Chapter 5

Flip-Flops, Registers, and Counters

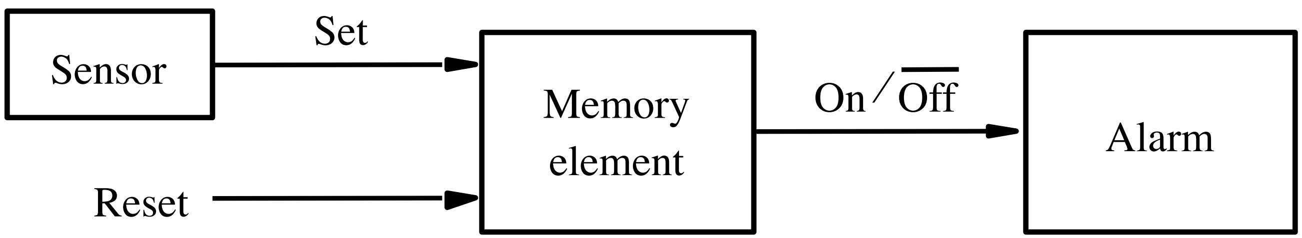


Figure 5.1. Control of an alarm system.

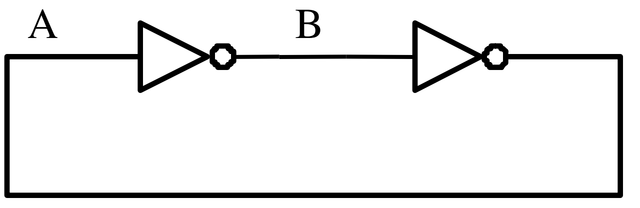


Figure 5.2.   A simple memory element.

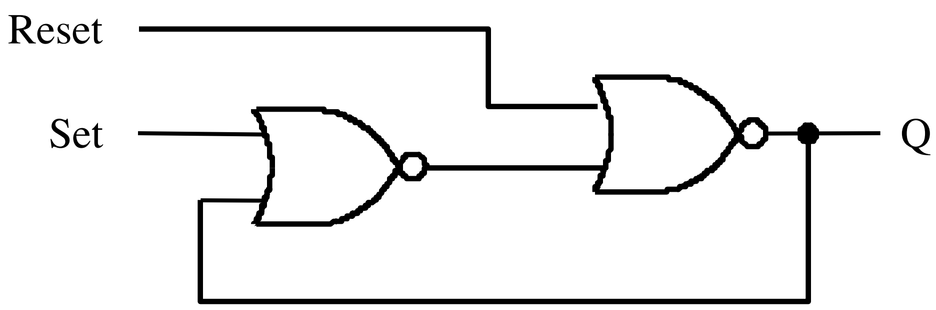
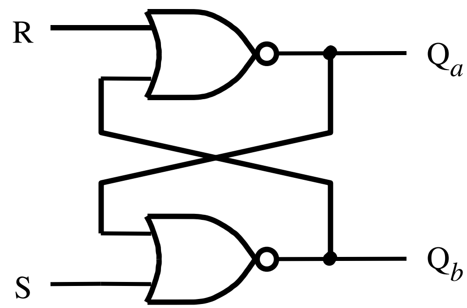


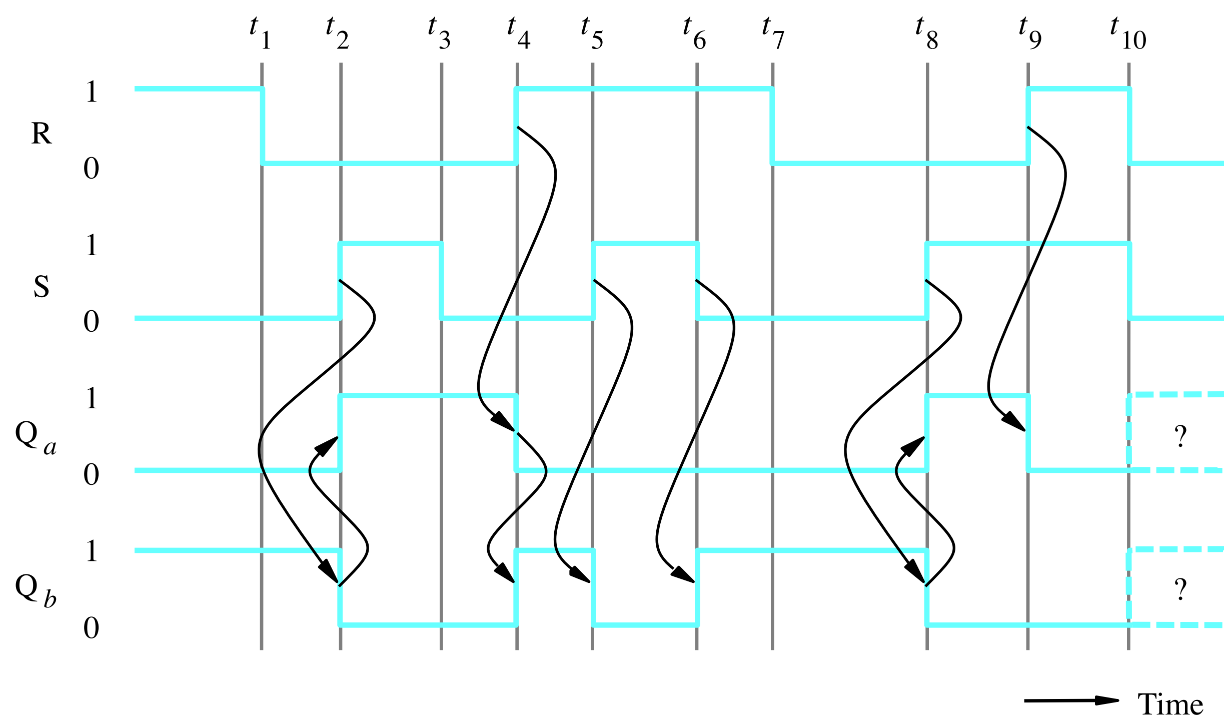
Figure 5.3. A memory element with NOR gates.



(a) Circuit

S	R	Q_a	Q_b	
0	0	0/1	1/0	(no change)
0	1	0	1	
1	0	1	0	
1	1	0	0	

(b) Truth table



(c) Timing diagram

Figure 5.4. A basic latch built with NOR gates.

Please see “portrait orientation” PowerPoint file for Chapter 5

Figure 5.5. Gated SR latch.

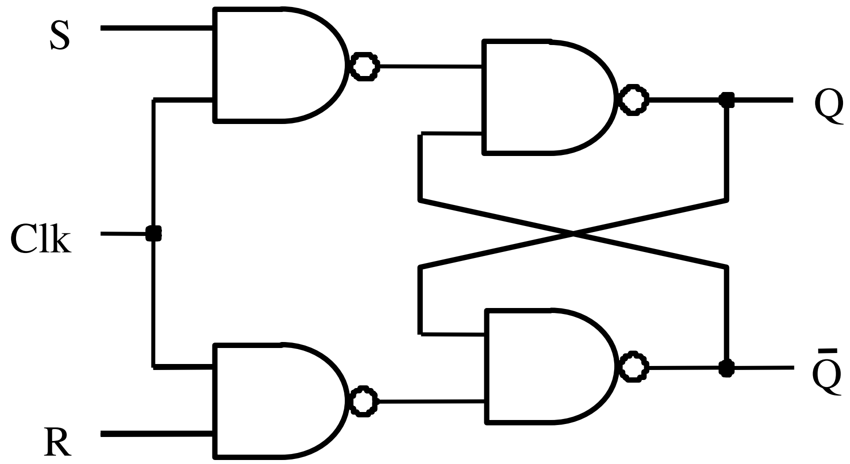


Figure 5.6. Gated SR latch with NAND gates.

Please see “portrait orientation” PowerPoint file for Chapter 5

Figure 5.7. Gated D latch.

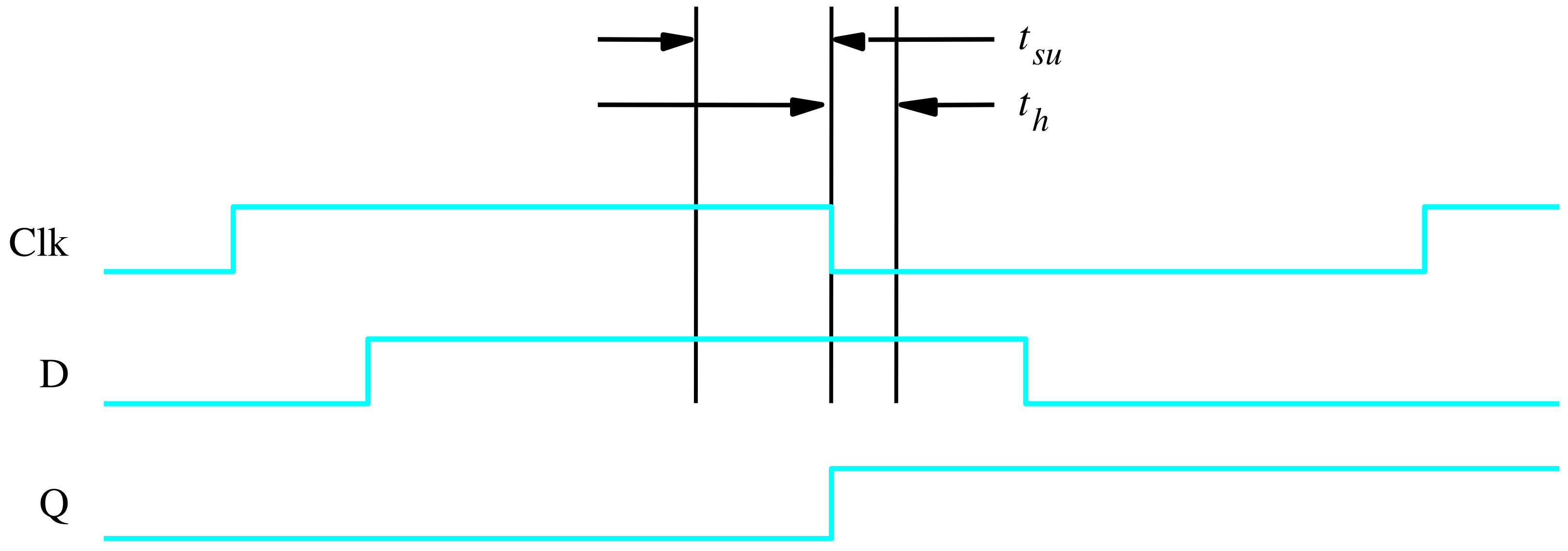


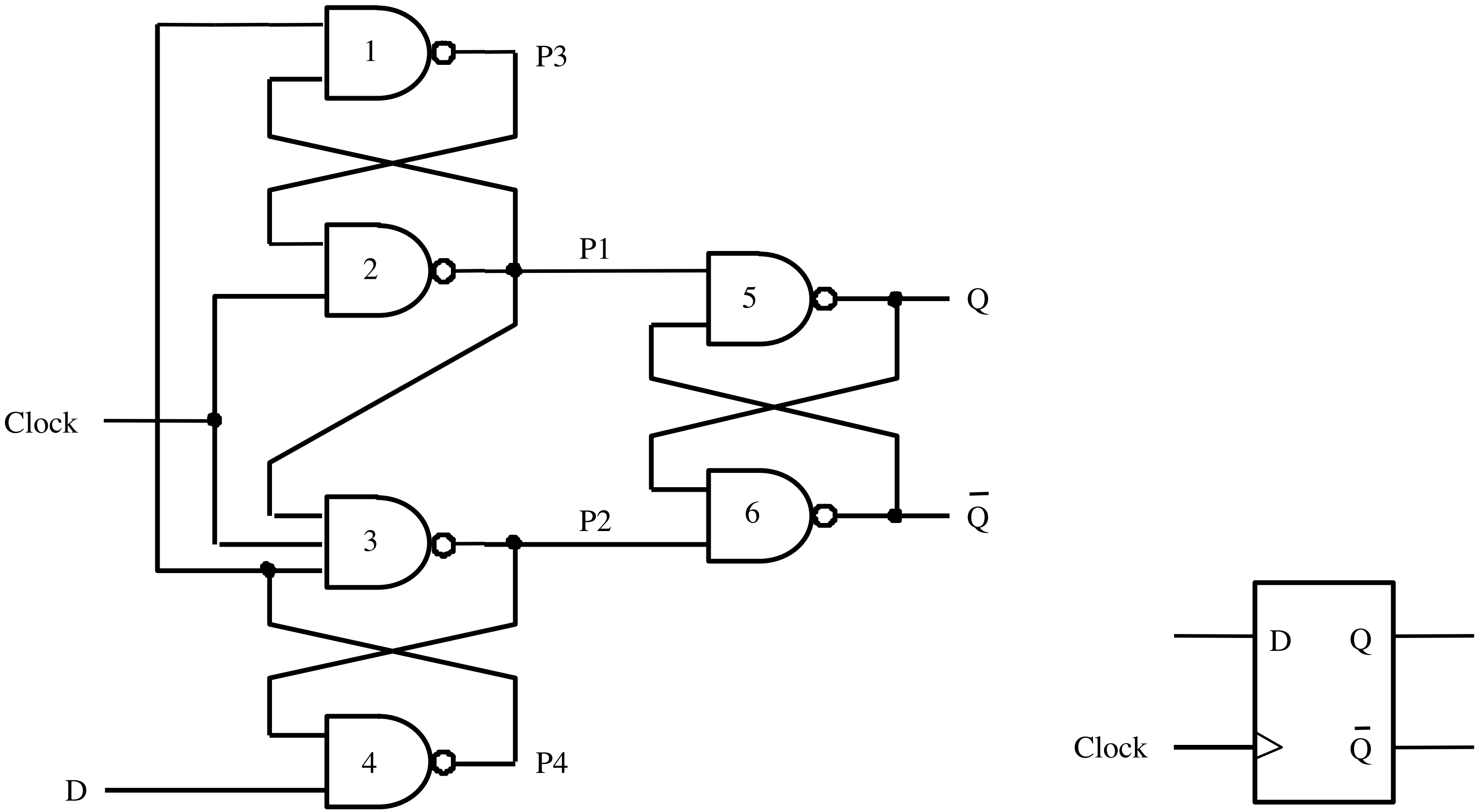
Figure 5.8. Setup and hold times.

Please see “portrait orientation” PowerPoint file for Chapter 5

Figure 5.9. Master-slave D flip-flop.

Please see “portrait orientation” PowerPoint file for Chapter 5

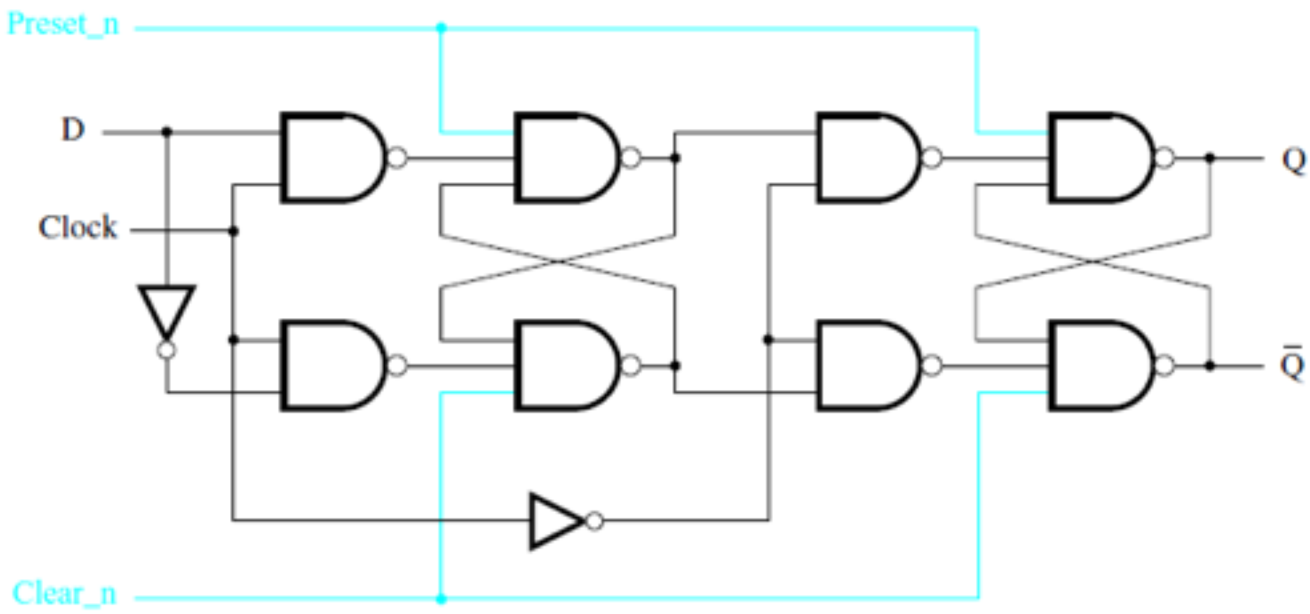
Figure 5.10. Comparison of level-sensitive and edge-triggered D storage elements.



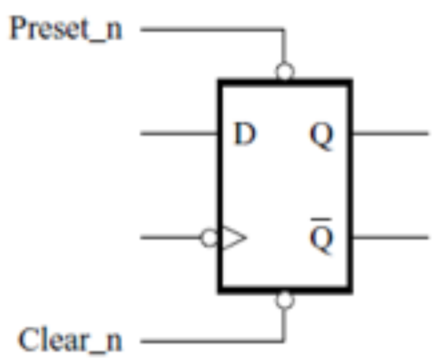
(a) Circuit

(b) Graphical symbol

Figure 5.11. A positive-edge-triggered D flip-flop.



(a) Circuit

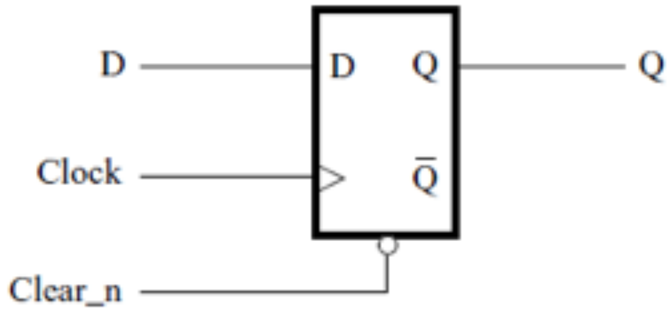


(b) Graphical symbol

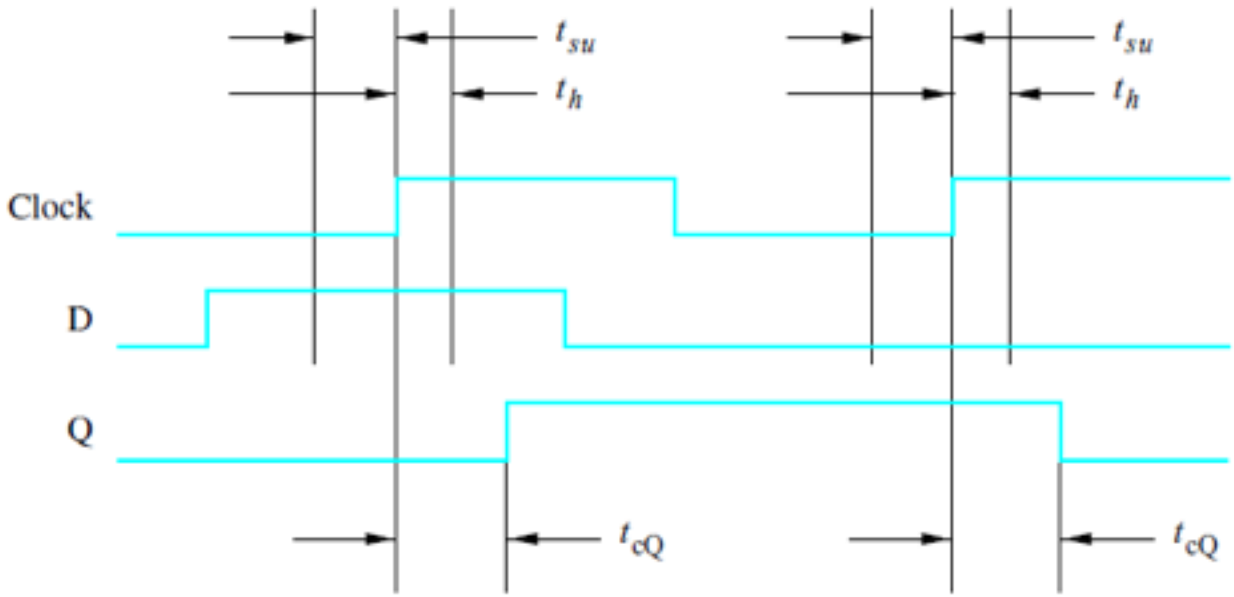
Figure 5.12. Master-slave D flip-flop with *Clear* and *Preset*.

Please see “portrait orientation” PowerPoint file for Chapter 5

Figure 5.13. Positive-edge-triggered D flip-flop with *Clear* and *Preset*.



(a) D flip-flop with asynchronous clear

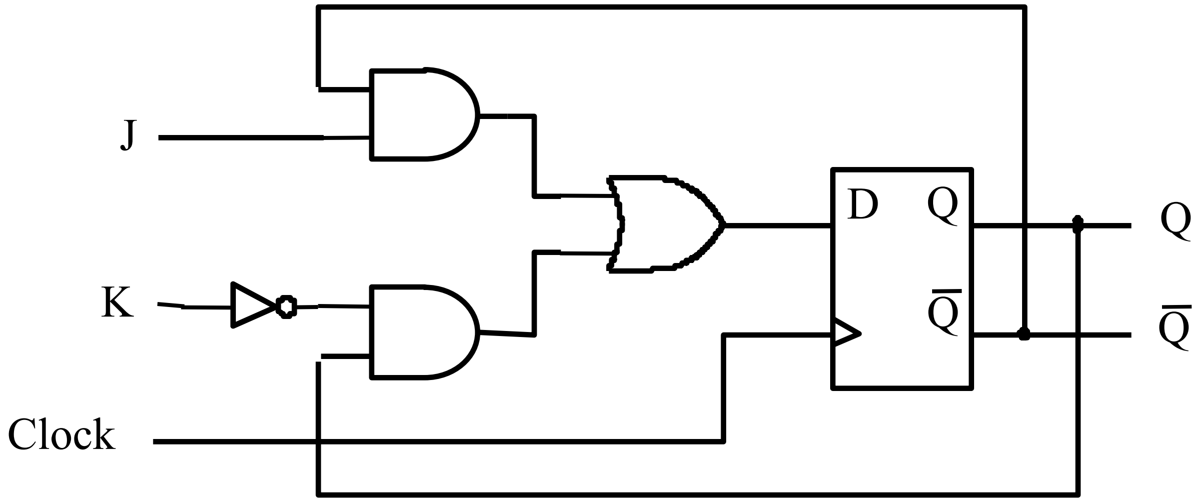


(b) Timing diagram

Figure 5.14. Timing for a flip-flop.

Please see “portrait orientation” PowerPoint file for Chapter 5

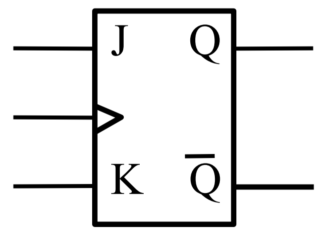
Figure 5.15. T flip-flop.



(a) Circuit

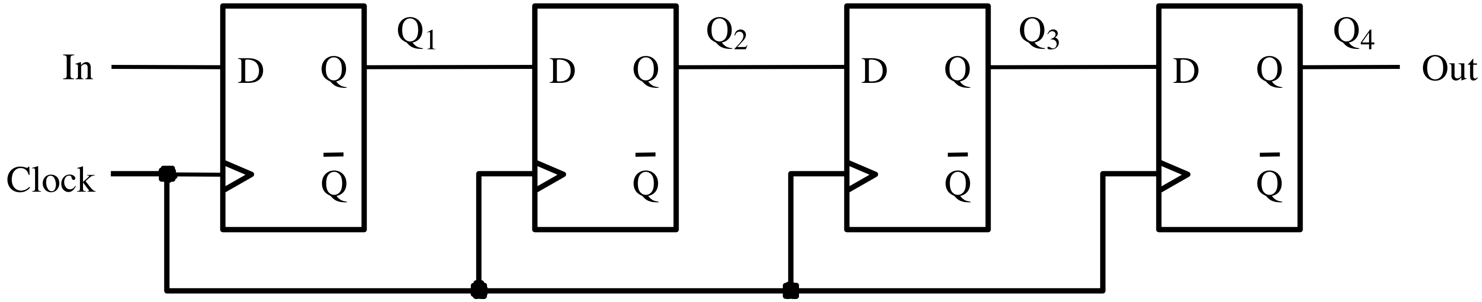
J	K	Q(t+1)
0	0	Q(t)
0	1	0
1	0	1
1	1	$\overline{Q}(t)$

(b) Truth table



(c) Graphical symbol

Figure 5.16. JK flip-flop.



(a) Circuit

	In	Q_1	Q_2	Q_3	Q_4 = Out
t_0	1	0	0	0	0
t_1	0	1	0	0	0
t_2	1	0	1	0	0
t_3	1	1	0	1	0
t_4	1	1	1	0	1
t_5	0	1	1	1	0
t_6	0	0	1	1	1
t_7	0	0	0	1	1

(b) A sample sequence

Figure 5.17. A simple shift register.

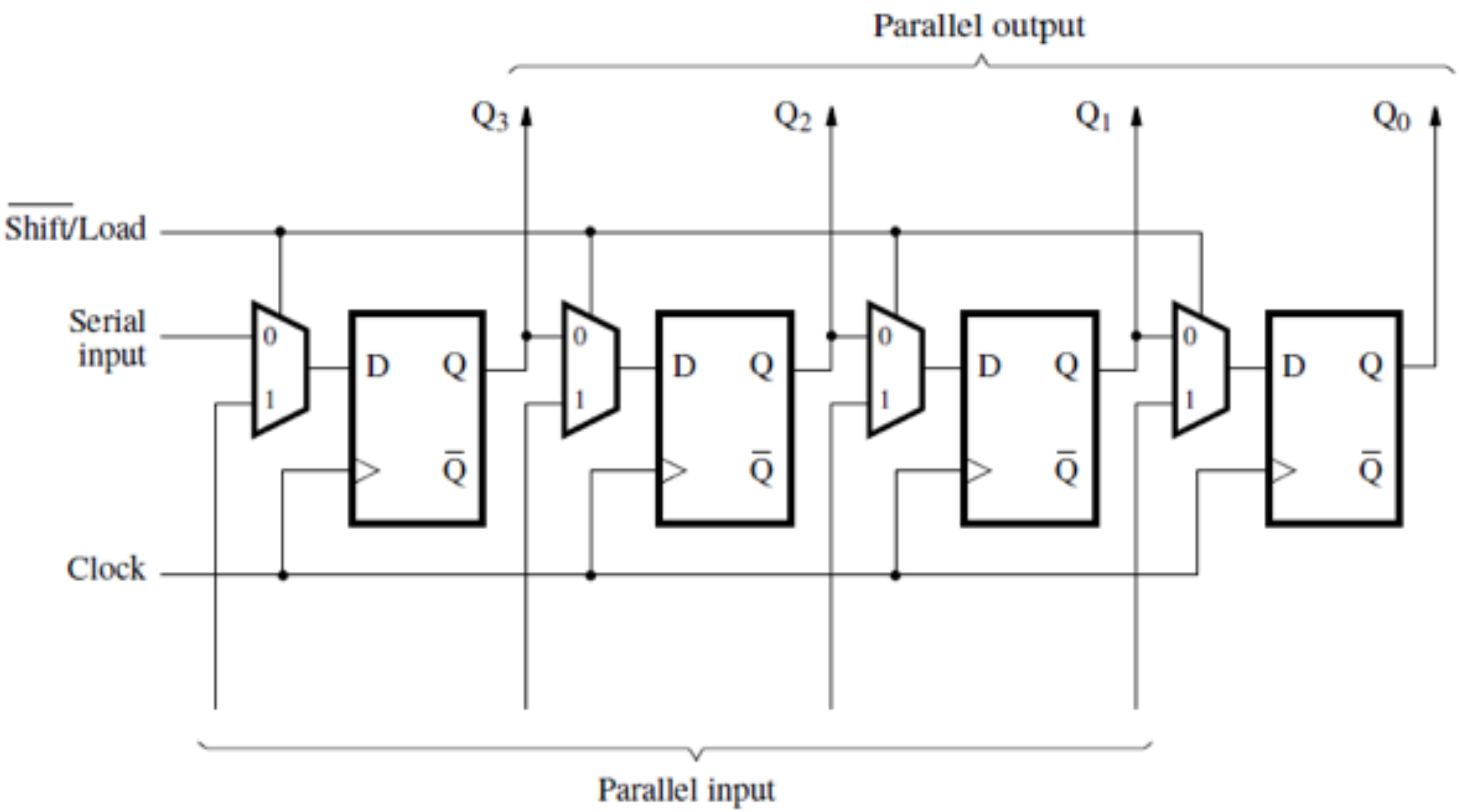
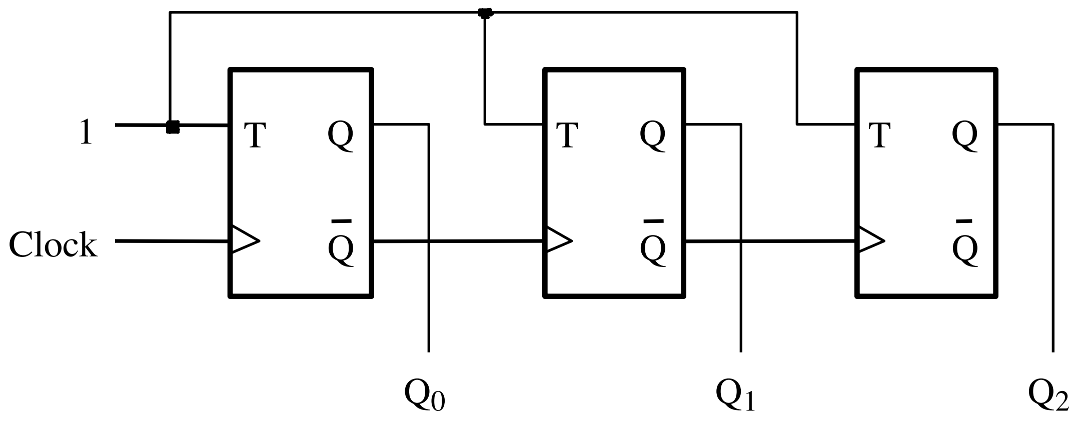
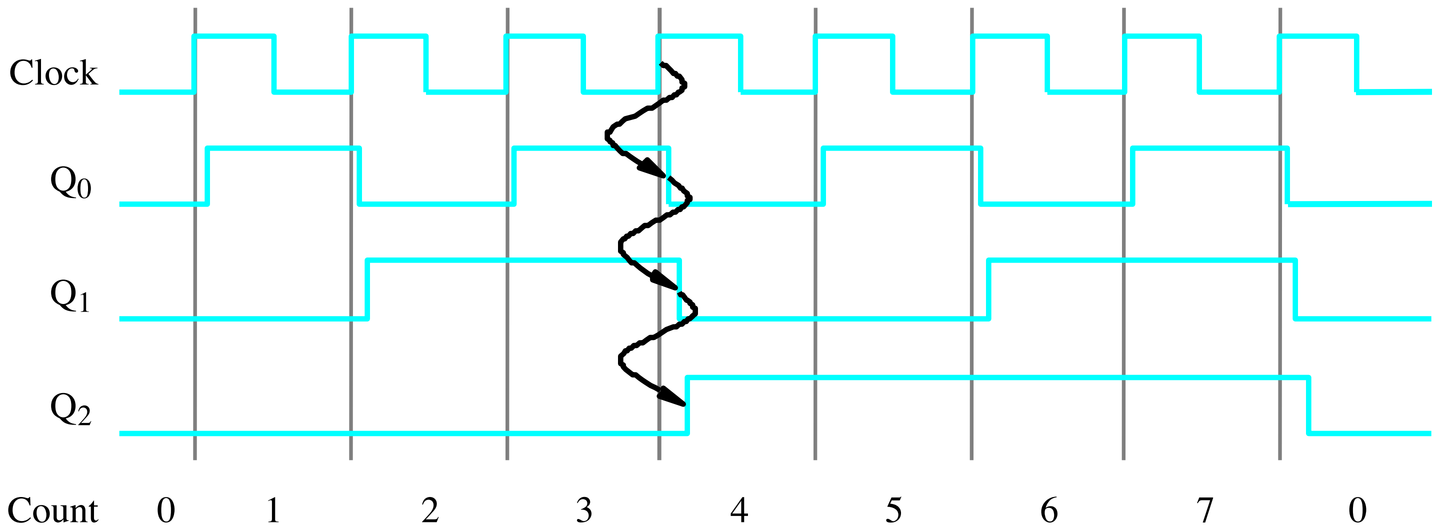


Figure 5.18. Parallel-access shift register.

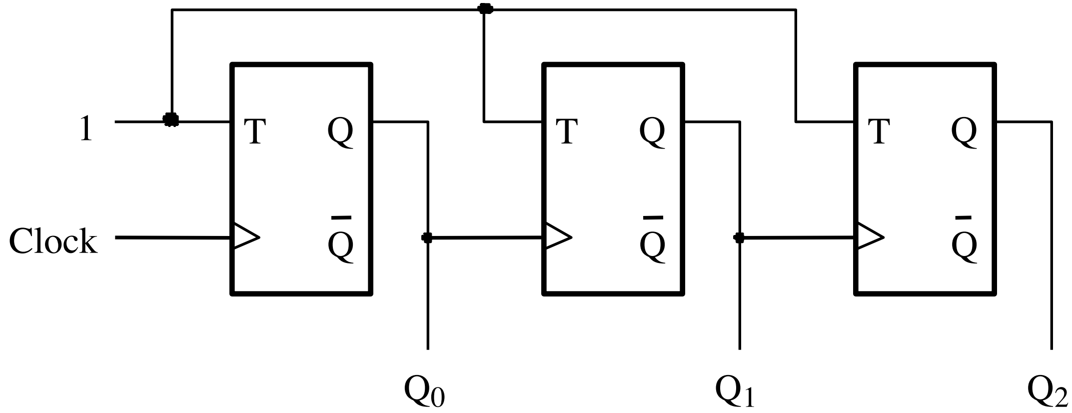


(a) Circuit

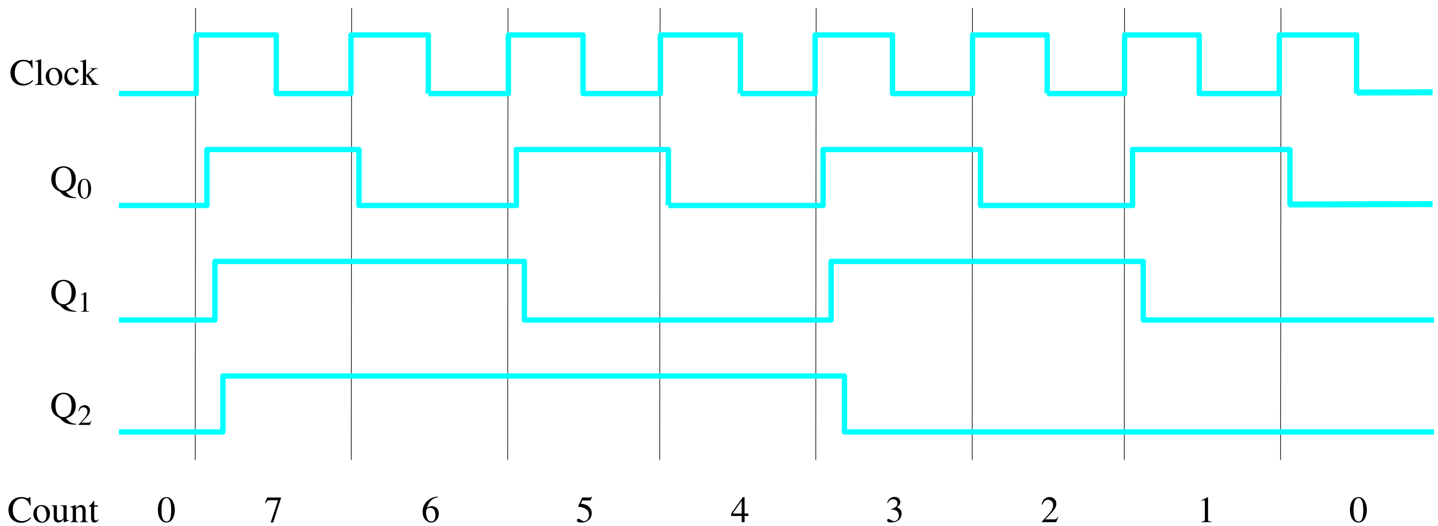


(b) Timing diagram

Figure 5.19. A three-bit up-counter.



(a) Circuit



(b) Timing diagram

Figure 5.20. A three-bit down-counter.

Clock cycle	Q_2	Q_1	Q_0
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1
8	0	0	0

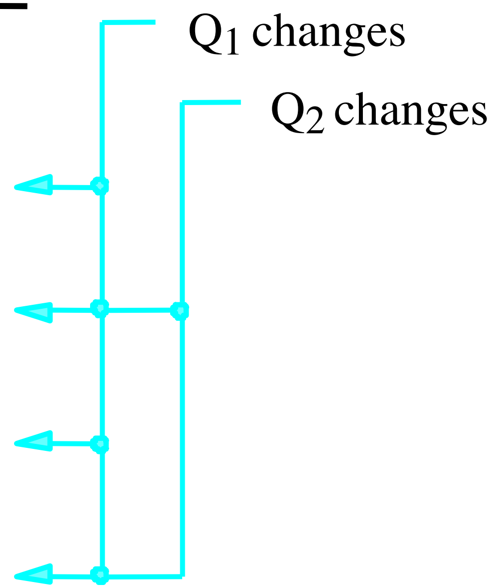
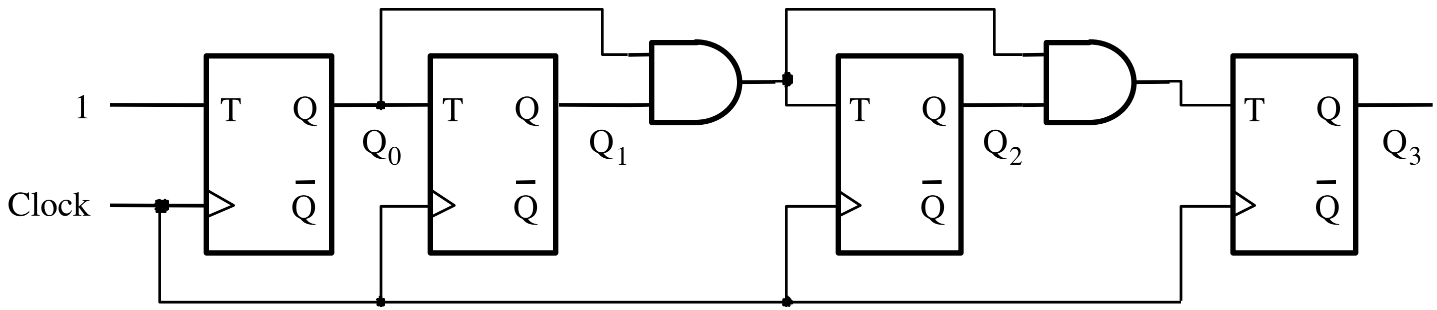
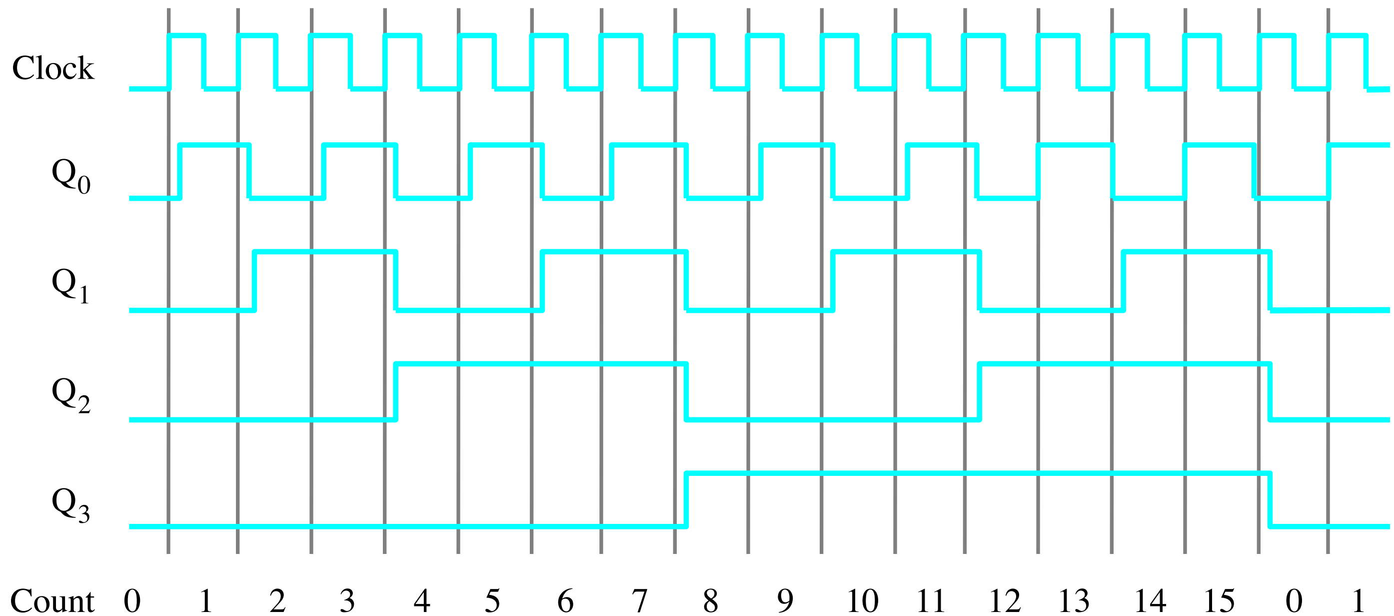


Table 5.1. Derivation of the synchronous up-counter.



(a) Circuit



(b) Timing diagram

Figure 5.21. A four-bit synchronous up-counter.

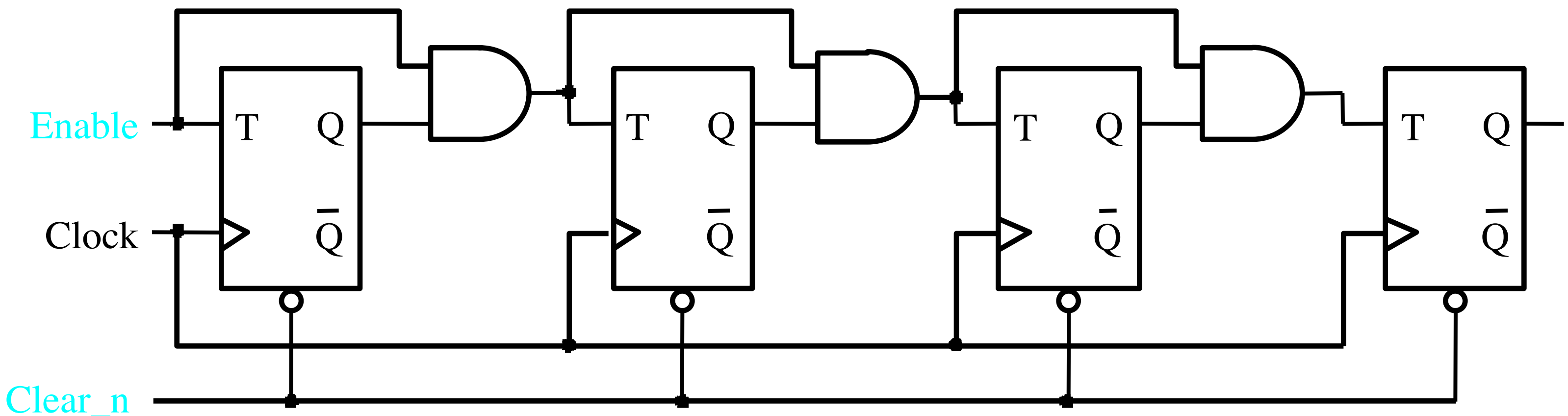


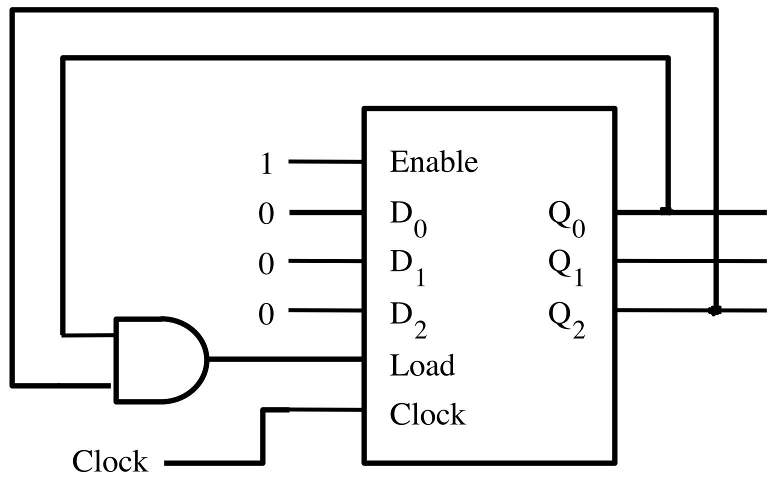
Figure 5.22. Inclusion of Enable and Clear capability.

Please see “portrait orientation” PowerPoint file for Chapter 5

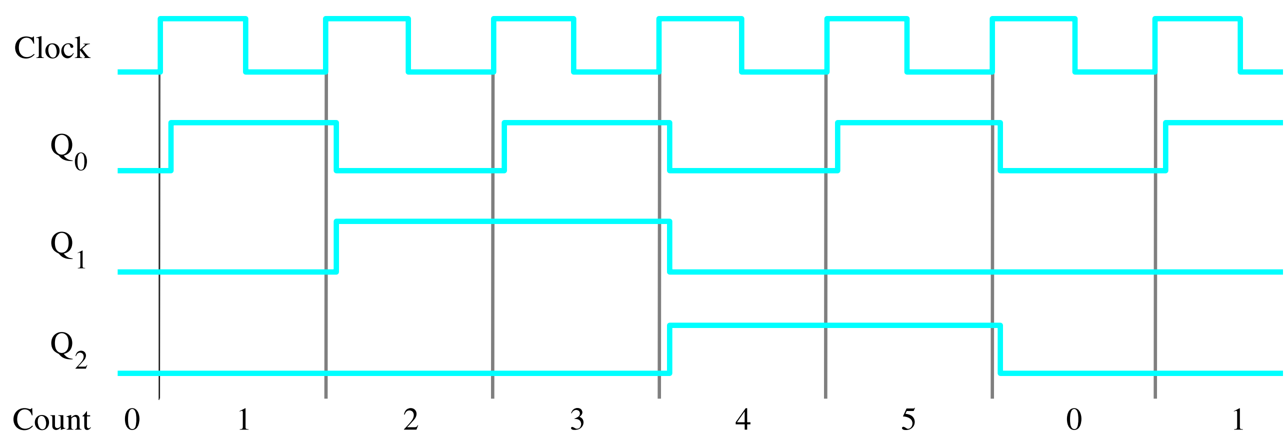
Figure 5.23. A four-bit counter with D flip-flops.

Please see “portrait orientation” PowerPoint file for Chapter 5

Figure 5.24. A counter with parallel-load capability.

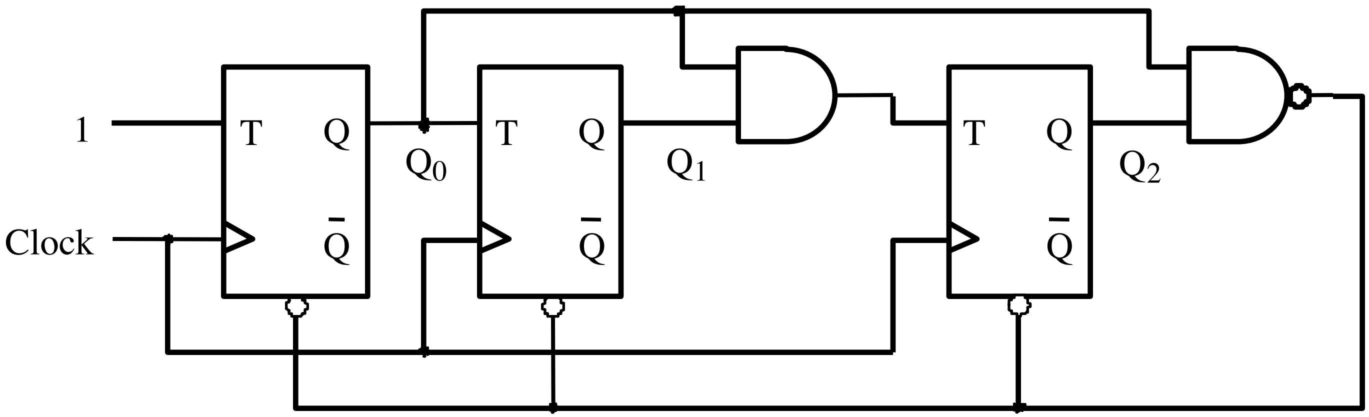


(a) Circuit

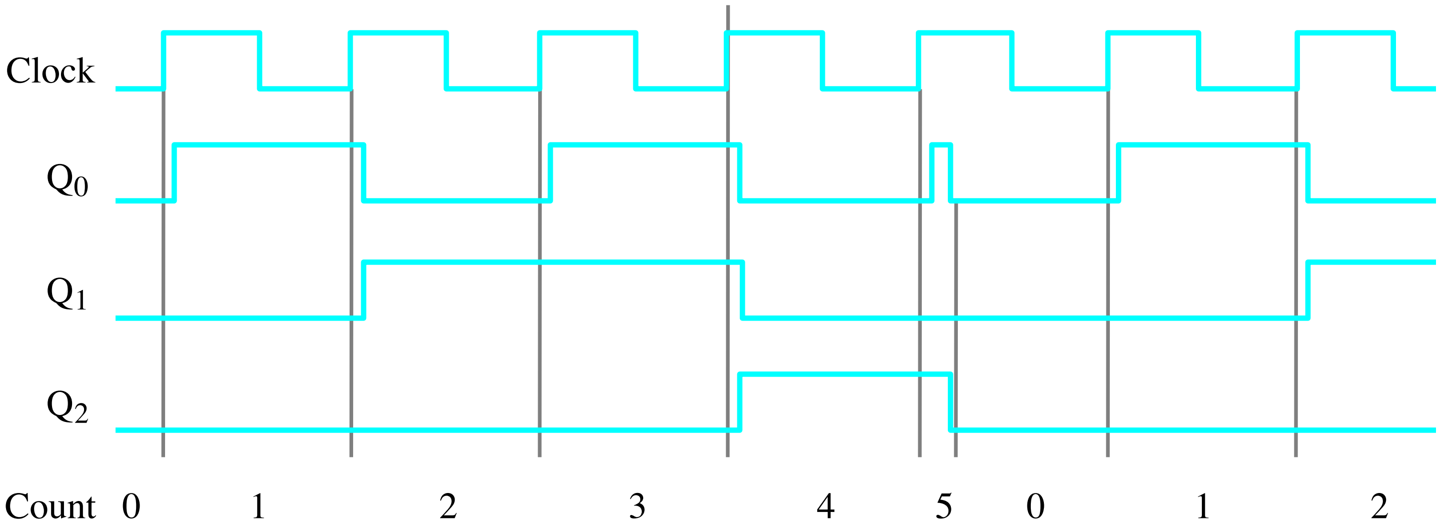


(b) Timing diagram

Figure 5.25. A modulo-6 counter with synchronous reset.



(a) Circuit



(b) Timing diagram

Figure 5.26. A modulo-6 counter with asynchronous reset.

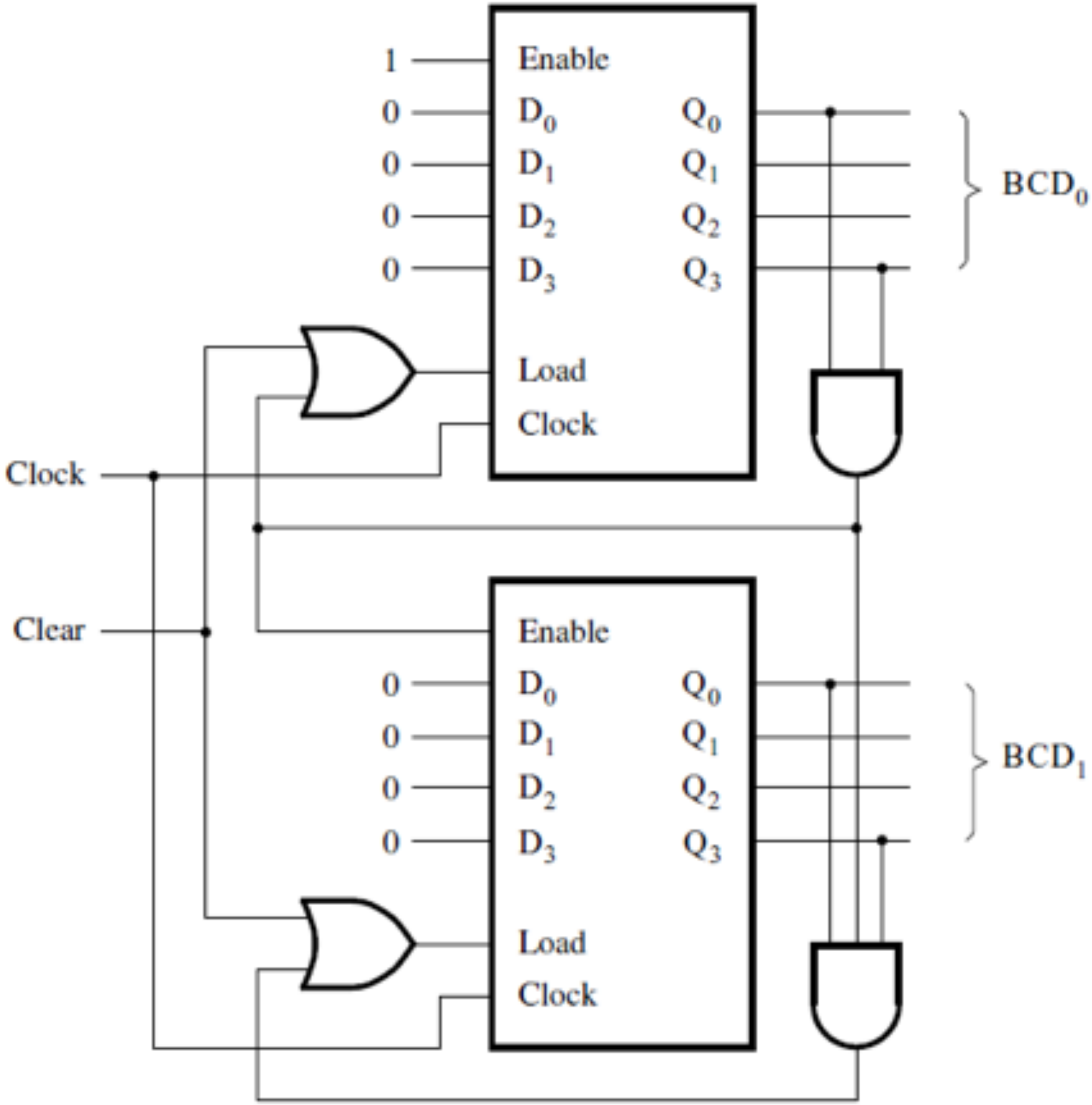


Figure 5.27. A two-digit BCD counter.

Please see “portrait orientation” PowerPoint file for Chapter 5

Figure 5.28. Ring counter.

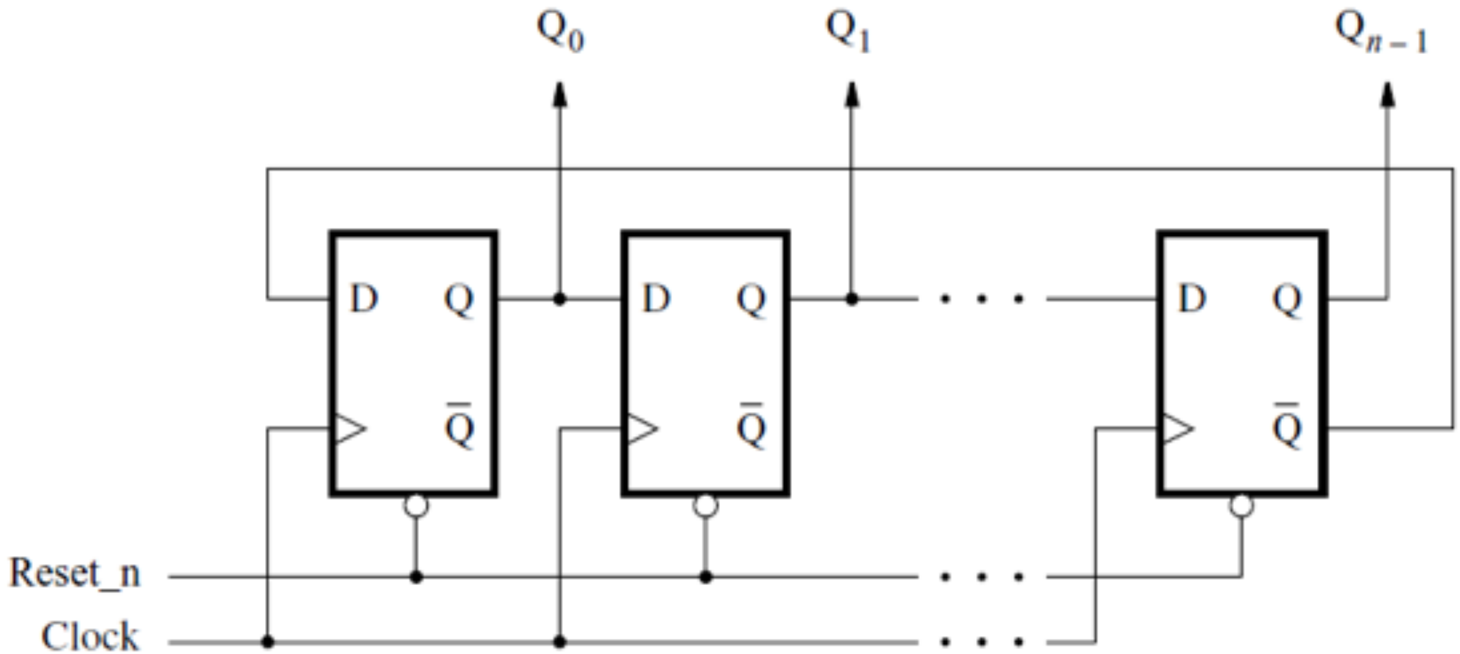


Figure 5.29. Johnson counter.

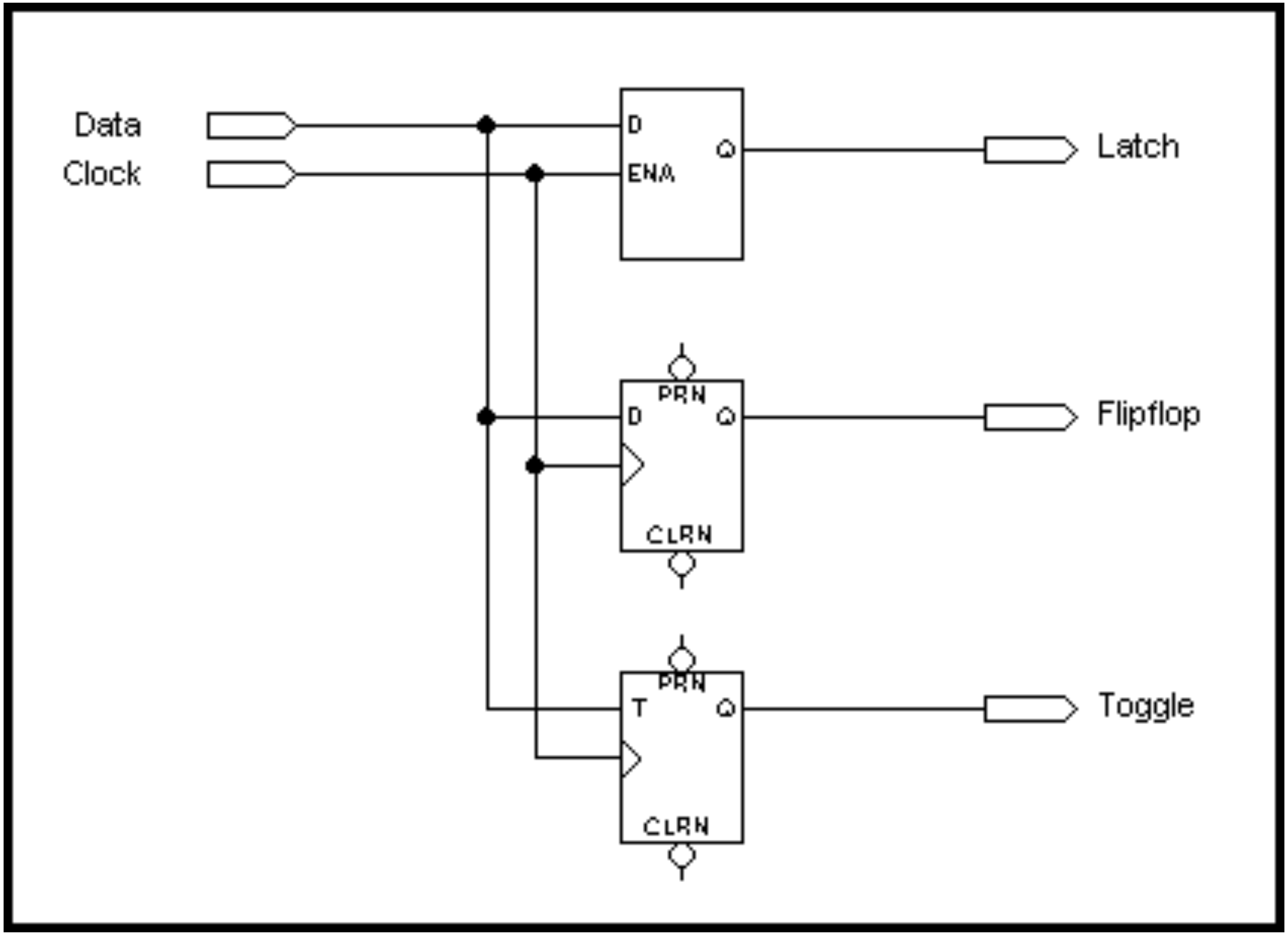


Figure 5.30. Three types of storage elements in a schematic.

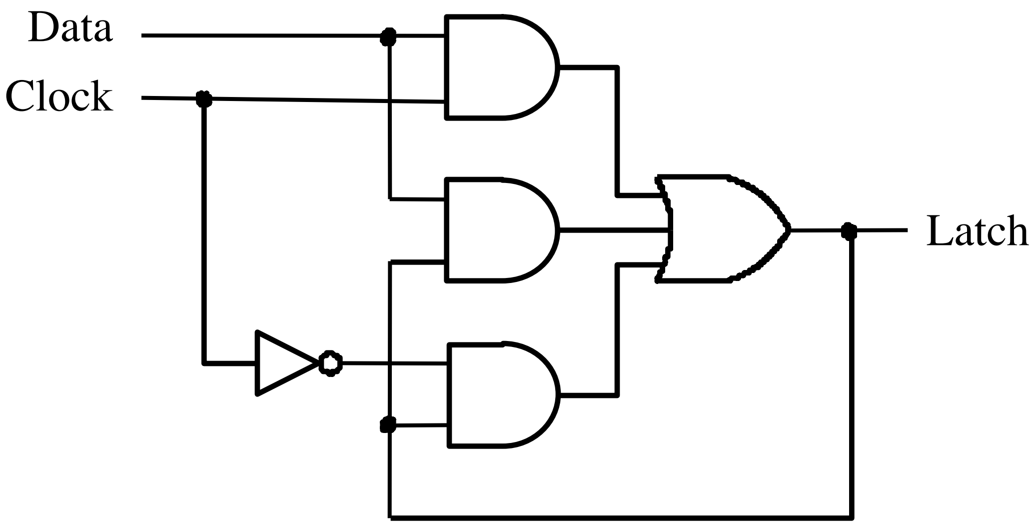


Figure 5.31. Gated D latch generated by CAD tools.

Please see “portrait orientation” PowerPoint file for Chapter 5

Figure 5.32. Implementation of the schematic in Figure 5.30 in a CPLD.

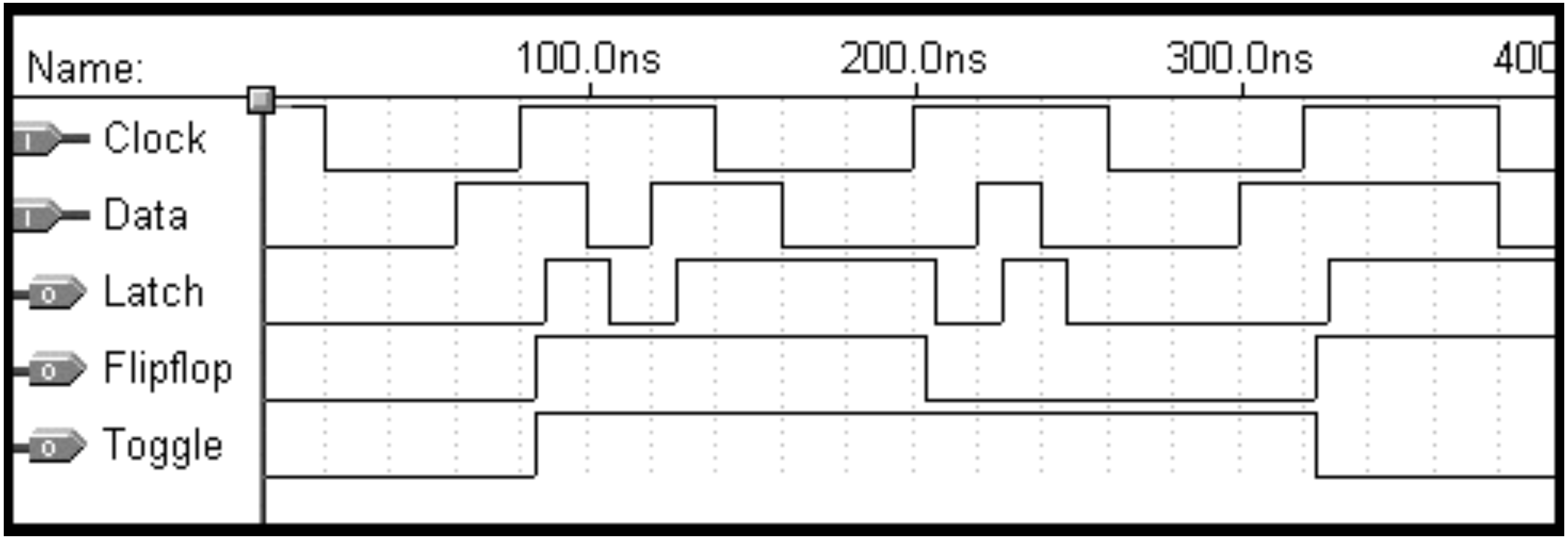


Figure 5.33. Timing simulation of storage elements in Figure 5.30.

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

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

endmodule
```

Figure 5.34. Code for a gated D latch.

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

	always @(posedge Clock)
		Q = D;

endmodule
```

Figure 5.35. Code for a D flip-flop.

```
module example5_3 (D, Clock, Q1, Q2);
	input D, Clock;
	output reg Q1, Q2;

	always @(posedge Clock)
	begin
		Q1 = D;
		Q2 = Q1;
	end

endmodule
```

Figure 5.36. Incorrect code for two cascaded flip-flops.

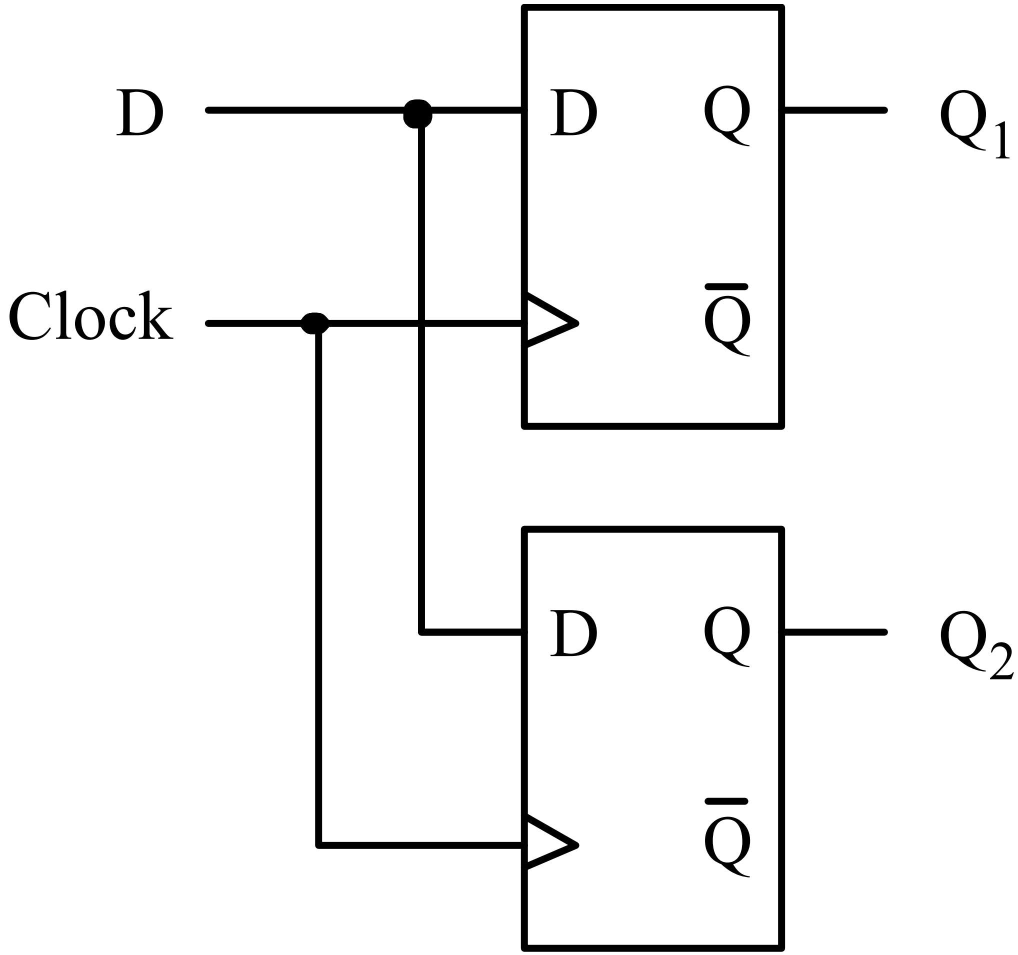


Figure 5.37. Circuit for Example 5.3.

```
module example5_4 (D, Clock, Q1, Q2);
	input D, Clock;
	output reg Q1, Q2;

	always @(posedge Clock)
	begin
		Q1 <= D;
		Q2 <= Q1;
	end

endmodule
```

Figure 5.38. Code for two cascaded flip-flops.

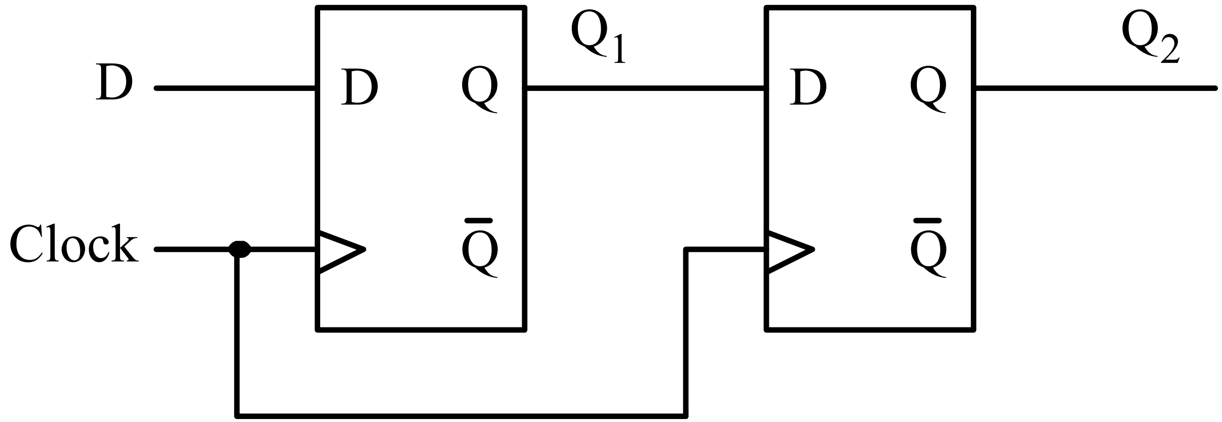


Figure 5.39. Circuit defined in Figure 5.38.

```
module example5_5 (x1, x2, x3, Clock, f, g);
	input x1, x2, x3, Clock;
	output reg f, g;

	always @(posedge Clock)
	begin
		f = x1 & x2;
		g = f | x3;
	end

endmodule
```

Figure 5.40. Code for Example 5.5.

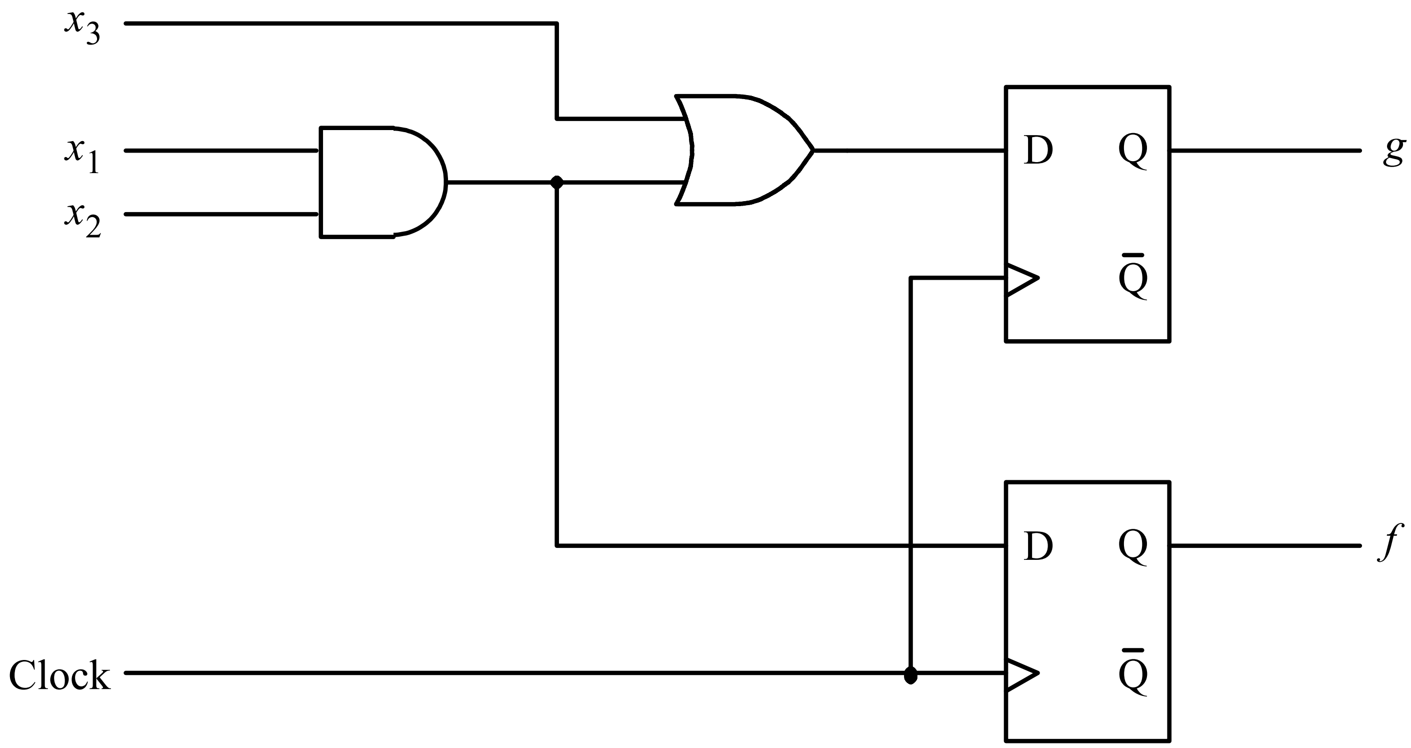


Figure 5.41. Circuit for Example 5.5.

```
module example5_6 (x1, x2, x3, Clock, f, g);
	input x1, x2, x3, Clock;
	output reg f, g;

	always @(posedge Clock)
	begin
		f <= x1 & x2;
		g <= f | x3;
	end

endmodule
```

Figure 5.42. Code for Example 5.6.

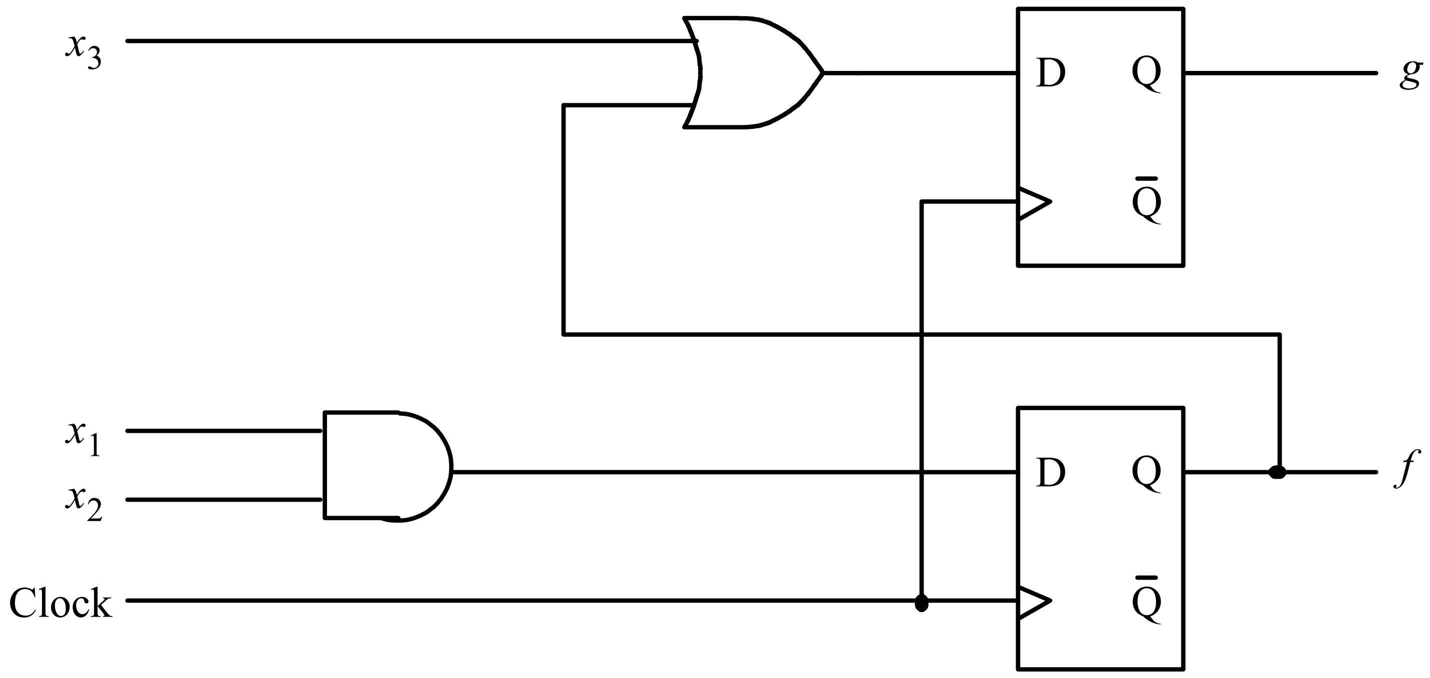


Figure 5.43. Circuit for Example 5.6.

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

	always @(negedge Resetn, posedge Clock)
		if (!Resetn)
		   Q <= 0;
		else
		   Q <= D;

endmodule
```

Figure 5.44. D flip-flop with asynchronous reset.

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

	always @(posedge Clock)
		if (!Resetn)
			Q <= 0;
		else
			Q <= D;

endmodule
```

Figure 5.45. D flip-flop with synchronous reset.

```
module regn (D, Clock, Resetn, Q);
	parameter n = 16;
	input [n-1:0] D;
	input Clock, Resetn;
	output reg [n-1:0] Q;

	always @(negedge Resetn, posedge Clock)
		if (!Resetn)
			Q <= 0;
		else
			Q <= D;

endmodule
```

Figure 5.46. Code for an *n*-bit register with asynchronous clear.

```
module muxdff (D0, D1, Sel, Clock, Q);
    input D0, D1, Sel, Clock;
    output reg Q;

    always @(posedge Clock)
        if (!Sel)
            Q <= D0;
        else
            Q <= D1;

endmodule
```

Figure 5.47. Code for a D flip-flop with a 2-to-1 multiplexer on the D input.

```
module muxdff (D0, D1, Sel, Clock, Q);
    input D0, D1, Sel, Clock;
    output reg Q;

    wire D;
    assign D = Sel ? D1 : D0;

    always @(posedge Clock)
        Q <= D;

endmodule
```

Figure 5.48. Alternative code for a D flip-flop with a 2-to-1 multiplexer on the D input.

```
module shift4 (R, L, w, Clock, Q);
	input [3:0] R;
	input L, w, Clock;
	output [3:0] Q;
	wire [3:0] Q;

	muxdff Stage3 (w, R[3], L, Clock, Q[3]);
	muxdff Stage2 (Q[3], R[2], L, Clock, Q[2]);
	muxdff Stage1 (Q[2], R[1], L, Clock, Q[1]);
	muxdff Stage0 (Q[1], R[0], L, Clock, Q[0]);

endmodule
```

Figure 5.49. Hierarchical code for a four-bit shift register.

```
module shift4 (R, L, w, Clock, Q);
    input [3:0] R;
    input L, w, Clock;
    output reg [3:0] Q;

    always @(posedge Clock)
        if (L)
            Q <= R;
        else
        begin
            Q[0] <= Q[1];
            Q[1] <= Q[2];
            Q[2] <= Q[3];
            Q[3] <= w;
        end

endmodule
```

Figure 5.50. Alternative code for a four-bit shift register.

```
module shiftn (R, L, w, Clock, Q);
    parameter n = 16;
    input [n-1:0] R;
    input L, w, Clock;
    output reg [n-1:0] Q;
    integer k;

    always @(posedge Clock)
        if (L)
          Q <= R;
        else
        begin
          for (k = 0; k < n-1; k = k+1)
              Q[k] <= Q[k+1];
              Q[n-1] <= w;
        end

endmodule
```

Figure 5.51. An *n*-bit shift register.

```
module upcount (Resetn, Clock, E, Q);
    input Resetn, Clock, E;
    output reg [3:0] Q;

    always @(negedge Resetn, posedge Clock)
        if (!Resetn)
            Q <= 0;
        else if (E)
            Q <= Q + 1;

endmodule
```

Figure 5.52. Code for a four-bit up-counter.

```
module upcount (R, Resetn, Clock, E, L, Q);
    input [3:0] R;
    input Resetn, Clock, E, L;
    output reg [3:0] Q;

    always @(negedge Resetn, posedge Clock)
        if (!Resetn)
            Q <= 0;
        else if (L)
            Q <= R;
        else if (E)
            Q <= Q + 1;

endmodule
```

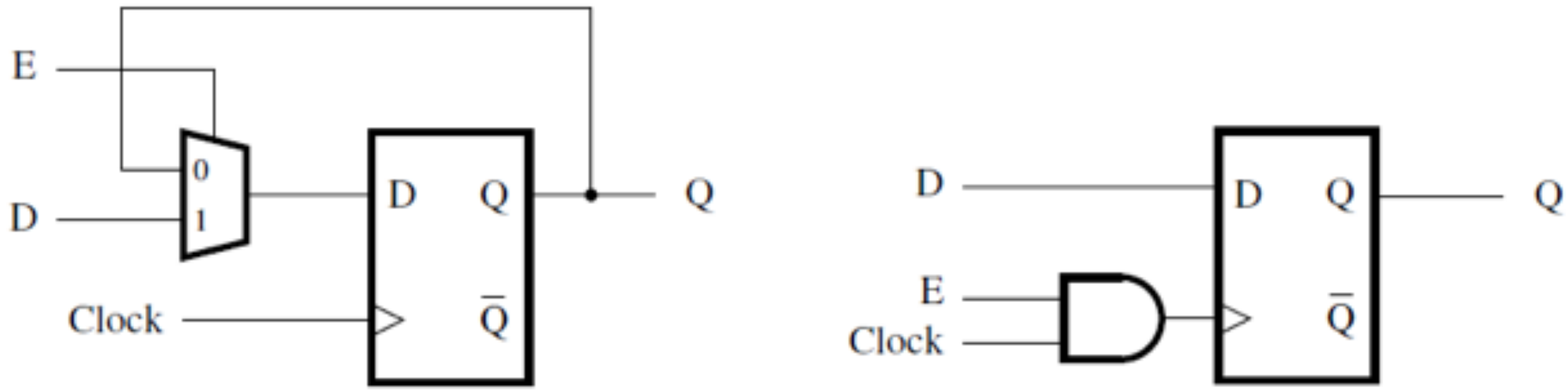
Figure 5.53. A four-bit up-counter with parallel load.

```
module downcount (R, Clock, E, L, Q);
	parameter n = 8;
	input [n-1:0] R;
	input Clock, L, E;
	output reg [n-1:0] Q;

	always @(posedge Clock)
		if (L)
			Q <= R;
		else if (E)
			Q <= Q - 1;

endmodule
```

Figure 5.54. A down-counter with a parallel load.

```
module updowncount (R, Clock, L, E, up_down, Q);
    parameter n = 8;
    input [n-1:0] R;
    input Clock, L, E, up_down;
    output reg [n-1:0] Q;
    always @(posedge Clock)
        if (L)
            Q <= R;
        else if (E)
            Q <= Q + (up_down ? 1 : -1);

endmodule
```

Figure 5.55. Code for an up/down counter.



(a) Using a multiplexer

(b) Clock gating

Figure 5.56. Providing an enable input for a D flip-flop.

```
module rege (D, Clock, Resetn, E, Q);
    input D, Clock, Resetn, E;
    output reg Q;

    always @(posedge Clock, negedge Resetn)
        if (Resetn == 0)
            Q <= 0;
        else if (E)
            Q <= D;

endmodule
```

Figure 5.57. Code for a D flip-flop with enable.

```
module regne (R, Clock, Resetn, E, Q);
    parameter n = 8;
    input [n-1:0] R;
    input Clock, Resetn, E;
    output reg [n-1:0] Q;

    always @(posedge Clock,  negedge Resetn)
        if (Resetn == 0)
            Q <= 0;
        else if (E)
            Q <= R;

endmodule
```

Figure 5.58. An *n*-bit register with an enable input.

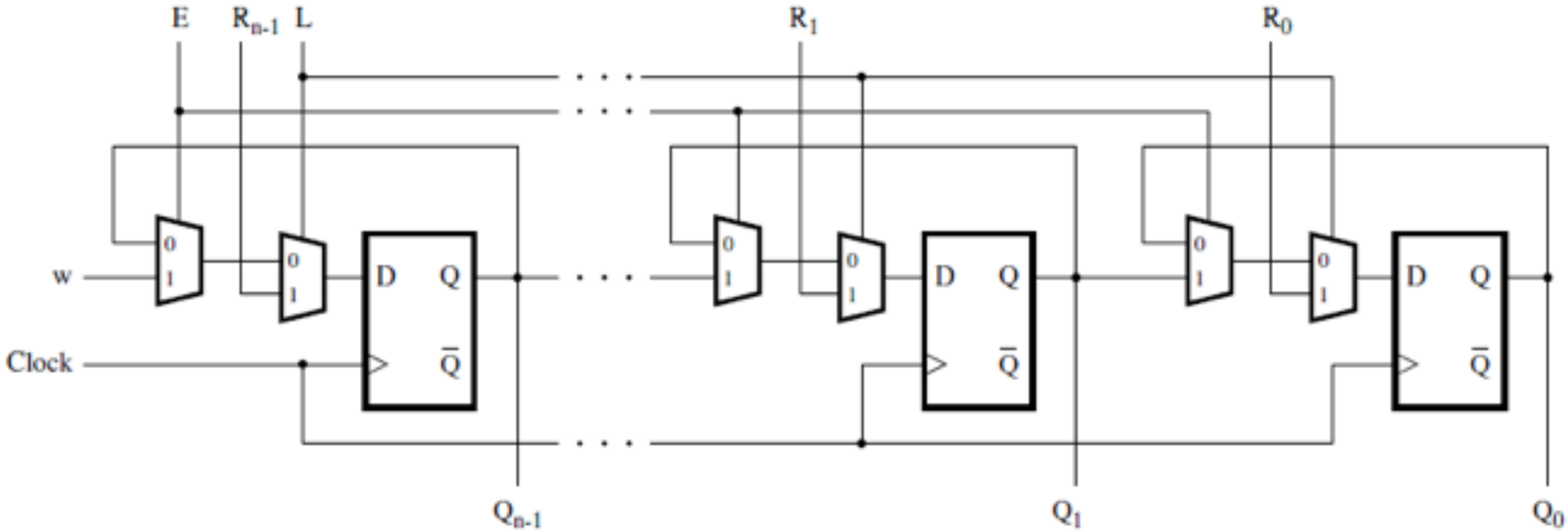


Figure 5.59. A shift register with parallel load and enable control inputs.

```
module shiftrne (R, L, E, w, Clock, Q);
  parameter n = 4;
  input [n-1:0] R;
  input L, E, w, Clock;
  output reg [n-1:0] Q;
  integer k;

  always @(posedge Clock)
  begin
    if (L)
      Q <= R;
    else if (E)
    begin
      Q[n-1] <= w;
    for (k = n-2; k >= 0; k = k-1)
      Q[k] <= Q[k+1];
    end
  end
endmodule
```

Figure 5.60. A left-to-right shift register with an enable input.

Please see "portrait orientation" PowerPoint file for Chapter 5

Figure 5.61. A reaction-timer circuit.

Please see “portrait orientation” PowerPoint file for Chapter 5

Figure 5.62. Code for the two-digit BCD counter in Figure 5.27.

```
module seg7 (bcd, leds);
  input [3:0] bcd;
  output reg [1:7] leds;

  always @(bcd)
    case (bcd)   //abcdefg
      0: leds = 7'b1111110;
      1: leds = 7'b0110000;
      2: leds = 7'b1101101;
      3: leds = 7'b1111001;
      4: leds = 7'b0110011;
      5: leds = 7'b1011011;
      6: leds = 7'b1011111;
      7: leds = 7'b1110000;
      8: leds = 7'b1111111;
      9: leds = 7'b1111011;
      default: leds = 7'bx;
    endcase

endmodule
```

Figure 5.63. Code for the BCD-to-7-segment decoder.

```
module reaction (Clock, Reset, c9, w, Pushn, LEDn, Digit1, Digit0);
    input Clock, Reset, c9, w, Pushn;
    output wire LEDn;
    output wire [1:7] Digit1, Digit0;
    reg LED;
    wire [3:0] BCD1, BCD0;

    always @(posedge Clock)
    begin
        if (!Pushn || Reset)
            LED <= 0;
        else if (w)
            LED <= 1;
    end

    assign LEDn = ~LED;
    BCDcount counter (c9, Reset, LED, BCD1, BCD0);
    seg7 seg1 (BCD1, Digit1);
    seg7 seg0 (BCD0, Digit0);

endmodule
```

Figure 5.64. Code for the reaction timer.

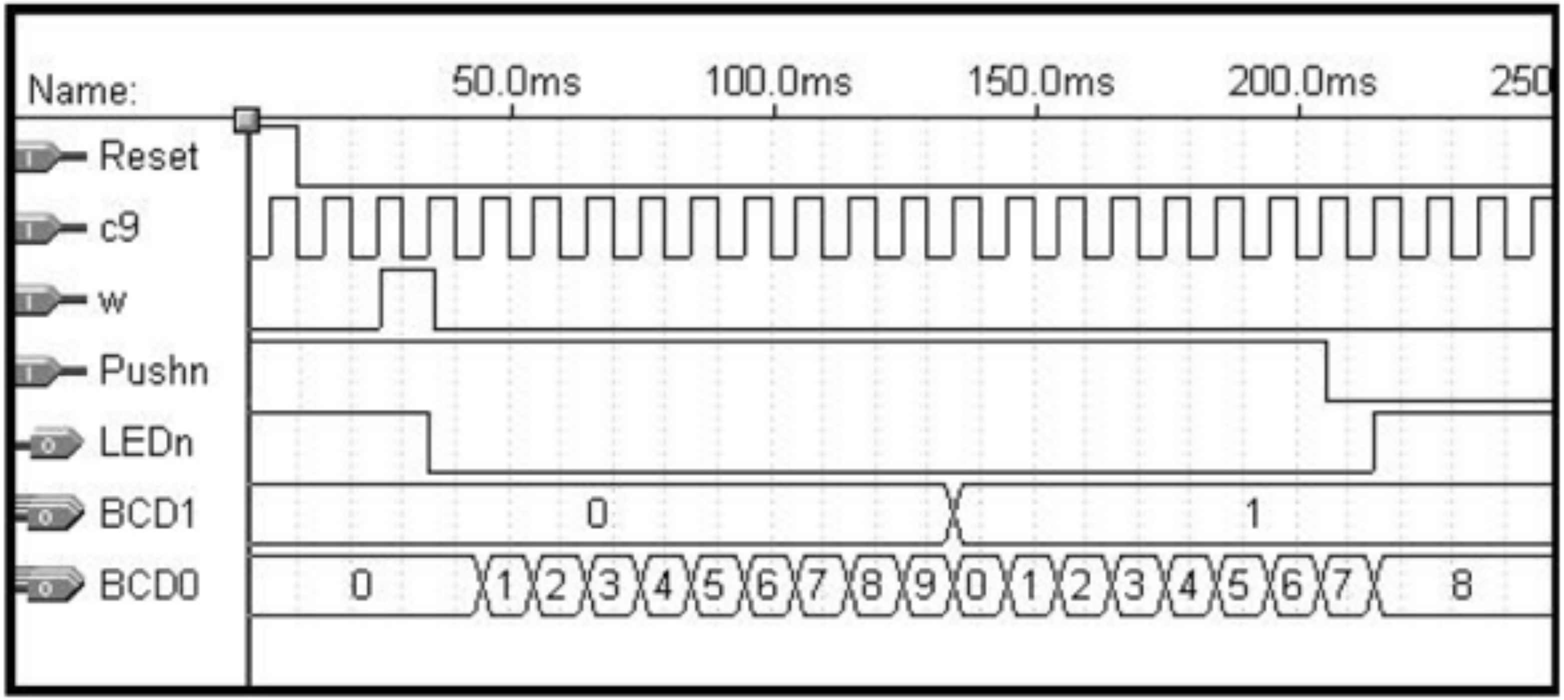


Figure 5.65. Simulation of the reaction-timer circuit.

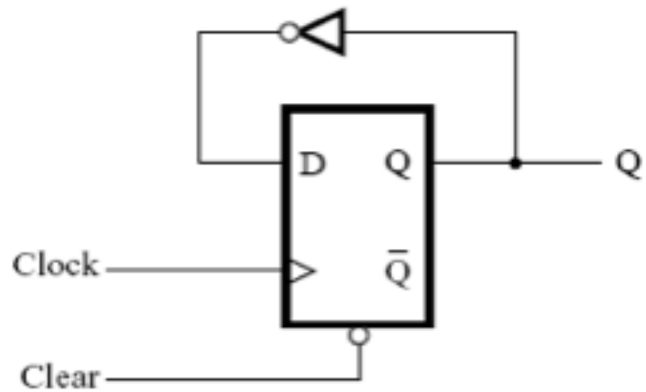


Figure 5.66. A simple flip-flop circuit.

Please see “portrait orientation” PowerPoint file for Chapter 5

Figure 5.67. A 4-bit counter.

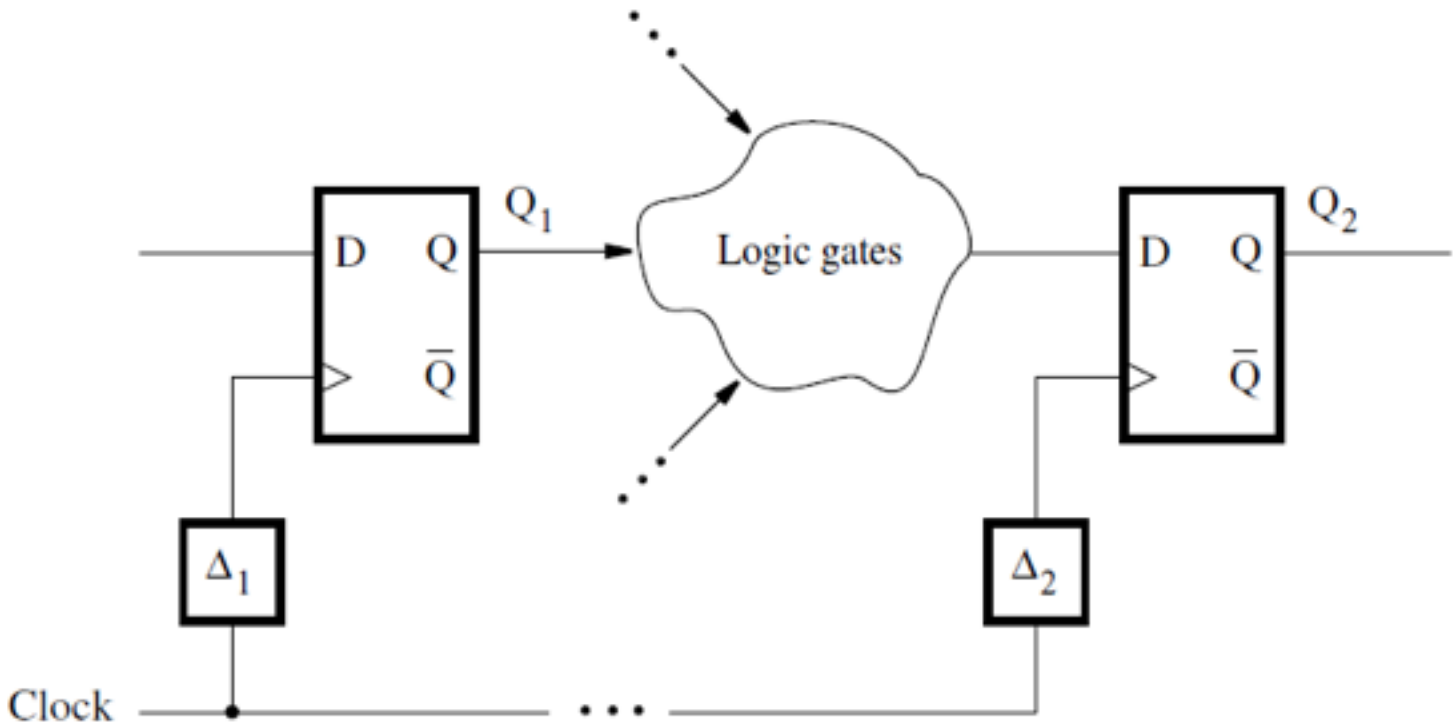


Figure 5.68. A general example of clock skew.

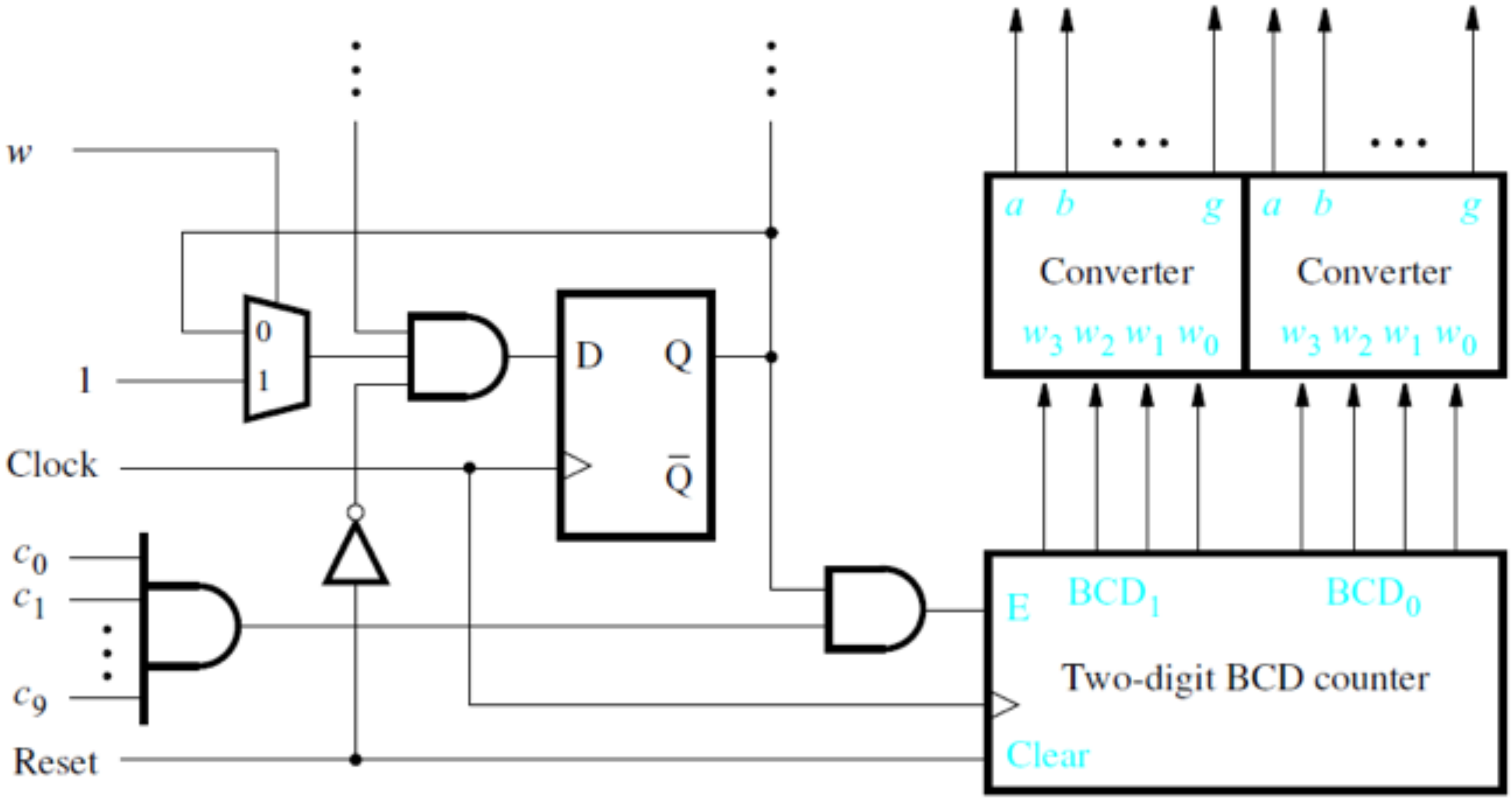
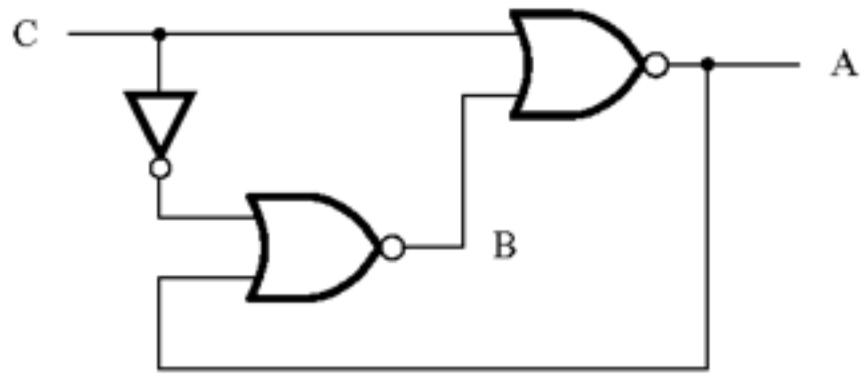
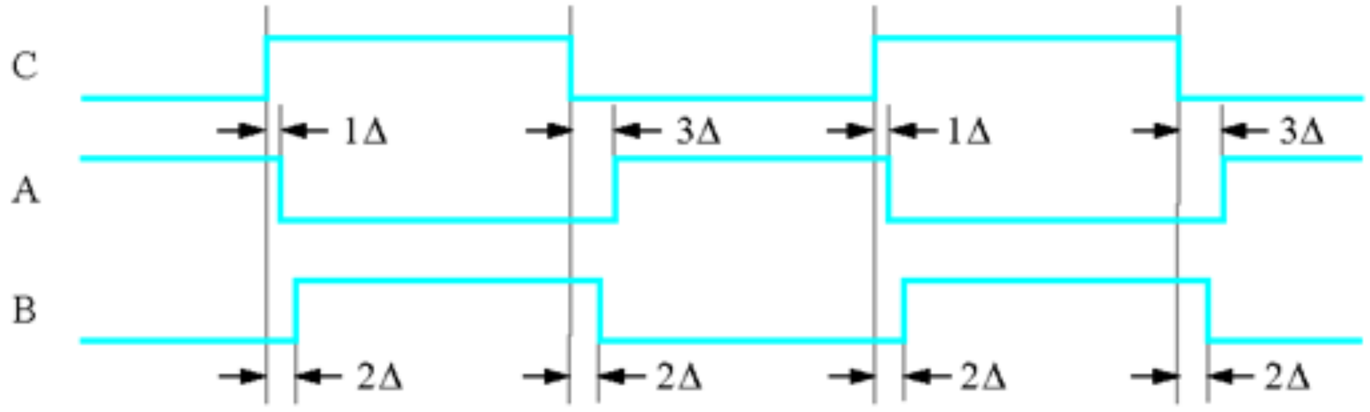


Figure 5.69. A modified version of the reaction-timer circuit.



(a) Circuit



(b) Timing diagram

Figure 5.70. Circuit for Example 5.18.

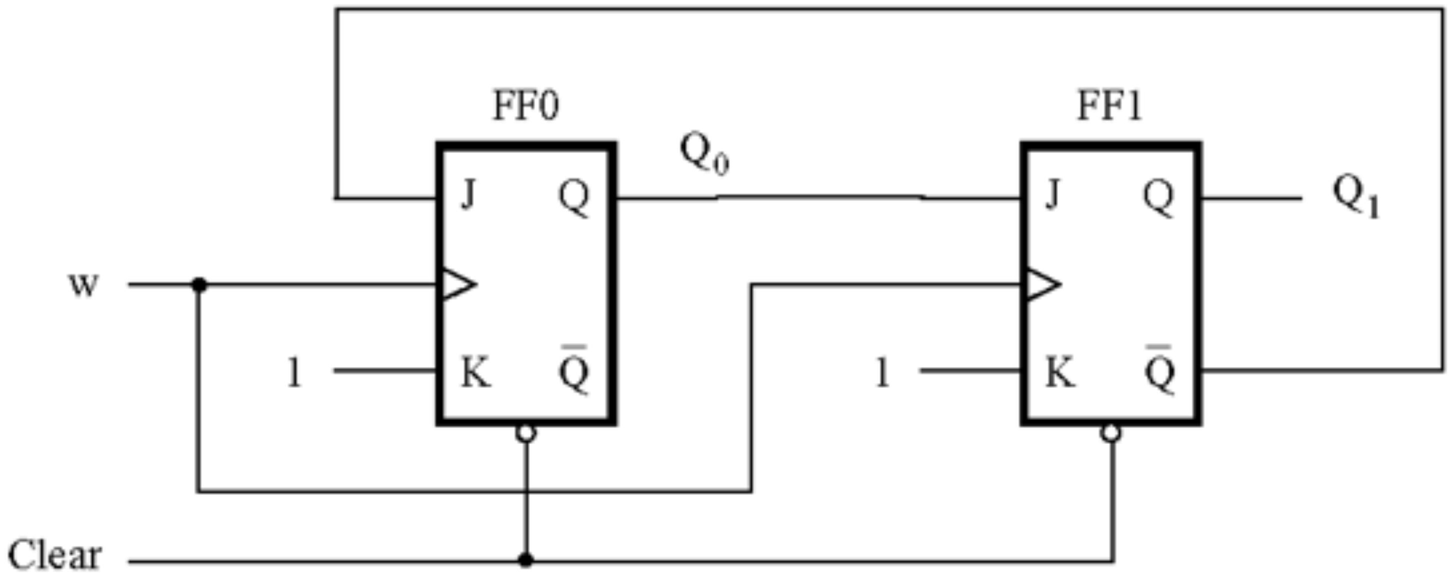


Figure 5.71. Circuit for Example 5.19.

Time interval	FF0			FF1		
	J_0	K_0	Q_0	J_1	K_1	Q_1
Clear	1	1	0	0	1	0
t_1	1	1	1	1	1	0
t_2	0	1	0	0	1	1
t_3	1	1	0	0	1	0
t_4	1	1	1	1	1	0

Figure 5.72. Summary of the behavior of the circuit in Figure 5.71.

Please see “portrait orientation” PowerPoint file for Chapter 5

Figure 5.73. Circuit for Example 5.20.

```
module vend (N, D, Q, Resetn, Coin, Z);
    input N, D, Q, Resetn, Coin;
    output Z;
    wire [4:0] X;
    reg [5:0] S;

    assign X[0] = N | Q;
    assign X[1] = D;
    assign X[2] = N;
    assign X[3] = D | Q;
    assign X[4] = Q;
    assign Z = S[5] | (S[4] & S[3] & S[2] & S[1]);
    always @(negedge Coin, negedge Resetn)
        if (Resetn = = 1’b0)
            S <= 5’b00000;
        else
            S <= {1’b0, X} + S;
    end

endmodule
```

Figure 5.74. Code for Example 5.21.

Please see "portrait orientation" PowerPoint file for Chapter 5

Figure 5.75. A faster 4-bit counter.

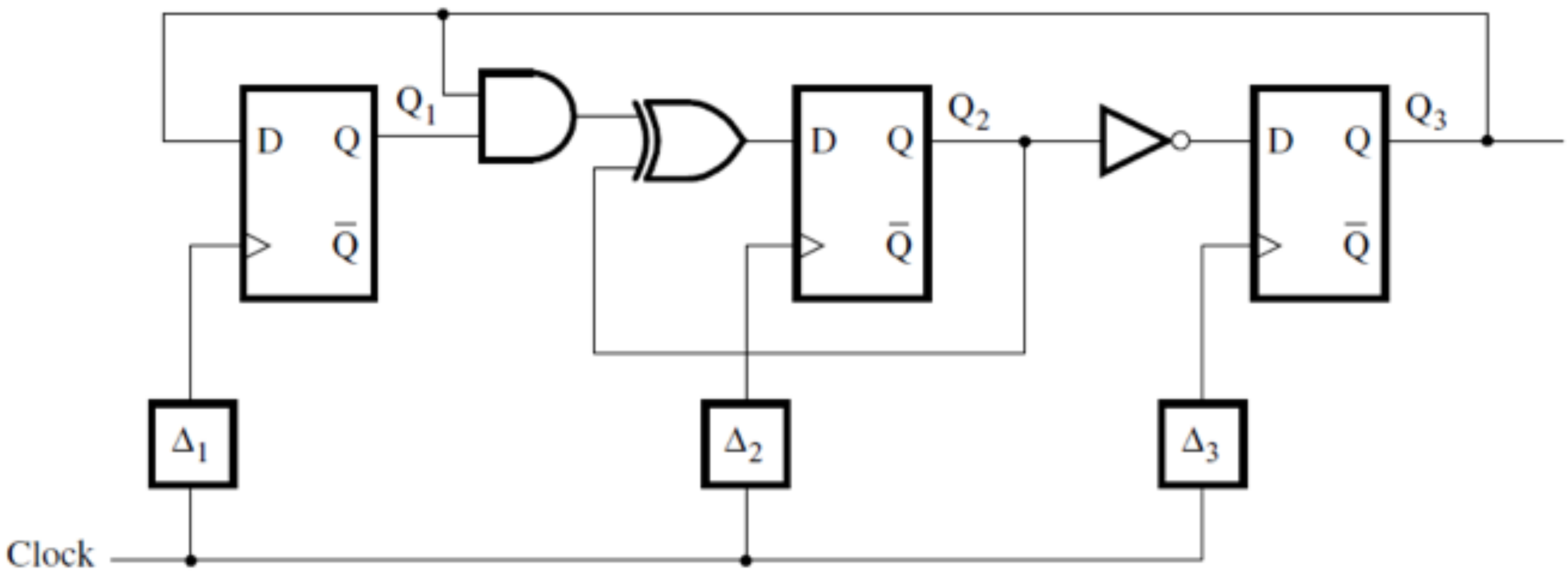


Figure 5.76. A circuit with clock skews.

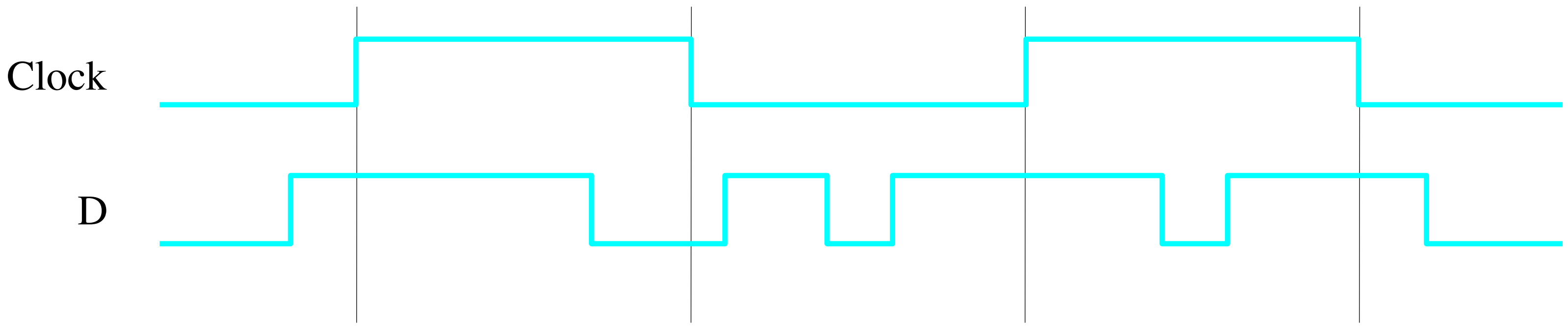


Figure P5.1. Timing diagram for Problem 5.1.

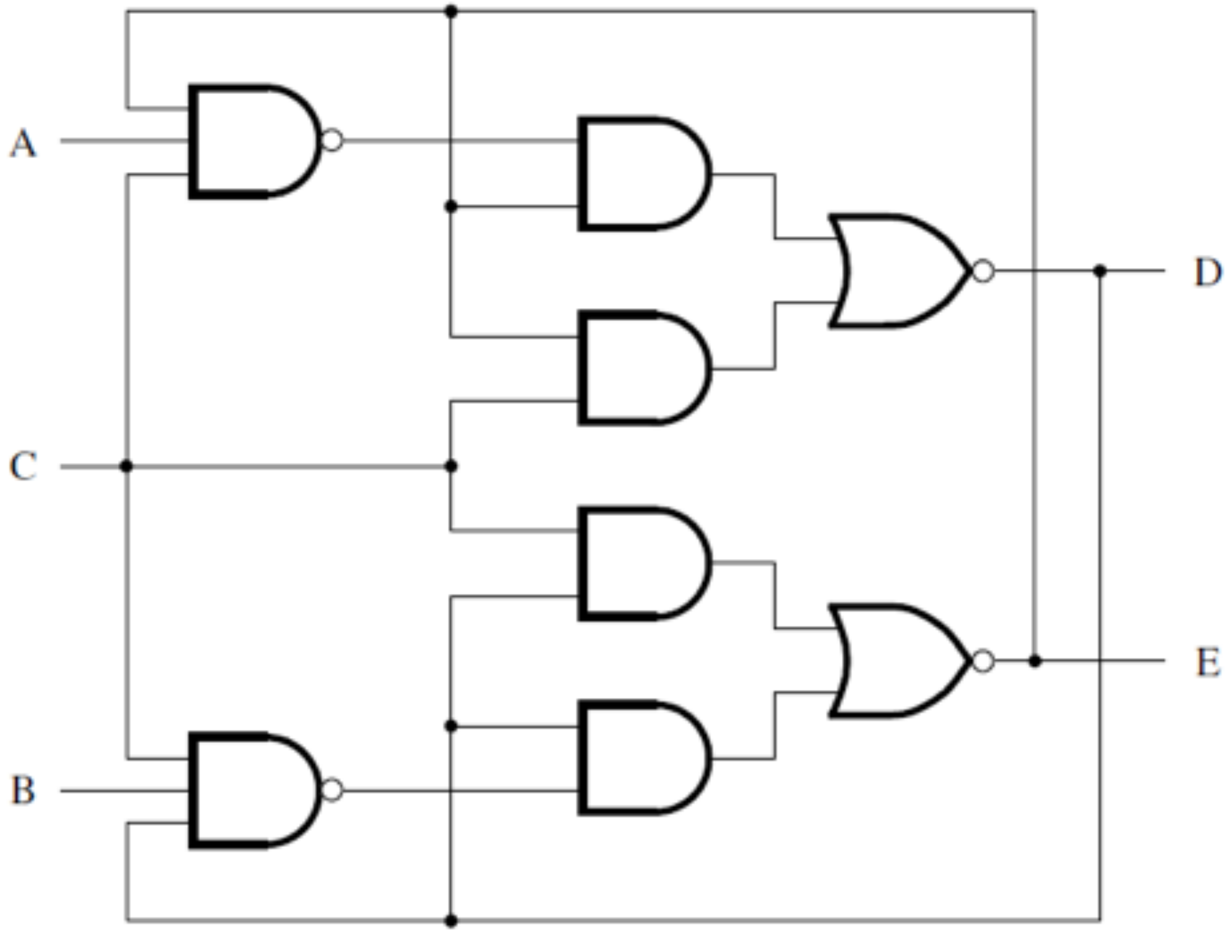


Figure P5.2. Circuit for Problem 5.8.

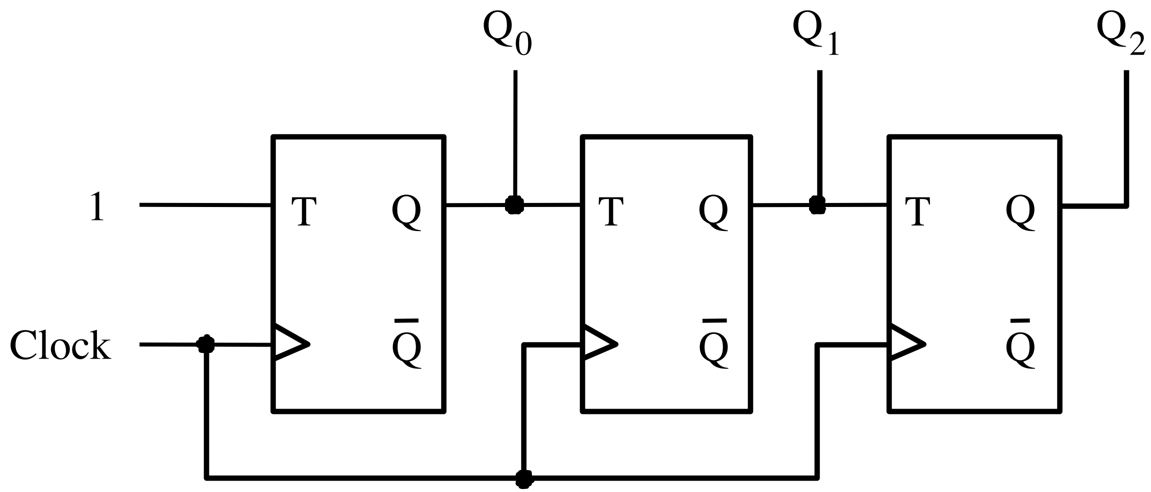


Figure P5.3. The circuit for Problem 5.17.

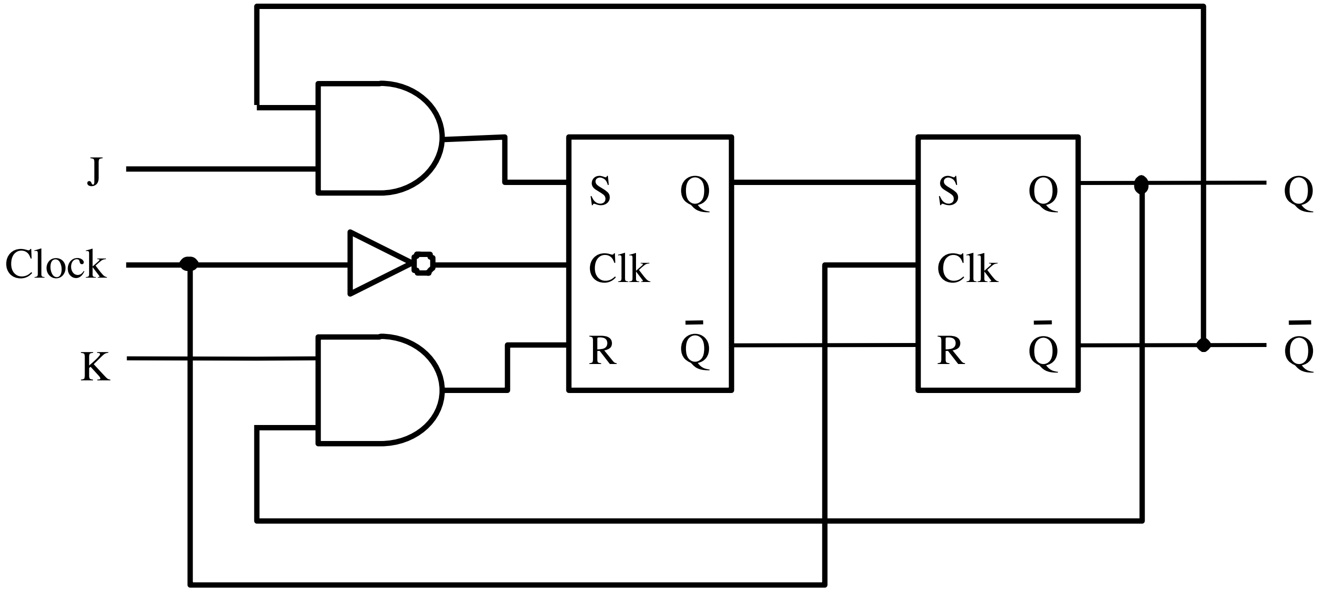


Figure P5.4. Circuit for Problem 5.18.

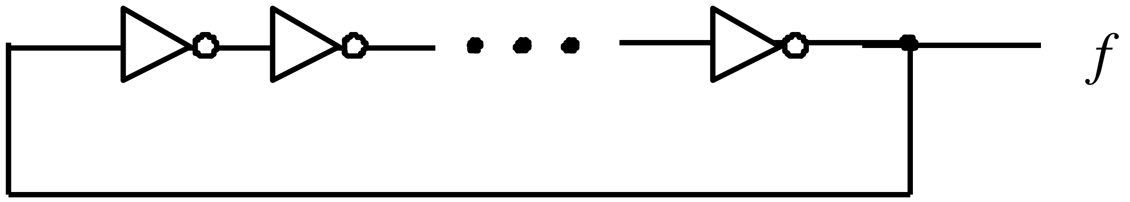


Figure P5.5. A ring oscillator.

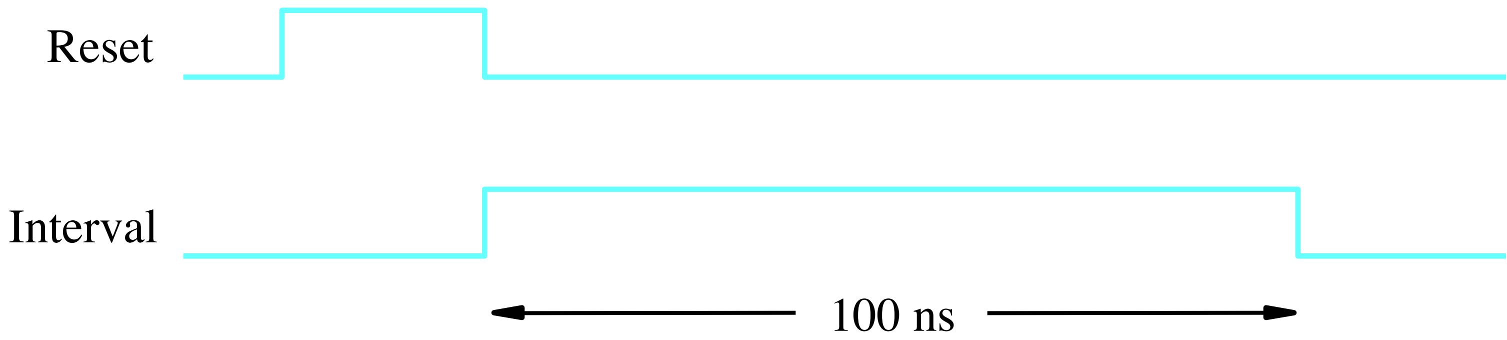


Figure P5.6. Timing of signals for Problem 5.24.

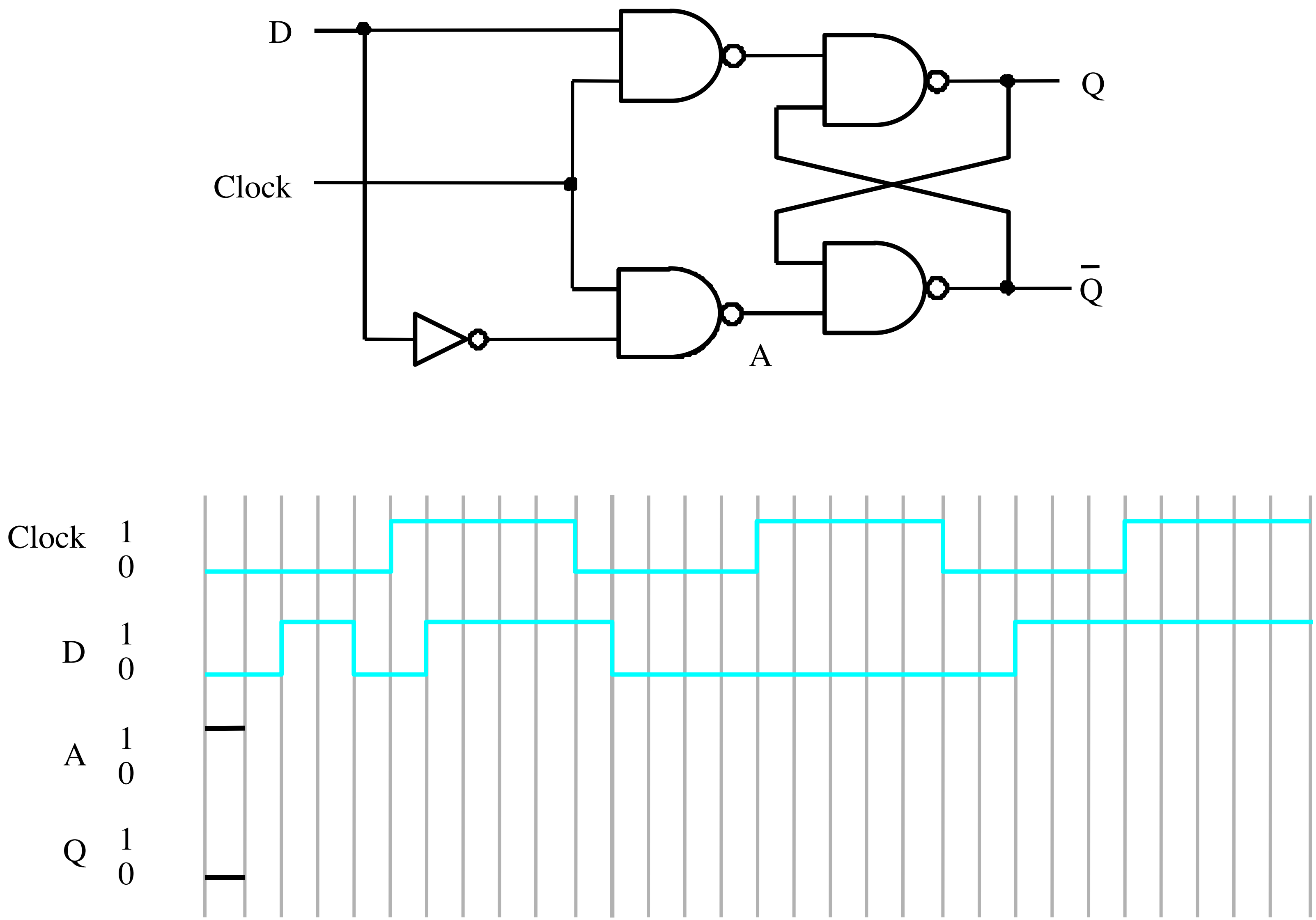


Figure P5.7. Circuit and timing diagram for Problem 5.25.

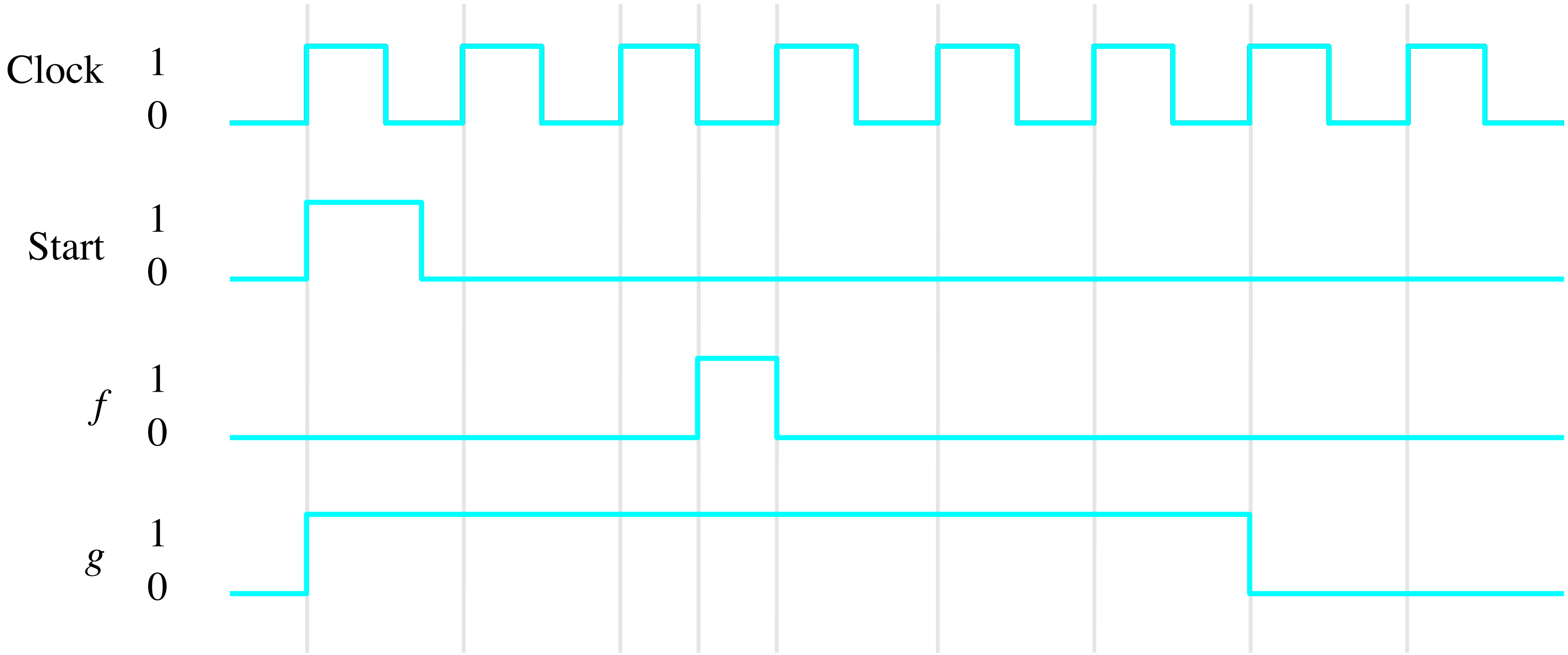


Figure P5.8. Timing diagram for Problem 5.26.

```
module lfsr (R, L, Clock, Q);
    input [0:2] R;
    input L, Clock;
    output reg [0:2] Q;

    always @(posedge Clock)
        if (L)
            Q <= R;
        else
            Q <= {Q[2], Q[0] ^ Q[2], Q[1]};

endmodule
```

Figure P5.9. Code for a linear-feedback shift register.

```
module lfsr (R, L, Clock, Q);
	input [0:2] R;
	input L, Clock;
	output reg [0:2] Q;

	always @(posedge Clock)
		if (L)
			Q <= R;
		else
			Q <= {Q[2], Q[0], Q[1] ^ Q[2]};

endmodule
```

Figure P5.10. Code for a linear-feedback shift register.

```
module lfsr (R, L, Clock, Q);
    input [0:2] R;
    input L, Clock;
    output reg [0:2] Q;

    always @(posedge Clock)
        if (L)
           Q <= R;
        else
        begin
           Q[0] = Q[2];
           Q[1] = Q[0] ^ Q[2];
           Q[2] = Q[1];
        end

endmodule
```

Figure P5.11. Code for Problem 7.30.

```
module lfsr (R, L, Clock, Q);
    input [0:2] R;
    input L, Clock;
    output reg [0:2] Q;

    always @(posedge Clock)
        if (L)
            Q <= R;
        else
        begin
            Q[0] = Q[2];
            Q[1] = Q[0];
            Q[2] = Q[1] ^ Q[2];
        end

endmodule
```

Figure P5.12. Code for Problem 5.31.