The digital system consists of two parts:

- **Datapath circuit** — used to store, manipulate, and transfer data.
- **Control circuit** — controls the operation of the datapath. Usually, its built using an FSM.

### VHDL Code for a D Flip-Flop with an Enable Input

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

ENTITY dff_e IS
  PORT ( R, Resetn, E, Clock : IN STD_LOGIC;
         Q : BUFFER STD_LOGIC);
END dff_e;

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

### VHDL Code for an N-bit Register with an Enable Input

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

ENTITY reg_e IS
  GENERIC (N : INTEGER := 4);
  PORT ( R : IN STD_LOGIC_VECTOR(N - 1 DOWNTO 0);
         Resetn : IN STD_LOGIC;
         E, Clock : IN STD_LOGIC;
         Q : OUT STD_LOGIC_VECTOR(N - 1 DOWNTO 0));
END reg_e;

ARCHITECTURE Behavior OF reg_e IS
BEGIN
  PROCESS (Resetn, Clock)
  BEGIN
    IF Resetn = '0' THEN
      Q <= (OTHERS => '0');
    ELSIF Clock'EVENT AND Clock = '1' THEN
      IF E = '1' THEN
        Q <= R;
      END IF;
    END IF;
  END PROCESS;
END Behavior;
```

### VHDL Code for a Right-to-Left Shift Register with Parallel Load and Enable

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

ENTITY shiftline_e IS
  GENERIC (N : INTEGER := 4);
  PORT ( R : IN STD_LOGIC_VECTOR(N - 1 DOWNTO 0);
         L, E, w : IN STD_LOGIC;
         Clock : IN STD_LOGIC;
         Q : BUFFER STD_LOGIC_VECTOR(N - 1 DOWNTO 0));
END shiftline_e;

ARCHITECTURE Behavior OF shiftline_e IS
BEGIN
  PROCESS (L, E, Clock)
  BEGIN
    ... con't
  END PROCESS;
END Behavior;
```

---

**Figure 10.1** A flip-flop with an enable input

**Figure 10.2** VHDL code for a D flip-flop with an enable input

**Figure 10.3** VHDL code for an n-bit register with an enable input

**Figure 10.4a** Code for a right-to-left shift register with an enable input
Figure 10.4b Code for a right-to-left shift register with an enable input (con't)

```vhdl
WAIT UNTIL Clock'EVENT AND Clock = '1';
IF E = '1' THEN
  IF L = '1' THEN
    Q <= R;
  ELSE
    Q(0) <= w;
    Genbits: FOR i IN 1 TO N - 1 LOOP
      Q(i) <= Q(i - 1);
    END LOOP;
  END IF;
END IF;
END PROCESS;
END Behavior;
```

Figure 10.5a Component declaration statements for building blocks

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;
PACKAGE components IS
  -- 2-to-1 multiplexer
  COMPONENT mux2to1
    PORT ( w0, w1 : IN STD_LOGIC;
           s : IN STD_LOGIC;
           f : OUT STD_LOGIC );
  END COMPONENT;

  -- D flip-flop with 2-to-1 multiplexer connected to D
  COMPONENT muxdff
    PORT ( D0, D1, Sel, Clock : IN STD_LOGIC;
           Q : OUT STD_LOGIC );
  END COMPONENT;

  -- n-bit register with enable
  COMPONENT regne
    GENERIC ( N: INTEGER := 4 );
    PORT ( R : IN STD_LOGIC_VECTOR(N - 1 DOWNTO 0);
           Resetn : IN STD_LOGIC;
           E, Clock : IN STD_LOGIC;
           Q : OUT STD_LOGIC_VECTOR(N - 1 DOWNTO 0) );
  END COMPONENT;

  -- n-bit right-to-left shift register with parallel load and enable
  COMPONENT shiftlne
    GENERIC ( N: INTEGER := 4 );
    PORT ( R : IN STD_LOGIC_VECTOR(N - 1 DOWNTO 0);
           L, E, w : IN STD_LOGIC;
           Clock : IN STD_LOGIC;
           Q : BUFFER STD_LOGIC_VECTOR(N - 1 DOWNTO 0) );
  END COMPONENT;

  -- n-bit left-to-right shift register with parallel load and enable
  COMPONENT shiftrne
    GENERIC ( N: INTEGER := 4 );
    PORT ( R : IN STD_LOGIC_VECTOR(N - 1 DOWNTO 0);
           L, E, w : IN STD_LOGIC;
           Clock : IN STD_LOGIC;
           Q : BUFFER STD_LOGIC_VECTOR(N - 1 DOWNTO 0) );
  END COMPONENT;

END components;
```

Figure 10.5b Component declaration statements for building blocks (con't)

```vhdl
-- n-bit right-to-left shift register with parallel load and enable
COMPONENT shiftlne
  GENERIC ( N: INTEGER := 4 );
  PORT ( R : IN STD_LOGIC_VECTOR(N - 1 DOWNTO 0);
         L, E, w : IN STD_LOGIC;
         Clock : IN STD_LOGIC;
         Q : BUFFER STD_LOGIC_VECTOR(N - 1 DOWNTO 0) );
END COMPONENT;

-- n-bit left-to-right shift register with parallel load and enable
COMPONENT shiftrne
  GENERIC ( N: INTEGER := 4 );
  PORT ( R : IN STD_LOGIC_VECTOR(N - 1 DOWNTO 0);
         L, E, w : IN STD_LOGIC;
         Clock : IN STD_LOGIC;
         Q : BUFFER STD_LOGIC_VECTOR(N - 1 DOWNTO 0) );
END COMPONENT;
```

Figure 10.5c Component declaration statements for building blocks (con't)

```vhdl
-- up-counter that counts from 0 to modulus
COMPONENT upcount
  GENERIC ( modulus: INTEGER := 8 );
  PORT ( Resetn : IN STD_LOGIC;
         Clock, E, L : IN STD_LOGIC;
         R : IN INTEGER RANGE 0 TO modulus - 1;
         Q : BUFFER INTEGER RANGE 0 TO modulus - 1 );
END COMPONENT;

-- down-counter that counts from modulus down to 0
COMPONENT downcnt
  GENERIC ( modulus: INTEGER := 8 );
  PORT ( Clock, E, L : IN STD_LOGIC;
         Q : BUFFER INTEGER RANGE 0 TO modulus - 1 );
END COMPONENT;
```

Figure 10.5d Component declaration statements for building blocks (con’t)

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;
PACKAGE components IS
  -- 2-to-1 multiplexer
  COMPONENT mux2to1
    PORT ( w0, w1 : IN STD_LOGIC;
           s : IN STD_LOGIC;
           f : OUT STD_LOGIC );
  END COMPONENT;

  -- D flip-flop with 2-to-1 multiplexer connected to D
  COMPONENT muxdff
    PORT ( D0, D1, Sel, Clock : IN STD_LOGIC;
           Q : OUT STD_LOGIC );
  END COMPONENT;

  -- n-bit register with enable
  COMPONENT regne
    GENERIC ( N: INTEGER := 4 );
    PORT ( R : IN STD_LOGIC_VECTOR(N - 1 DOWNTO 0);
           Resetn : IN STD_LOGIC;
           E, Clock : IN STD_LOGIC;
           Q : OUT STD_LOGIC_VECTOR(N - 1 DOWNTO 0) );
  END COMPONENT;

  -- n-bit right-to-left shift register with parallel load and enable
  COMPONENT shiftlne
    GENERIC ( N: INTEGER := 4 );
    PORT ( R : IN STD_LOGIC_VECTOR(N - 1 DOWNTO 0);
           L, E, w : IN STD_LOGIC;
           Clock : IN STD_LOGIC;
           Q : BUFFER STD_LOGIC_VECTOR(N - 1 DOWNTO 0) );
  END COMPONENT;

  -- n-bit left-to-right shift register with parallel load and enable
  COMPONENT shiftrne
    GENERIC ( N: INTEGER := 4 );
    PORT ( R : IN STD_LOGIC_VECTOR(N - 1 DOWNTO 0);
           L, E, w : IN STD_LOGIC;
           Clock : IN STD_LOGIC;
           Q : BUFFER STD_LOGIC_VECTOR(N - 1 DOWNTO 0) );
  END COMPONENT;

END components;
```

Static Random Access Memory

- **SRAM** is used when a large amount of data needs to be stored.
- **SRAM block** is 2-dimensional array of **SRAM cells** where each cell stores 1-bit.
- To store \( m \) elements of \( n \)-bits, the aspect ratio of the SRAM array would be \( m \times n \).
\[ B = 0; \]
\[ \text{while } A \neq 0 \text{ do} \]
\[ \quad \text{if } a_0 = 1 \text{ then} \]
\[ \qquad B = B + 1; \]
\[ \quad \text{End if;} \]
\[ \quad \text{Right-shift } A; \]
\[ \text{End while;} \]
LIBRARY ieee;
USE ieee.std_logic_1164.all;
LIBRARY work;
USE work.components.all;

ENTITY bitcount IS
PORT( Clock, Resetn : IN STD_LOGIC;
     LA, s : IN STD_LOGIC;
     Data : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
     B : BUFFER INTEGER RANGE 0 to 8;
     Done : OUT STD_LOGIC);
END bitcount;

ARCHITECTURE Behavior OF bitcount IS

TYPE State_type IS (S1, S2, S3);
SIGNAL y : State_type;
SIGNAL A : STD_LOGIC_VECTOR(7 DOWNTO 0);
SIGNAL z, EA, LB, EB, low : STD_LOGIC;

BEGIN
  FSM_transitions: PROCESS (Resetn, Clock)
  BEGIN
    IF Resetn = '0' THEN
      y <= S1;
    ELSEIF (Clock'EVENT AND Clock = '1') THEN
      CASE y IS
      WHEN S1 =>
        IF s = '0' THEN y <= S1; ELSE y <= S2; END IF;
      WHEN S2 =>
        IF z = '0' THEN y <= S2; ELSE y <= S3; END IF;
      WHEN S3 =>
        IF s = '1' THEN y <= S3; ELSE y <= S1; END IF;
      END CASE;
    END IF;
  END PROCESS;

  FSM_outputs: PROCESS (y, s, A(0), z)
  BEGIN
    EA <= '0'; LB <= '0'; EB <= '0'; Done <= '0';
    CASE y IS
    WHEN S1 =>
      LB <= '1'; EB <= '1';
      IF s = '0' AND LA = '1' THEN EA <= '1'; ELSE EA <= '0'; END IF;
    WHEN S2 =>
      EA <= '1';
      IF A(0) = '1' THEN EB <= '1'; ELSE EB <= '0'; END IF;
    WHEN S3 =>
      Done <= '1';
    END CASE;
  END PROCESS;

END Behavior;

Figure 10.13a VHDL code for the bit-counting circuit

Figure 10.13b VHDL code for the bit-counting circuit (cont)

Figure 10.13c VHDL code for the bit-counting circuit (cont)

Figure 10.14 Simulation results for the bit-counting circuit

Figure 10.15 An algorithm for multiplication

(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 10.16 ASM chart for the multiplier
Figure 10.17 Datapath circuit for the multiplier

Figure 10.18 ASM chart for the multiplier control circuit

Figure 10.19a VHDL code for the multiplier circuit

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;
USE work.components.all;
ENTITY multiply IS
  GENERIC ( N, NN : INTEGER := 8, 16 );
  PORT ( Clock : IN STD_LOGIC;
          Resetn : IN STD_LOGIC;
          LA, LB, s : IN STD_LOGIC;
          DataA : IN STD_LOGIC_VECTOR(N-1 DOWNTO 0);
          DataB : IN STD_LOGIC_VECTOR(N-1 DOWNTO 0);
          P : BUFFER STD_LOGIC_VECTOR(NN-1 DOWNTO 0);
          Done : OUT STD_LOGIC )
END multiply;
ARCHITECTURE Behavior OF multiply IS
  TYPE State_type IS ( S1, S2, S3 );
  SIGNAL y: State_type;
  SIGNAL Psel, z, EA, EB, EP, Zero: STD_LOGIC;
  SIGNAL B, N_Zeros: STD_LOGIC_VECTOR(N-1 DOWNTO 0);
  SIGNAL A, Ain, DataP, Sum: STD_LOGIC_VECTOR(NN-1 DOWNTO 0);
BEGIN
  FSM_transitions: PROCESS ( Resetn, Clock )
  BEGIN
    IF Resetn = '0' THEN
      y <= S1;
    ELSIF ( Clock'EVENT AND Clock = '1') THEN
      CASE y IS
        WHEN S1 =>
          IF s = '0' THEN y <= S1; ELSE y <= S2; END IF;
        WHEN S2 =>
          IF z = '0' THEN y <= S2; ELSE y <= S3; END IF;
        WHEN S3 =>
          IF s = '1' THEN y <= S3; ELSE y <= S1; END IF;
      END CASE;
    END IF;
  END PROCESS;
  FSM_outputs: PROCESS ( y, s, LA, LB, B(0) )
  BEGIN
    EP <= '0'; EA <= '0'; EB <= '0'; Done <= '0'; Psel <= '0';
    CASE y IS
      WHEN S1 =>
        EP <= '1';
        IF s = '0' AND LA = '1' THEN EA <= '1'; ELSE EA <= '0'; END IF;
        IF s = '0' AND LB = '1' THEN EB <= '1'; ELSE EB <= '0'; END IF;
      WHEN S2 =>
        EA <= '1'; EB <= '1'; Psel <= '1';
        IF B(0) = '1' THEN EP <= '1';
        ELSE EP <= '0'; END IF;
      WHEN S3 =>
        Done <= '1';
    END CASE;
  END PROCESS;
  -- Define the datapath circuit
  Zero <= '0'; N_Zeros <= OTHERS => '0';
  Ain <= N_Zeros & DataA;
  ShiftA : shiftlne GENERIC MAP ( N => NN )
    PORT MAP ( Ain, LA, EA, Zero, Clock, A );
  ShiftB : shiftrne GENERIC MAP ( N => N )
    PORT MAP ( DataB, LB, EB, Zero, Clock, B );
  z <= '1' WHEN B = N_Zeros ELSE '0';
  Sum <= A + P;
  -- Define the 2n2 - to - 1 multiplexers for DataP
  GenMUX FOR ( N TO NN GENERATE
    Multi_map301 PORT MAP ( Zero, Sum, P, DataP ) );
  END GENERATE;
  RegP : regne GENERIC MAP ( N => NN )
    PORT MAP ( DataP, Resetn, EP, Clock, P );
  END Behavior;
END multiply;

Figure 10.19b VHDL code for the multiplier circuit (con't)

Figure 10.19c VHDL code for the multiplier circuit (con't)

Figure 10.20 Simulation results for the multiplier circuit
An algorithm for division

1. Initialize $R$ with 0.
2. For $i$ from 0 to $n - 1$, do:
   - Left-shift $R$.
   - If $R \geq B$ then:
     - $q_i = 1$;
     - $R = R - B$;
   - Else:
     - $q_i = 0$;
     - End if.
   - End for.

(c) Pseudo-code

An example using decimal numbers

Using binary numbers

(c) An example of division using $n = 8$ clock cycles

Figure 10.21 An algorithm for division

Figure 10.22 ASM chart for the divider

Figure 10.23 Datapath circuit for the divider

Figure 10.24 ASM chart for the divider control circuit

Figure 10.25 An example of division using $n = 8$ clock cycles

Figure 10.26 An example of division using $n = 8$ clock cycles
Figure 10.27 Datapath circuit for the enhanced divider

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arITH.all;
USE work.components.all;
ENTITY divider IS
  GENERIC (N: INTEGER := 8);
  PORT (Clock : IN STD_LOGIC;
        Resetn : IN STD_LOGIC;
        s, LA, EB : IN STD_LOGIC;
        DataA : IN STD_LOGIC_VECTOR(N-1 DOWNTO 0);
        DataB : IN STD_LOGIC_VECTOR(N-1 DOWNTO 0);
        R, Q : BUFFER STD_LOGIC_VECTOR(N-1 DOWNTO 0);
        Done : OUT STD_LOGIC);
END divider;
ARCHITECTURE Behavior OF divider IS
  TYPE State_type IS (S1, S2, S3);
  SIGNAL y : State_type;
  SIGNAL Zero, Cout, z : STD_LOGIC;
  SIGNAL EA, Rsel, LR, ER, ER0, LC, EC, R0 : STD_LOGIC;
  SIGNAL A, B, DataR : STD_LOGIC_VECTOR(N-1 DOWNTO 0);
  SIGNAL Sum : STD_LOGIC_VECTOR(N DOWNTO 0); -- adder outputs
  SIGNAL Count : INTEGER RANGE 0 TO N - 1;
  ...
BEGIN
  FSM_transitions: PROCESS (Resetn, Clock)
  BEGIN
    IF Resetn = '0' THEN y <= S1;
    ELSIF (Clock'EVENT AND Clock = '1') THEN
      CASE y IS
        WHEN S1 =>
          IF s = '0' THEN y <= S1; ELSE y <= S2; END IF;
        WHEN S2 =>
          IF z = '0' THEN y <= S2; ELSE y <= S3; END IF;
        WHEN S3 =>
          IF s = '1' THEN y <= S3; ELSE y <= S1; END IF;
      END CASE;
    END IF;
  END PROCESS;
  FSM_outputs: PROCESS (s, y, Cout, z)
  BEGIN
    LR <= '0'; ER <= '0'; ER0 <= '0';
    LC <= '0'; EC <= '0'; EA <= '0'; Done <= '0';
    Rsel <= '0';
    CASE y IS
      WHEN S1 =>
        ...
      WHEN S2 =>
        ...
      WHEN S3 =>
        Done <= '1';
    END CASE;
  END PROCESS;
  ...
END Behavior;

Figure 10.28a VHDL code for the divider circuit

BEGIN
  FSM_transitions: PROCESS (Resetn, Clock)
  BEGIN
    IF Resetn = '0' THEN y <= S1;
    ELSIF (Clock'EVENT AND Clock = '1') THEN
      CASE y IS
        WHEN S1 =>
          IF s = '0' THEN y <= S1; ELSE y <= S2; END IF;
        WHEN S2 =>
          IF z = '0' THEN y <= S2; ELSE y <= S3; END IF;
        WHEN S3 =>
          IF s = '1' THEN y <= S3; ELSE y <= S1; END IF;
      END CASE;
    END IF;
  END PROCESS;
  FSM_outputs: PROCESS (s, y, Cout, z)
  BEGIN
    LR <= '0'; ER <= '0'; ER0 <= '0';
    LC <= '0'; EC <= '0'; EA <= '0'; Done <= '0';
    Rsel <= '0';
    CASE y IS
      WHEN S1 =>
        ...
      WHEN S2 =>
        ...
      WHEN S3 =>
        Done <= '1';
    END CASE;
  END PROCESS;
  ...
END Behavior;

Figure 10.28b VHDL code for the divider circuit (con't)

BEGIN
  FSM_transitions: PROCESS (Resetn, Clock)
  BEGIN
    IF Resetn = '0' THEN y <= S1;
    ELSIF (Clock'EVENT AND Clock = '1') THEN
      CASE y IS
        WHEN S1 =>
          IF s = '0' THEN y <= S1; ELSE y <= S2; END IF;
        WHEN S2 =>
          IF z = '0' THEN y <= S2; ELSE y <= S3; END IF;
        WHEN S3 =>
          IF s = '1' THEN y <= S3; ELSE y <= S1; END IF;
      END CASE;
    END IF;
  END PROCESS;
  FSM_outputs: PROCESS (s, y, Cout, z)
  BEGIN
    LR <= '0'; ER <= '0'; ER0 <= '0';
    LC <= '0'; EC <= '0'; EA <= '0'; Done <= '0';
    Rsel <= '0';
    CASE y IS
      WHEN S1 =>
        ...
      WHEN S2 =>
        ...
      WHEN S3 =>
        Done <= '1';
    END CASE;
  END PROCESS;
  ...
END Behavior;

Figure 10.28c VHDL code for the divider circuit (con't)

BEGIN
  FSM_transitions: PROCESS (Resetn, Clock)
  BEGIN
    IF Resetn = '0' THEN y <= S1;
    ELSIF (Clock'EVENT AND Clock = '1') THEN
      CASE y IS
        WHEN S1 =>
          IF s = '0' THEN y <= S1; ELSE y <= S2; END IF;
        WHEN S2 =>
          IF z = '0' THEN y <= S2; ELSE y <= S3; END IF;
        WHEN S3 =>
          IF s = '1' THEN y <= S3; ELSE y <= S1; END IF;
      END CASE;
    END IF;
  END PROCESS;
  FSM_outputs: PROCESS (s, y, Cout, z)
  BEGIN
    LR <= '0'; ER <= '0'; ER0 <= '0';
    LC <= '0'; EC <= '0'; EA <= '0'; Done <= '0';
    Rsel <= '0';
    CASE y IS
      WHEN S1 =>
        ...
      WHEN S2 =>
        ...
      WHEN S3 =>
        Done <= '1';
    END CASE;
  END PROCESS;
  ...
END Behavior;

Figure 10.28d VHDL code for the divider circuit (con't)
An algorithm for finding the mean of \( k \) numbers:

\[
\text{Sum} = 0; \quad \text{for } i = k-1 \text{ down to } 0 \text{ do }
\text{Sum} = \text{Sum} + R_i; \\
M = \text{Sum} / k;
\]

(a) Pseudo-code

(b) ASM chart

Datapath circuit for the mean operation

ASM chart for the control circuit

Schematic of the mean circuit with an SRAM block

Simulation results for the mean circuit using SRAM

Pseudo-code for the sort operation

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
\( R_j = B; \)
\( R_j = A; \)
end if;
end for;
end for;
Figure 10.36 ASM chart for the sort operation

Figure 10.37 A part of the datapath circuit for the sort operation

Figure 10.38 A part of the datapath circuit for the sort operation

Figure 10.39 ASM chart for the control circuit

Figure 10.40a VHDL code for the sort operation

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE work.components.all;
ENTITY sort IS
GENERIC ( N : INTEGER := 4 );
PORT ( Clock, Resetn : IN STD_LOGIC;
s, WrInit, Rd : IN STD_LOGIC;
DataIn : IN STD_LOGIC_VECTOR(N - 1 DOWNTO 0);
RAdd : IN INTEGER RANGE 0 TO 3;
DataOut : BUFFER STD_LOGIC_VECTOR(N - 1 DOWNTO 0);
Done : BUFFER STD_LOGIC );
END sort;
ARCHITECTURE Behavior OF sort IS
TYPE State_type IS ( S1, S2, S3, S4, S5, S6, S7, S8, S9 );
SIGNAL y : State_type;
SIGNAL Ci, Cj : INTEGER RANGE 0 TO 3;
SIGNAL Rin : STD_LOGIC_VECTOR(3 DOWNTO 0);
TYPE RegArray IS ARRAY(3 DOWNTO 0) OF STD_LOGIC_VECTOR(N - 1 DOWNTO 0);
SIGNAL R : RegArray ;
SIGNAL RData, ABMux : STD_LOGIC_VECTOR(N - 1 DOWNTO 0);
SIGNAL Int, Csel, Wr, BltA : STD_LOGIC;
SIGNAL CMux, IMux : INTEGER RANGE 0 TO 3;
SIGNAL Ain, Bin, Aout, Bout : STD_LOGIC;
SIGNAL LI, LJ, EI, EJ, zi, zj : STD_LOGIC;
SIGNAL Zero : INTEGER RANGE 3 DOWNTO 0; -- parallel data for Ci = 0
SIGNAL A, B, ABData : STD_LOGIC_VECTOR(N - 1 DOWNTO 0);
BEGIN
FSM_transitions: PROCESS ( Resetn, Clock )
BEGIN
IF Resetn = '0' THEN
  y <= S1;
ELSIF ( Clock'EVENT AND Clock = '1') THEN
  CASE y IS
    WHEN S1 =>
      IF s = '0' THEN y <= S1;
      ELSE y <= S2;
    END IF;
    WHEN S2 => y <= S3;
    WHEN S3 => y <= S4;
    WHEN S4 => IF BltA = '1' THEN y <= S6; ELSE y <= S8; END IF;
    WHEN S6 => y <= S7;
    WHEN S7 => y <= S8;
    WHEN S8 =>
      IF zj = '0' THEN y <= S4;
      ELSEIF zi = '0' THEN y <= S2;
      ELSE y <= S9;
    END IF;
    WHEN OTHERS => y <= S1;
  END CASE;
END IF;
END PROCESS FSM_transitions;
END Behavior ;
WHEN S9 => IF y = 'T' THEN y <= S9; ELSE y <= S1; END IF;
END CASE;
END IF;
END PROCESS;

-- define the outputs generated by the FSM
Int <= '0' WHEN y = S1 ELSE '1';
Done <= '1' WHEN y = S9 ELSE '0';
FSM_outputs: PROCESS (y, zi, zj)
BEGIN
Li <= '0'; LJ <= '0'; EI <= '0'; EJ <= '0'; Csel <= '0';
Wr <= '0'; Ain <= '0'; Bin <= '0'; Aout <= '0'; Bout <= '0';
CASE y IS
WHEN S1 => Li <= '1'; EI <= '1';
WHEN S2 => Ain <= '1'; LJ <= '1'; EJ <= '1';
WHEN S3 => EJ <= '1';
WHEN S4 => Bin <= '1'; Csel <= '1';
WHEN S5 => -- no outputs asserted in this state
WHEN S6 => Csel <= '1'; Wr <= '1'; Aout <= '1';
WHEN S7 => Wr <= '1'; Bout <= '1';
WHEN S8 => Ain <= '1';
IF zj = '0' THEN
EJ <= '1';
ELSE
EJ <= '0';
END IF;
END IF;
WHEN S9 => -- Done is assigned 1 by conditional signal assignment
END CASE;
END PROCESS;

-- define the datapath circuit
Zero <= 0;
GenReg : FOR i IN 0 TO 3 GENERATE
Reg: regne GENERIC MAP (N => N)
PORT MAP (RData, Resetn, Rin (i), Clock, R(i));
END GENERATE;
RegA : regne GENERIC MAP (N => N)
PORT MAP (ABData, Resetn, Ain, Clock, A);
RegB : regne GENERIC MAP (N => N)
PORT MAP (ABData, Resetn, Bin, Clock, B);
BltA <= '1' WHEN B < A ELSE '0';
ABMux <= A WHEN Bout = '0' ELSE B;
RData <= ABMux WHEN WrInit = '0' ELSE DataIn;
OuterLoop : upcount GENERIC MAP (modulus => 4)
PORT MAP (Resetn, Clock, EI, LI, Zero, Ci);
InnerLoop : upcount GENERIC MAP (modulus => 4)
PORT MAP (Resetn, Clock, EJ, LJ, Cj);
CMux <= Ci WHEN Csel = '0' ELSE Cj;
IMux <= CMux WHEN Int = '1' ELSE Radd;
WITH IMux Select
ABData <= R(0) WHEN 0,
R(1) WHEN 1,
R(2) WHEN 2,
R(3) WHEN OTHERS;
RinDec : PROCESS (WrInit, Wr, IMux)
BEGIN
IF (WrInit OR Wr) = '1' THEN
CASE IMux IS
WHEN 0 => Rin <= "0001";
WHEN 1 => Rin <= "0010";
WHEN 2 => Rin <= "0100";
WHEN OTHERS => Rin <= "1000";
END CASE;
ELSE Rin <= "0000";
END IF;
END PROCESS;
Zi <= '1' WHEN Ci = 2 ELSE '0';
Zj <= '1' WHEN Cj = 3 ELSE '0';
DataOut <= OTHERS => Z;
END Behavior;
END Process;

Figure 10.40c VHDL code for the sort operation (con't)
Asynchronous Inputs to FFs

- Inputs generated asynchronously may violate setup and hold times of flip-flops.
- FF may take on a value between 0 and 1.
- This condition is called a metastable state.
- There is no guarantee of how long the circuit will persist in this state.
- Care must be taken to reduce the probability of having synchronization failure.
Figure 10.47: Asynchronous inputs

Figure 10.48: Switch debouncing circuit

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