Introduction to Sequential Circuits

Latches
Flip-flops
Registers

Definitions

- Combinational circuit
  - Output depends only on present inputs
  - Example: Half adder
- Sequential circuit
  - Output depends not just on present inputs, but on past sequence of inputs
  - Stores information, also known as having “state”
  - Simple example: a circuit that counts up in binary

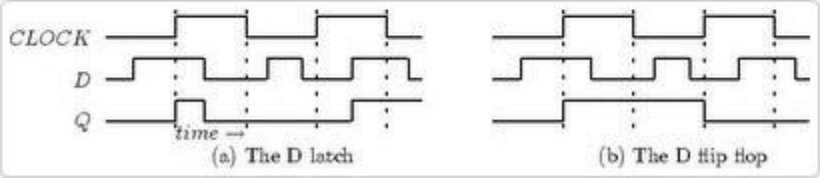
A flip-flop is Edge sensitive: Output only changes on rising (or falling) edge of clock.
A latch is Level sensitive: Output changes whenever clock/Enable is high (or low)

A common implementation of a flip-flop is a pair of latches (Master/Slave flop).

Latches are sometimes called "transparent latches", because they are transparent (input directly connected to output) when the clock is high.

The clock to a latch is primarily called the "enable".

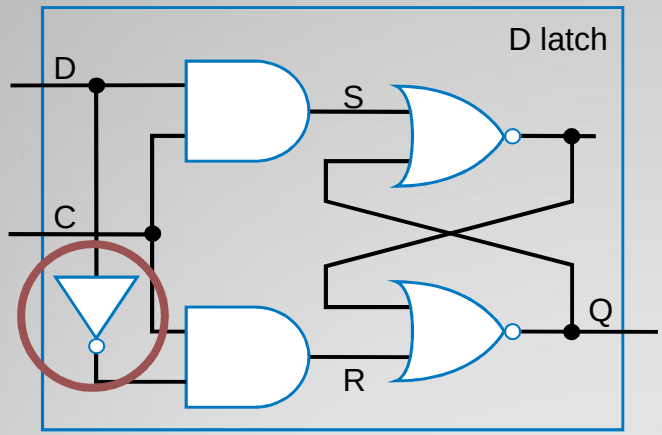
For more information have a look at the picture below.



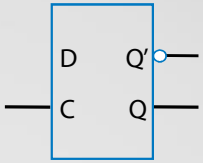
From The Digital Electronics Blog

Latch vs Flip-flop

Level-Sensitive D-Latch



Designer does not have to worry about S and R both being 1 by the addition of an inverter.

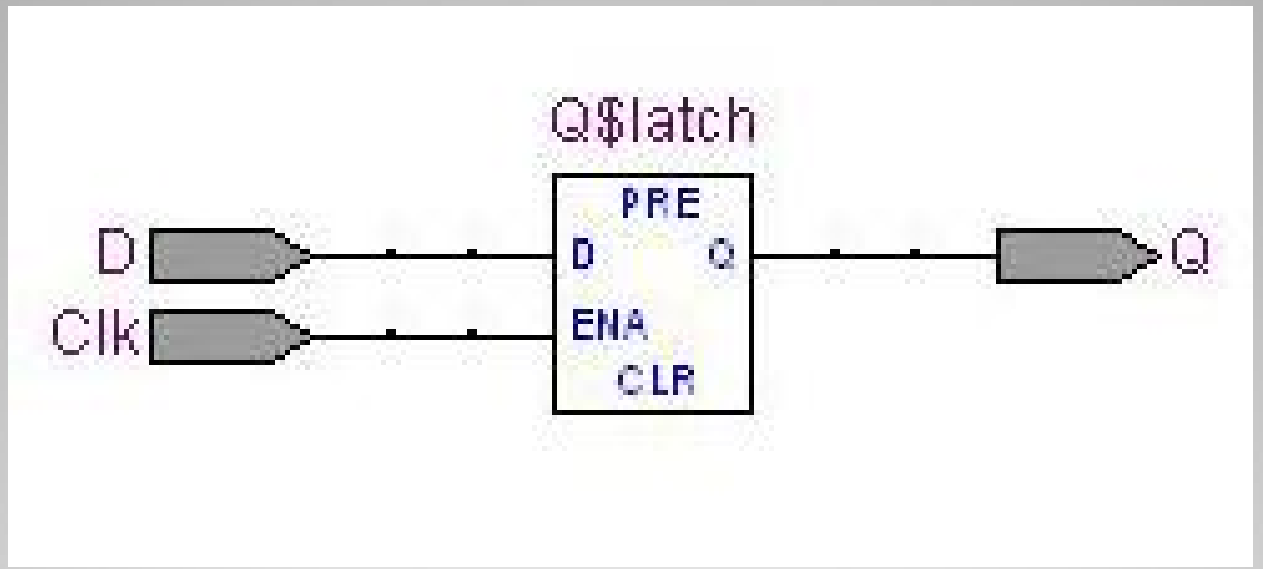


D latch symbol

```
module D_Latch (D, Clk, Q);
    input D, Clk;
    output reg Q;

    always @(D, Clk)
       if (Clk)
         Q = D;

endmodule
```

- If you enter this in Quartus, you'll get several warnings (**inferred latches**)
- We really don't want **inferred latches**.
- If you want a latch, use the Quartus primitive (shown later)

Figure 7.35. Code for a gated D latch.



What Quartus Built

RTL View

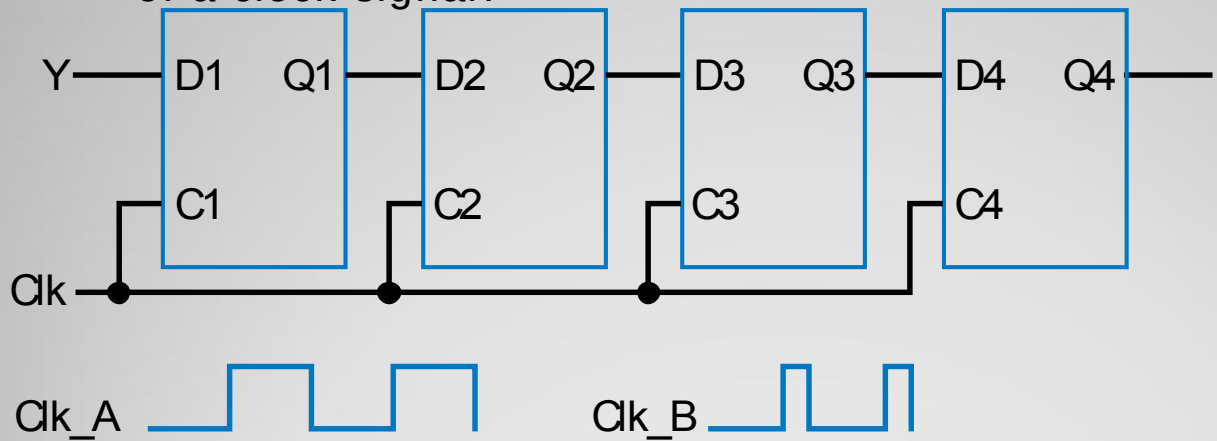
```
module D_Latch (D, Clk, Q);
       input D, Clk;
       output Q;

       latch D1( D, Clk, Q );
       //LATCH <instance_name> (.d(<input_wire>),
       //   .ena(<input_wire>), .q(<output_wire>));
       // latch D2( .d(D), .ena(Clk), .q(Q) );

endmodule
```

(Quartus built the same circuit as previously shown)

D-Latch Using Quartus Primitive

Problem with Level-Sensitive D Latch: Attempted Shift Register

- D latch still has problem (as does SR latch)
  - When C=1, through how many latches will a signal travel?
  - Depends on for how long C=1
    - Clk_A -- signal may travel through multiple latches
    - Clk_B -- signal may travel through fewer latches
  - Hard to pick C that is just the right length
    - Can we design bit storage that only stores a value on the rising edge of a clock signal?



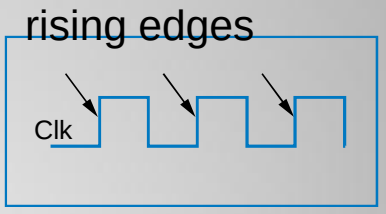
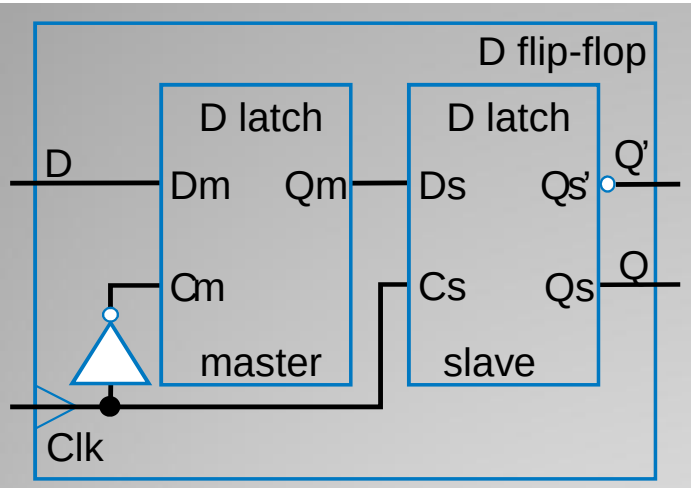
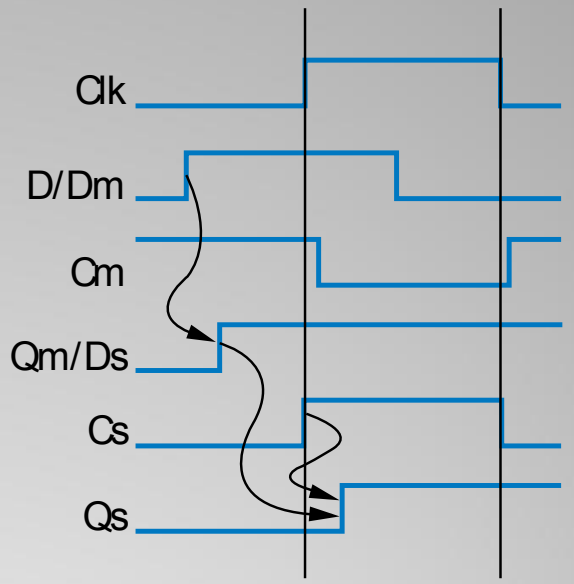
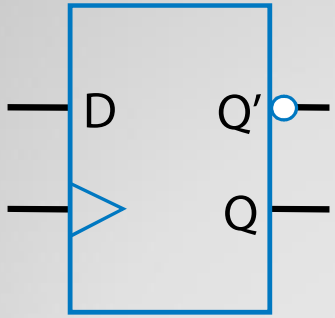


Chart from F. Vahid

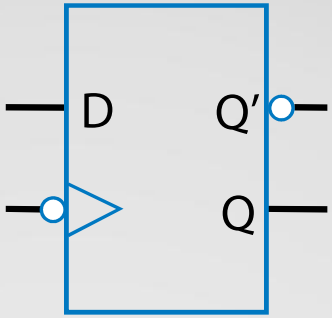






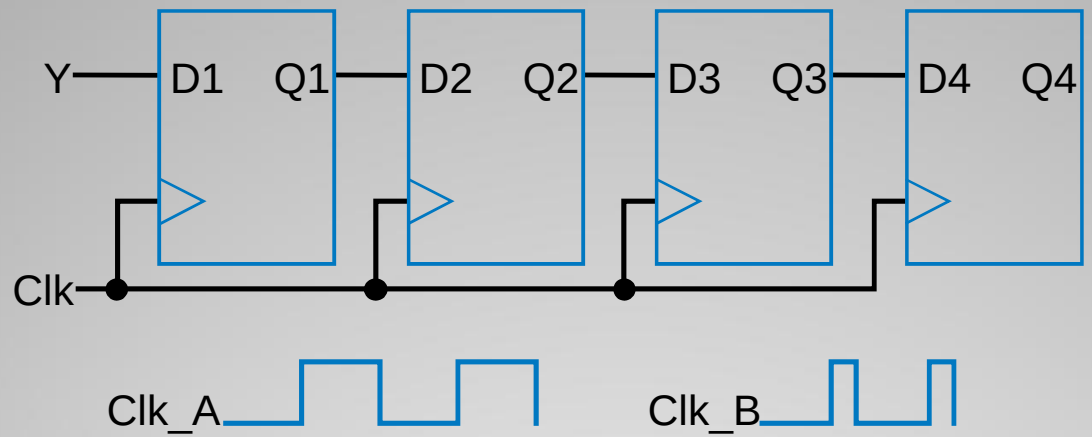
Positive Edge DFF

or



Negative Edge DFF

The D-Flip-Flop Solution



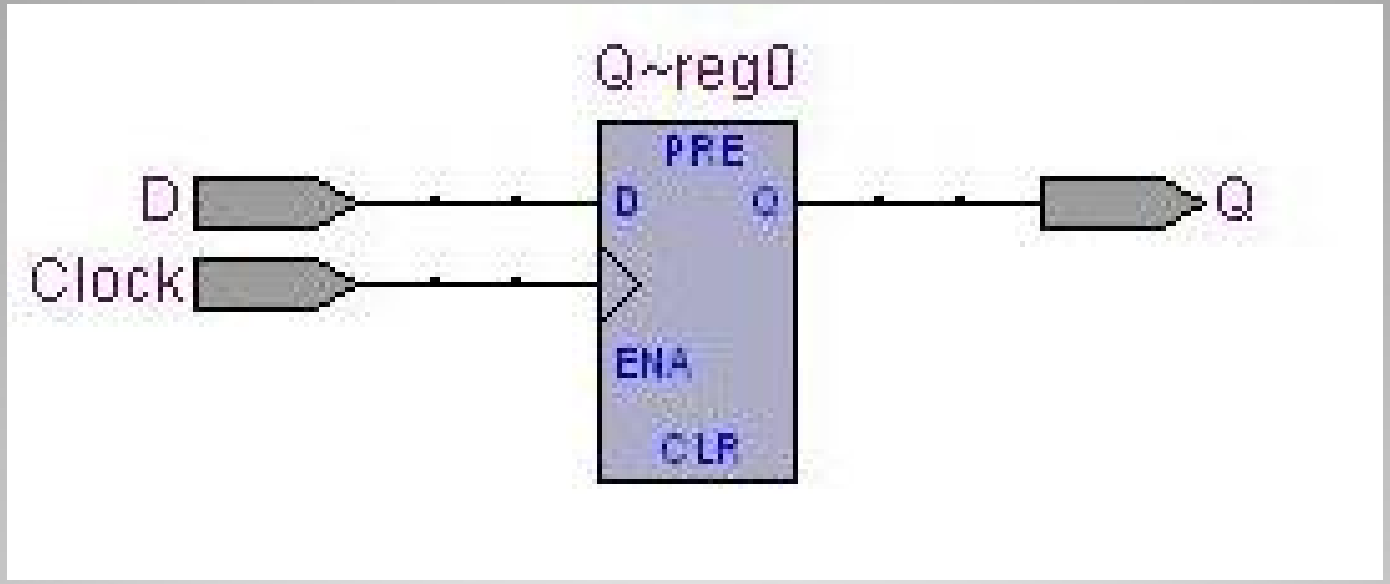
Solves the problem mentioned earlier: Data will advance exactly one FF per (positive) clock edge, regardless of the clock period or duty cycle.

D Flip-Flops Wired Together: Correct Shift Register

```
module DFlipFlop( D, Clock, Q );
    input D, Clock;
    output reg Q;

    always @( posedge Clock )
        Q <= D;  // non-blocking

endmodule
```

Figure 7.36. Code for D flip-flop

What Quartus Built (RTL View)



Note the unused ports (inputs):

- ENA = Enable
- CLR = Clear
- PRE = Preset (opposite of Clear)

Deprecated Hardware:

Latches:

1. Use flops, not latches
2. Latch-based designs are susceptible to timing problems
3. The transparent phase of a latch can let a signal "leak" through a latch — causing the signal to affect the output one clock cycle too early
4. It's possible for a latch-based circuit to simulate correctly, but not work in real hardware, because the timing delays on the real hardware don't match those predicted in synthesis

Flip-flops:

1. Limit yourself to D-type flip-flops
2. Some FPGA and ASIC cell libraries include only D-type flip flops. Others, such as Altera's APEX FPGAs, can be configured as D, T, JK, or SR flip-flops.

- For every signal in your design, know whether it should be a flip-flop or combinational. Examine the log file e.g. dc shell.log to see if the flip-flops in your circuit match your expectations, and to check that you don't have any latches in your design.
- Do not assign a signal to itself (e.g. a <= a; is bad). If the signal is a flop, use an enable to cause the signal to hold its value. If the signal is combinational, then assigning a signal to itself will cause combinational loops, which are very bad.

Some Advice

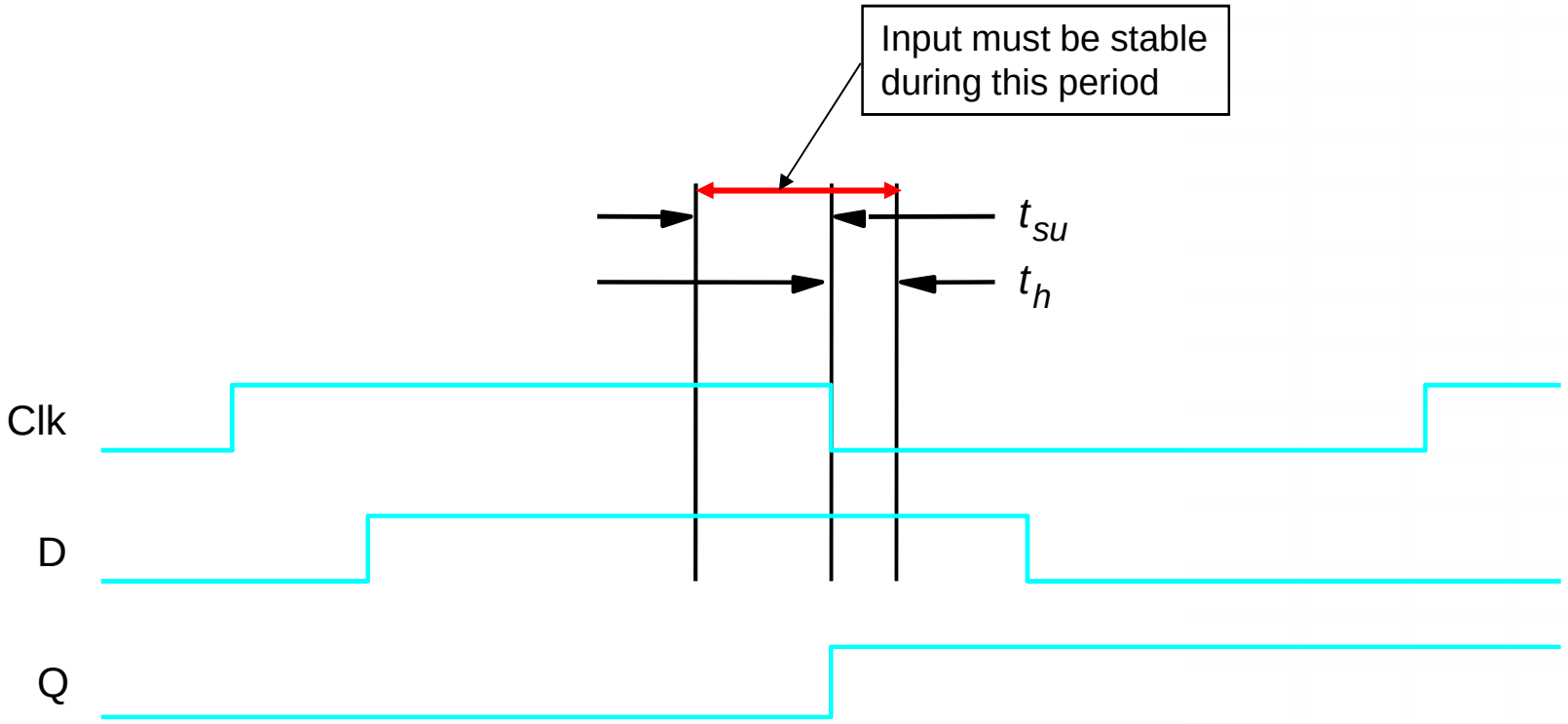
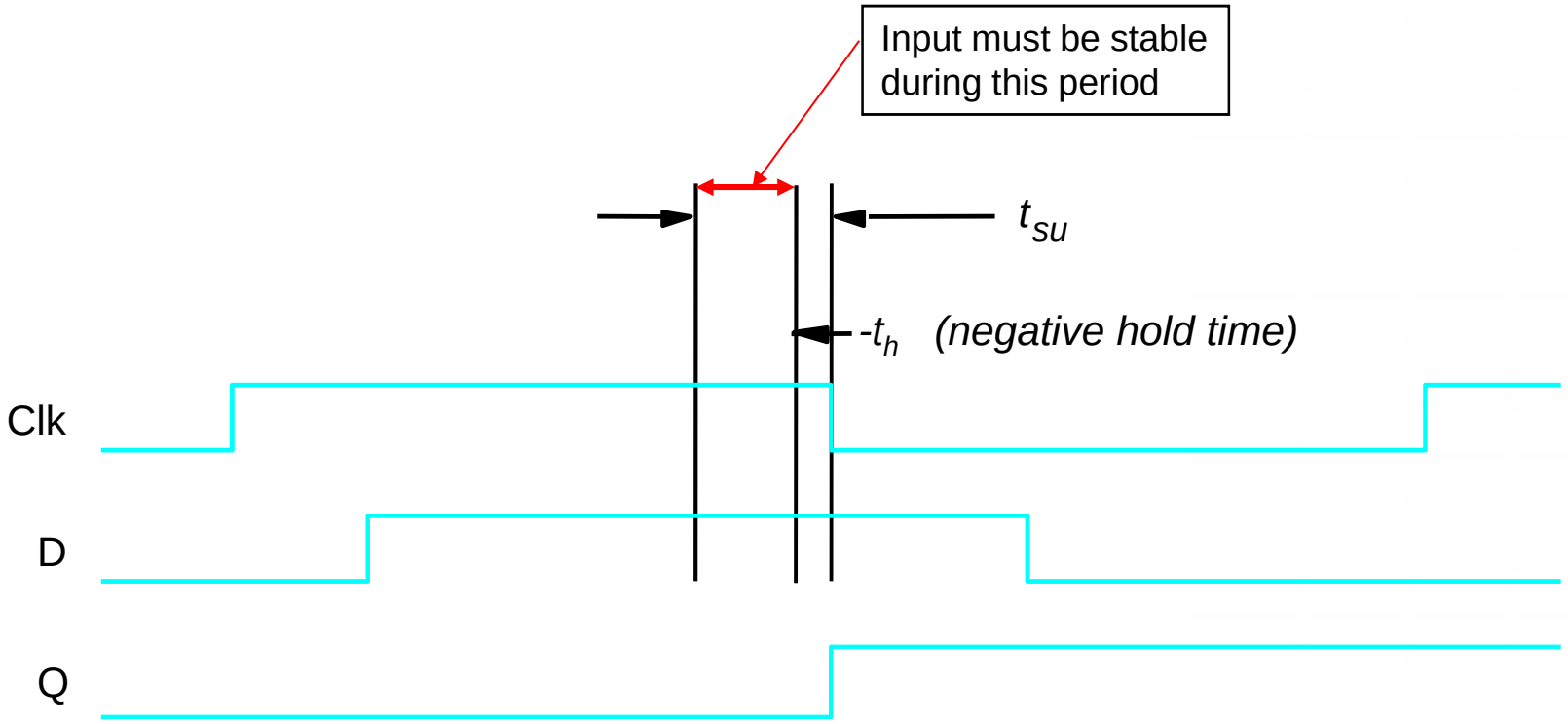
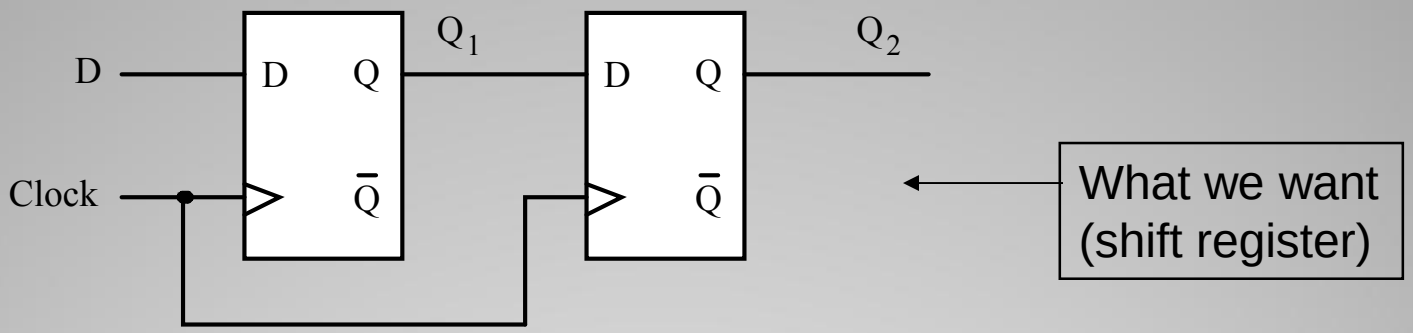


Figure 7.9. Setup and hold times.

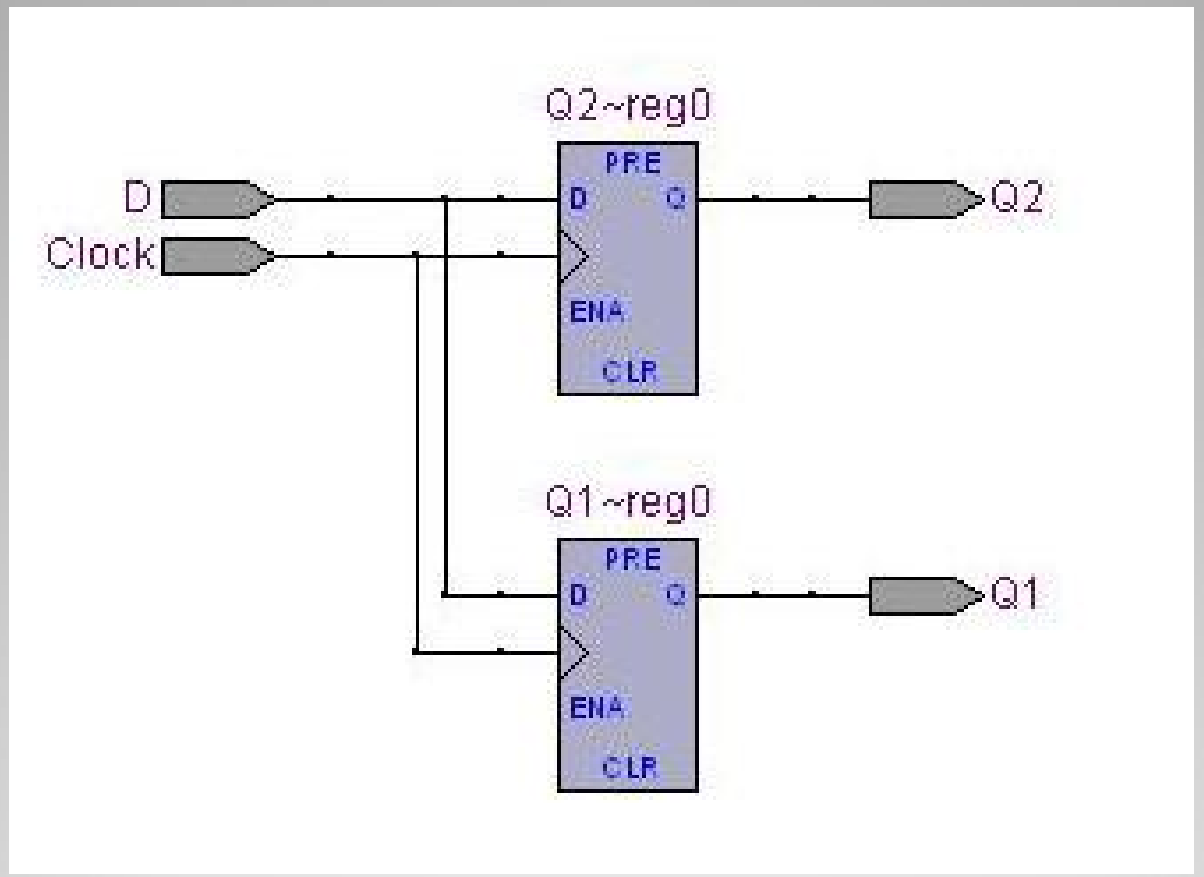


Setup and <u>negative</u> hold times.

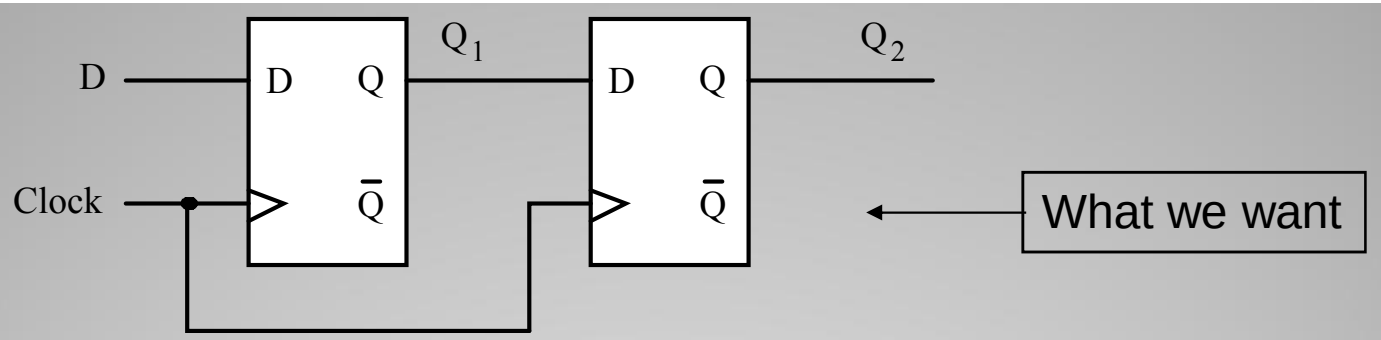



```
module example7_3 (D, Clock, Q1, Q2);
        input D, Clock;
        output reg Q1, Q2;
        always @(posedge Clock)
        begin
                Q1 = D;  // blocking
                Q2 = Q1;
        end
endmodule
```

Two cascaded flip-flops

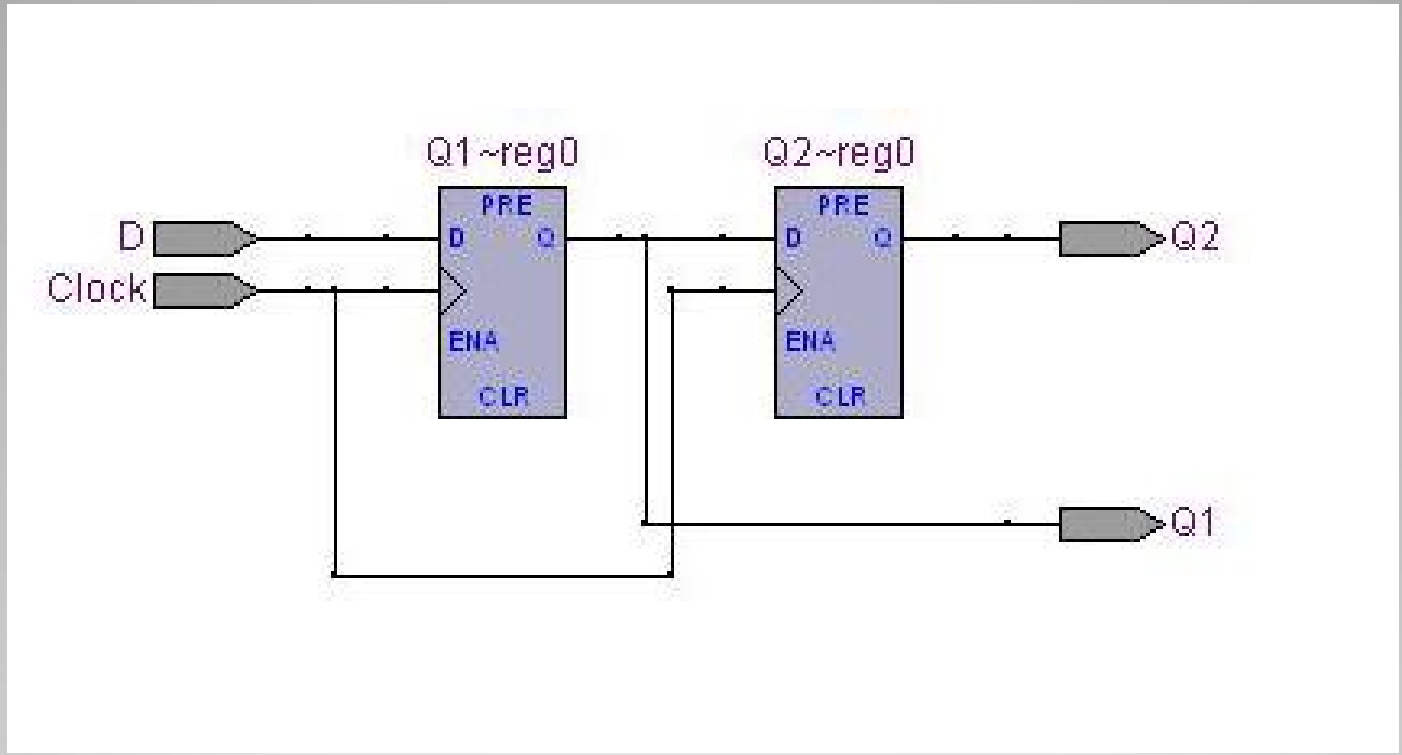


What Quartus Built




```
module example7_3 (D, Clock, Q1, Q2);
       input D, Clock;
       output reg Q1, Q2;
       always @(posedge Clock)
       begin
         Q1 <= D; // non-blocking
         Q2 <= Q1;
       end
endmodule
```

Figure 7.37. Correct code for two cascaded flip-flops



What Quartus Built

Blocking/Non-blocking Uses

- Use blocking assignments (=) to model combinational logic.
- Use non-blocking assignments (<=) to model sequential logic.
- This helps avoid race conditions that can develop.

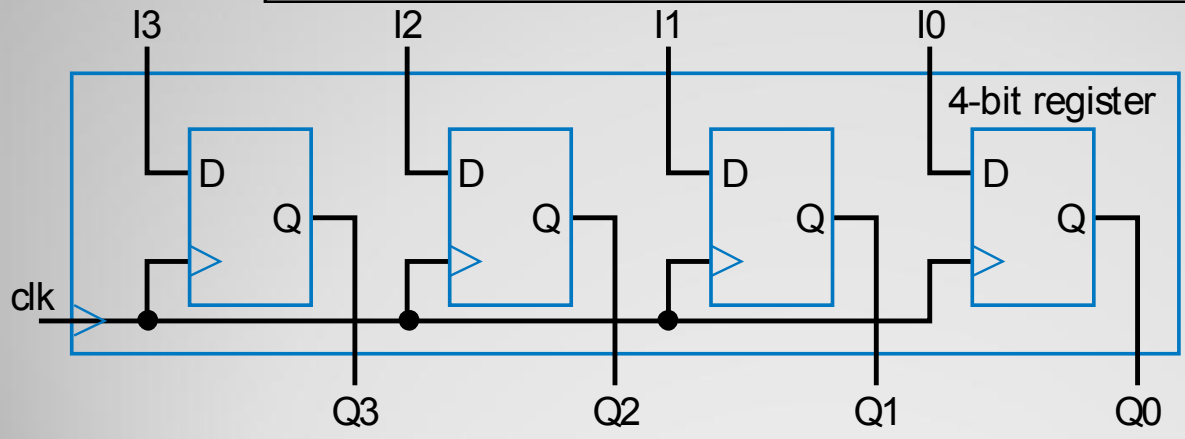
[Middle English *registre*, from Old French, from Medieval Latin *registrum*, alteration of Late Latin *regesta*, from Latin neuter pl. past participle of *regerere*, to record : *re-*, re- + *gerere*, to carry.]

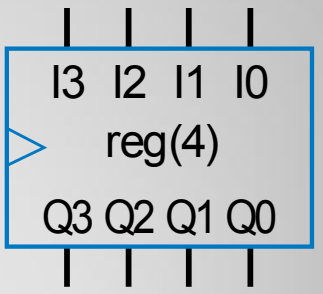
Origin of ‘register’

Basic Register

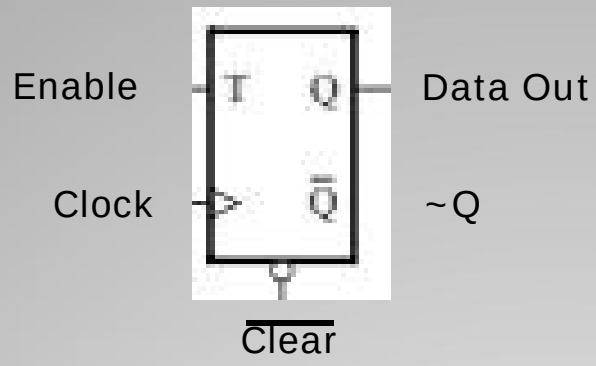
- A Register is a collection of bit storage flip-flops as shown below.
- Each Q is brought out to a pin; they are not cascaded together.
- Assembly programmers are familiar with the term Register.
- In Verilog:

```
...
   reg [3:0] A;
   always @( posedge Clk ) begin
     A <= X;  // non-blocking
   end
...
```





T-Flip Flop (Toggle Flip Flop)




```
module TFFx( T, Clk, ClrN, Q, QN );
  input T, Clk, ClrN;
  output reg Q;
  output QN = ~Q;
  always @( posedge Clk ) begin
    if ( T ) Q <= ~Q;
    else Q <= Q;
  end  // need to implement Clear
endmodule
```

Note: TFF and tff are Quartus primitive names, so we can't use these for our module names.

```
module TFFx( T, Clk, ClrN, Q, QN );
  input T, Clk, ClrN;
  output reg Q;
  output QN = ~Q;
  always @( posedge Clk ) begin
    if ( ~ClrN )
      Q <= 1'b0;
    else
      if ( T )
        Q <= ~Q;
      else
        Q <= Q;
  end
endmodule
```

TFF with Clear

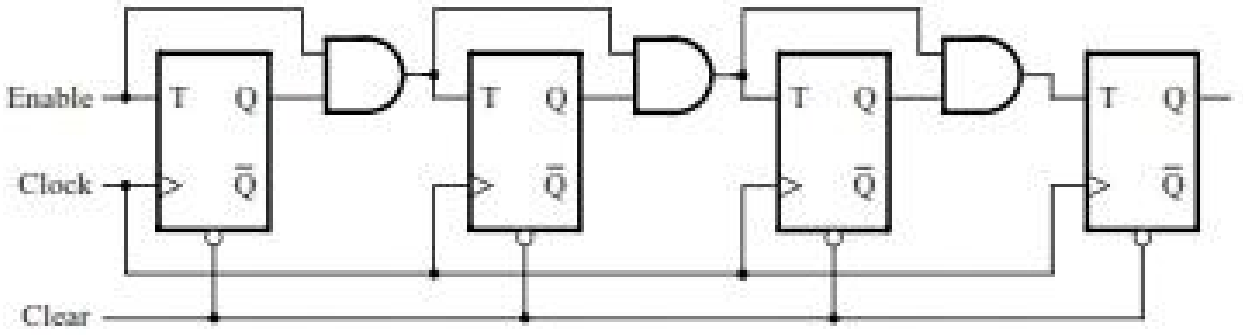
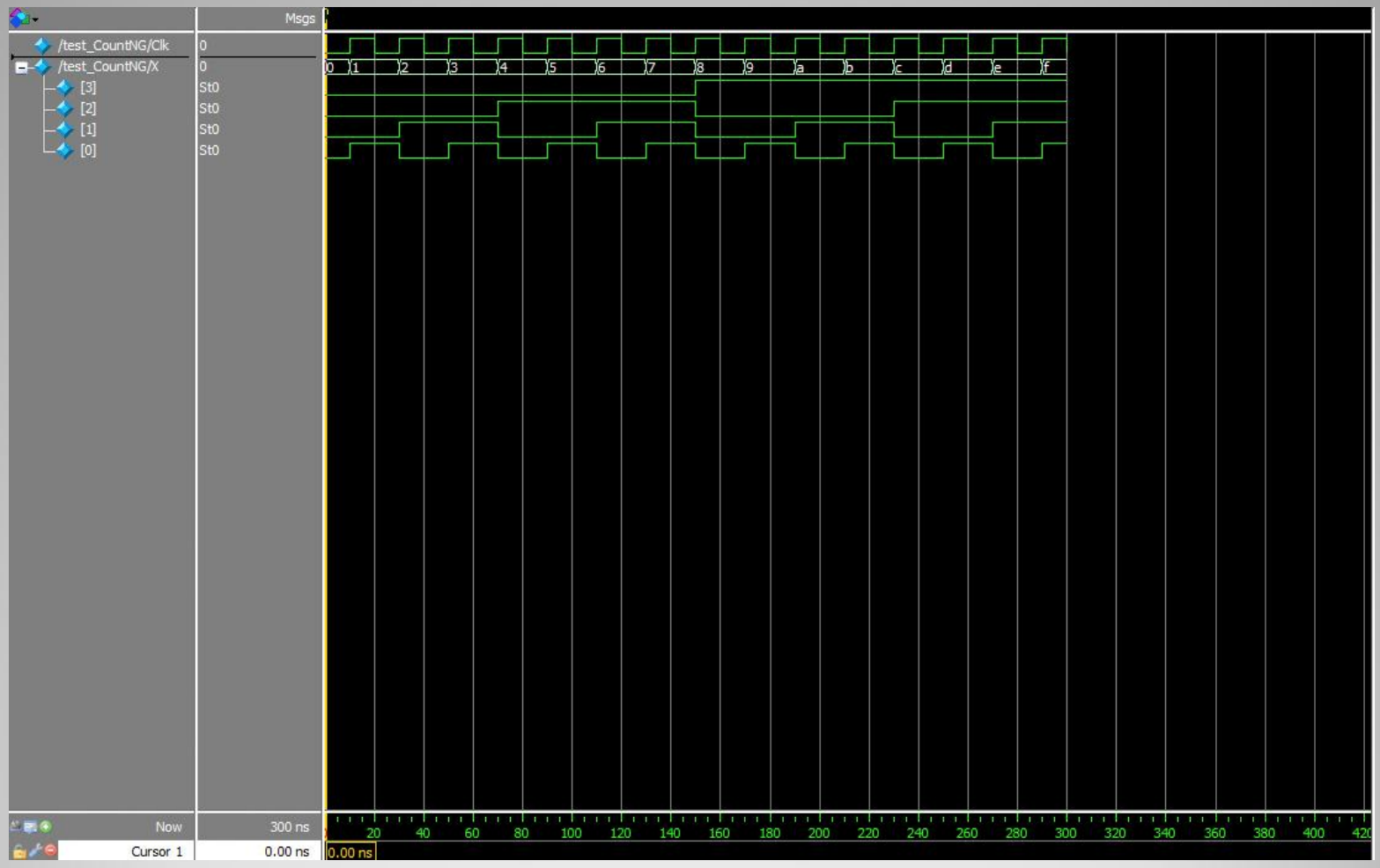


Figure 1. A 4-bit counter.

Counter out of TFFs



ModelSim Output

```
reg Clk;
always begin
  Clk = 0;
  #10;
  Clk = 1;
  #10;
end
```

This clock (Clk) would have a period of 20 time units

How to Generate a Clock Signal in ModelSim