- **Sequential Basics**: Storage elements Counters[book 2], Sequential circuit timing : Propagation Delays, Setup, and Hold Times, Timing Conditions for Proper Operations, Glitches In Sequential Circuits, Synchronous Design. Tristate Logic and Busses [book 1].
- Design flow of ASIC and FPGA based systems. Logic Synthesis, Synthesis Design Flow Coding guidelines[book1,2,3,6]

- Sequential circuits are the mainstay of digital systems.
- Sequential circuit elements that are widely used in digital systems for storing information and for counting events. We then see how a system can be built from two main sections: a data- path and a control section.
- Discussion of a clocked synchronous timing methodology based on the abstraction of discrete time. This methodology is central to design of complex digital systems.

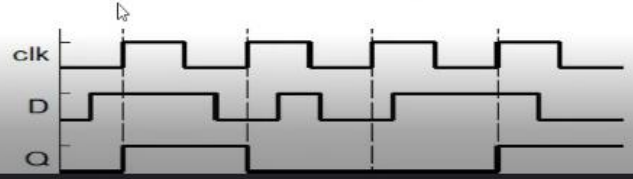
MADHURA R

S TO R A GE EL EMEN TS

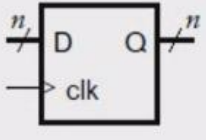
- Sequential circuit as one whose outputs depend not only on the current values of inputs, but also on the previous values of inputs.
- Sequential circuits are commonly regu- lated by a periodic clock signal that divides the passage of time into dis- crete clock cycles.

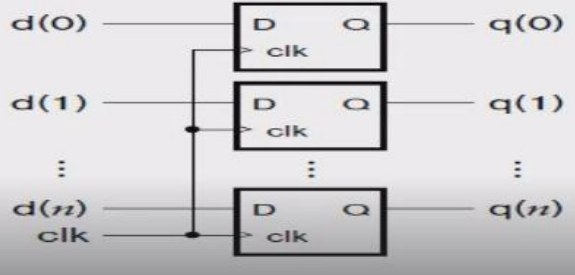
D Flip Flop

- The flip-flop is edge-triggered, meaning that on each rising edge of the clk input, the current value of the D input is stored within the flip-flop and reflected on the Q output. We illustrated use of D flip-flops in sequential circuits in where we stored the previous two values of an input signal on successive clock edges so that we could detect a given sequence of input values.
- While it is possible to implement a flip-flop as a combination of gates, it is not very instructive to do so. Moreover, flip-flops are provided as
-

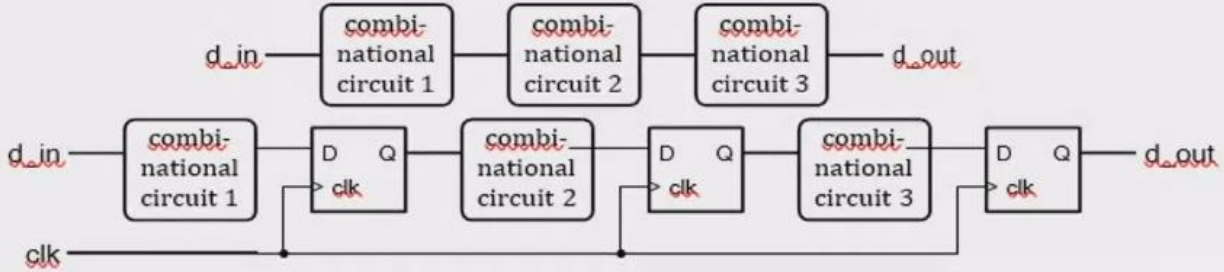


- A group of flip-flops used in this way is called a *register*. Each flip-flop in the register stores one bit of the code word of the stored value, as shown in Figure . The circuit at the top of the figure shows that each bit of an input and an output signal is connected to the input and output, respectively, of one of the flip-flops, and that the clock signal is connected in common to the clock input of all of the flip-flops. When there is a rising edge on the clock input, each flip- flop in the register updates its stored bit from the signal connected to its data input and drives the new value on its data output. The symbol for the register is shown





Circuit composed of combinational subcircuits (top), and a pipeline containing the same subcircuits.

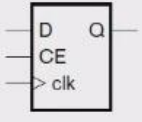


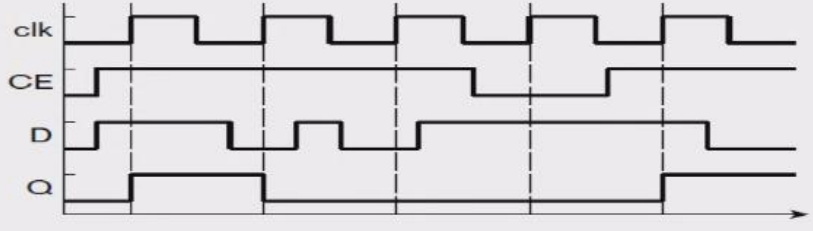
One use for a register constructed from simple D flip-flops is as a *pipeline register* in a sequential design., focusing on the use of pipelining as a technique for improving performance of a digital system. For now, consider the circuit outlined at the top of Figure . Successive values of data arriving at the input are processed by a number of combinational subcircuits, for example, by arithmetic subcircuits built from components described in Chapter 3. The total propagation delay of the circuit is the sum of the propagation delays of the individual subcircuits. This total delay must be less than the interval between arriving data values, otherwise data values may be lost. If the total delay is too long, we can divide the circuit into segments by inserting a register after each subcircuit, as shown

EXAMPLE 4.1 Develop a Verilog model for a pipelined circuit that computes the average of corresponding values in three streams of input values, a, b and c. The pipeline consists of three stages: the first stage sums values of a and b and saves the value of c; the second stage adds on the saved value of c; and the third stage divides by three. The inputs and output are all signed fixed-point numbers indexed from 5 down to −8.

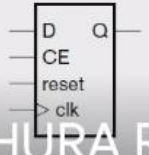
```
module average_pipeline ( output reg signed [5:-8] avg,
                          input      signed [5:-8] a, b, c,
                          input                    clk );

  wire signed [5:-8] a_plus_b, sum, sum_div_3;
  reg  signed [5:-8] saved_a_plus_b, saved_c, saved_sum;

  assign a_plus_b = a + b;

  always @(posedge clk) begin  // Pipeline register 1
    saved_a_plus_b <= a_plus_b;
    saved_c        <= c;
  end

  assign sum = saved_a_plus_b + saved_c;

  always @(posedge clk)  // Pipeline register 2
    saved_sum <= sum;

  assign sum_div_3 = saved_sum * 14'b00000001010101;

  always @(posedge clk)  // Pipeline register 3
    avg <= sum_div_3;

endmodule
```

A D flip-flop with clock-enable input. With Timing Diagram





A D flip-flop with clock-enable and reset inputs.




```
always @(posedge clk)
  if (ce) q <= d;
```

MADHURA R

EXAMPLE 4.2 Develop a Verilog model for an accumulator that calculates the sum of a sequence of fixed-point numbers. Each input number is signed with 4 pre-binary-point and 12 post-binary-point bits. The accumulated sum has 8 pre-binary-point and 12 post-binary-point bits. A new number arrives at the input during a clock cycle when the data_en control input is 1. The accumulated sum is cleared to 0 when the reset control input is 1. Both control inputs are synchronous.

SOLUTION The module requires a clock input, two control inputs, a data input and a data output, as follows:

```
module accumulator
  ( output reg signed [7:-12] data_out,
    input      signed [3:-12] data_in,
    input                     data_en, clk, reset );

  wire signed [7:-12] new_sum;

  assign new_sum = data_out + data_in;

  always @(posedge clk)
    if      (reset)   data_out <= 20'b0;
    else if (data_en) data_out <= new_sum;

endmodule
```

EXAMPLE 4.3 The symbol in Figure 4.10 shows a negative-edge-triggered flip-flop with clock enable, negative-logic asynchronous preset and clear, and both active-high and active-low outputs. It is illegal for both preset and clear to be active together. Develop a Verilog model for this flip-flop.

SOLUTION The module definition is

```
module flip_flop_n ( output reg Q,
                     output     Q_n,
                     input      pre_n, clr_n, D,
                     input      clk_n, CE );

  always @( negedge clk_n or
            negedge pre_n or negedge clr_n ) begin
    if (!pre_n && !clr_n)
      $display("Illegal inputs: pre_n and clr_n both 0");
    if      (!pre_n) Q <= 1'b1;
    else if (!clr_n) Q <= 1'b0;
    else if (CE)     Q <= D;
  end

  assign Q_n = ~Q;
endmodule
```