational circuits: Truth tables
  - Sequential circuits: State diagrams
- First draw the states
  - States ≡ Unique circuit configurations
- Second draw the transitions between states
  - Transitions ≡ Changes in state caused by inputs

Example: SR Latch

- Begin by drawing the states
  - States ≡ Unique circuit configurations
  - Possible values for feedback (Q, Q')

S	R	Q
0	0	hold
0	1	0
1	0	1
1	1	disallow





Thank You ☺