Chapter 7: Flip-flops, Registers, Counters and a simple Processor

- Until now we have only considered combinational logic i.e. logic without memory where any change to an input directly effects the output.
- Simple example in figure 7.1 and 7.2

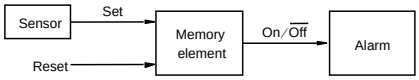


 Figure 7.1 Control of an alarm system

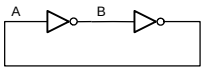
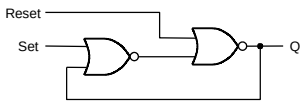


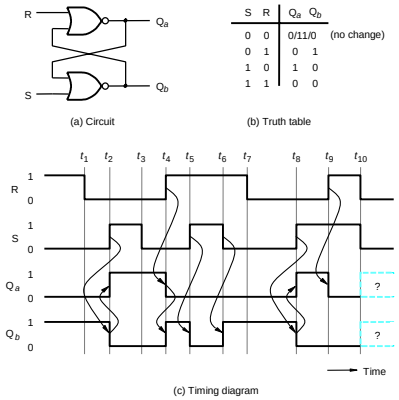
 Figure 7.2 A simple memory element

7.1 Basic Latch (Afr: Grendel)

- Set (Afr: stel) and Reset (Afr: herstel) signals are required to change the state of a memory element.
- Example built with two NOR gates in figure 7.4 and 7.5
- Often called a SR latch.



July 2000 Figure 7.4 A memory element with NOR gates 216

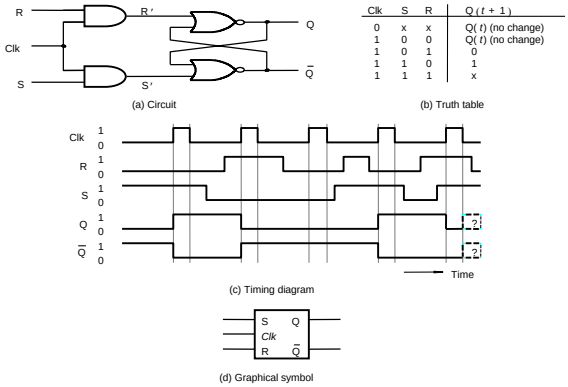


July 2000 Figure 7.5 A latch built with NOR gates 217

7.2 Gated SR latch

- We often need to isolate a latch from the inputs using a gate, enable or clock signal.
- In figure 7.6 S and R may change while clk is low (latch disabled).
- Q(t) = Q(t+1)

July 2000 Univ. of Stellenbosch - Digital Systems 144 218



July 2000 Figure 7.6 Gated SR latch 219

7.2.1 Gated SR latch with NAND gates.

- NAND gates require fewer transistors to implement. Will therefore get preference.
- See figure 7.7

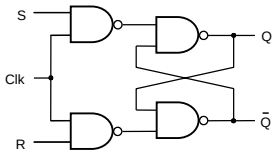
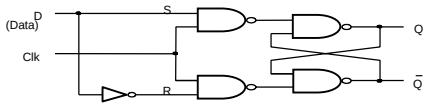


Figure 7.7 Gated SR latch with NAND gates

7.3 Gated D Latch

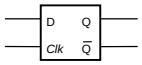
- D Latch, D-type latch (Afr: D-tipe grendel)
- Figure 7.8.
- Data Latch – very useful for implementing Data memory.
- Level sensitive vs. edge triggered.
- Transparent latch.
- Data and load signals.



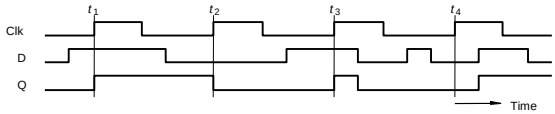
(a) Circuit

Clk	D	Q(t + 1)
0	x	Q(t)
1	0	0
1	1	1

(b) Truth table



(c) Graphical symbol



(d) Timing diagram

Figure 7.8 Gated D latch

7.3.1 Effects of Propagation Delays

- Setup and hold times
- Propagation delay times
- Figure 7.9

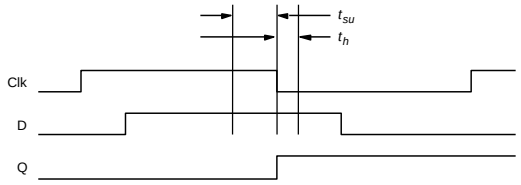


Figure 7.9 Setup and hold times

7.4 Master-Slave and Edge-Triggered D Flip-Flops

- In level sensitive latches the output will change with the input while the clock is active.
- Edge-triggered flip-flips only change the output at the edge of the clock.
- (Afr: Meester-slaaf en kantgesnellerde D-tipe wipkringe)

7.4.1 Master-Slave D Flip-Flop

- In figure 7.10 the master flip-flop follows the input while the clock is high and the slave flip-flop follows the output of the master while the clock is low.
- The output only changes at the negative edge of the clock and is insensitive to the input value during the rest of the clock cycle.

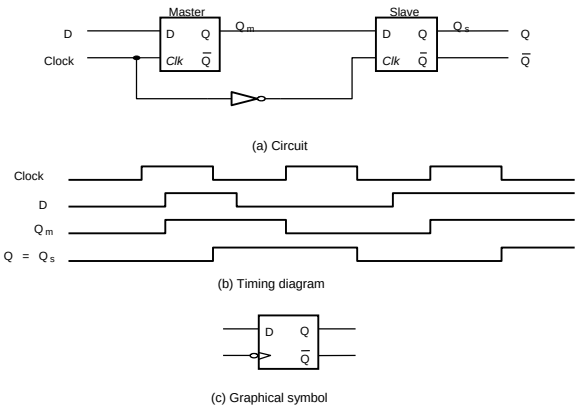


Figure 7.10 Master-slave D flip-flop

Level-sensitive vs. Edge-triggering

- Example see figure 7.12

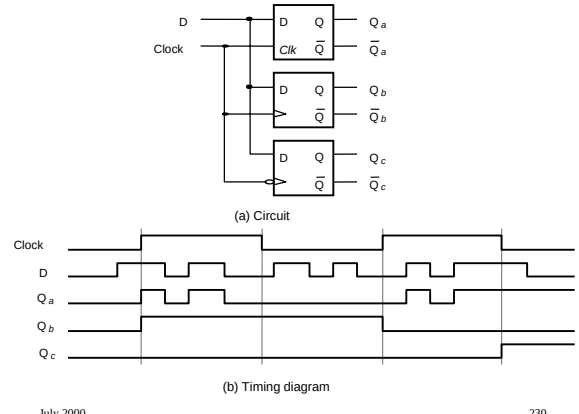
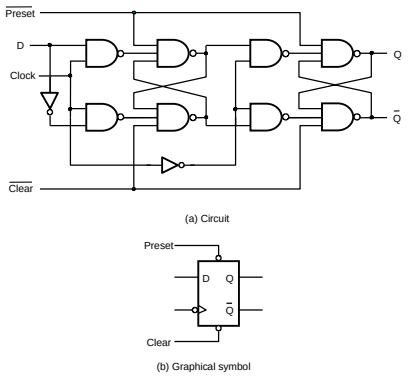


Figure 7.12 Comparison of level-sensitive and edge-triggered

7.4.3D Flip-Flops with Clear and Preset

- Preset and clear are asynchronous with the clock.
- Higher priority.
- Usually used to initialize a flip-flop after power is turned on.
- See figure 7.13.

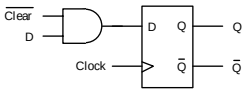


July 2000 Figure 7.13 Master-slave D flip-flop with *Clear* and *Preset* 232

Synchronous reset

- See figure 7.15

July 2000 Univ. of Stellenbosch - Digital Systems 144 233



July 2000 Figure 7.15 Synchronous reset for a D flip-flop 234

7.5 T Flip-Flop

- Toggle means “going to the opposite state when a clock transition occurs”.
- Useful to build counters.
- See figure 7.16.

July 2000 Univ. of Stellenbosch - Digital Systems 144 235

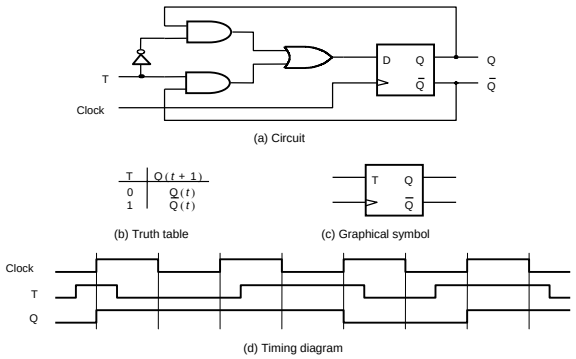


Figure 7.16 T flip-flop

7.6 JK Flip-Flop

- D input defined as
  D = JQ' + K'Q
- Combines behaviors of SR ant T flip-flops
- Behaves as a SR flip-flop when J = S and K = R for all input values except J = K = 1. For the latter it behaves as a T flip-flop.
- See figure 7.17.

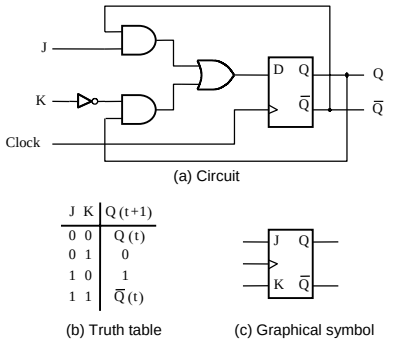


Figure 7.17 JK flip-flop

7.7 Summary of Technology

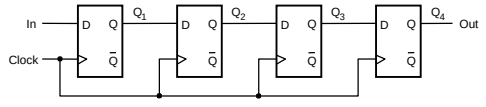
- Basic Latch – two NOR gates or two NAND gates.
- Gated Latch (SR or D)
- Flip-Flop (changes state at clock transition)
  - Edge-triggered flip-flop
  - Master-slave flip-flop

7.8 Registers

- Now moving from gate level to register level logic circuits.
- When a set of n identical flip-flops is used to store n bits, it is referred to as a *register*.
- ALL REGISTERS RELY ON A SINGLE UNBUFFERED CLOCK AND THE INHERENT DELAY OF THE CIRCUITS.

7.8.1 Shift Register

- Shifts the bits in a register one position per clock pulse from left-to-right or from right-to-left.
- See figure 7.18.



(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 7.18 A simple shift register

7.8.2 Parallel-Access Shift Register

- A shift register that can be loaded and read in parallel.
- Often used to convert parallel-to-serial and serial-to-parallel data streams.
- See figure 7.19.

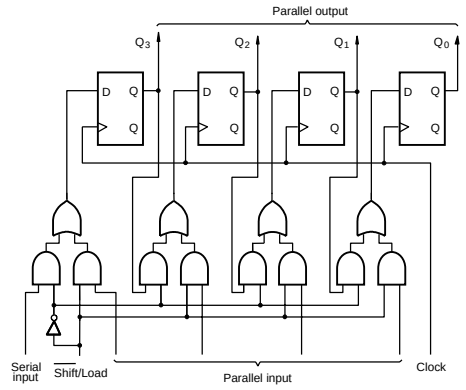


Figure 7.19 A simple shift register

7.9 Counters

- Counters are circuits that add or subtract values by 1.
- Simpler circuits than full adders will suffice.
- Two main types:
  - Asynchronous or ripple counters.
  - Synchronous counters.

7.9.1 Asynchronous Counters

- Up-counter with T flip-flops (figure 7.20)
- Down counter with T flip-flops (figure 7.21)

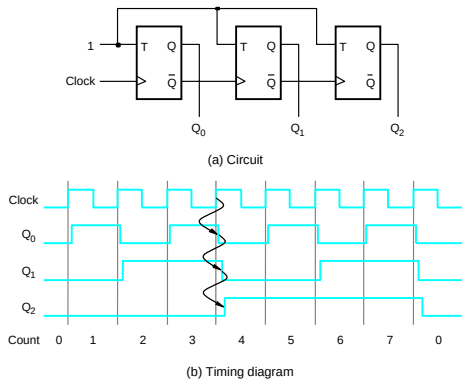
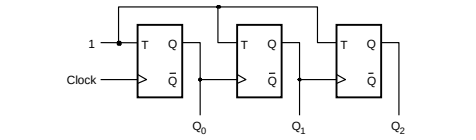
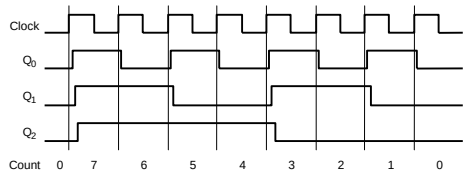


Figure 7.20 A three-bit up-counter



(a) Circuit



(b) Timing diagram

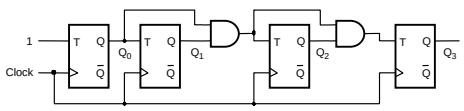
Figure 7.21 A three-bit down-counter

7.9.2 Synchronous Counters

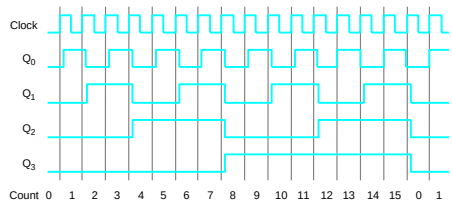
- Synchronous Counters with T flip-flops
  - Observe pattern in table 7.1 resulting in:
  - $T_0 = 1$
  - $T_1 = Q_0$
  - $T_2 = Q_0Q_1$
  - $T_3 = Q_0Q_1Q_2$
  - $T_n = Q_0Q_{1....}Q_{n-1}$
  - Implementation in figure 7.22
  - Note effect on speed of counter.
  - Enable and clear added in figure 7.23.

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

Table 7.1 Derivation of the synchronous up-counter



(a) Circuit



(b) Timing diagram

Figure 7.22 A four-bit synchronous up-counter

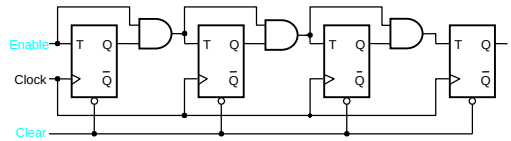


Figure 7.23 Inclusion of enable and clear capability

- Synchronous counter with D flip-flops
  - Formal proof in chapter 8.
  - $D_0 = Q_0 \oplus$ Enable
  - $D_1 = Q_1 \oplus Q_0$ Enable
  - $D_2 = Q_2 \oplus Q_1Q_0$ Enable
  - See figure 7.24
- 7.9.3 Counters with parallel load
  - See figure 7.25

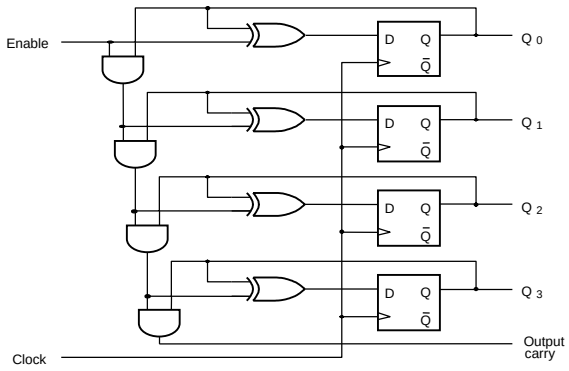


Figure 7.24 A four-bit counter with D flip-flop

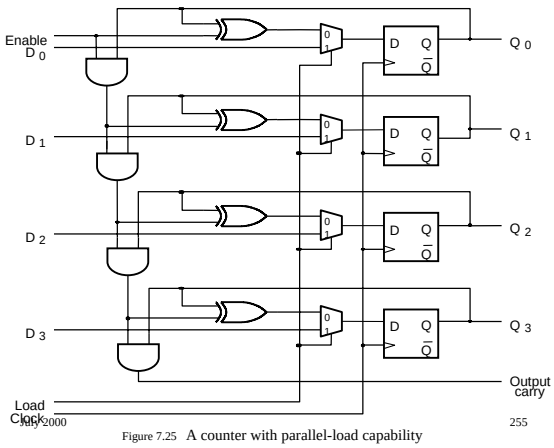
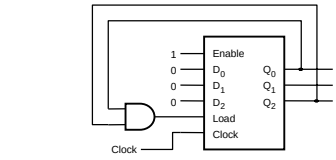


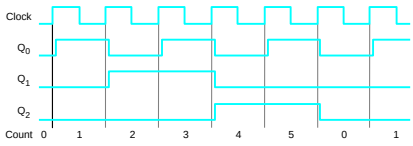
Figure 7.25 A counter with parallel-load capability

7.10 Reset Synchronization

- Circuit in figure 7.25 can be used to build a modulo-n counter. See figure 7.26 as an example of a modulo-6 counter using *synchronous load.*
- The circuit in figure 7.27 uses asynchronous reset inputs, but has a potential problem of cutting state 5 short – not recommended!

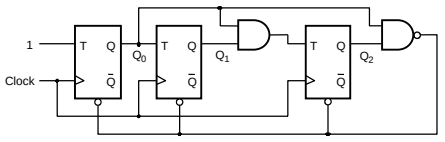


(a) Circuit

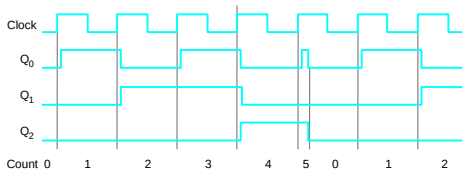


(b) Timing diagram

 Figure 7.26 A modulo-6 counter with synchronous reset



(a) Circuit

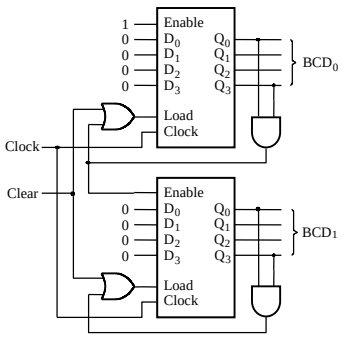


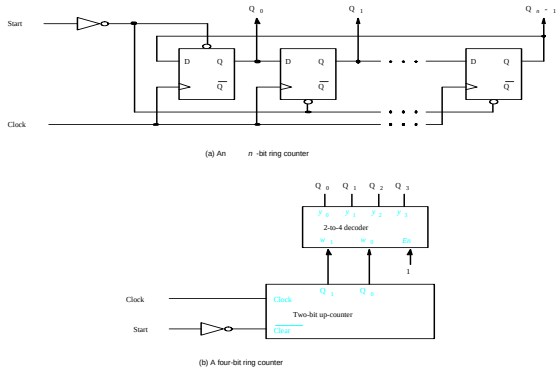
(b) Timing diagram

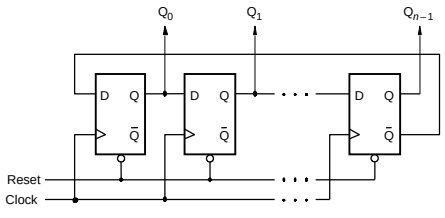
 Figure 7.27 A modulo-6 counter with asynchronous reset

7.11 Other types of counters

- 7.11.1 BCD counter (figure 7.28)
  - Counts decimal
- 7.11.2 Ring counter (figure 7.29)
  - One hot code
- 7.11.3 Johnson counter (figure 7.30)
  - Only one bit changes at a time







7.11.4 Remarks on counter design

- A number of “intuitive” counter designs have been presented in this chapter.
- A more general approach will be discussed in the next chapter.

Rest of chapter 7

- The rest of chapter 6 will not be tested in the examination.
- Some concepts are discussed in order to give the students a broader *system* perspective of the digital building blocks are typically used in building more complex digital systems such as computers.

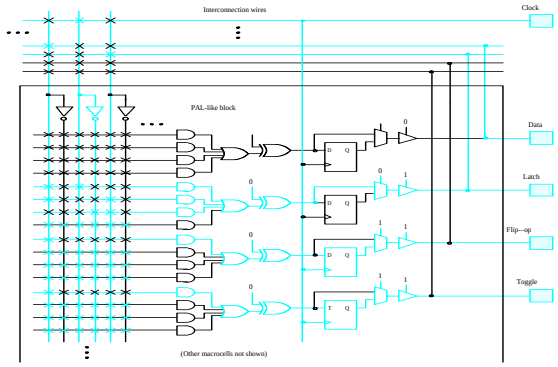


Figure 7.33 Implementation of a circuit in a CPLD

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY flipflop IS
      PORT (   D, Resetn, Clock    : IN      STD_LOGIC ;
               Q                   : OUT     STD_LOGIC) ;
END flipflop ;

ARCHITECTURE Behavior OF flipflop IS
BEGIN
      PROCESS ( Resetn, Clock )
      BEGIN
            IF Resetn = '0' THEN
                  Q <= '0' ;
            ELSIF Clock'EVENT AND Clock = '1' THEN
                  Q <= D ;
            END IF ;
      END PROCESS ;
END Behavior ;
```

 Figure 7.40 D flip-flop with asynchronous reset

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY reg8 IS
      PORT (   D                   : IN      STD_LOGIC_VECTOR(7 DOWNTO 0) ;
               Resetn, Clock       : IN      STD_LOGIC ;
               Q                   : OUT     STD_LOGIC_VECTOR(7 DOWNTO 0) ) ;
END reg8 ;

ARCHITECTURE Behavior OF reg8 IS
BEGIN
      PROCESS ( Resetn, Clock )
      BEGIN
            IF Resetn = '0' THEN
                  Q <= "00000000" ;
            ELSIF Clock'EVENT AND Clock = '1' THEN
                  Q <= D ;
            END IF ;
      END PROCESS ;
END Behavior ;
```

 Figure 7.46 Code for an eight-bit register with asynchronous clear

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE ieee.std_logic_unsigned.all ;
ENTITY upcount IS
    PORT (  Clock, Resetn, E    : IN    STD_LOGIC ;
            Q                   : OUT   STD_LOGIC_VECTOR (3 DOWNTO 0)) ;
END upcount ;

ARCHITECTURE Behavior OF upcount IS
    SIGNAL Count : STD_LOGIC_VECTOR (3 DOWNTO 0) ;
BEGIN
    PROCESS ( Clock, Resetn )
    BEGIN
        IF Resetn = '0' THEN
            Count <= "0000" ;
        ELSIF (Clock'EVENT AND Clock = '1') THEN
            IF E = '1' THEN
                Count <= Count + 1 ;
            ELSE
                Count <= Count ;
            END IF ;
        END IF ;
    END PROCESS ;
    Q <= Count ;
END Behavior ;
```


Figure 7.53 Code for a four-bit up-counter


```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY upcount IS
    PORT (  R                   : IN        INTEGER RANGE 0 TO 15 ;
            Clock, Resetn, L    : IN        STD_LOGIC ;
            Q                   : BUFFER    INTEGER RANGE 0 TO 15 ) ;
END upcount ;

ARCHITECTURE Behavior OF upcount IS
BEGIN
    PROCESS ( Clock, Resetn )
    BEGIN
        IF Resetn = '0' THEN
            Q <= 0 ;
        ELSIF (Clock'EVENT AND Clock = '1') THEN
            IF L = '1' THEN
                Q <= R ;
            ELSE
                Q <= Q + 1 ;
            END IF;
        END IF;
    END PROCESS;
END Behavior;
```

Figure 7.54 A four-bit counter with parallel load, using INTEGER signals

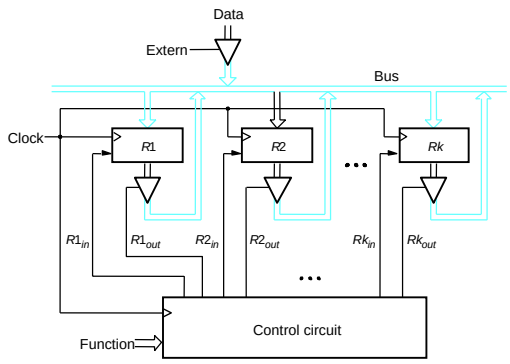


Figure 7.56 A digital system with *k* registers

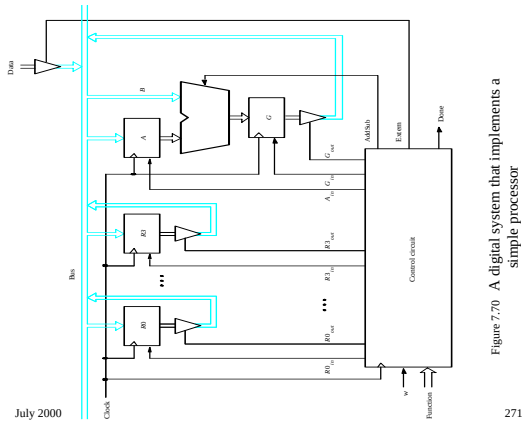


Figure 7.70 A digital system that implements a simple processor

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 7.2 Operations performed in the processor

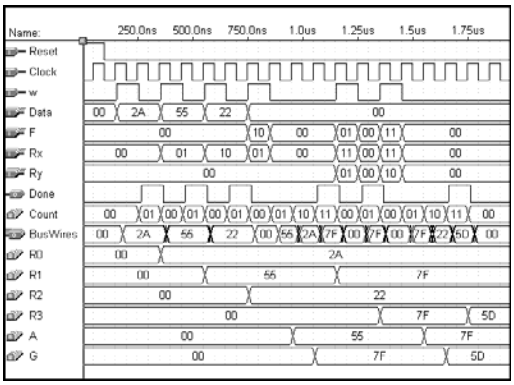


Figure 7.76 Timing simulation of the processor

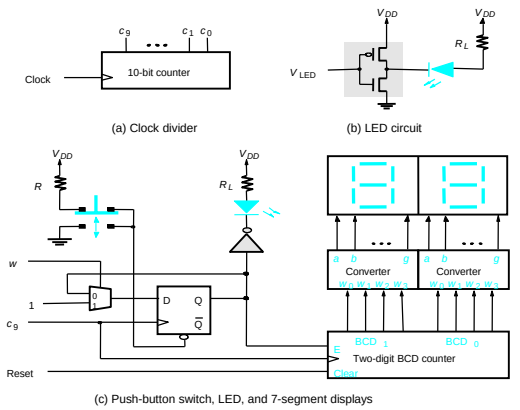


Figure 7.77 A reaction-timer circuit