Flip-Flops

Outline:

2. *Timing noise*

   → Signal races, glitches

   → FPGA example (“assign” → bad)

- *Synchronous circuits and memory*

   → Logic gate example

4. *Flip-Flop memory*

   → RS-latch example

- *D and JK flip-flops*

   → Flip-flops in FPGAs

- *Synchronous circuit design with FPGAs*

   → FPGA example (“always” → good).

   → Parallel circuit design with FPGAs.

Timing noise

Amplitude Noise

A digital circuit is very immune to amplitude noise, since it can only have two values (Low or High, True or False, 0 or 1). Digital electronics circuits typically have error rates smaller than 1 part in 10^9 (no error correction).

Timing Noise

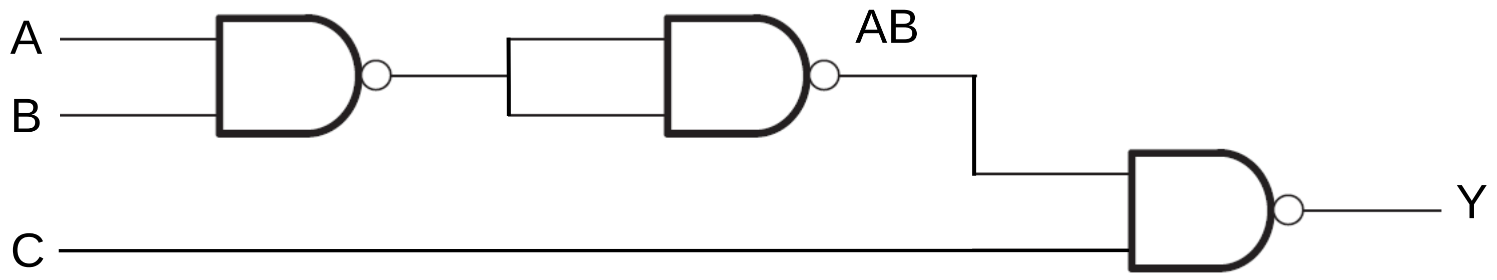
Just like an analog circuit, a digital circuit can experience timing noise. Fortunately, good clocks are cheap and easily available, and a good design will eliminate the effects of timing noise.

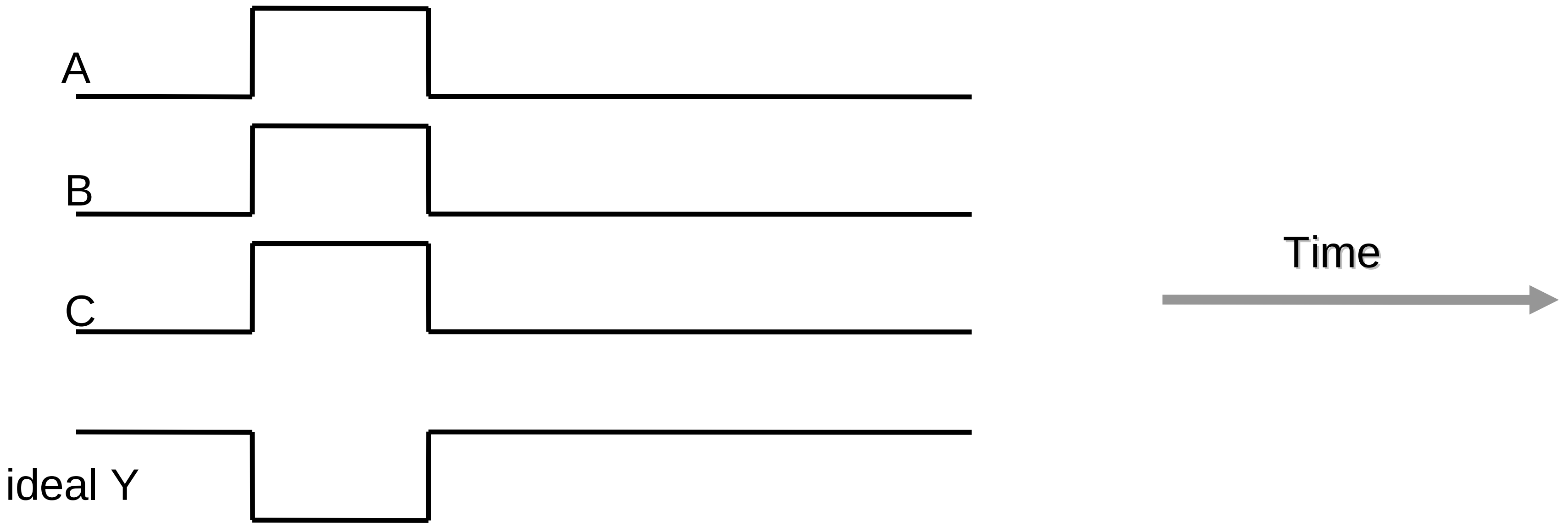
Timing issues/errors can easily produce amplitude noise (bit errors).

Signal Race

The timing delays produced by wires and logic gates can produce unwanted (illogical) outputs.

Example: 3-input NAND gate

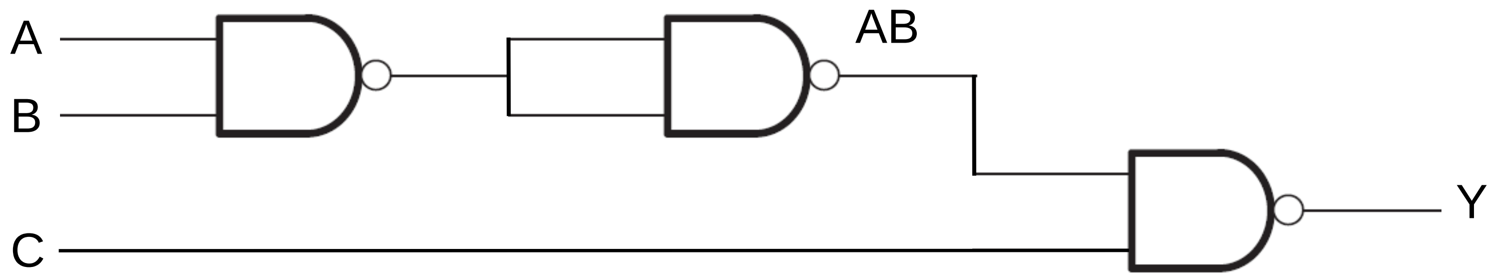


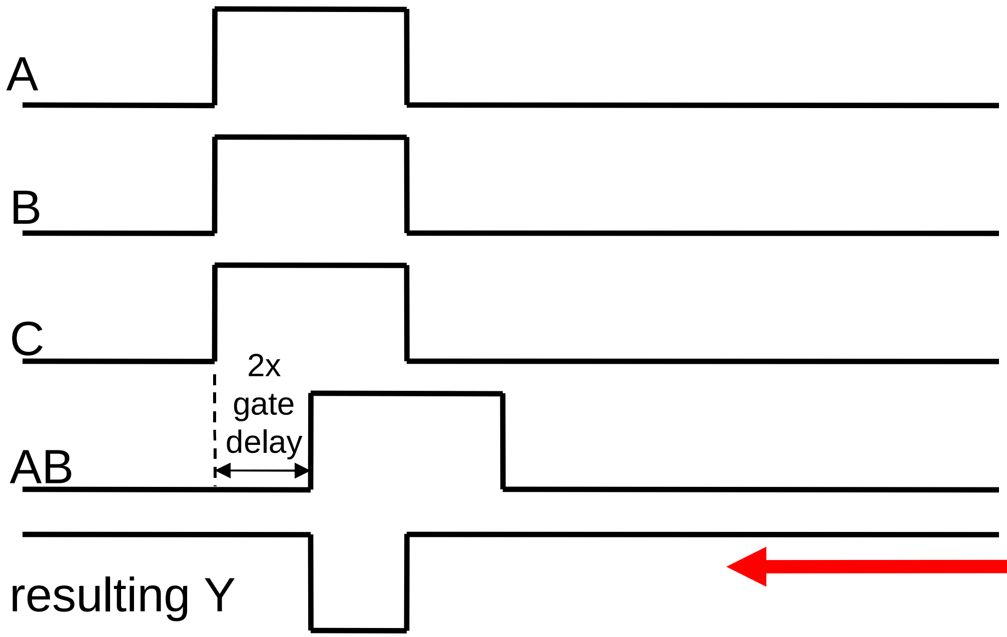


Signal Race

The timing delays produced by wires and logic gates can produce unwanted (illogical) outputs.

Example: 3-input NAND gate



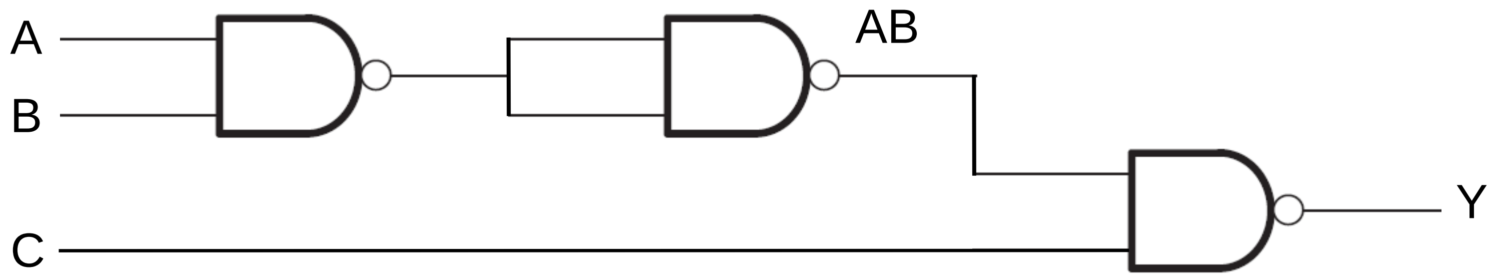


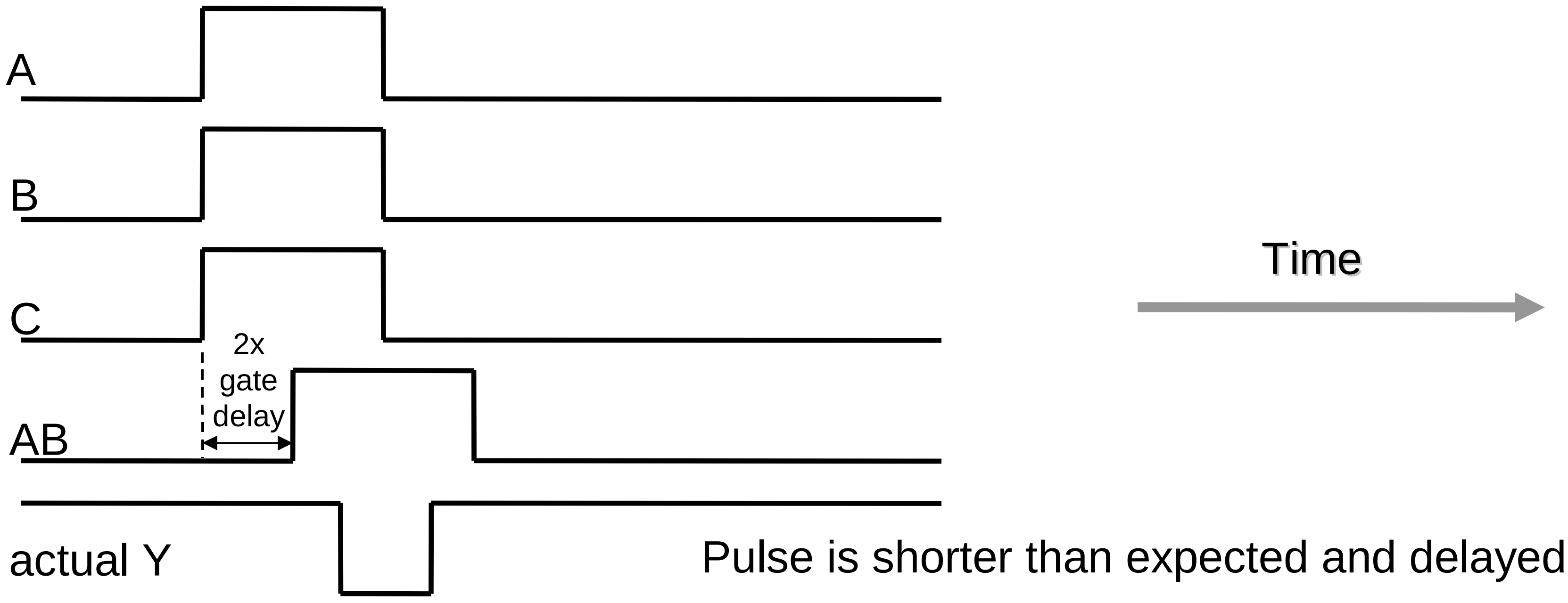
If gate delays are too long output pulse could disappear

Signal Race

The timing delays produced by wires and logic gates can produce unwanted (illogical) outputs.

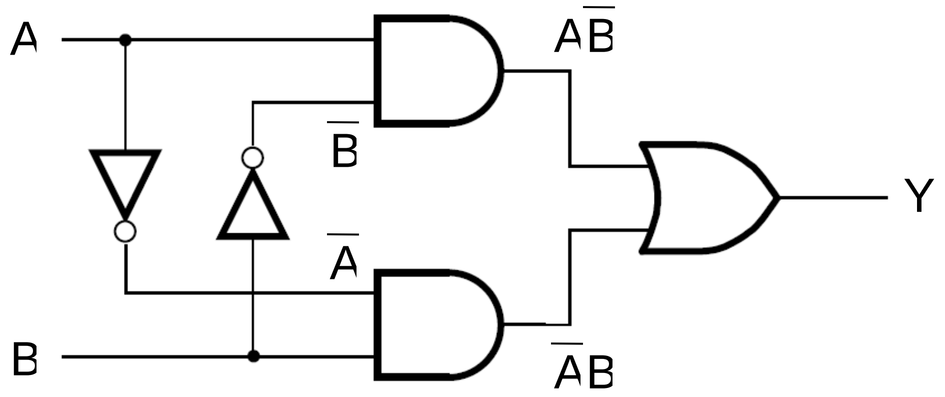
Example: 3-input NAND gate





Pulse is shorter than expected and delayed

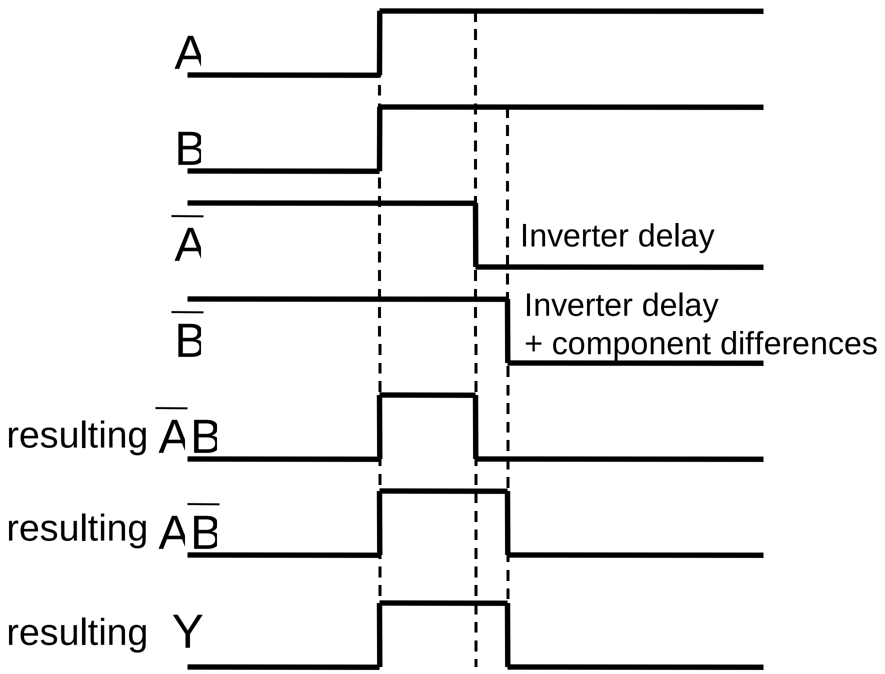
Signal Race with Glitch



[diagram courtesy of Altera Inc.]

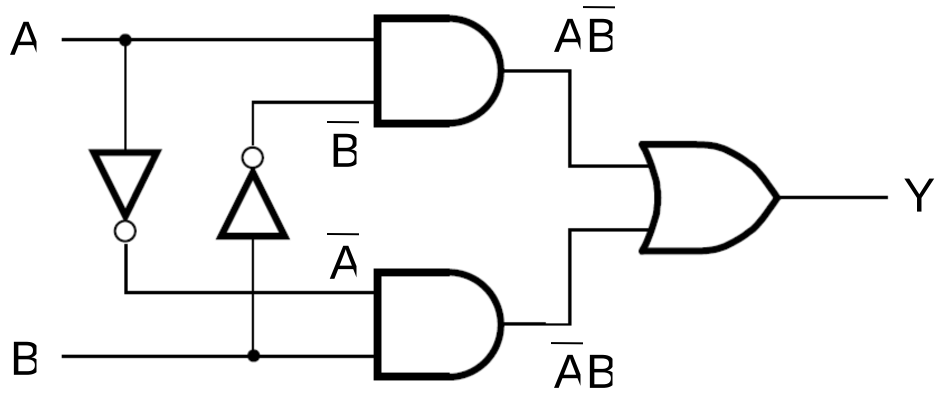
XOR

A	B	Y
L	L	L
L	H	H
H	L	H
H	H	L



[Figure adapted from ***Principles of Electronics: Analog & Digital*** by L. R. Fortney]

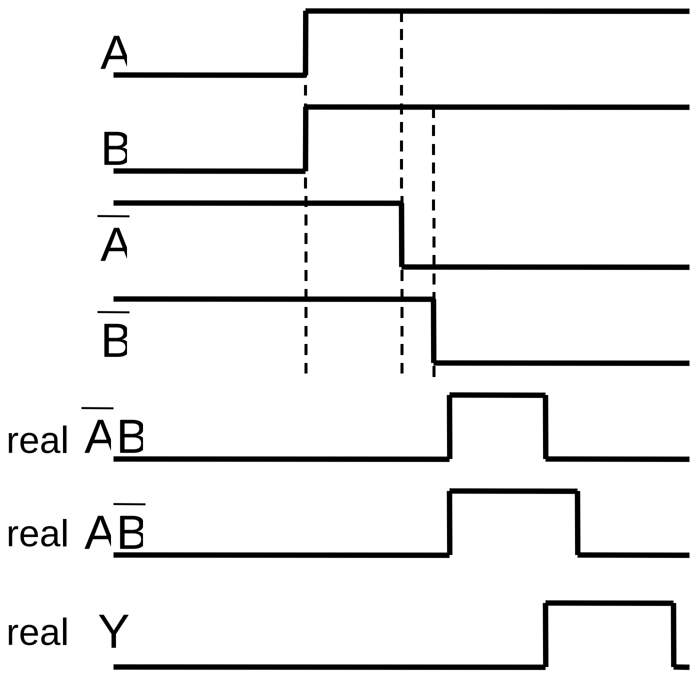
Signal Race with Glitch



[diagram courtesy of Altera Inc.]

XOR

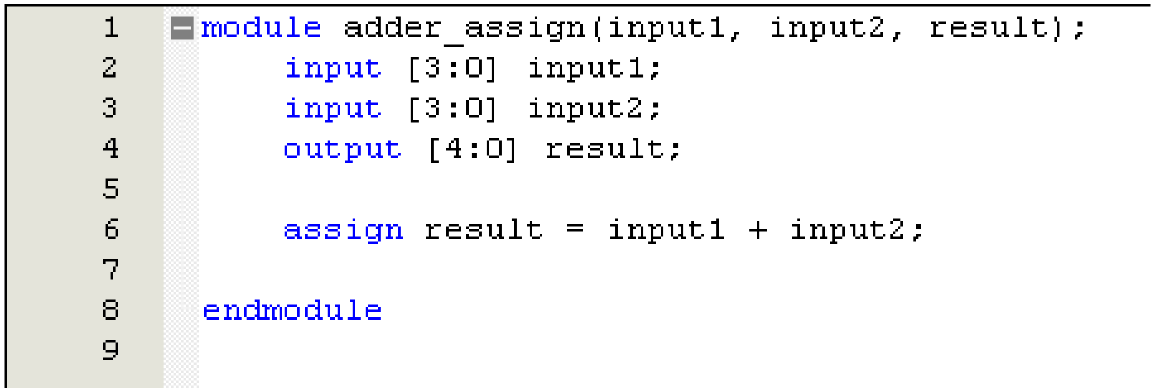
A	B	Y
L	L	L
L	H	H
H	L	H
H	H	L



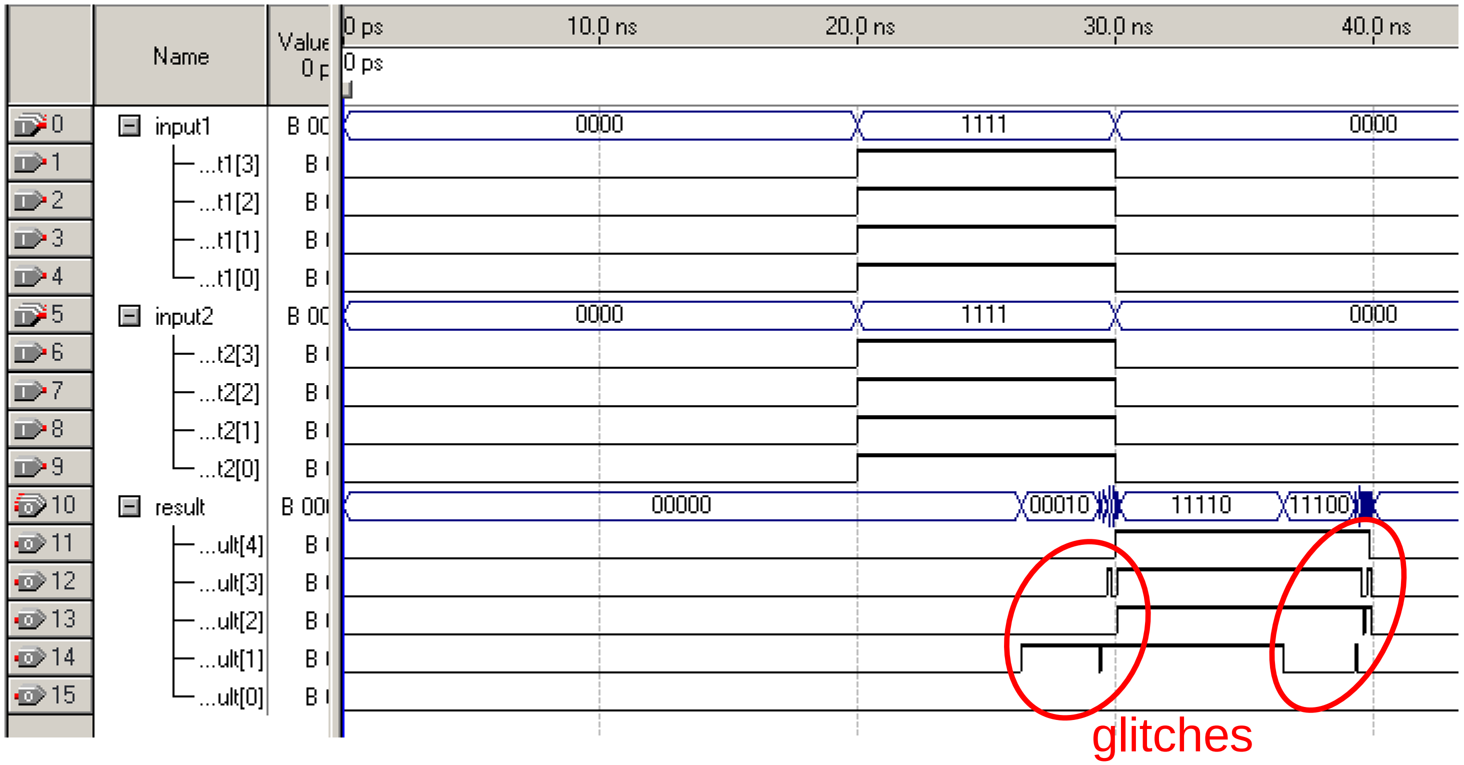
[Figure adapted from ***Principles of Electronics: Analog & Digital*** by L. R. Fortney]

Glitches with FPGAs

Quartus II will simulate glitches



```
module adder_assign(input1, input2, result);
    input [3:0] input1;
    input [3:0] input2;
    output [4:0] result;

    assign result = input1 + input2;

endmodule
```



Asynchronous Design

Asynchronous design requires very careful attention to signal delays to avoid producing glitches and other spurious signals.

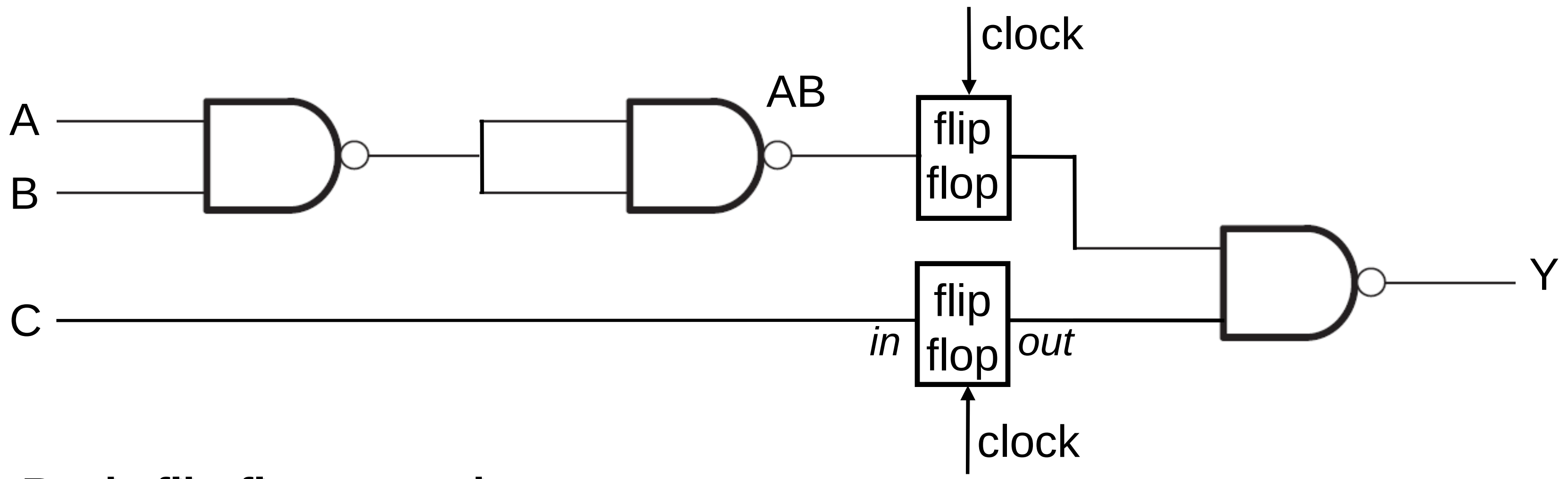
Glitches will produce false data and can produce very wrong results

e.g. a glitch on the most-significant-bit will produce a factor of 2 error.

Asynchronous design can produce very fast digital circuits, but is generally avoided due to more difficult design.

Synchronous Design

The use of **memory** and a **clock** can eliminate signal races and glitches.



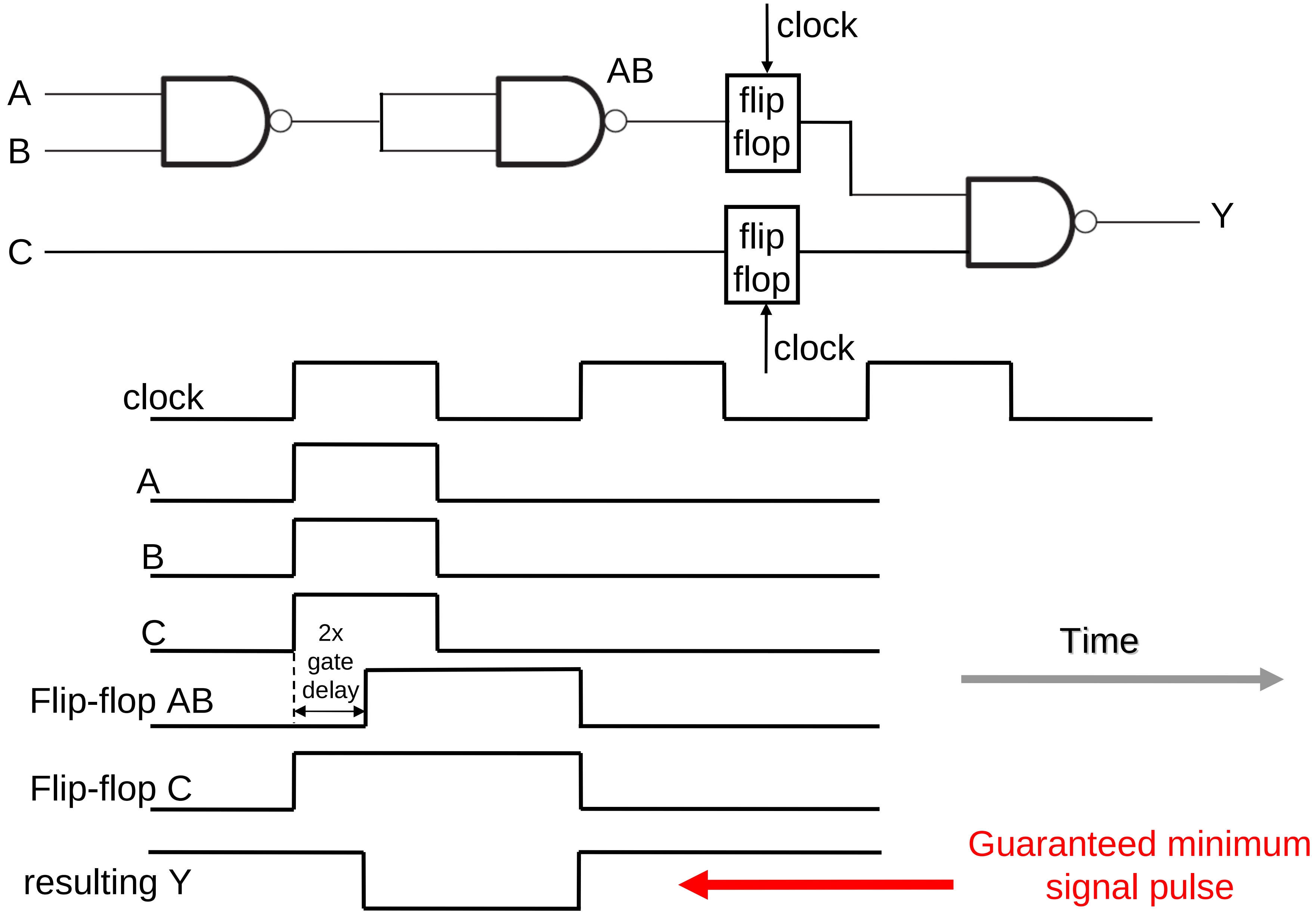
Basic flip-flop operation

The flip-flop will record and output the value at the input if the **clock** is HIGH. If the **clock** goes LOW, then the flip-flop does not change its value or output.

Glitches are eliminated if

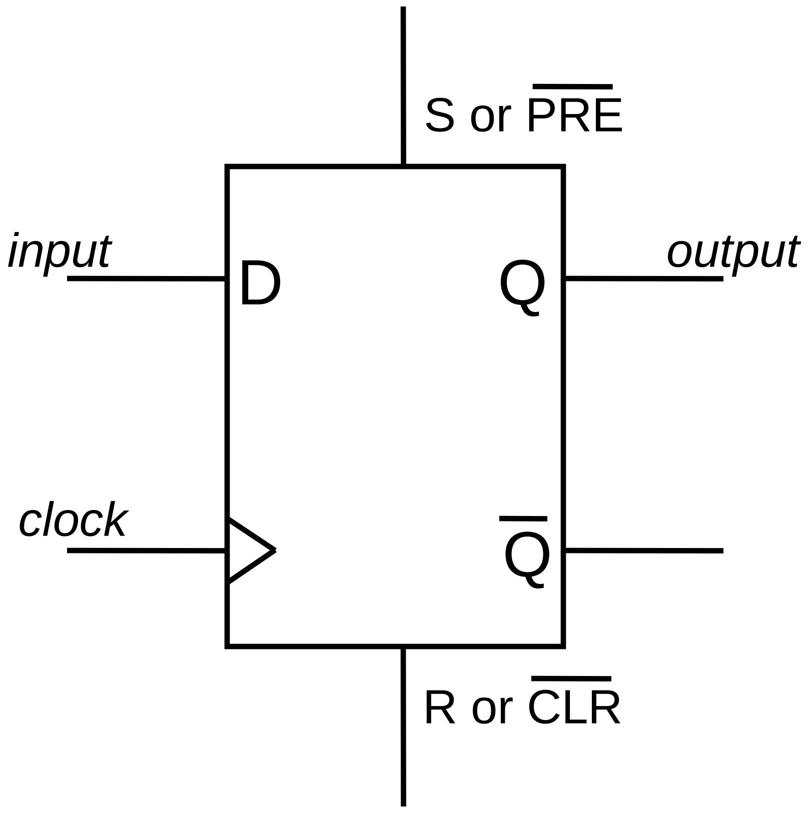
1. The clock HIGH and LOW times are longer than any gate delays.
2. The inputs are synchronized to the clock.

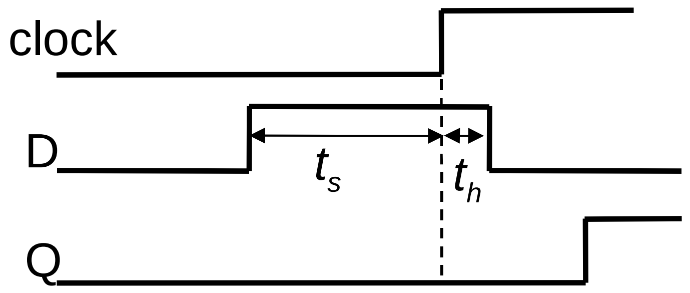
Synchronous Timing



D-type Edge-Triggered Flip-Flop

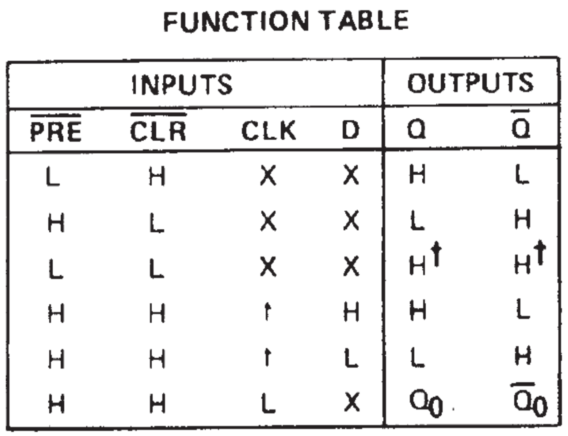
Generally, the flip-flop changes state on a clock signal “edge”, not the level. The flip-flop takes the value *just before* the clock “edge”.





For 74LS74: minimum t_s = *20 ns*

minimum t_h = *5 ns*



FUNCTION TABLE

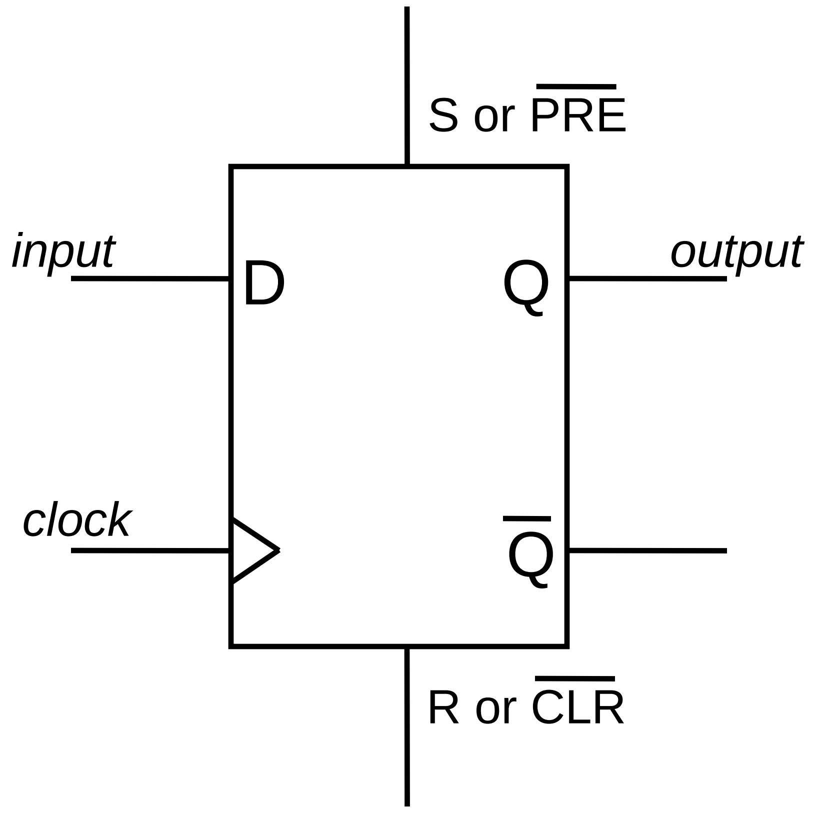
INPUTS				OUTPUTS	
$\overline{\text{PRE}}$	$\overline{\text{CLR}}$	CLK	D	Q	$\overline{\text{Q}}$
L	H	X	X	H	L
H	L	X	X	L	H
L	L	X	X	H†	H†
H	H	↑	H	H	L
H	H	↑	L	L	H
H	H	L	X	Q_0	$\overline{Q}_0$

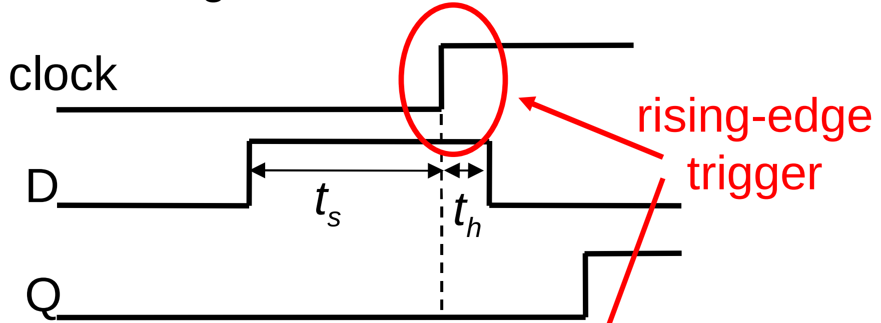
[Texas Instruments 74LS74 flip-flop datasheet]

Note: A flip-flop saves information (i.e. 1 bit); it does not modify it.

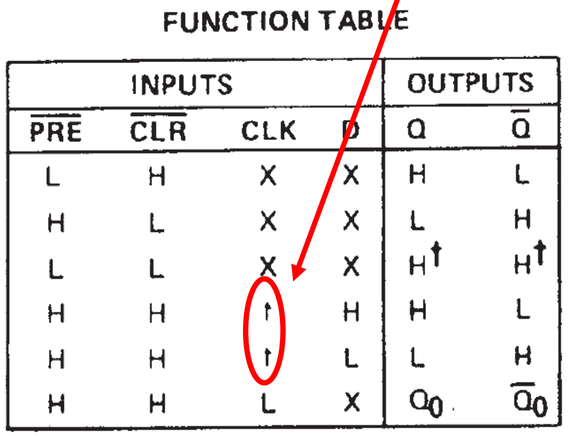
D-type Edge-Triggered Flip-Flop

Generally, the flip-flop changes state on a clock signal “edge”, not the level. The flip-flop takes the value *just before* the clock “edge”.





For 74LS74: minimum t_s = *20 ns*
minimum t_h = *5 ns*



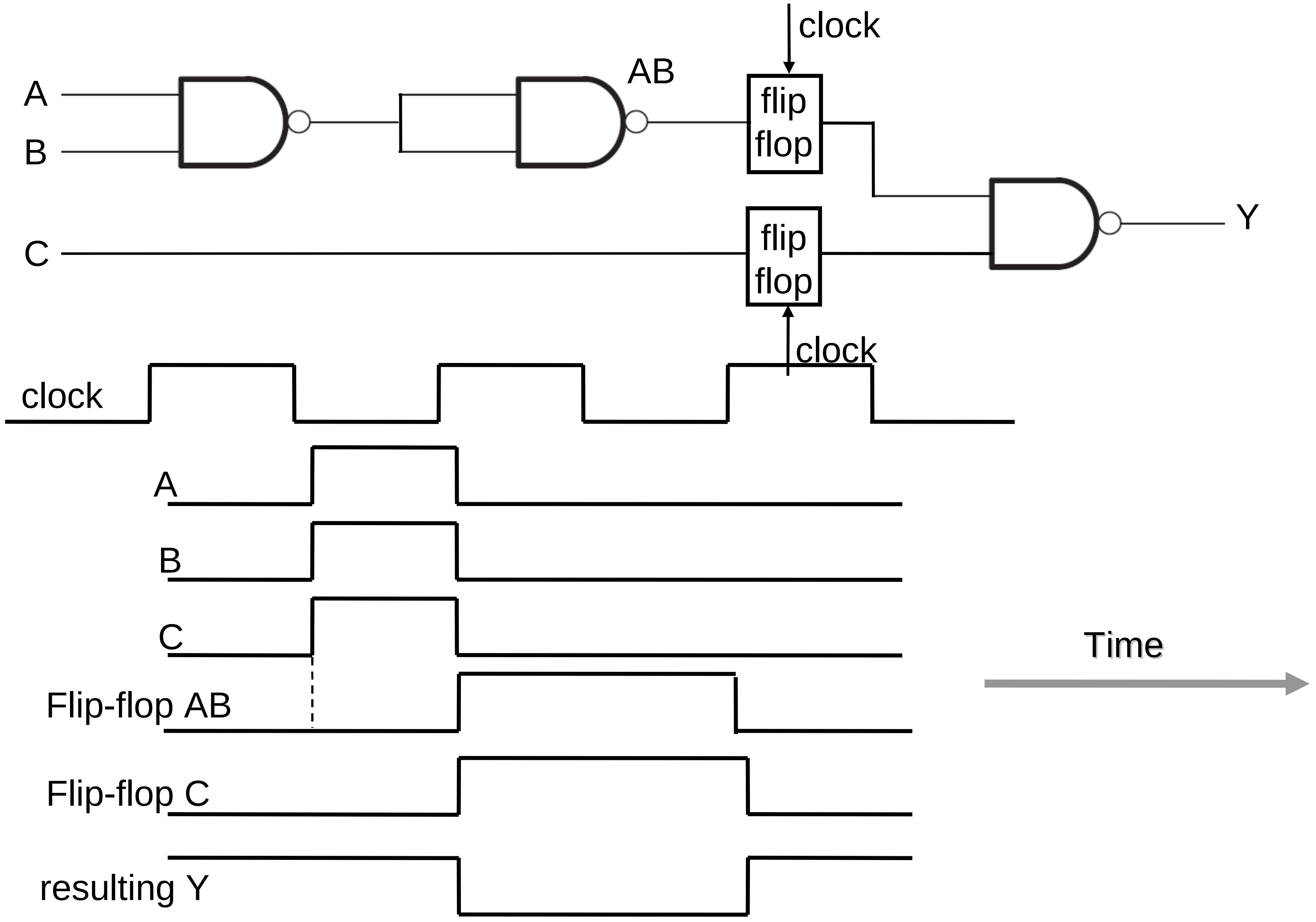
FUNCTION TABLE

INPUTS				OUTPUTS	
$\overline{\text{PRE}}$	$\overline{\text{CLR}}$	CLK	D	Q	$\overline{\text{Q}}$
L	H	X	X	H	L
H	L	X	X	L	H
L	L	X	X	H†	H†
H	H	↑	H	H	L
H	H	↑	L	L	H
H	H	L	X	Q_0	$\overline{Q}_0$

[Texas Instruments 74LS74 flip-flop datasheet]

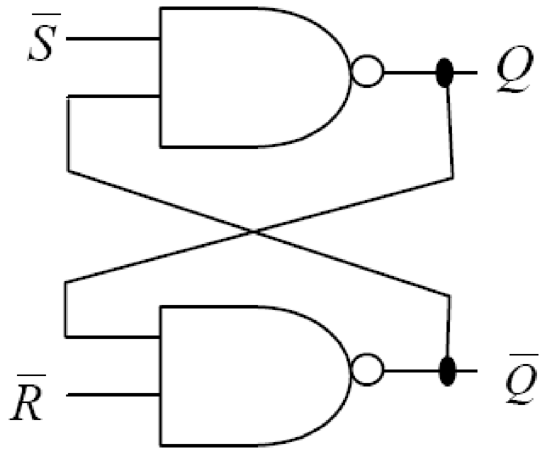
Note: A flip-flop saves information (i.e. 1 bit); it does not modify it.

Synchronous Timing (revisited)



How does a flip-flop work?

Basic flip-flop: the SR latch



Logic table

$\overline{S}$	$\overline{R}$	Q	$\overline{Q}$
1	1	Q_0	$\overline{Q}_0$
0	1	1	0
1	0	0	1
0	0	1	1

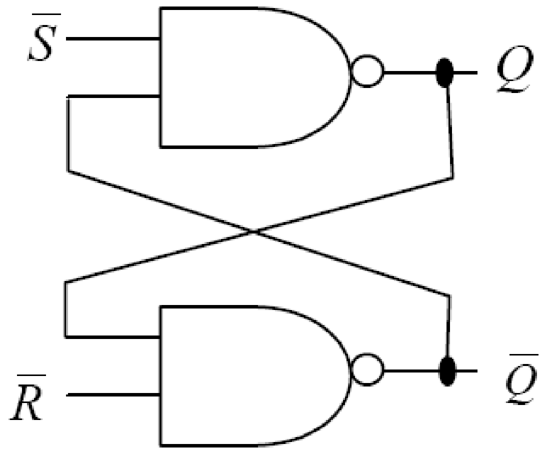
Q_0 = value before S&R changes

$\overline{R} = 0$ & $\overline{S} = 0$:

- $\overline{S} = 0$ & assume $\overline{Q} = 0$ $\rightarrow$ $Q = 1$.
  $\overline{S} = 0$ & assume $\overline{Q} = 1$ $\rightarrow$ $Q = 1$.
- $\overline{R} = 0$ & assume $Q = 0$ $\rightarrow$ $\overline{Q} = 1$.
  $\overline{R} = 0$ & assume $Q = 1$ $\rightarrow$ $\overline{Q} = 1$.

How does a flip-flop work?

Basic flip-flop: the SR latch



Logic table

$\overline{S}$	$\overline{R}$	Q	$\overline{Q}$
1	1	Q_0	$\overline{Q}_0$
0	1	1	0
1	0	0	1
0	0	1	1

Q_0 = value before S&R changes

$\overline{R}$ = 0 & $\overline{S}$ = 0:

- $\overline{S}$ = 0 & assume $\overline{Q}$ = 0 → Q = 1.

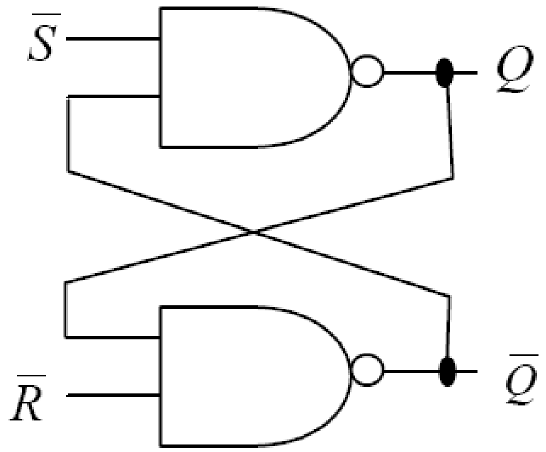
  $\overline{S}$ = 0 & assume $\overline{Q}$ = 1 → Q = 1.
- $\overline{R}$ = 0 & assume Q = 0 → $\overline{Q}$ = 1.

  $\overline{R}$ = 0 & assume Q = 1 → $\overline{Q}$ = 1.

consistent → $\boxed{\overline{R}=0 \;\&\; \overline{S}=0 \rightarrow Q=1 \;\&\; \overline{Q}=1}$

How does a flip-flop work?

Basic flip-flop: the SR latch



Logic table

$\overline{S}$	$\overline{R}$	Q	$\overline{Q}$
1	1	Q_0	$\overline{Q}_0$
0	1	1	0
1	0	0	1
0	0	1	1 ✓

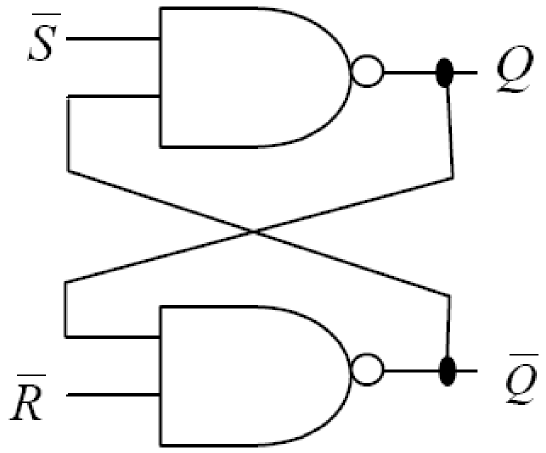
Q_0 = value before S&R changes

$\overline{R} = 0$ & $\overline{S} = 1$:

- $\overline{S} = 1$ & assume $\overline{Q} = 0 \rightarrow Q = 1$.
  $\overline{S} = 1$ & assume $\overline{Q} = 1 \rightarrow Q = 0$.
- $\overline{R} = 0$ & assume $Q = 0 \rightarrow \overline{Q} = 1$.
  $\overline{R} = 0$ & assume $Q = 1 \rightarrow \overline{Q} = 1$.

How does a flip-flop work?

Basic flip-flop: the SR latch



Logic table

$\bar{S}$	$\bar{R}$	Q	$\bar{Q}$
1	1	Q_0	$\bar{Q}_0$
0	1	1	0
1	0	0	1
0	0	1	1 ✓

Q_0 = value before S&R changes

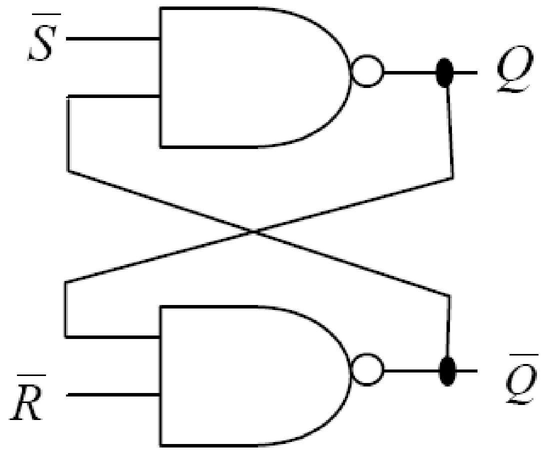
$\bar{R} = 0$ & $\bar{S} = 1$:

- $\bar{S} = 1$ & assume $\bar{Q} = 0 \rightarrow Q = 1$.
  $\bar{S} = 1$ & assume $\boxed{\bar{Q} = 1 \rightarrow Q = 0.}$
- $\bar{R} = 0$ & assume $\boxed{Q = 0 \rightarrow \bar{Q} = 1.}$
  $\bar{R} = 0$ & assume $Q = 1 \rightarrow \bar{Q} = 1$.

consistent ➡ $\boxed{\bar{R}=0 \text{ \& } \bar{S}=1 \rightarrow Q=0 \text{ \& } \bar{Q}=1}$

How does a flip-flop work?

Basic flip-flop: the SR latch



Logic table

$\bar{S}$	$\bar{R}$	Q	$\bar{Q}$	
1	1	Q_0	$\bar{Q}_0$	
0	1	1	0	
1	0	0	1	✓
0	0	1	1	✓

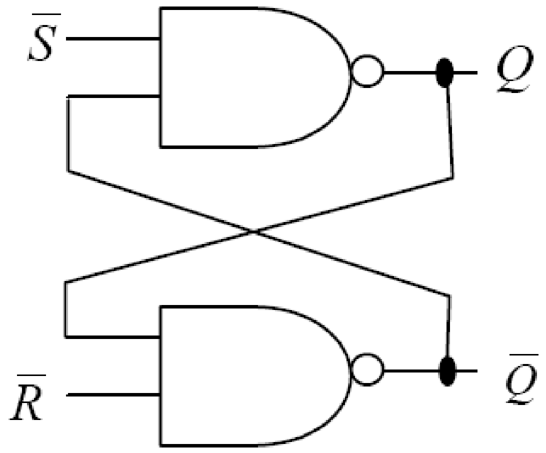
Q_0 = value before S&R changes

$\bar{R} = 1$ & $\bar{S} = 0$:

→ The opposite of $\bar{R} = 0$ & $\bar{S} = 1$ by symmetry.

How does a flip-flop work?

Basic flip-flop: the SR latch



Logic table

$\bar{S}$	$\bar{R}$	Q	$\bar{Q}$	
1	1	Q_0	$\bar{Q}_0$	
0	1	1	0	✓
1	0	0	1	✓
0	0	1	1	✓

Q_0 = value before S&R changes

$\bar{R} = 1$ & $\bar{S} = 1$:

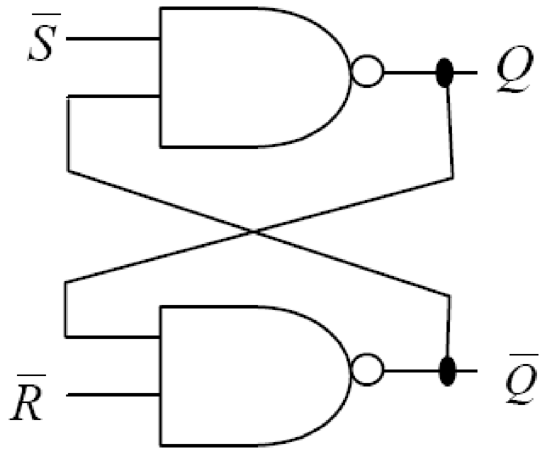
- $\bar{S} = 1$ & assume $\bar{Q} = 0$ → $Q = 1$.

  $\bar{S} = 1$ & assume $\bar{Q} = 1$ → $Q = 0$.
- $\bar{R} = 1$ & assume $Q = 0$ → $\bar{Q} = 1$.

  $\bar{R} = 1$ & assume $Q = 1$ → $\bar{Q} = 0$.

How does a flip-flop work?

Basic flip-flop: the SR latch



Logic table

$\overline{S}$	$\overline{R}$	Q	$\overline{Q}$	
1	1	Q_0	$\overline{Q}_0$	
0	1	1	0	✓
1	0	0	1	✓
0	0	1	1	✓

Q_0 = value before S&R changes

$\overline{R} = 1$ & $\overline{S} = 1$:

- $\overline{S} = 1$ & assume $\overline{Q} = 0 \rightarrow Q = 1.$

  $\overline{S} = 1$ & assume $\overline{Q} = 1 \rightarrow Q = 0.$

- $\overline{R} = 1$ & assume $Q = 0 \rightarrow \overline{Q} = 1.$

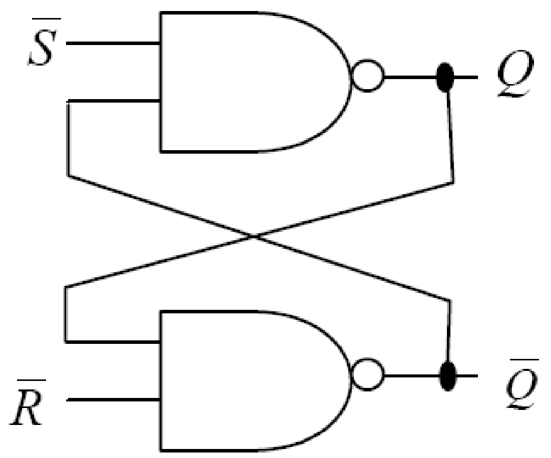
  $\overline{R} = 1$ & assume $Q = 1 \rightarrow \overline{Q} = 0.$

consistent → $\boxed{\overline{R}=1 \text{ \& } \overline{S}=1 \rightarrow Q=1 \text{ \& } \overline{Q}=0}$

consistent → $\boxed{\overline{R}=1 \text{ \& } \overline{S}=1 \rightarrow Q=0 \text{ \& } \overline{Q}=1}$

How does a flip-flop work?

Basic flip-flop: the SR latch



Logic table

$\overline{S}$	$\overline{R}$	Q	$\overline{Q}$	
1	1	Q_0	$\overline{Q}_0$	
0	1	1	0	✓
1	0	0	1	✓
0	0	1	1	✓

Q_0 = value before S&R changes

$\overline{R}$ = 1 & $\overline{S}$ = 1:

- $\overline{S}$ = 1 & assume $\overline{Q}$ = 0 → Q = 1.

  $\overline{S}$ = 1 & assume $\overline{Q}$ = 1 → Q = 0.
- $\overline{R}$ = 1 & assume Q = 0 → $\overline{Q}$ = 1.

  $\overline{R}$ = 1 & assume Q = 1 → $\overline{Q}$ = 0.

consistent → $\boxed{\overline{R}=1 \text{ \& } \overline{S}=1 \rightarrow Q=1 \text{ \& } \overline{Q}=0}$

consistent → $\boxed{\overline{R}=1 \text{ \& } \overline{S}=1 \rightarrow Q=0 \text{ \& } \overline{Q}=1}$

Two settings are possible

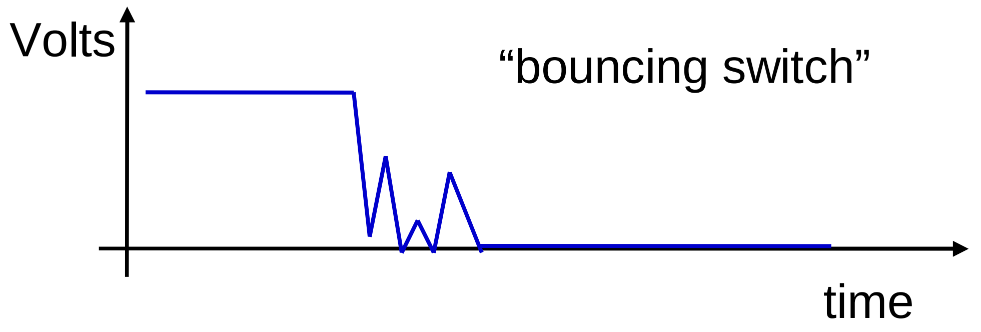
→ i.e. flip-flop keeps its state.

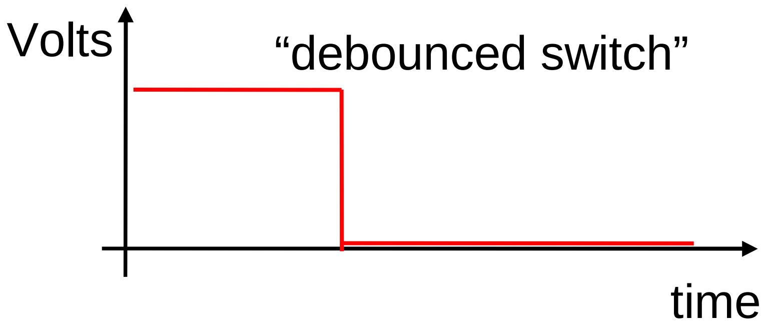
SR Latch Switch Debouncer

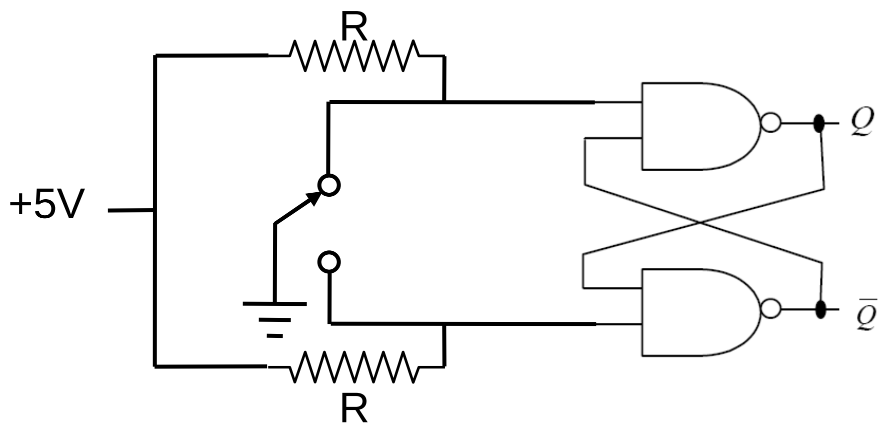
SR latch flip-flops are not used much for memory, but they are used for debouncing switches.

Switch Bounce:

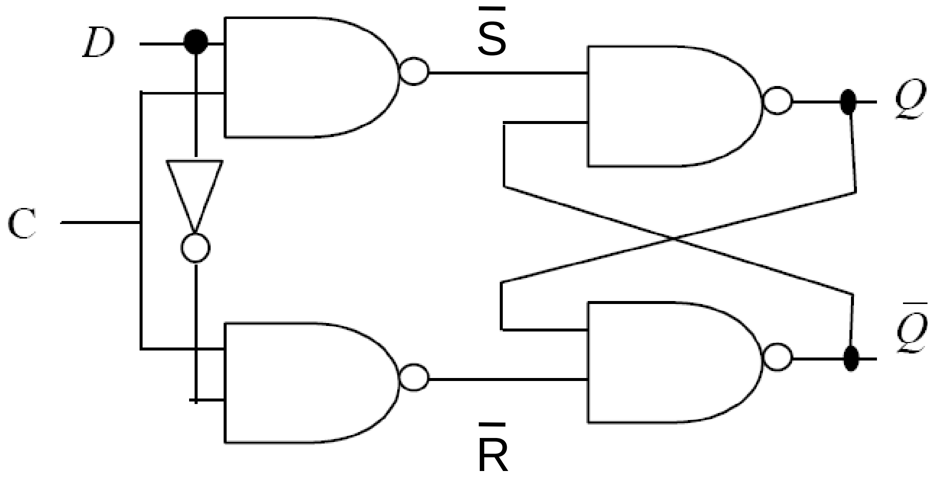
When a switch is toggled it will not go smoothly from HIGH to LOW, or vice versa.







Clocked D-type Latch



Logic table

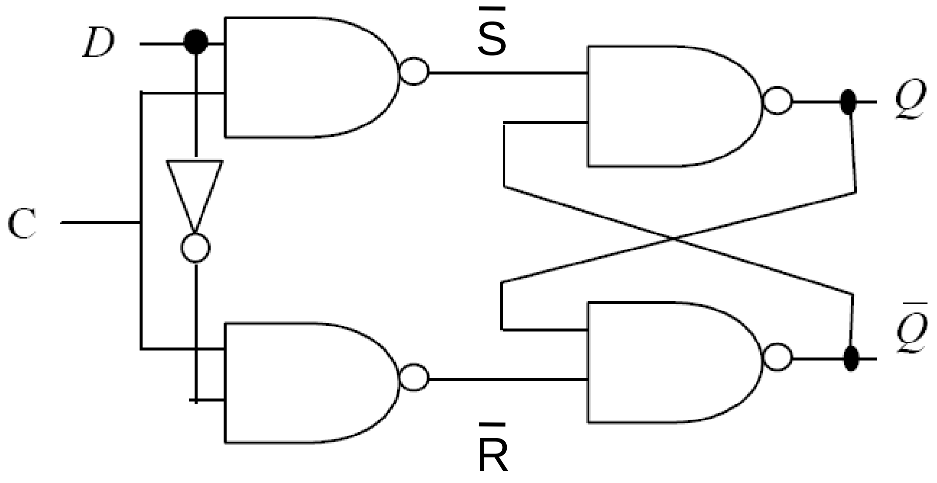
D	C	Q	$\bar{Q}$
X	0	Q_0	$\bar{Q}_0$
1	1	1	0
0	1	0	1

Clock Circuit Analysis:

- C = 1 & D = 1 → $\bar{S}$ = 0 & $\bar{R}$ = 1.
  C = 1 & D = 0 → $\bar{S}$ = 1 & $\bar{R}$ = 0.

- C = 0 & D = 1 → $\bar{S}$ = 1 & $\bar{R}$ = 1.
  C = 0 & D = 0 → $\bar{S}$ = 1 & $\bar{R}$ = 1.

Clocked D-type Latch



Logic table

D	C	Q	$\overline{Q}$
X	0	Q_0	$\overline{Q}_0$
1	1	1	0
0	1	0	1

Clock Circuit Analysis:

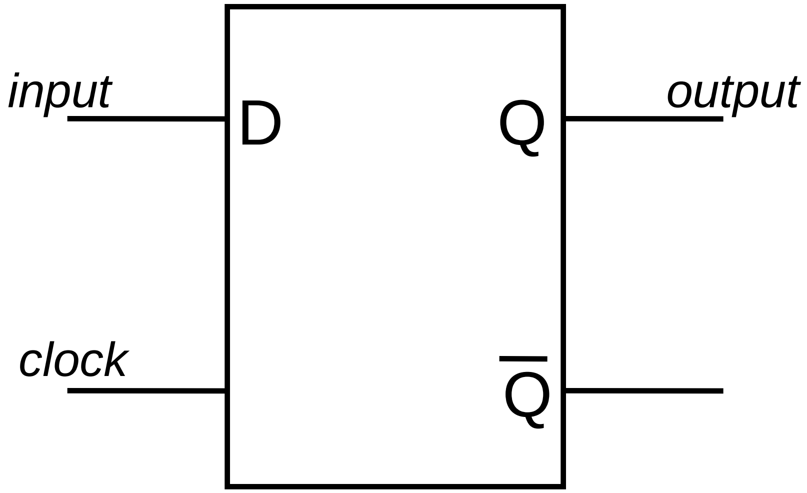
- C = 1 & D = 1 → $\overline{S}$ = 0 & $\overline{R}$ = 1.
  C = 1 & D = 0 → $\overline{S}$ = 1 & $\overline{R}$ = 0.

→ Clock HIGH: D sets the flip-flop state

- C = 0 & D = 1 → $\overline{S}$ = 1 & $\overline{R}$ = 1.
  C = 0 & D = 0 → $\overline{S}$ = 1 & $\overline{R}$ = 1.

→ Clock LOW: flip-flop state is locked

Clocked D-type Latch



Logic table

D	C	Q	$\overline{Q}$
X	0	Q_0	$\overline{Q}_0$
1	1	1	0
0	1	0	1

Clock Circuit Analysis:

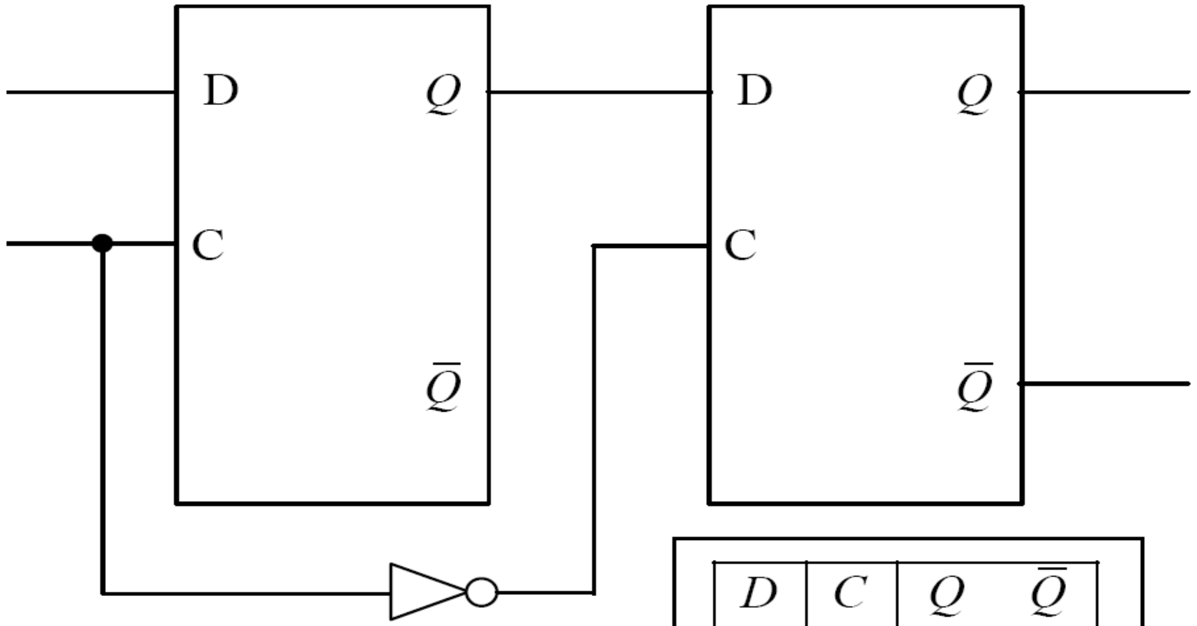
- C = 1 & D = 1 → $\overline{S}$ = 0 & $\overline{R}$ = 1.
  C = 1 & D = 0 → $\overline{S}$ = 1 & $\overline{R}$ = 0.

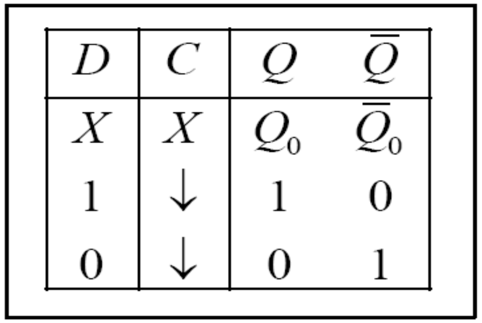
  → Clock HIGH: D sets the flip-flop state

- C = 0 & D = 1 → $\overline{S}$ = 1 & $\overline{R}$ = 1.
  C = 0 & D = 0 → $\overline{S}$ = 1 & $\overline{R}$ = 1.

  → Clock LOW: flip-flop state is locked

Master-Slave D-type Flip-Flop

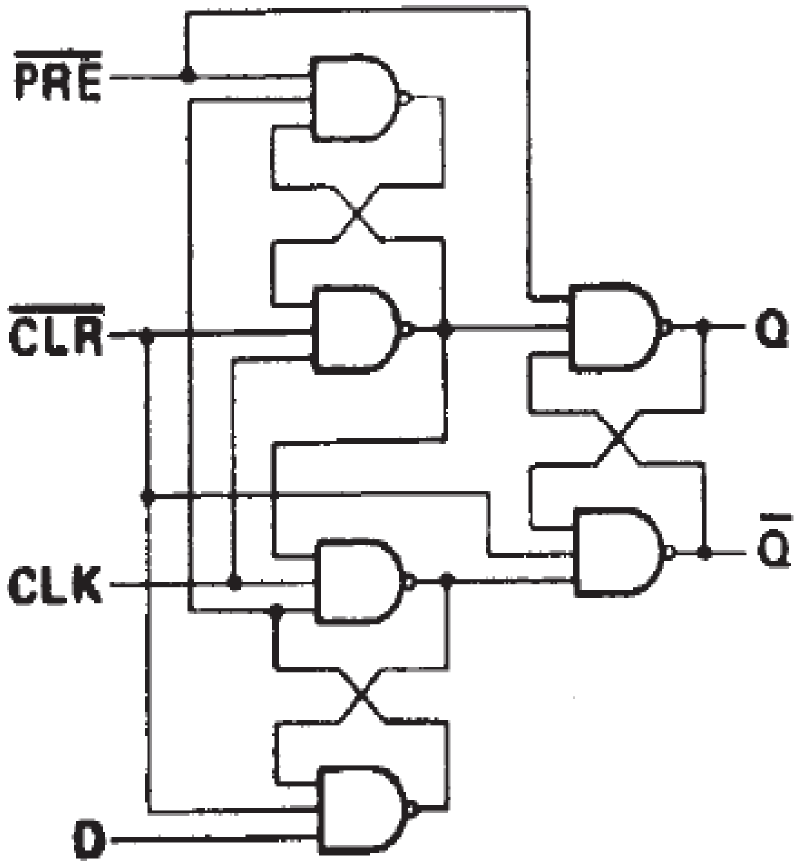




D	C	Q	$\bar{Q}$
X	X	Q_0	$\bar{Q}_0$
1	$\downarrow$	1	0
0	$\downarrow$	0	1

Note: The flip-flop triggers on a the falling edge of the clock.

74LS74 D-type edge-triggered flip-flop



FUNCTION TABLE

INPUTS				OUTPUTS	
$\overline{\text{PRE}}$	$\overline{\text{CLR}}$	CLK	D	Q	$\overline{\text{Q}}$
L	H	X	X	H	L
H	L	X	X	L	H
L	L	X	X	H†	H†
H	H	↑	H	H	L
H	H	↑	L	L	H
H	H	L	X	Q_0	$\overline{Q}_0$

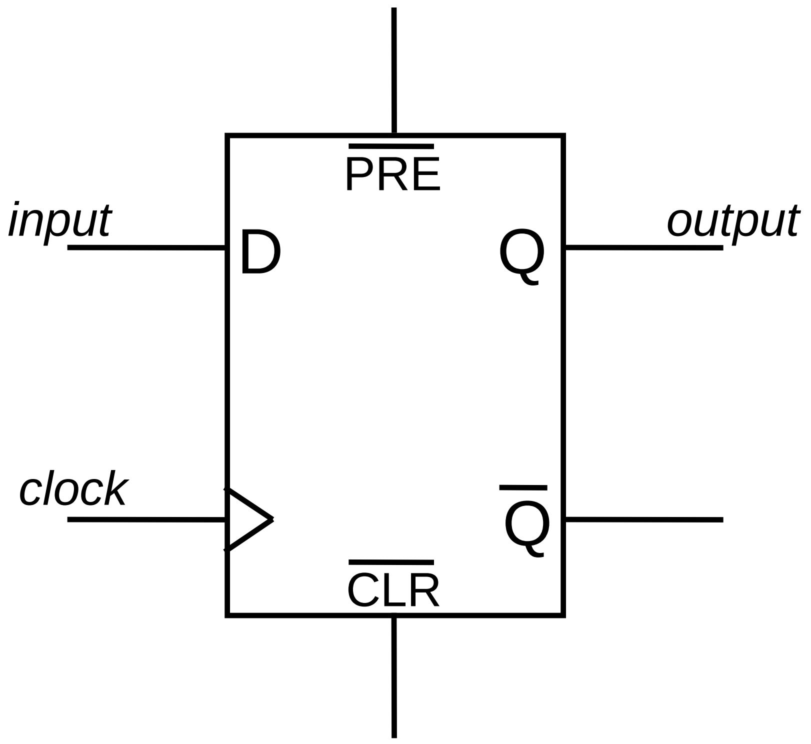
[Texas Instruments 74LS74 flip-flop datasheet]

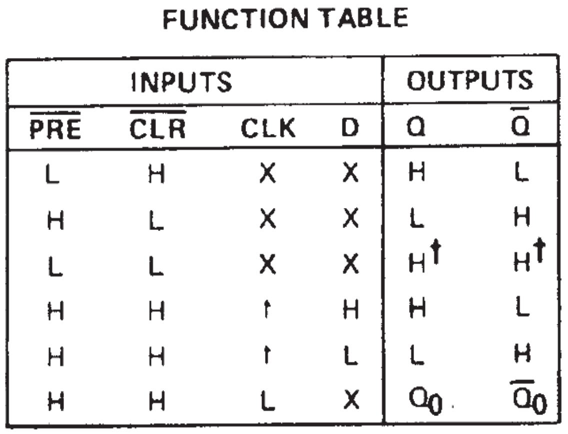
Both $\overline{\text{PRE}}$ and $\overline{\text{CLR}}$ behave like $\overline{\text{S}}$ and $\overline{\text{R}}$ inputs, respectively, on the SR latch.

IMPORTANT: *Both $\overline{\text{PRE}}$ and $\overline{\text{CLR}}$ must be high for normal D-type operation.*

Note: The flip-flop triggers on the rising edge of the clock.

74LS74 D-type edge-triggered flip-flop





FUNCTION TABLE

INPUTS				OUTPUTS	
$\overline{\text{PRE}}$	$\overline{\text{CLR}}$	CLK	D	Q	$\overline{\text{Q}}$
L	H	X	X	H	L
H	L	X	X	L	H
L	L	X	X	H†	H†
H	H	↑	H	H	L
H	H	↑	L	L	H
H	H	L	X	Q_0	$\overline{Q}_0$

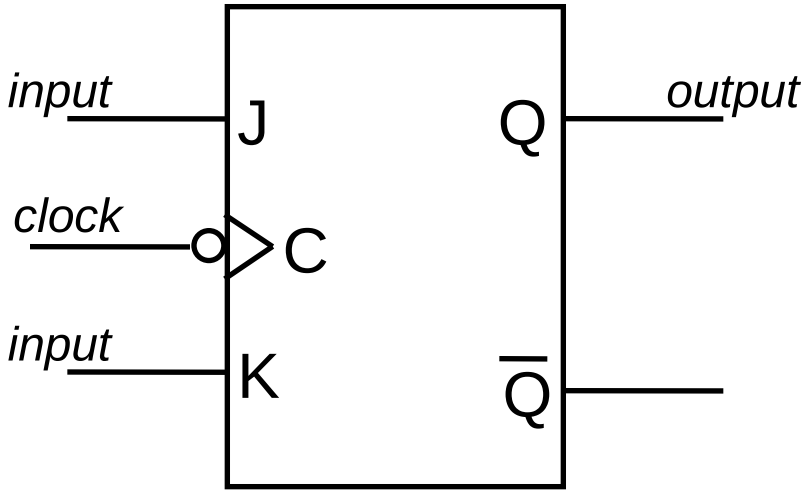
[Texas Instruments 74LS74 flip-flop datasheet]

Both $\overline{\text{PRE}}$ and $\overline{\text{CLR}}$ behave like $\overline{\text{S}}$ and $\overline{\text{R}}$ inputs, respectively, on the SR latch.

IMPORTANT: *Both $\overline{\text{PRE}}$ and $\overline{\text{CLR}}$ must be high for normal D-type operation.*

Note: The flip-flop triggers on the rising edge of the clock.

JK-type flip-flop



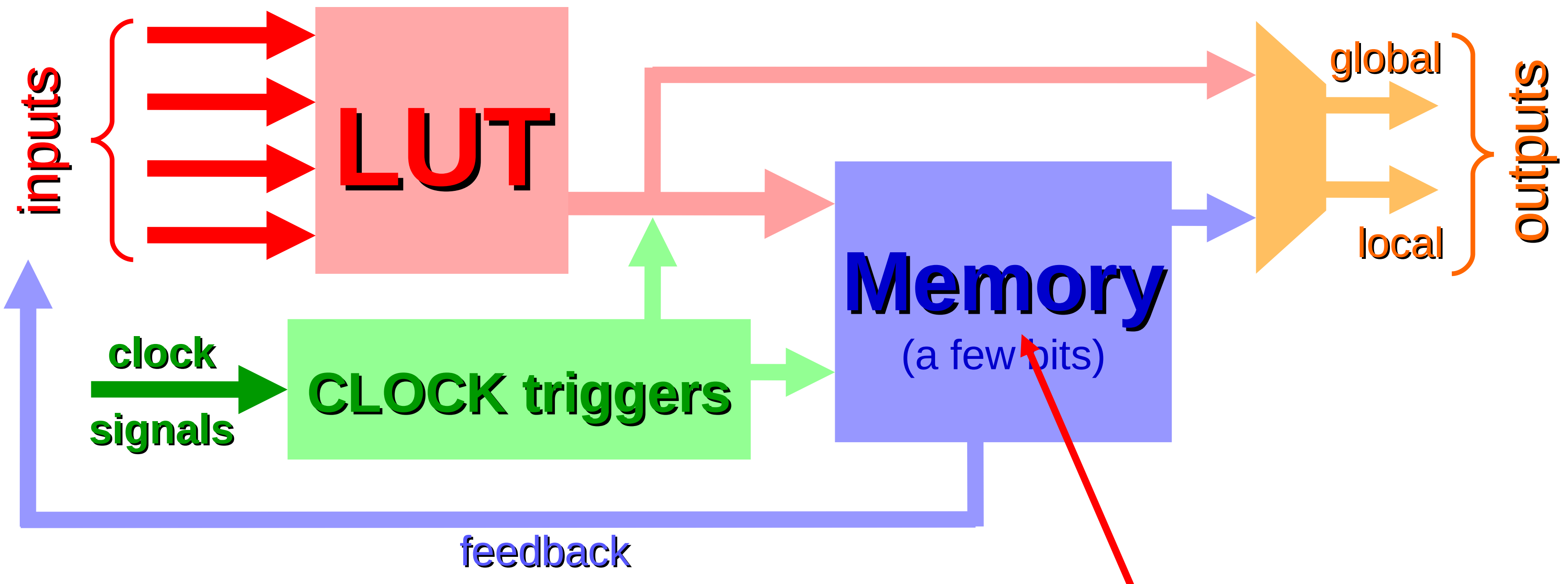
Logic table
for clock falling edge

J	K	Q_{n+1}
0	0	Q_n
1	0	0
0	1	1
1	1	$\overline{Q}_n$

JK-type flip-flops are used in counters.

Flip-flops in FPGAs

Architecture of a single Logic Element

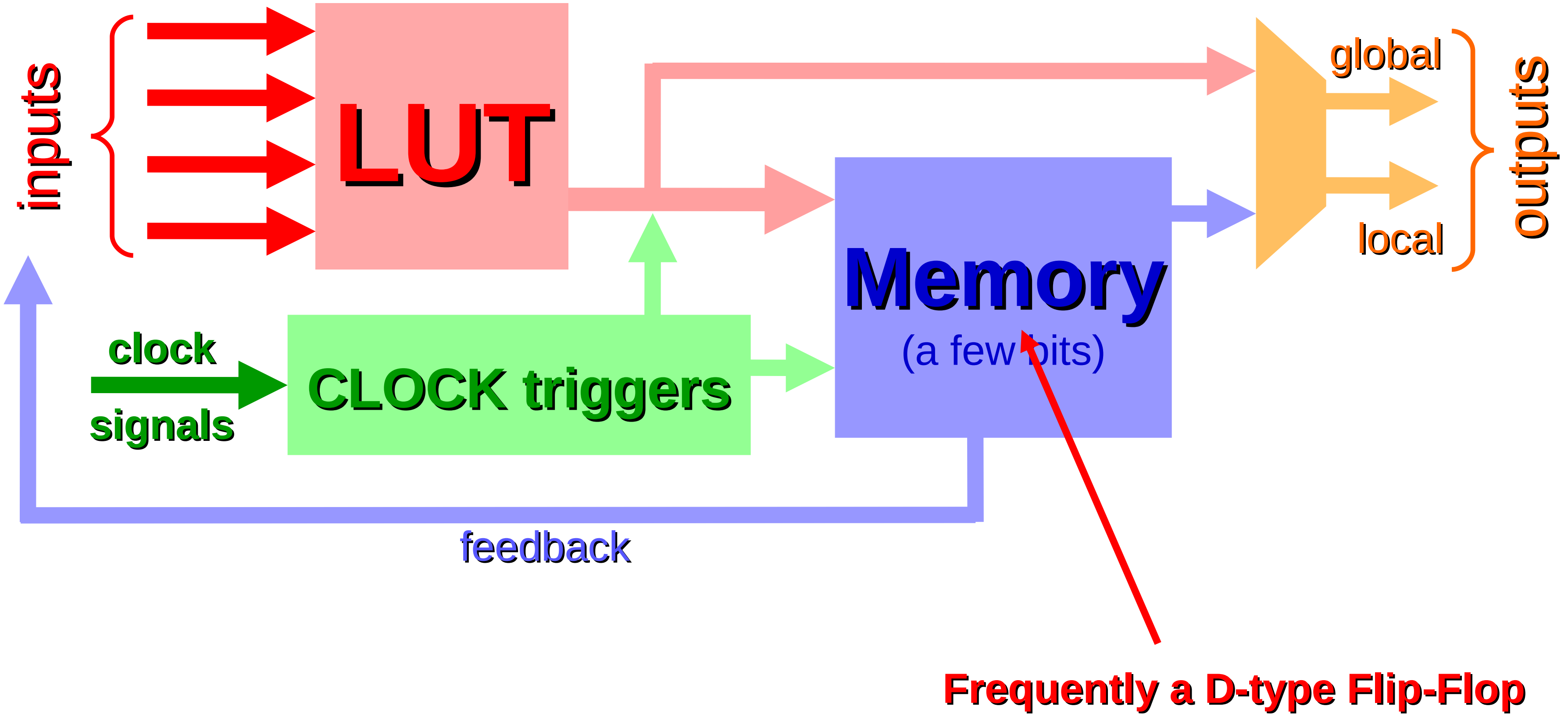


Frequently a D-type Flip-Flop

→ FPGAs are already set-up for synchronous circuit designs

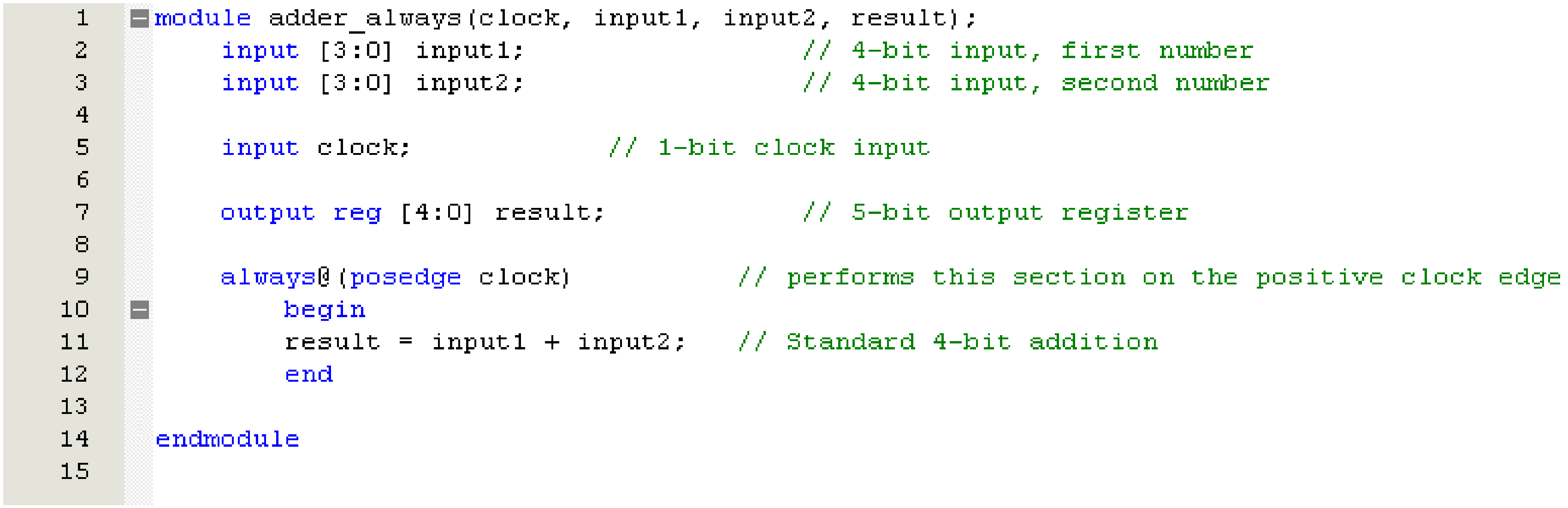
Flip-flops in FPGAs

Architecture of a single Logic Element



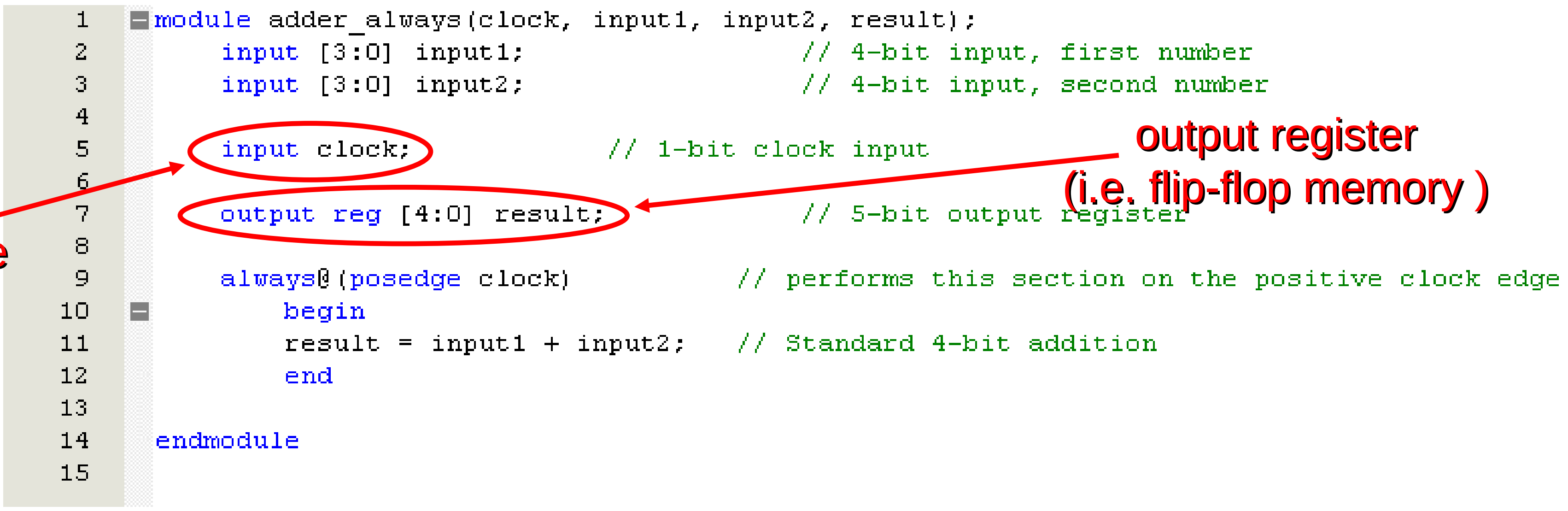
FPGAs are already set-up for synchronous circuit designs

Synchronous programming in Verilog (I)



```verilog
module adder_always(clock, input1, input2, result);
    input [3:0] input1;                 // 4-bit input, first number
    input [3:0] input2;                 // 4-bit input, second number

    input clock;            // 1-bit clock input

    output reg [4:0] result;            // 5-bit output register

    always@(posedge clock)          // performs this section on the positive clock edge
        begin
        result = input1 + input2;   // Standard 4-bit addition
        end

endmodule
```

Synchronous programming in Verilog (I)

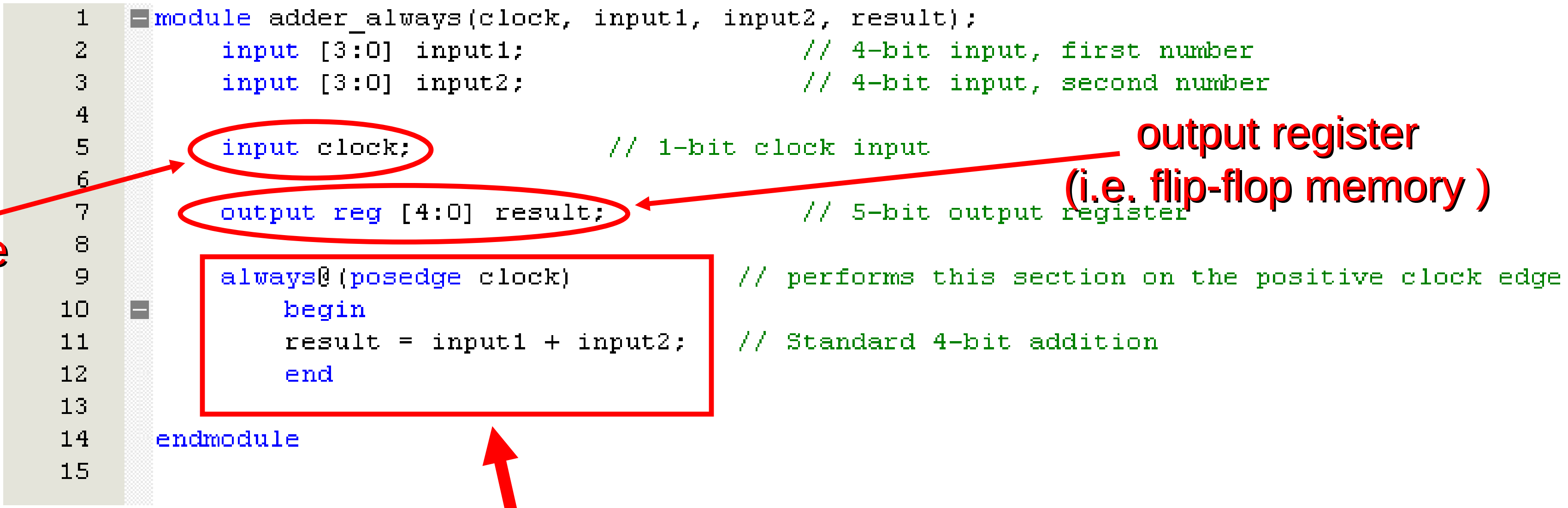


```
module adder_always(clock, input1, input2, result);
    input [3:0] input1;                   // 4-bit input, first number
    input [3:0] input2;                   // 4-bit input, second number

    input clock;              // 1-bit clock input

    output reg [4:0] result;              // 5-bit output register

    always@(posedge clock)            // performs this section on the positive clock edge
        begin
        result = input1 + input2;    // Standard 4-bit addition
        end

endmodule
```

Clock variable

output register (i.e. flip-flop memory)

Synchronous programming in Verilog (I)



```
module adder_always(clock, input1, input2, result);
    input [3:0] input1;                    // 4-bit input, first number
    input [3:0] input2;                    // 4-bit input, second number

    input clock;                // 1-bit clock input

    output reg [4:0] result;               // 5-bit output register

    always@(posedge clock)               // performs this section on the positive clock edge
       begin
       result = input1 + input2;         // Standard 4-bit addition
       end

endmodule
```

Clock variable

output register (i.e. flip-flop memory)

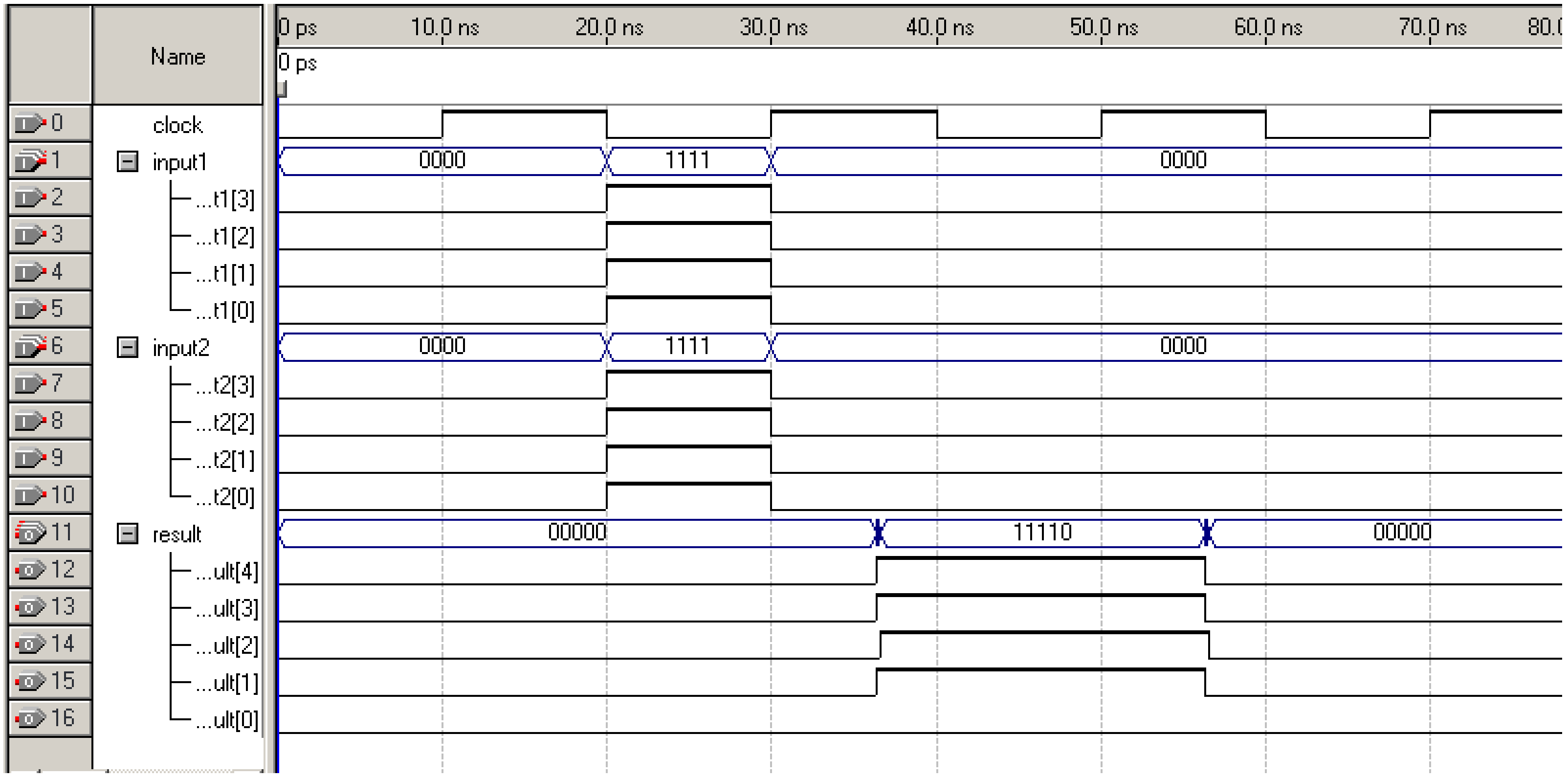
Read as "always at the positive clock edge do the following … "

"always" is the core command for synchronous programming, it should be used as frequently as possible.

"assign" should be used as little as possible. It is only useful for DC-type signals (signals that don't change).

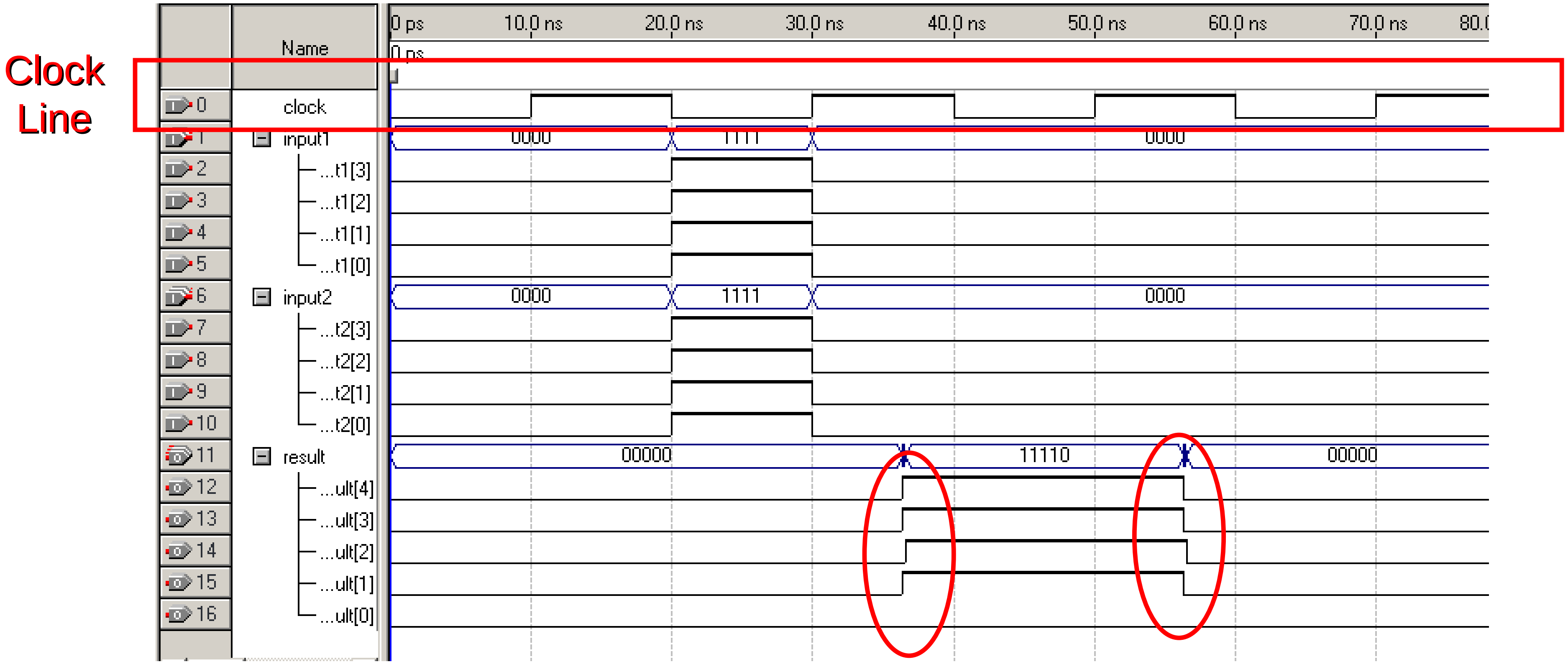
Synchronous programming in Verilog (II)

Quartus II circuit simulation



Synchronous programming in Verilog (II)

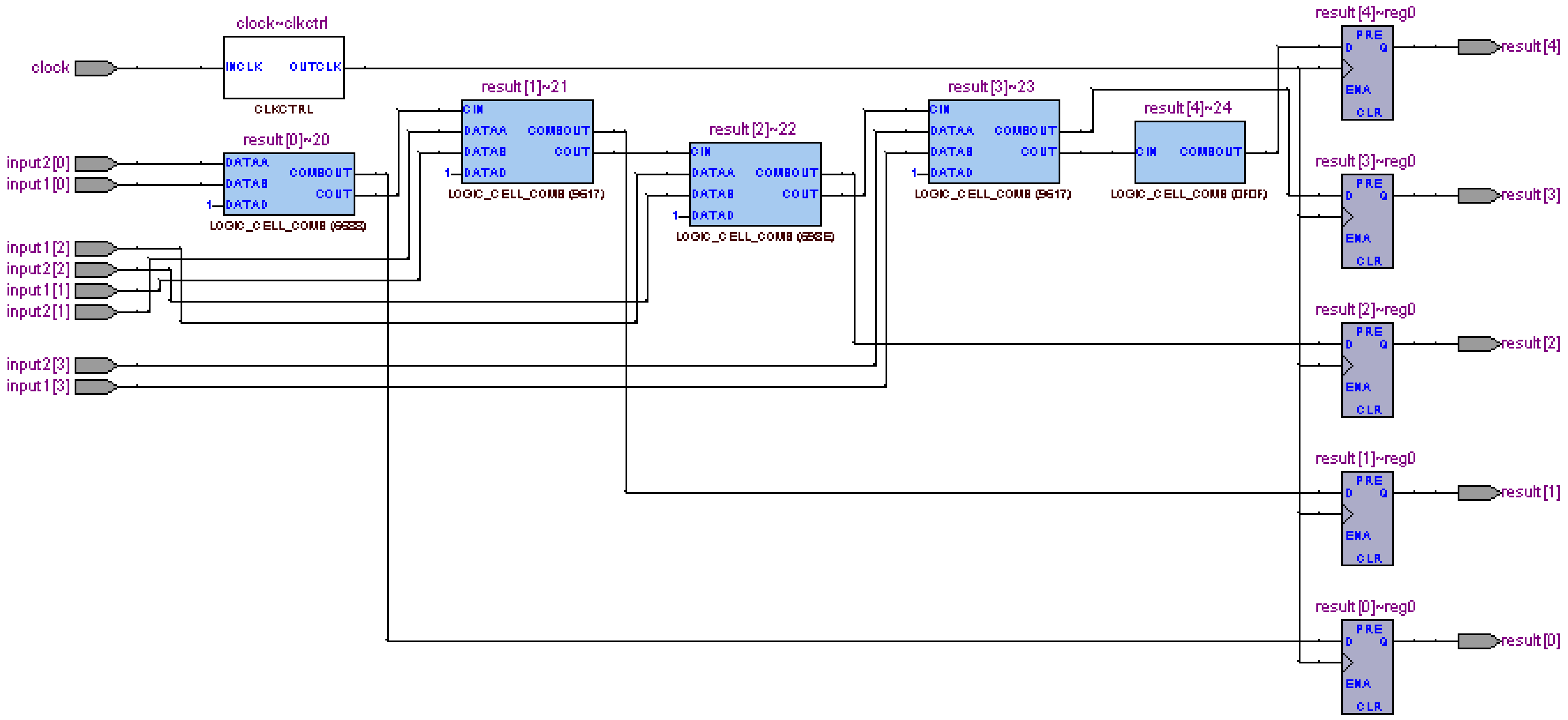
Quartus II circuit simulation



No more glitches

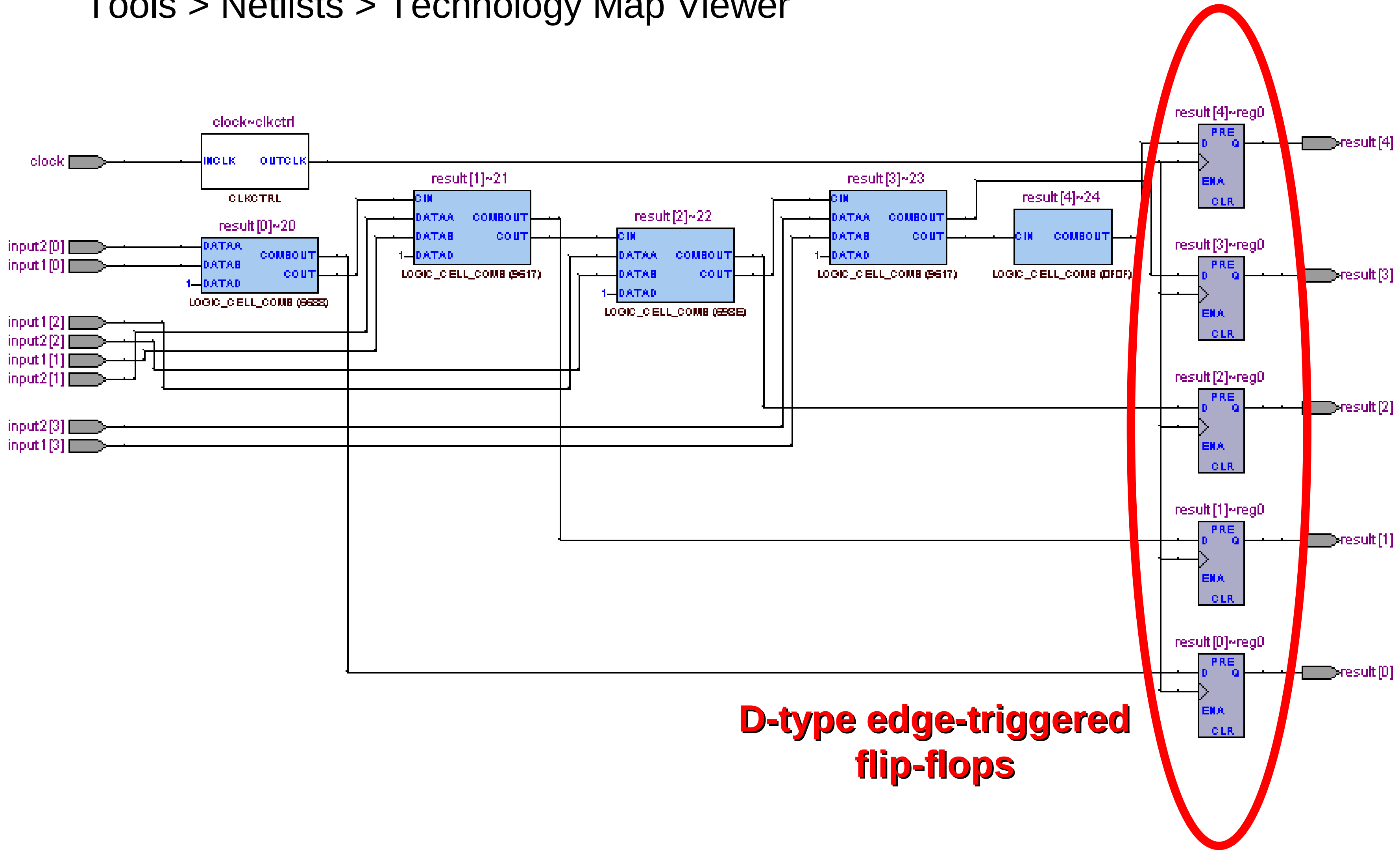
How did the FPGA implement the circuit?

Tools > Netlists > Technology Map Viewer



How did the FPGA implement the circuit?

Tools > Netlists > Technology Map Viewer



D-type edge-triggered flip-flops

Always use "always"

– A. Stummer, U. of Toronto.

Parallel programming in Verilog

- The "always" structure is used for exploiting the parallel processing features of the FPGA.
- Parallel processing must almost always be synchronous if several processes exchange data.

Parallel and Sequential processing examples:

Sequential

```
always@ (negedge clock)
        begin
        a = b;
        c = a;
        end
```

Parallel

```
always@ (negedge clock)
        begin
        a <= b;
        c <= a;
        end
```

Parallel programming in Verilog

- The “always” structure is used for exploiting the parallel processing features of the FPGA.
- Parallel processing must almost always be synchronous if several processes exchange data.

Parallel and Sequential processing examples:

Sequential

```
always@ (negedge clock)
        begin
        a = b;
        c = a;
        end
```

→ c = b

Parallel

```
always@ (negedge clock)
        begin
        a <= b;   } executed
        c <= a;   } simultaneously
        end
```

→ a = b

c = a (previous value)