Chapter 7

Digital System Design

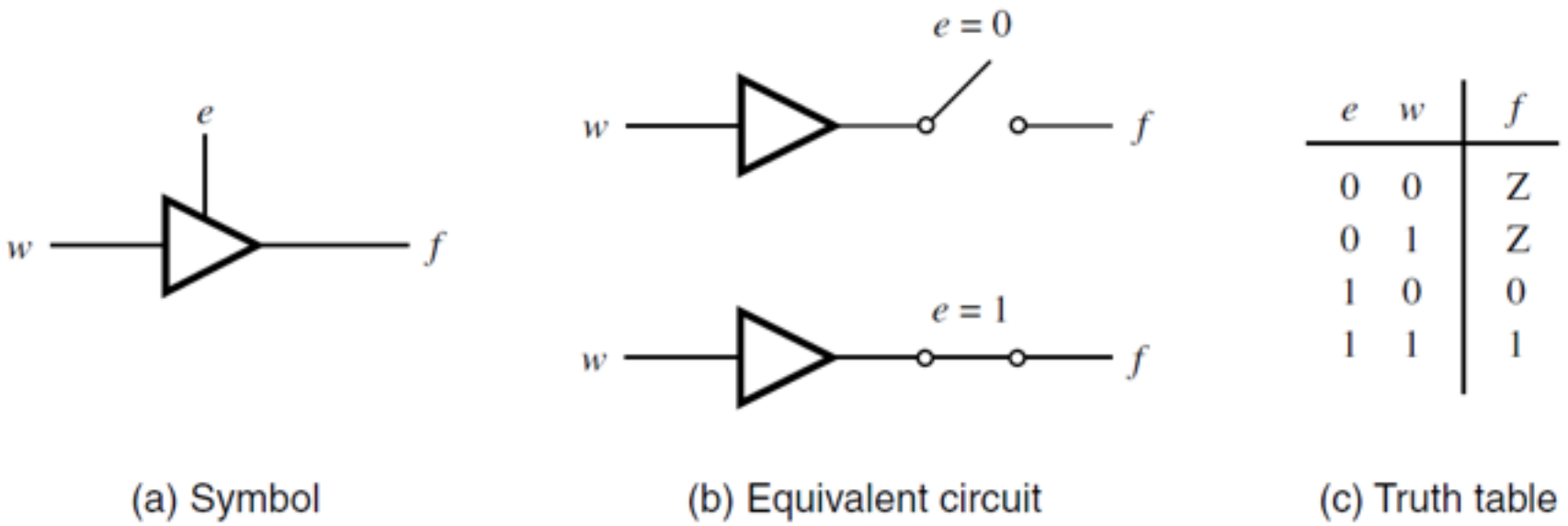


Figure 7.1. Tri-state driver.

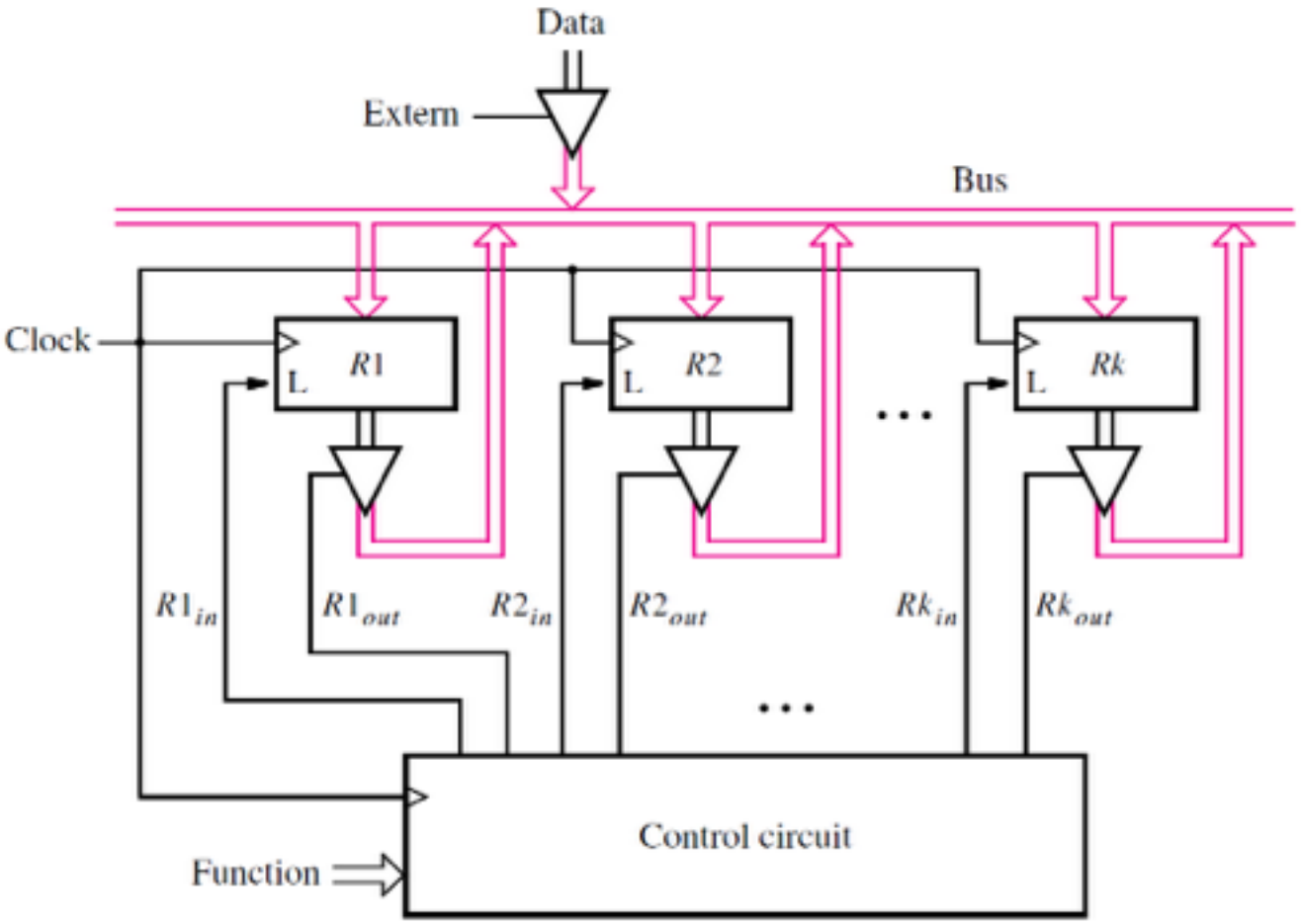


Figure 7.2. A digital system with *k* registers.

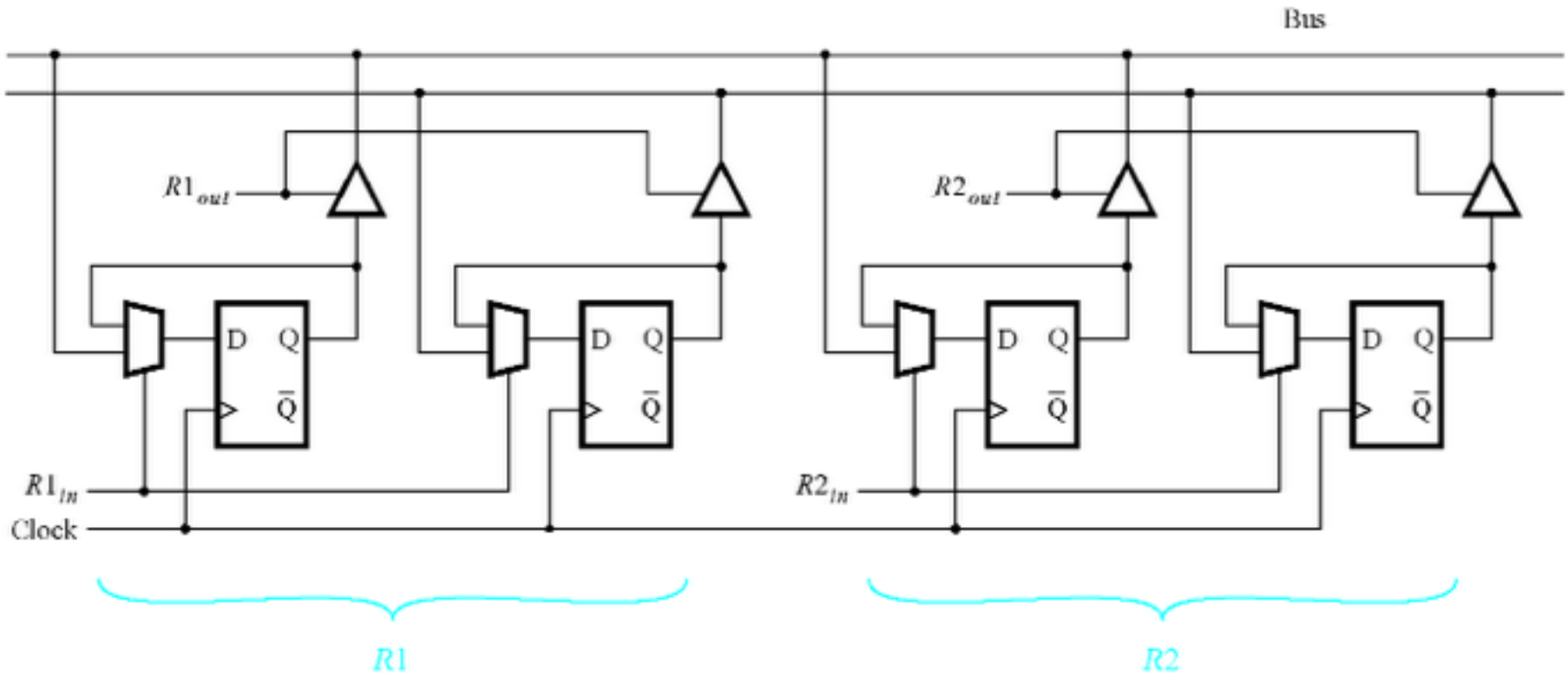


Figure 7.3. Details for connecting registers to a bus.

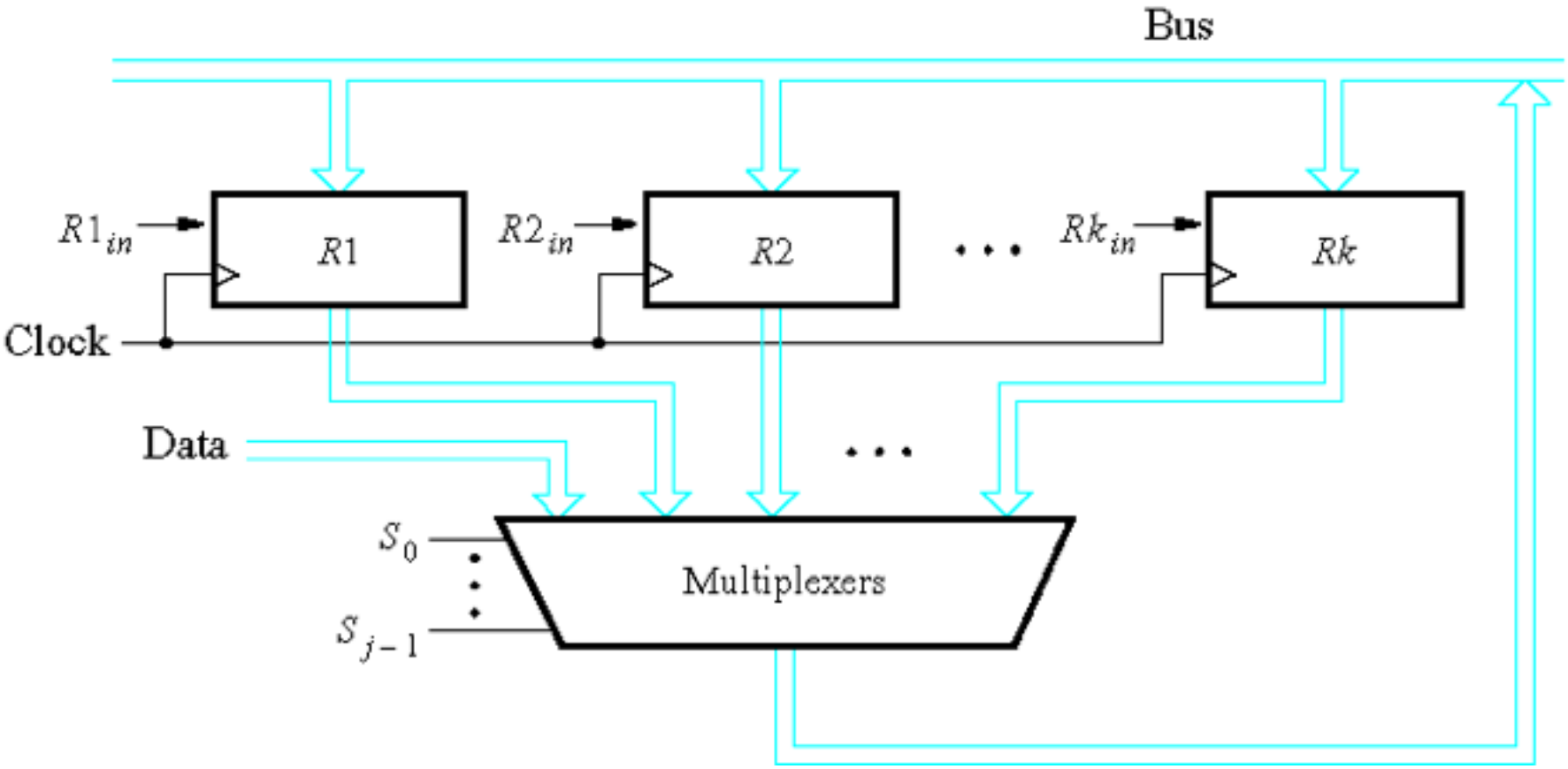


Figure 7.4. Using multiplexers to implement a bus.

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

    always @(posedge Clock)
        if (L)
            Q < = R;

endmodule
```

Figure 7.5. Code for an *n*-bit register of the type in Figure 7.2.

```
module trin (Y, E, F);
    parameter n = 8;
    input [n-1:0] Y;
    input E;
    output wire [n-1:0] F;

    assign F = E ? Y : 'bz;

endmodule
```

Figure 7.6. Code for an *n*-bit tri-state module.

Please see “portrait orientation” PowerPoint file for Chapter 7

Figure 7.7. A digital system like the one in Figure 7.2.

Please see “portrait orientation” PowerPoint file for Chapter 7

Figure 7.8. Using multiplexers to implement a bus.

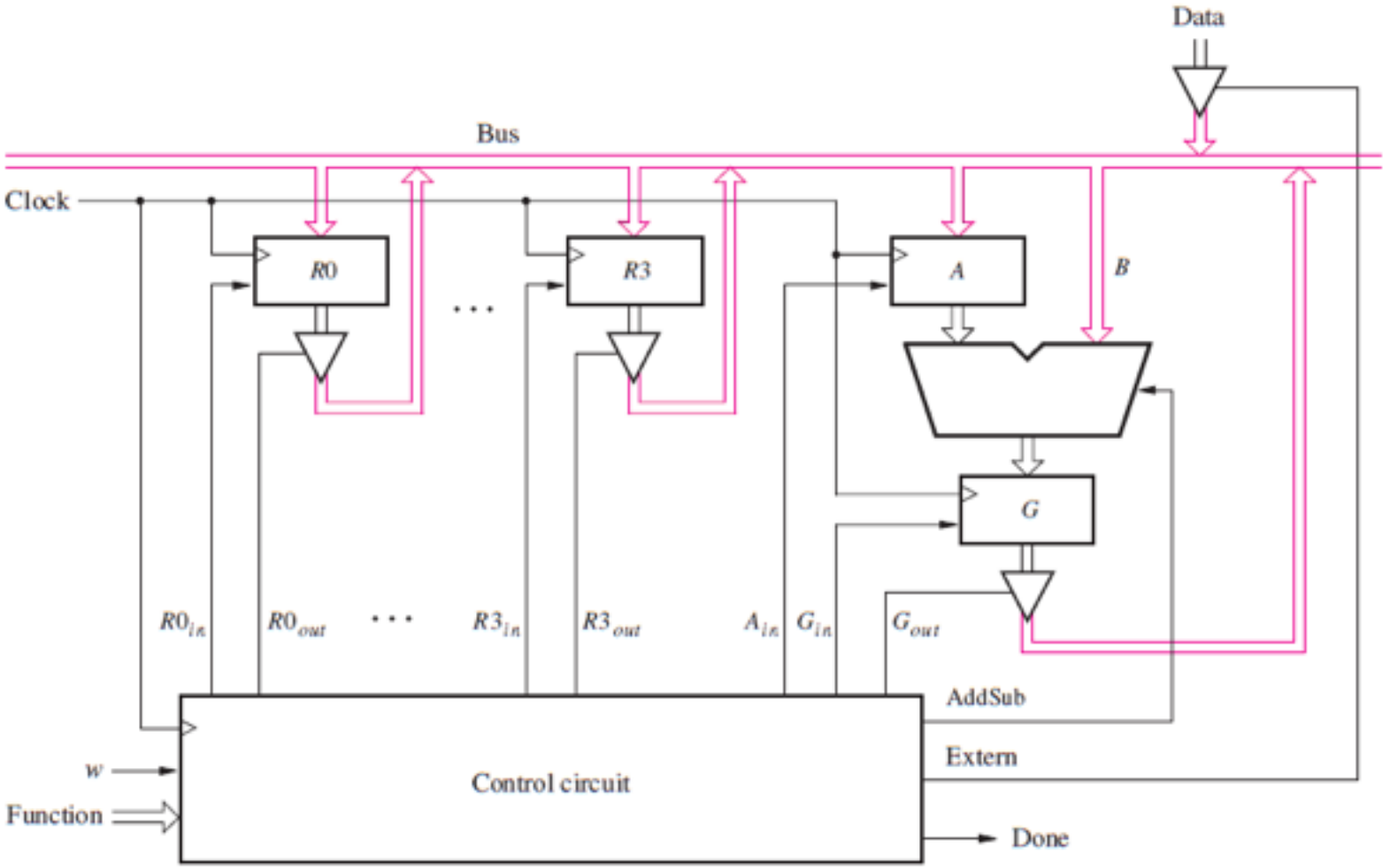


Figure 7.9. A digital system that implements a simple processor.

Operation	Function performed
Load Rx, $Data$	$Rx \leftarrow Data$
Move Rx, Ry	$Rx \leftarrow [Ry]$
Add Rx, Ry	$Rx \leftarrow [Rx] + [Ry]$
Sub Rx, Ry	$Rx \leftarrow [Rx] - [Ry]$

Table 7.1. Operations performed in the processor.

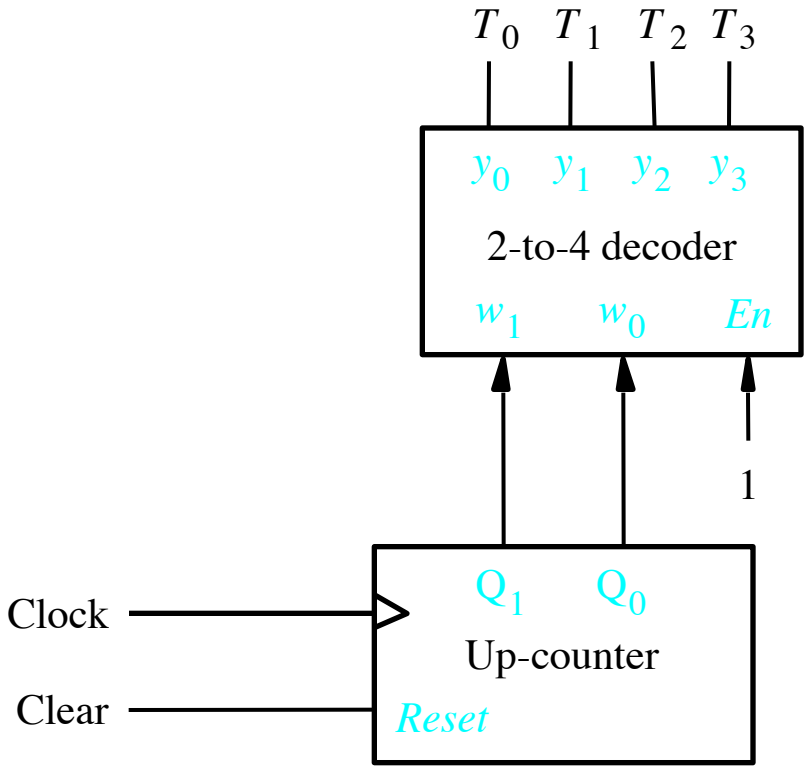


Figure 7.10. A part of the control circuit for the processor.

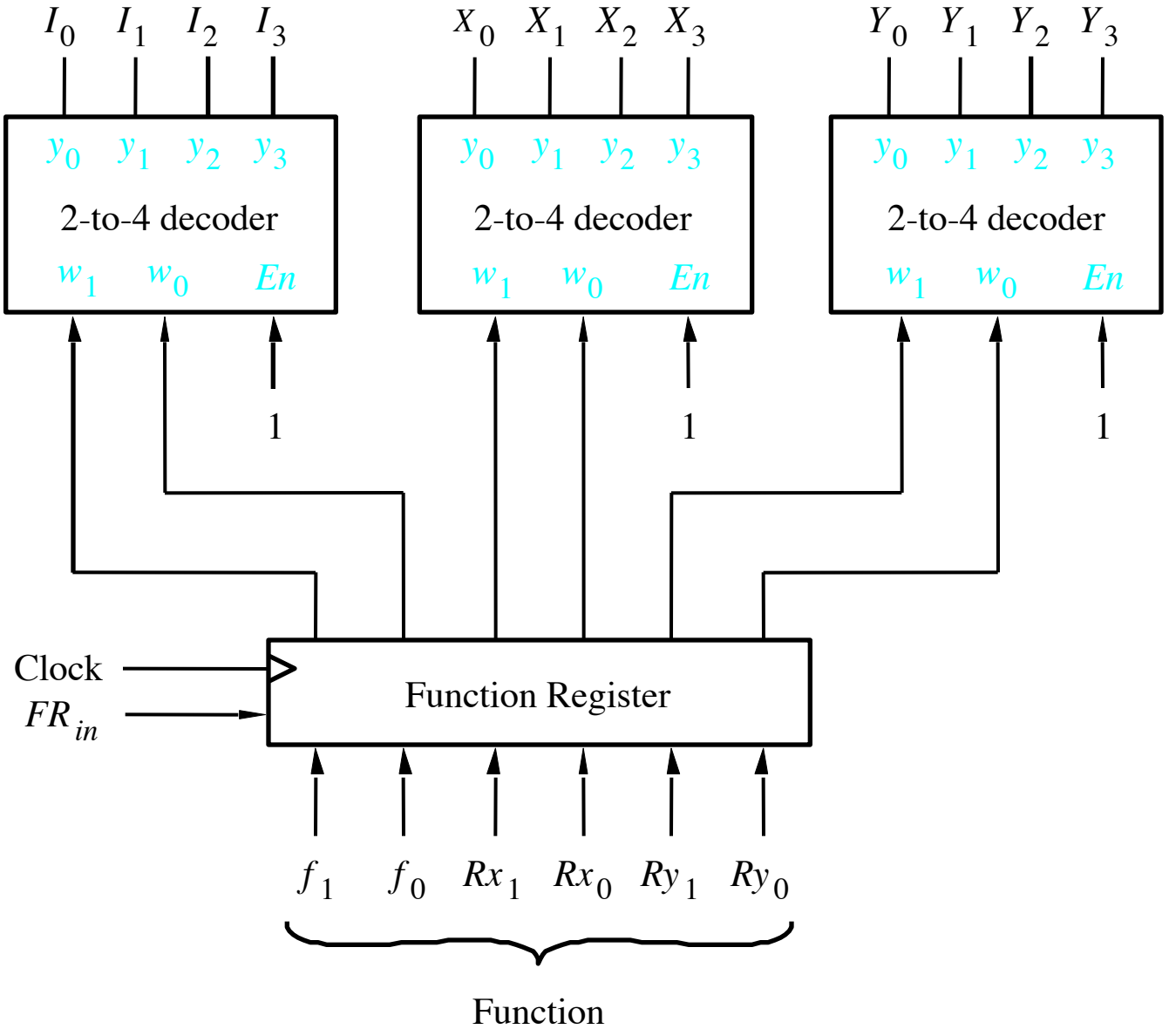


Figure 7.11. The function register and decoders.

	T_1	T_2	T_3
(Load): I_0	$Extern_{in}$, $R_{in} \leftarrow X$, $Done$		
(Move): I_1	$R_{in} \leftarrow X$, $R_{out} \leftarrow Y$, $Done$		
(Add): I_2	$R_{out} \leftarrow X$, A_{in}	$R_{out} \leftarrow Y$, G_{in}, $AddSub \leftarrow 0$	G_{out}, $R_{in} \leftarrow X$, $Done$
(Sub): I_3	$R_{out} \leftarrow X$, A_{in}	$R_{out} \leftarrow Y$, G_{in}, $AddSub \leftarrow 1$	G_{out}, $R_{in} \leftarrow X$, $Done$

Table 7.2. Control signals asserted in each operation/time step.

```
module upcount (Clear, Clock, Q);
    input Clear, Clock;
    output reg [1:0] Q;

    always @(posedge Clock)
        if (Clear)
            Q <= 0;
        else
            Q <= Q + 1;

endmodule
```

Figure 7.12. A two-bit up-counter with synchronous reset.

Please see “portrait orientation” PowerPoint file for Chapter 7

Figure 7.13. Code for the processor.

Please see “portrait orientation” PowerPoint file for Chapter 7

Figure 7.14. Alternative code for the processor.

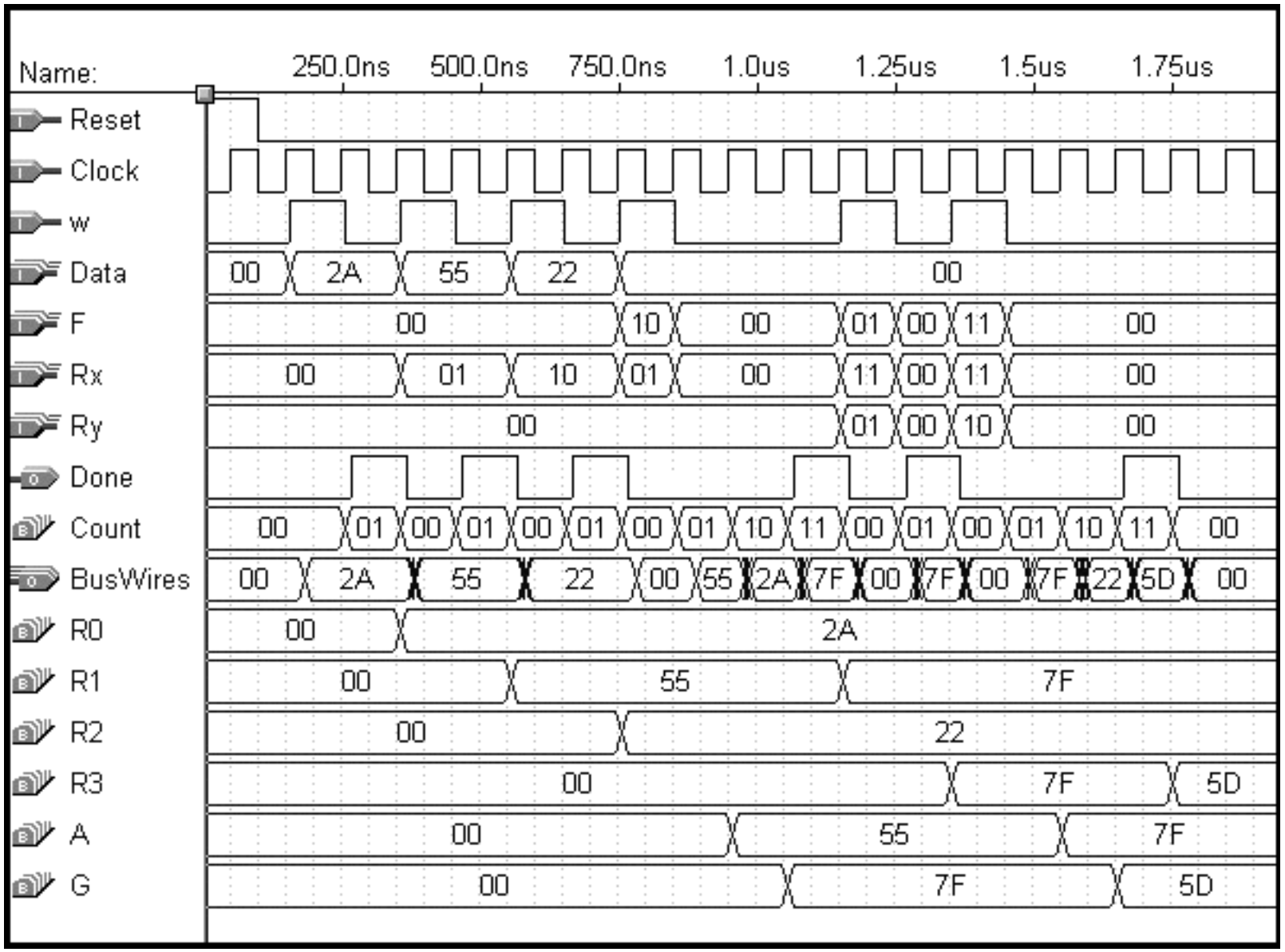


Figure 7.15. Timing simulation for the Verilog code in Figure 7.14.

```
B = 0 ;
while A ≠ 0 do
        if a0 = 1 then
                B = B + 1 ;
        end if;
        Right-shift A ;
end while;
```

Figure 7.16 Pseudo-code for the bit counter.

Please see “portrait orientation” PowerPoint file for Chapter 7

Figure 7.17. ASM chart for the pseudo-code in Figure 7.16.

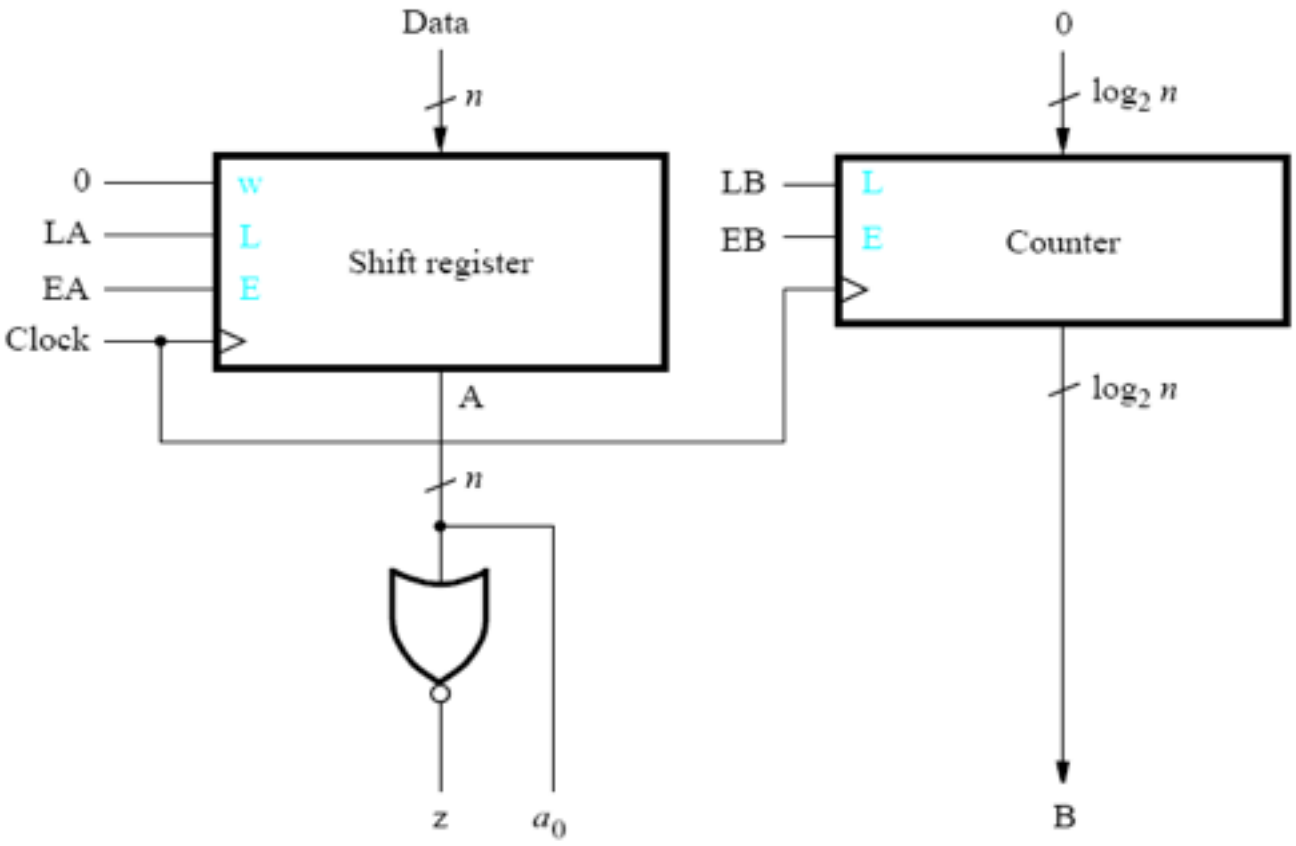


Figure 7.18. Datapath for the ASM chart in Figure 7.17.

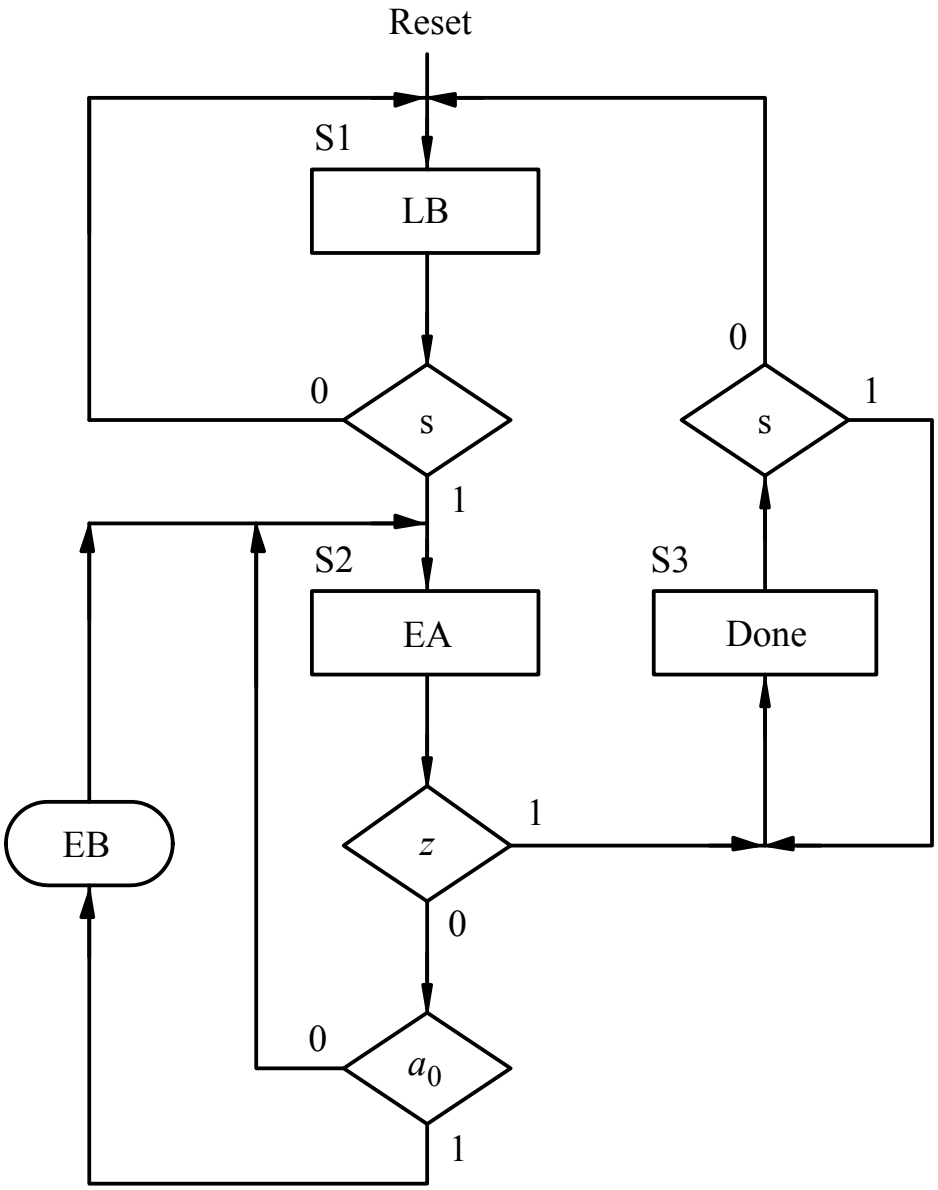


Figure 7.19. ASM chart for the bit counter control circuit.

Please see “portrait orientation” PowerPoint file for Chapter 7

Figure 7.20. Verilog code for the bit-counting circuit.

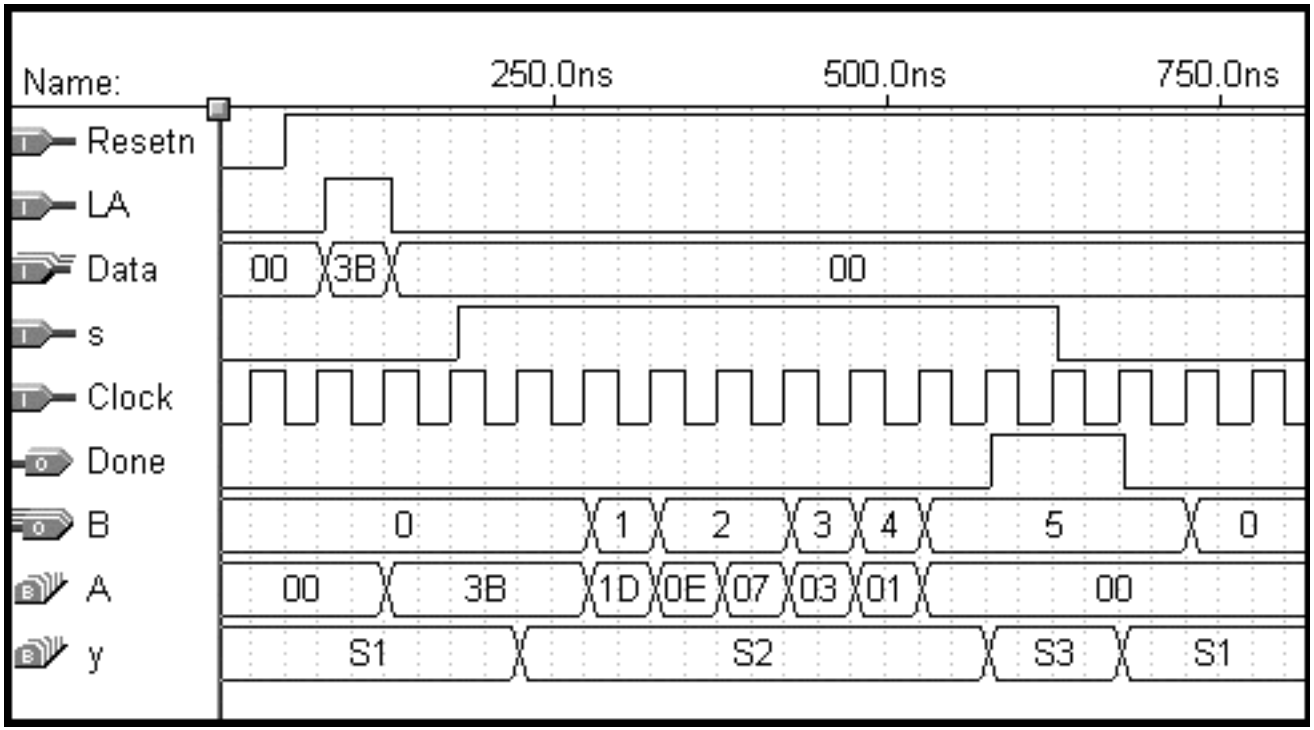
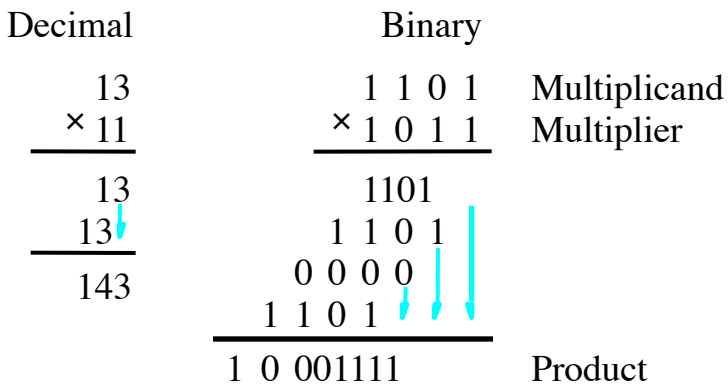


Figure 7.21. Simulation results for the bit-counting circuit.



(a) Manual method

```
P = 0 ;
for i = 0 to n – 1 do
        if b_i = 1 then
                P = P + A ;
        end if;
        Left-shift A ;
end for;
```

(b) Pseudo-code

Figure 7.22. An algorithm for multiplication.

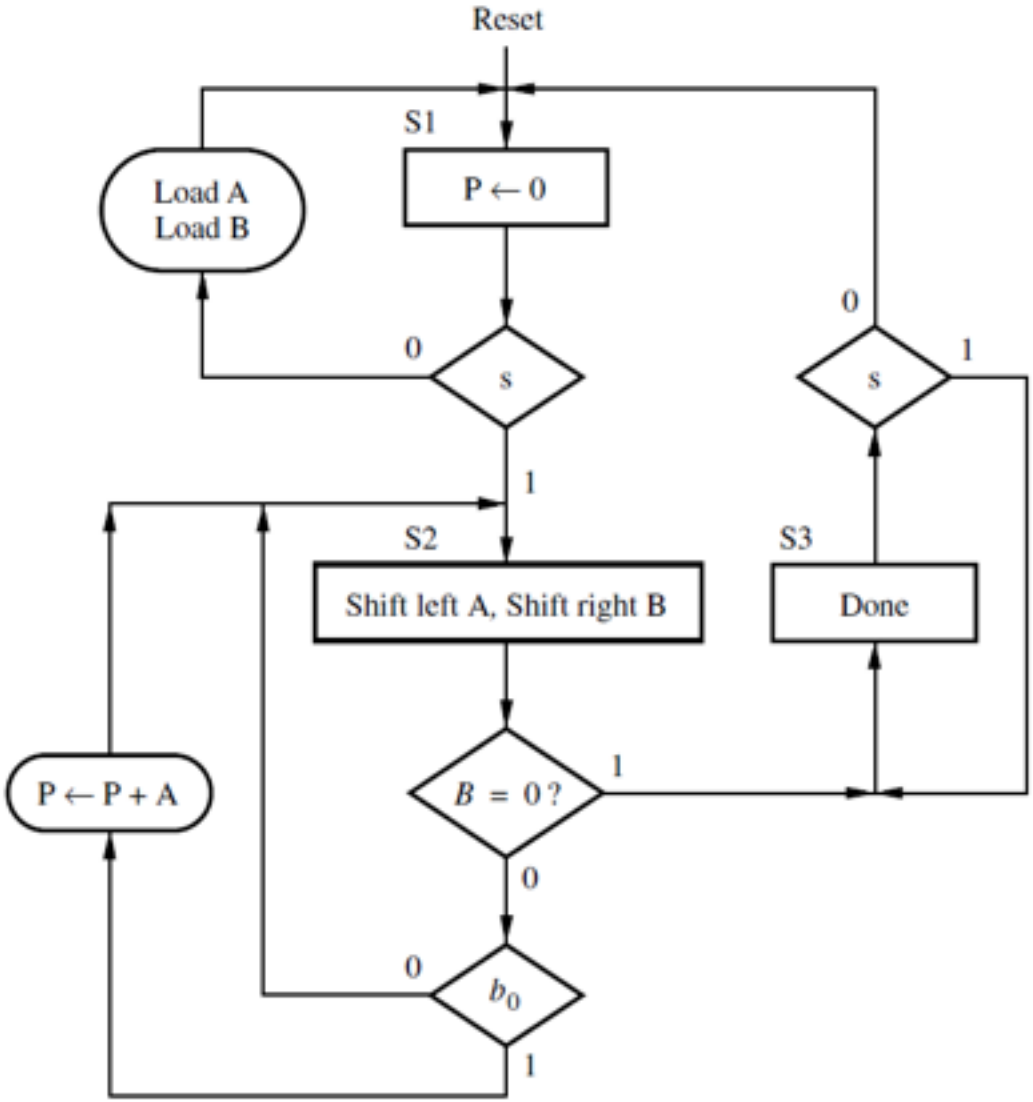


Figure 7.23. ASM chart for the multiplier.

Please see “portrait orientation” PowerPoint file for Chapter 7

Figure 7.24. Datapath circuit for the multiplier.

Please see “portrait orientation” PowerPoint file for Chapter 7

Figure 7.25. ASM chart for the multiplier control circuit.

Please see “portrait orientation” PowerPoint file for Chapter 7

Figure 7.26. Verilog code for the multiplier circuit.

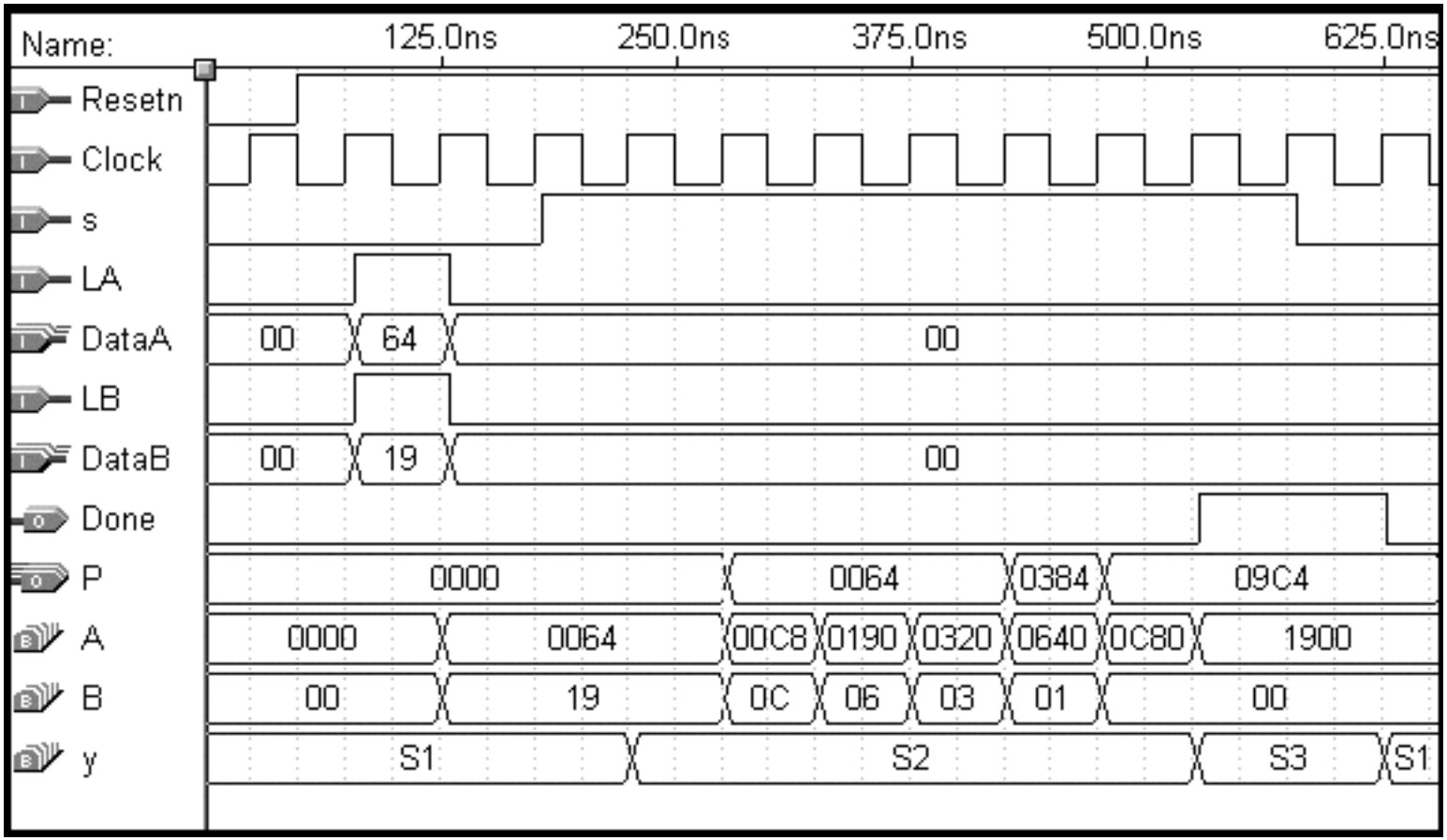
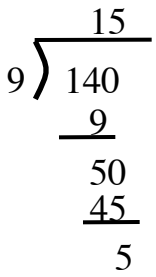
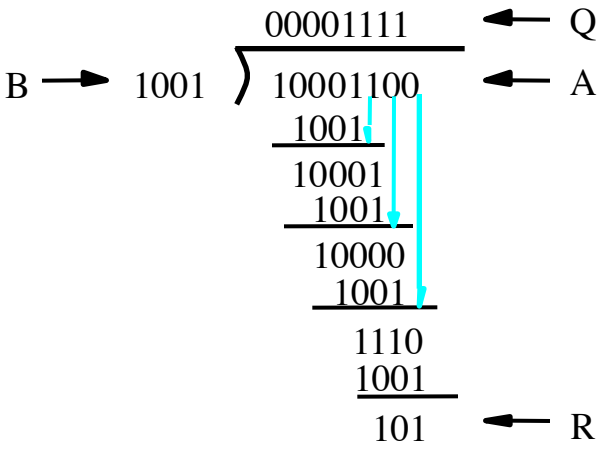


Figure 7.27. Simulation results for the multiplier circuit.



(a) An example using decimal numbers



(b) Using binary numbers

```
R = 0 ;
for i = 0 to n – 1 do
        Left-shift R||A   ;
        if R ≥ B then
                q_i = 1 ;
                R = R – B ;
        else
                q_i = 0 ;
        end if;
end for;
```

(c) Pseudo-code

Figure 7.28. An algorithm for division.

Please see “portrait orientation” PowerPoint file for Chapter 7

Figure 7.29. ASM chart for the divider.

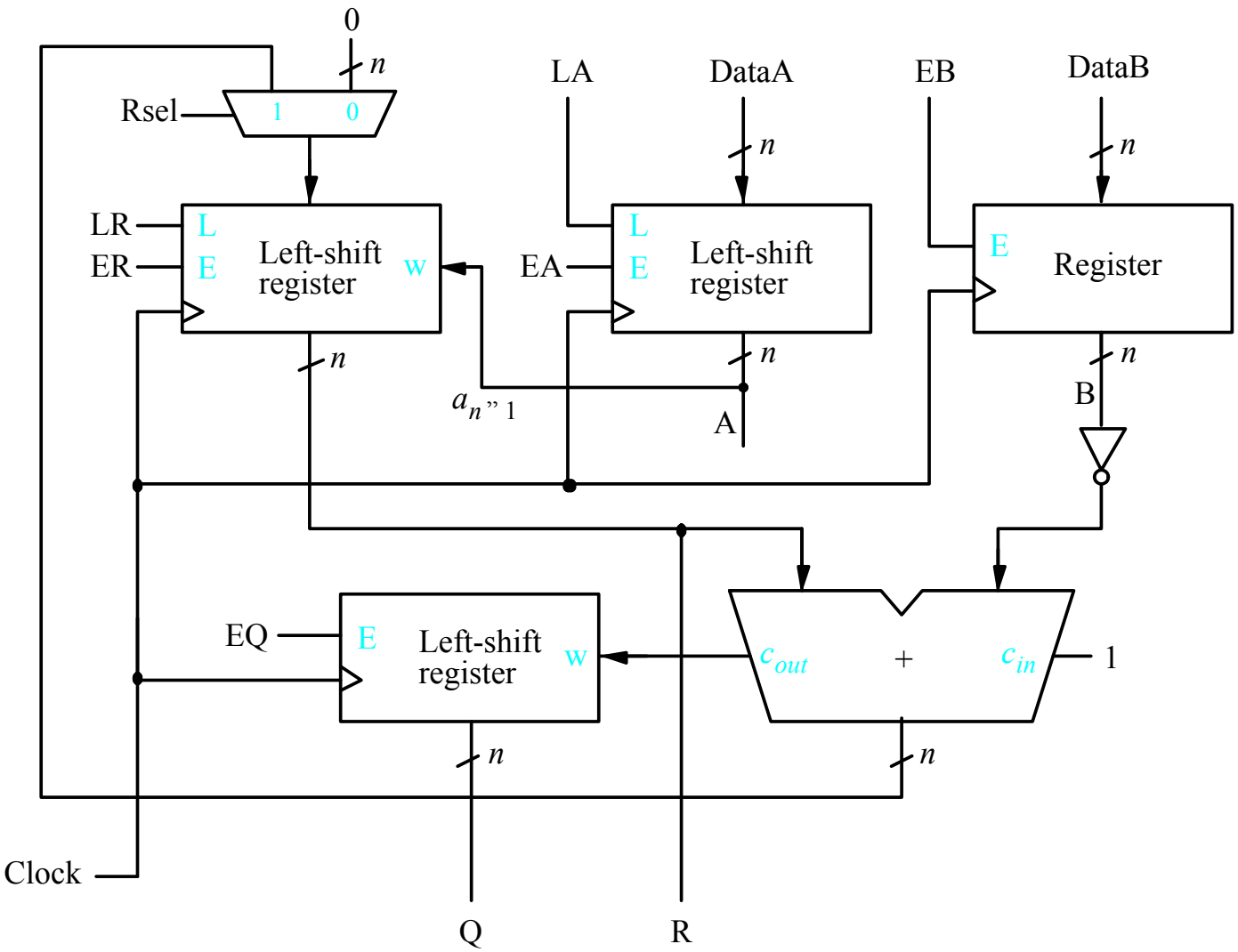


Figure 7.30. Datapath circuit for the divider.

Please see “portrait orientation” PowerPoint file for Chapter 7

Figure 7.31. ASM chart for the divider control circuit.

B → 1001) 10001100 ← A

	Clock cycle	R								rr_0	A/Q							
	Load A, B	0	0	0	0	0	0	0	0	0	1	0	0	0	1	1	0	0
0	Shift left	0	0	0	0	0	0	0	0	1	0	0	0	1	1	0	0	0
1	Shift left, $Q_0 \leftarrow 0$	0	0	0	0	0	0	0	1	0	0	0	1	1	0	0	0	0
2	Shift left, $Q_0 \leftarrow 0$	0	0	0	0	0	0	1	0	0	0	1	1	0	0	0	0	0
3	Shift left, $Q_0 \leftarrow 0$	0	0	0	0	0	1	0	0	0	1	1	0	0	0	0	0	0
4	Shift left, $Q_0 \leftarrow 0$	0	0	0	0	1	0	0	0	1	1	0	0	0	0	0	0	0
5	Subtract, $Q_0 \leftarrow 1$	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	1
6	Subtract, $Q_0 \leftarrow 1$	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	1
7	Subtract, $Q_0 \leftarrow 1$	0	0	0	0	0	1	1	1	0	0	0	0	0	0	1	1	1
8	Subtract, $Q_0 \leftarrow 1$	0	0	0	0	0	1	0	1	0	0	0	0	0	1	1	1	1

Figure 7.32. An example of division using $n = 8$ clock cycles.

Please see “portrait orientation” PowerPoint file for Chapter 7

Figure 7.33. ASM chart for the enhanced divider control circuit.

Please see “portrait orientation” PowerPoint file for Chapter 7

Figure 7.34. Datapath circuit for the enhanced divider.

Please see “portrait orientation” PowerPoint file for Chapter 7

Figure 7.35. Verilog code for the divider circuit.

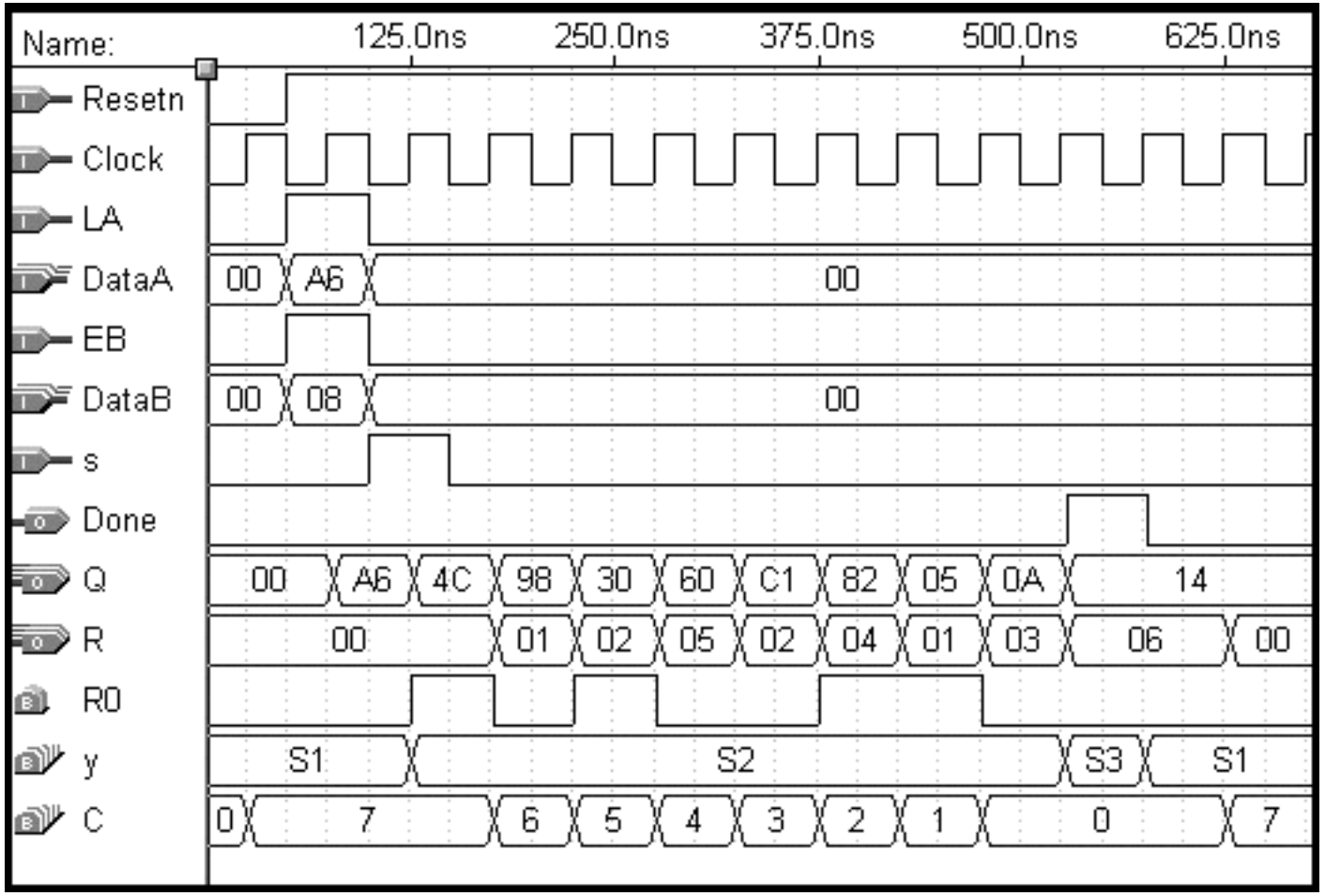


Figure 7.36. Simulation results for the divider circuit.

Please see “portrait orientation” PowerPoint file for Chapter 7

Figure 7.37. An algorithm for finding the mean of k numbers.

Please see “portrait orientation” PowerPoint file for Chapter 7

Figure 7.38. Datapath circuit for the mean operation.

Please see “portrait orientation” PowerPoint file for Chapter 10

Figure 7.39. ASM chart for the mean operation control circuit.

```
for i = 0 to k − 2 do
        A = R_i ;
        for j = i + 1 to k − 1 do
                B = R_j ;
                if B < A then
                        R_i = B ;
                        R_j = A ;
                        A = R_i ;
                end if ;
        end for;
end for;
```

Figure 7.40. Pseudo-code for the sort operation.

Please see “portrait orientation” PowerPoint file for Chapter 7

Figure 7.41. ASM chart for the sort operation.

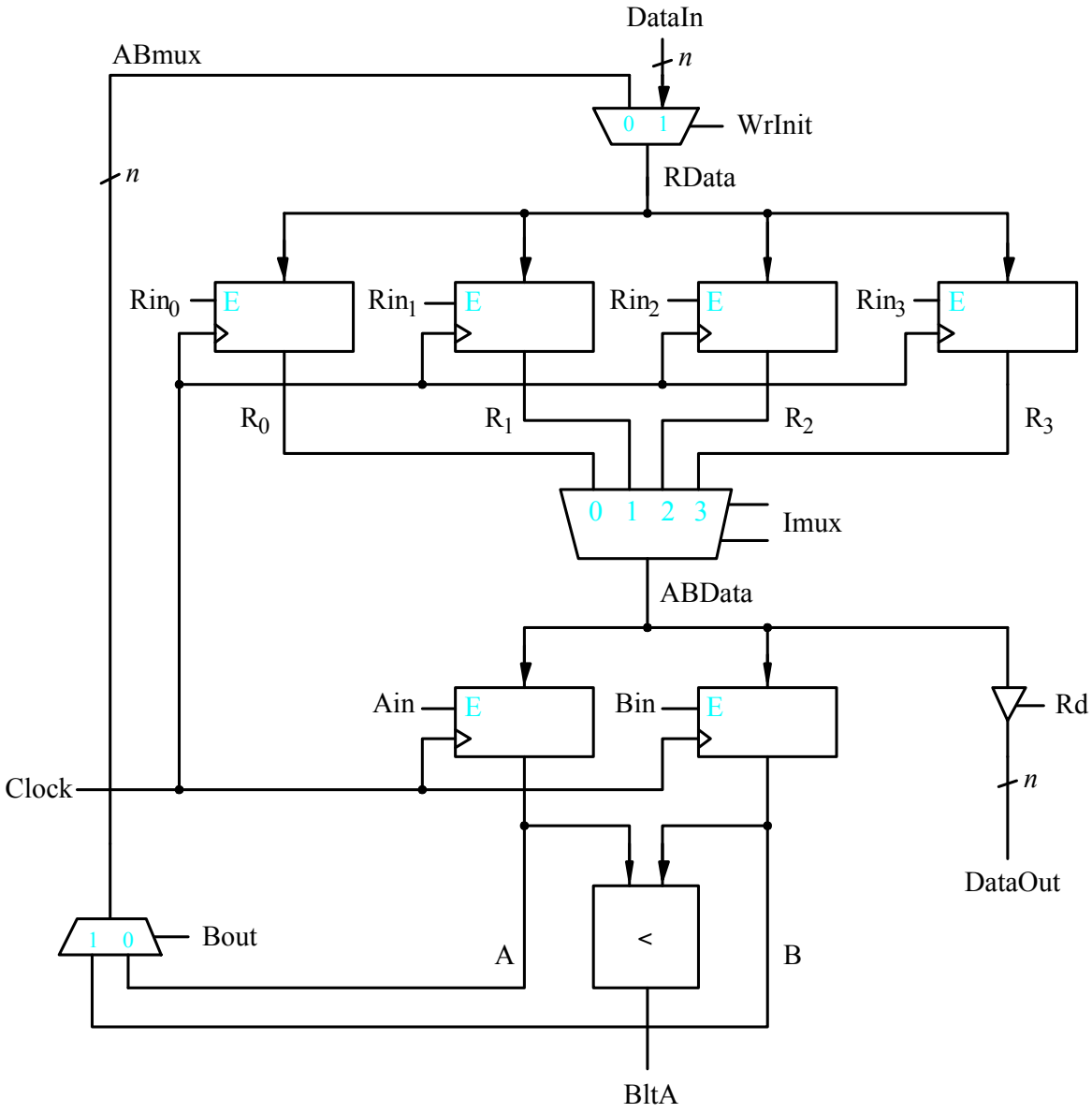


Figure 7.42. A part of the datapath circuit for the sort operation.

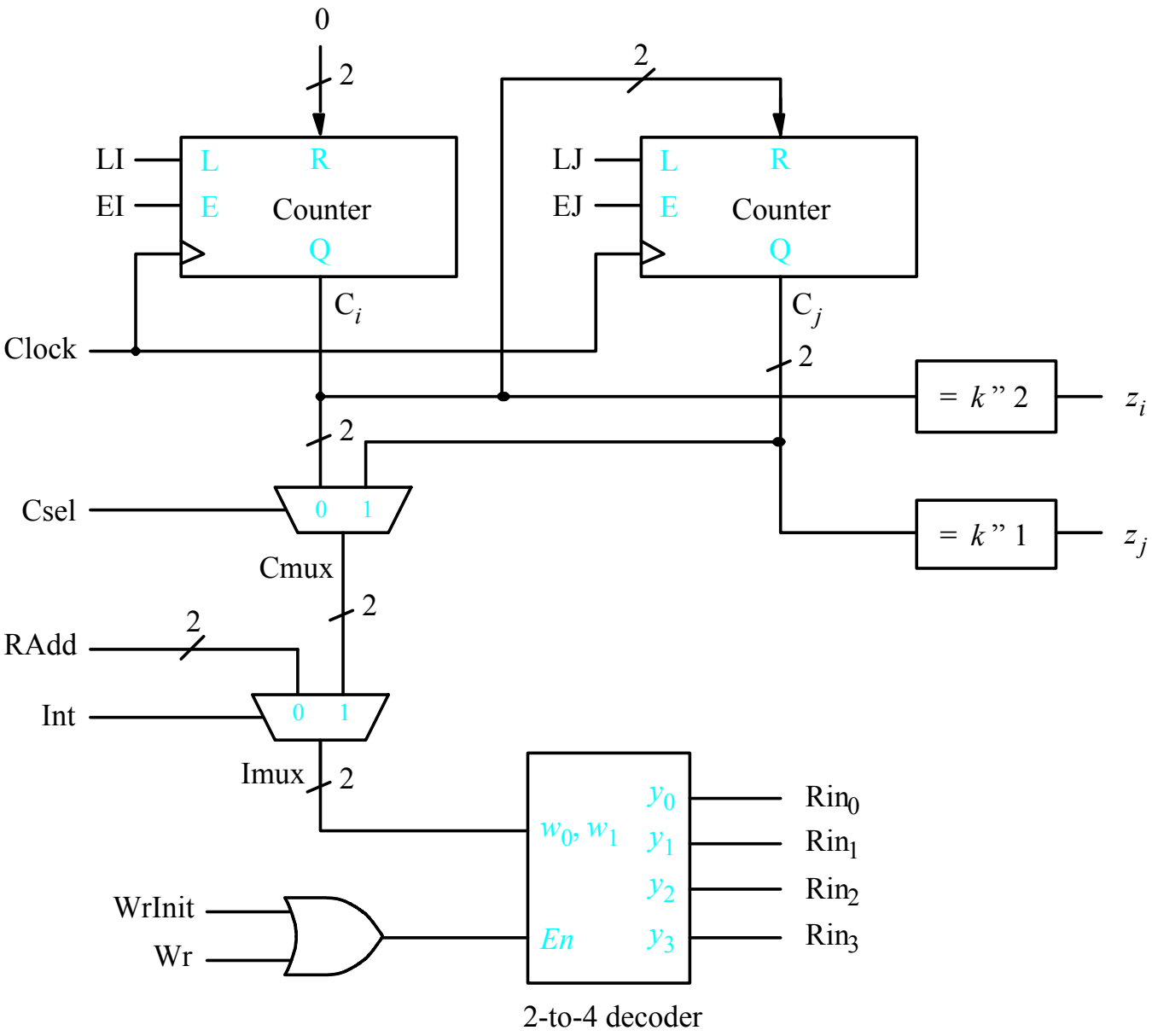


Figure 7.43. A part of the datapath circuit for the sort operation.

Please see “portrait orientation” PowerPoint file for Chapter 7

Figure 7.44. ASM chart for the control circuit.

Please see “portrait orientation” PowerPoint file for Chapter 7

Figure 7.45. Verilog code for the sorting circuit.

Please see “portrait orientation” PowerPoint file for Chapter 7

Figure 7.46. Simulation results for the sort operation.

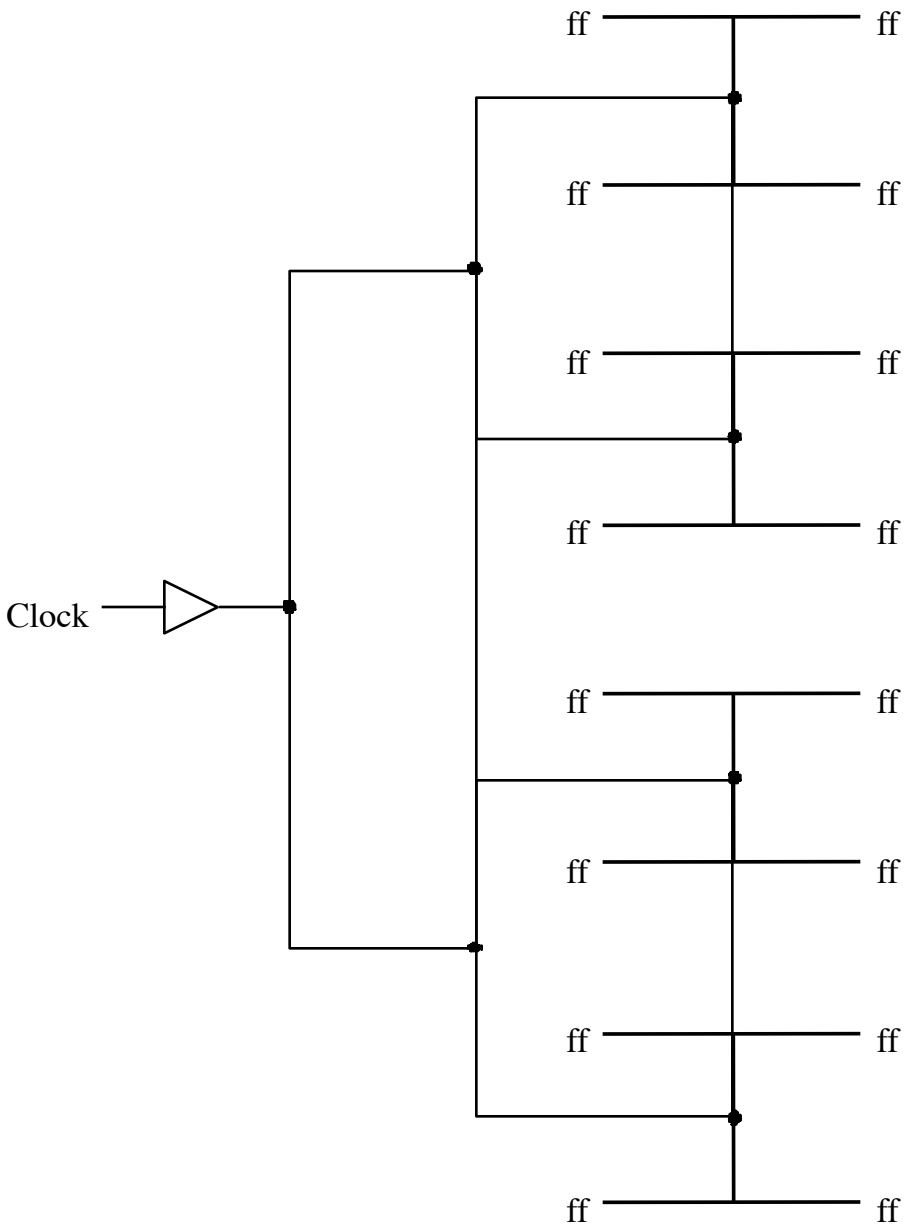


Figure 7.47. An H tree clock distribution network.

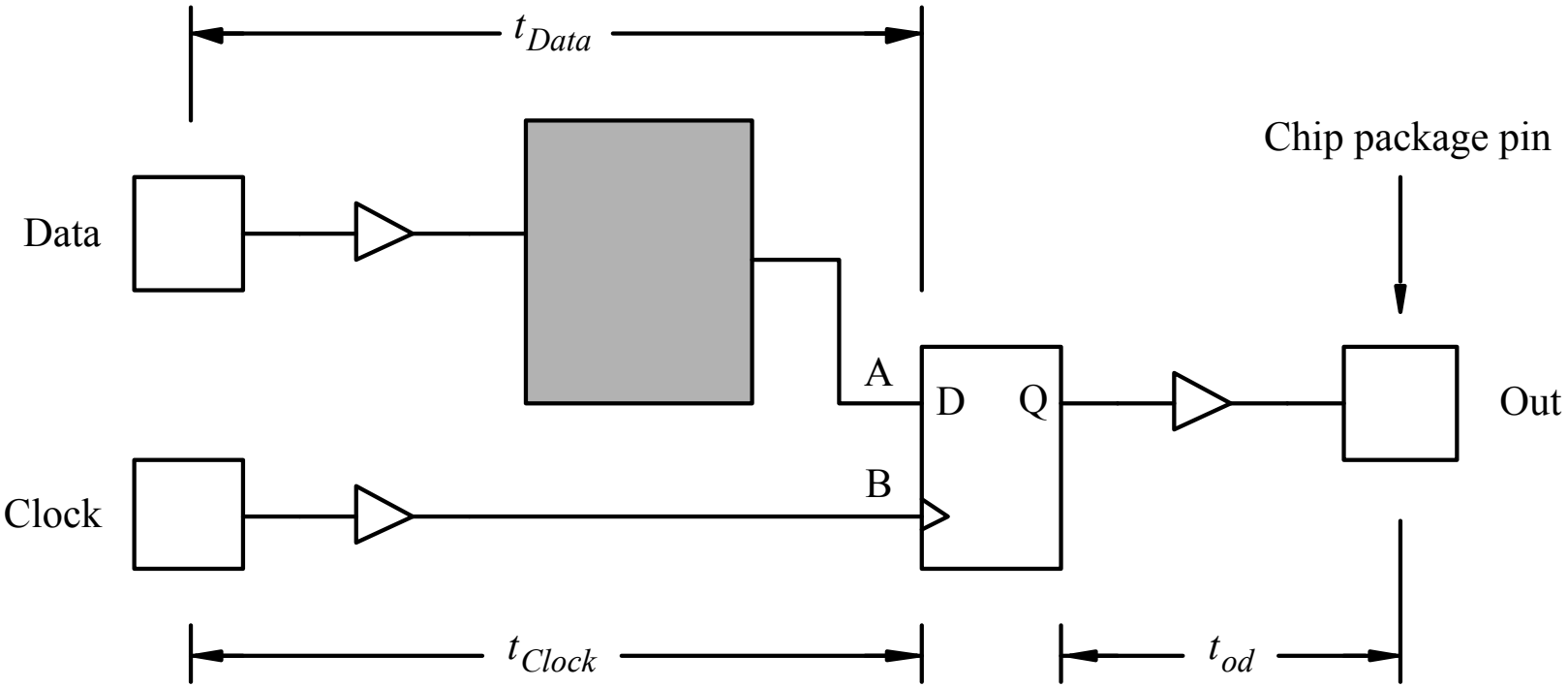


Figure 10.48. A flip-flop in an integrated circuit.

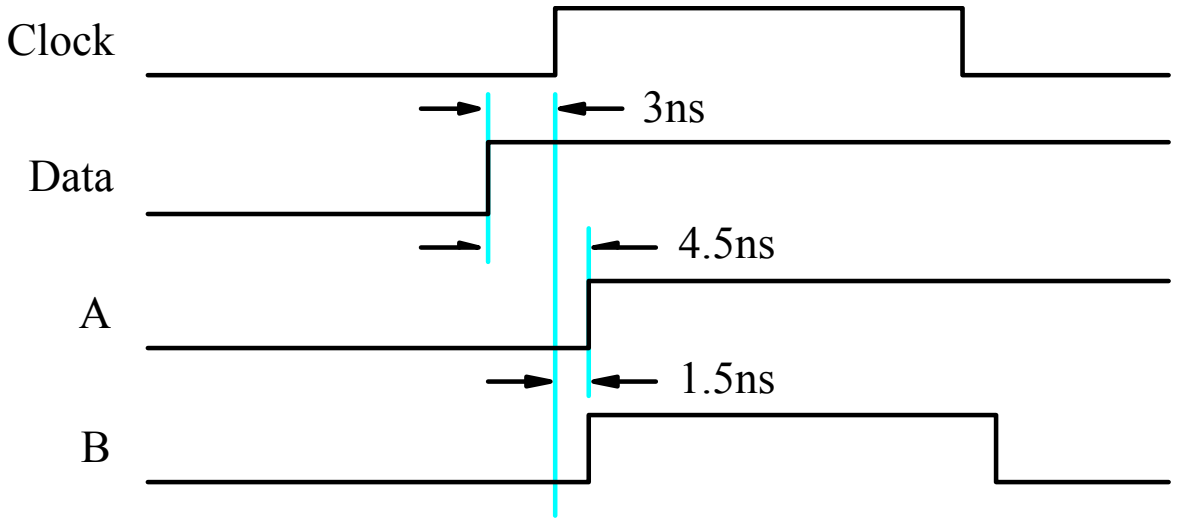


Figure 7.49. Flip-flop timing in a chip.

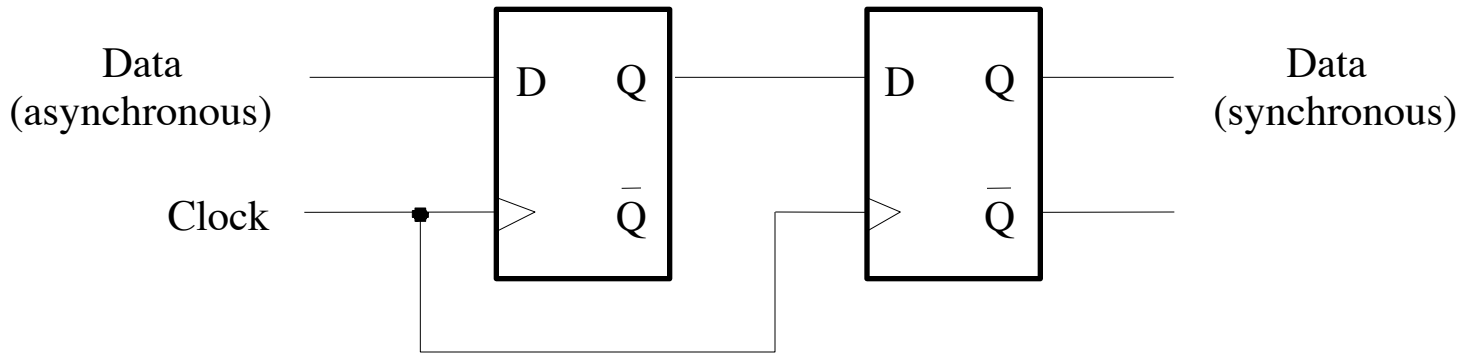
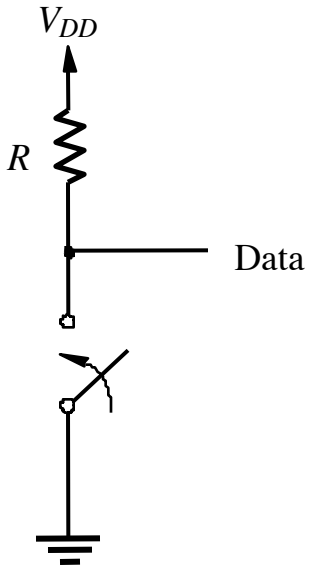
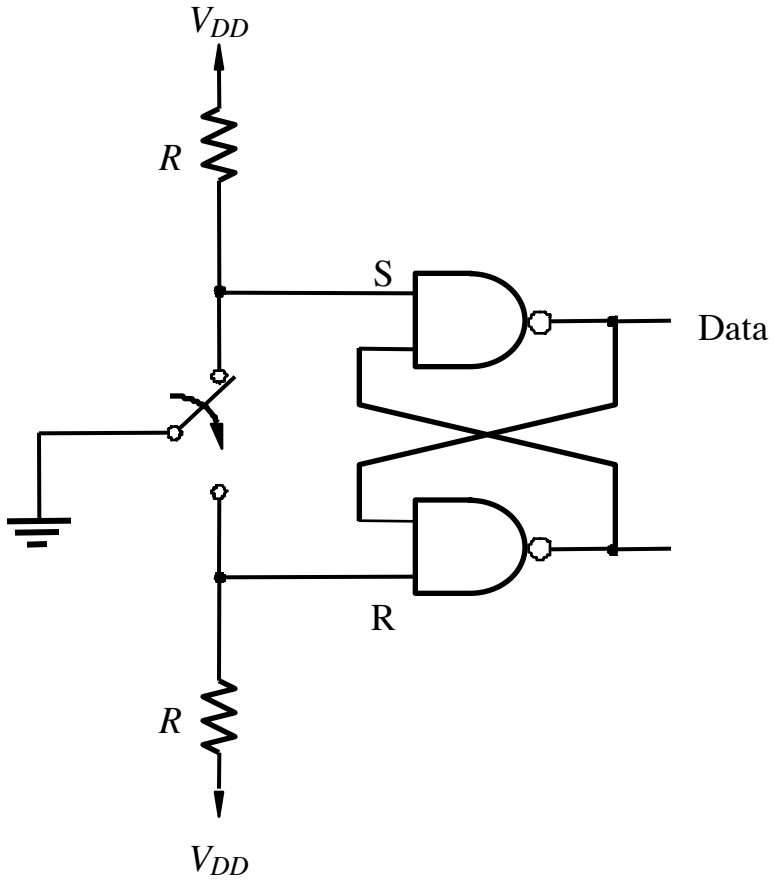


Figure 7.50. Asynchronous inputs.



(a) Single-pole single-throw switch



(b) Single-pole double-throw switch with a basic SR latch

Figure 7.51. Switch debouncing circuit.

```
Q = 0 ;
R = A ;
while ((R − B) > 0) do
        R = R − B ;
        Q = Q + 1 ;
end while ;
```

Figure P7.1. Pseudo-code for integer division.

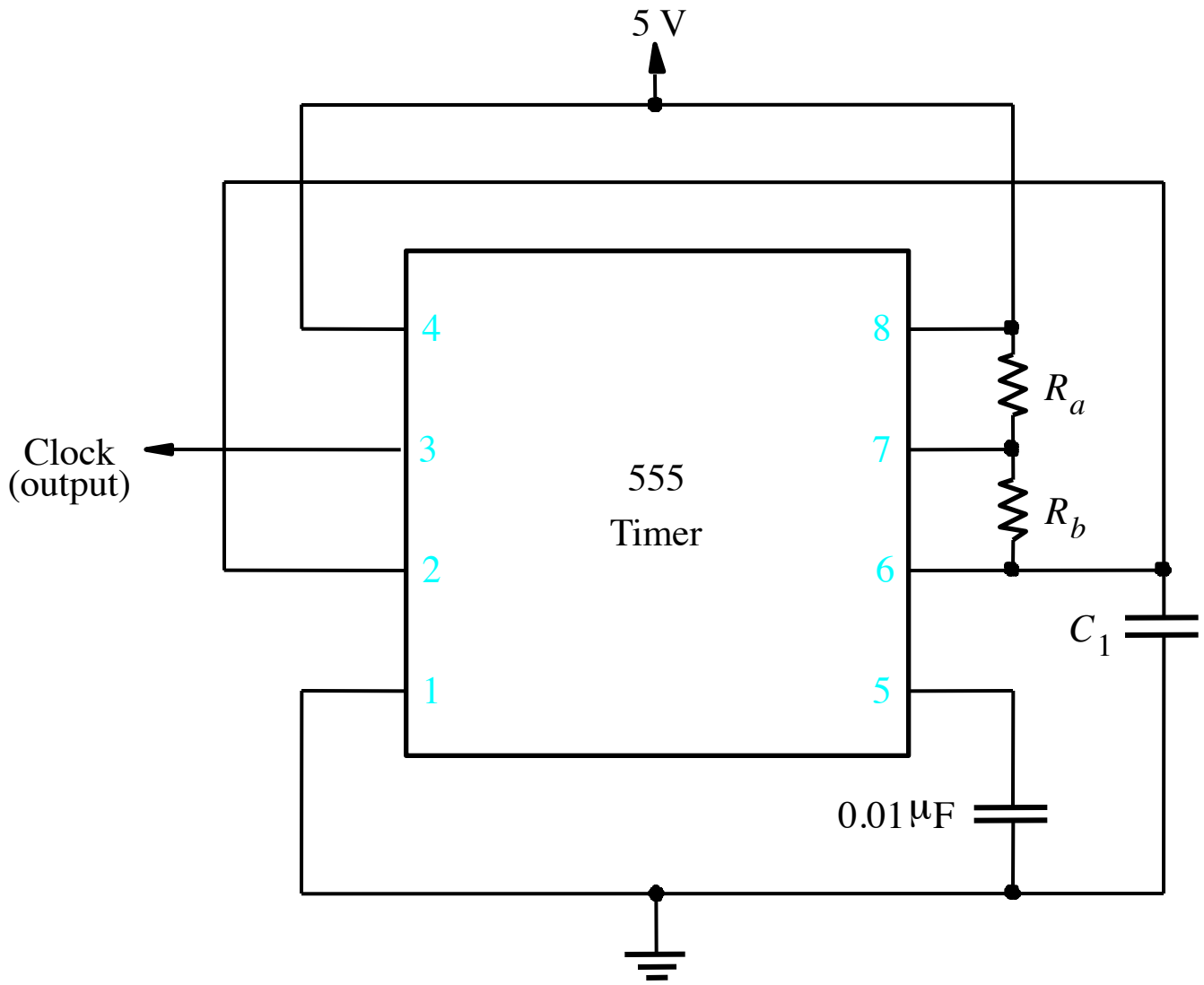


Figure P7.2. The 555 programmable timer chip.