REGISTER-TRANSFER LEVEL (RTL) SYSTEMS

- EXECUTION GRAPHS
- CLASSIFICATION OF EXECUTION GRAPHS
  according to sequencing structure
- ORGANIZATION OF SYSTEMS: functional and control units
- RTL SYSTEM ORGANIZATION: data and control subsystems.
- μVHDL SPECIFICATION OF RTL SYSTEMS
- REGISTER TRANSFER, REGISTER-TRANSFER GROUP, AND REGISTER-TRANSFER SEQUENCE
- ANALYSIS AND DESIGN PROCESS FOR RTL SYSTEMS
- DESIGN OF SERIAL-PARALLEL MULTIPLIER

REGISTER TRANSFER LEVEL (RTL)

1. DATA SUBSYSTEM (datapath) AND CONTROL SUBSYSTEM

2. THE STATE OF DATA SUBSYSTEM:
   CONTENTS OF A SET OF REGISTERS

3. THE FUNCTION OF THE SYSTEM PERFORMED AS A SEQUENCE OF REGISTER TRANSFERS (in one or more clock cycles)

4. A REGISTER TRANSFER:

   A TRANSFORMATION PERFORMED ON A DATA WHILE THE DATA TRANSFERRED FROM ONE REGISTER TO ANOTHER

5. THE SEQUENCE OF REGISTER TRANSFERS CONTROLLED BY THE CONTROL SUBSYSTEM (a sequential system)

EXAMPLE OF EXECUTION GRAPH

$$P_7(x) = \sum_{i=0}^{7} p_i x^i$$

TWO ALGORITHMS:

$$P_7(x) = ((((((p_7x + p_6)x + p_5)x + p_4)x + p_3)x + p_2)x + p_1)x + p_0$$

$$P_7(x) = (x^2)(x^2)[x^2(p_7x + p_6) + (p_5x + p_4)] + x^2(p_3x + p_2) + (p_1x + p_0)$$

Example 13.1

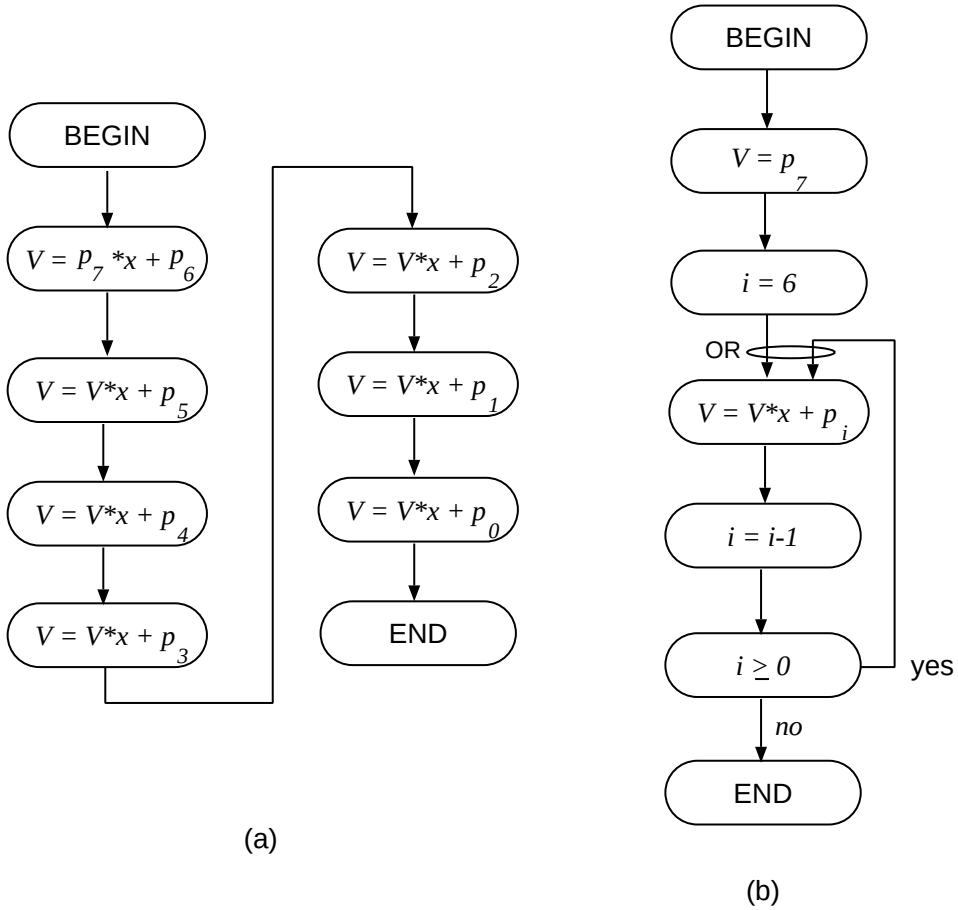


Figure 13.1: SEQUENTIAL EXECUTION GRAPHS FOR POLYNOMIAL EVALUATION: a) UNFOLDED; AND b) LOOP.

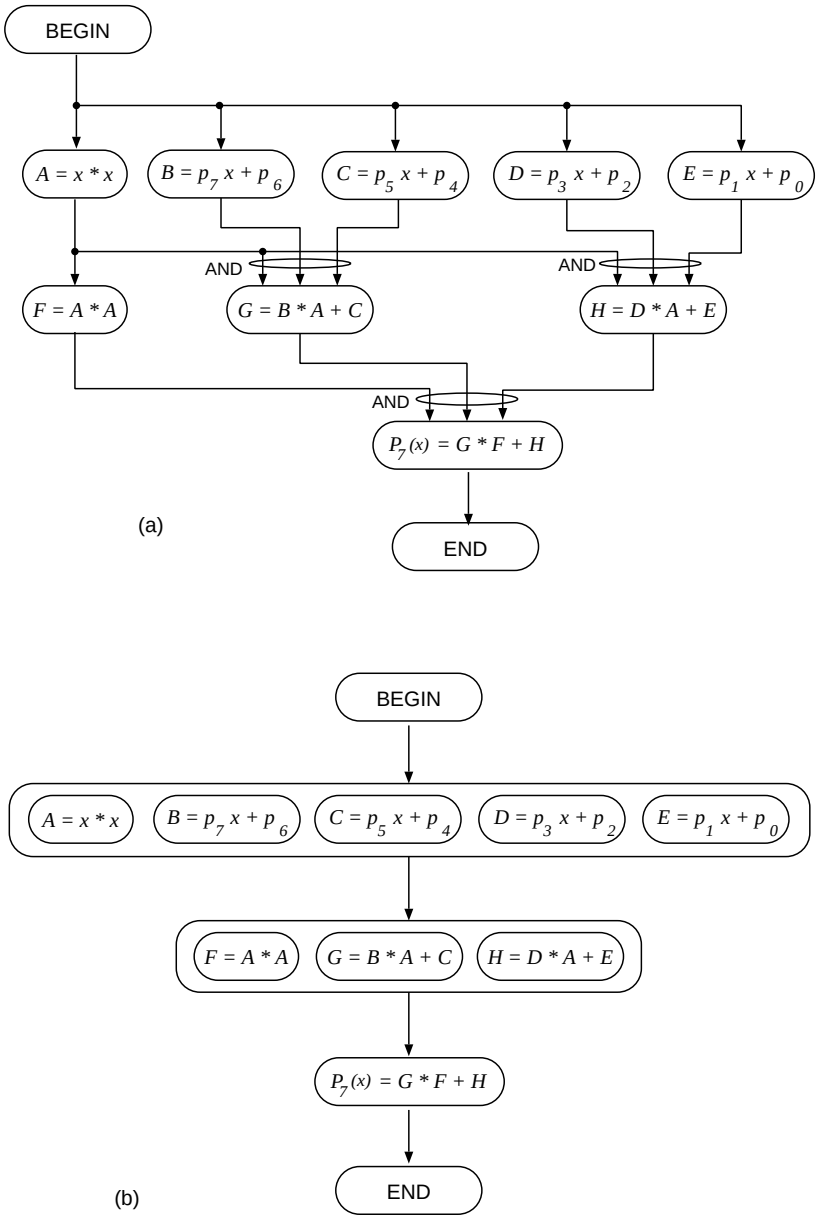


Figure 13.2: CONCURRENT EXECUTION GRAPHS FOR POLYNOMIAL EVALUATION: a) CONCURRENT; AND b) GROUP-SEQUENTIAL.

CONCURRENT INTO SEQUENTIAL

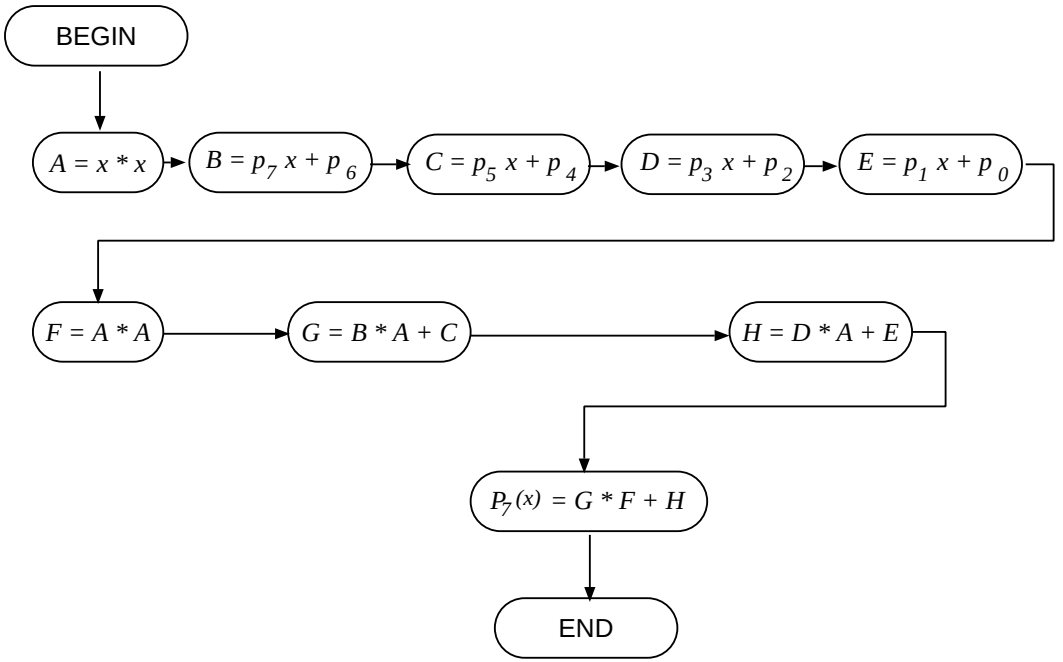


Figure 13.3: TRANSFORMATION OF A CONCURRENT EXECUTION GRAPH INTO A SEQUENTIAL ONE.

ORGANIZATION OF SYSTEMS

TWO FUNCTIONS:

- DATA TRANSFORMATIONS $\Longleftrightarrow$ FUNCTIONAL UNITS (operators)
- CONTROL OF DATA TRANSFORMATIONS
  AND THEIR SEQUENCING $\Longleftrightarrow$ CONTROL UNITS

TYPES OF SYSTEMS WITH RESPECT TO FUNCTIONAL UNITS:

- NONSHARING SYSTEM
- SHARING SYSTEM
- UNIMODULE SYSTEM

TYPES OF CONTROL

- CENTRALIZED CONTROL
- DECENTRALIZED CONTROL
- SEMICENTRALIZED CONTROL

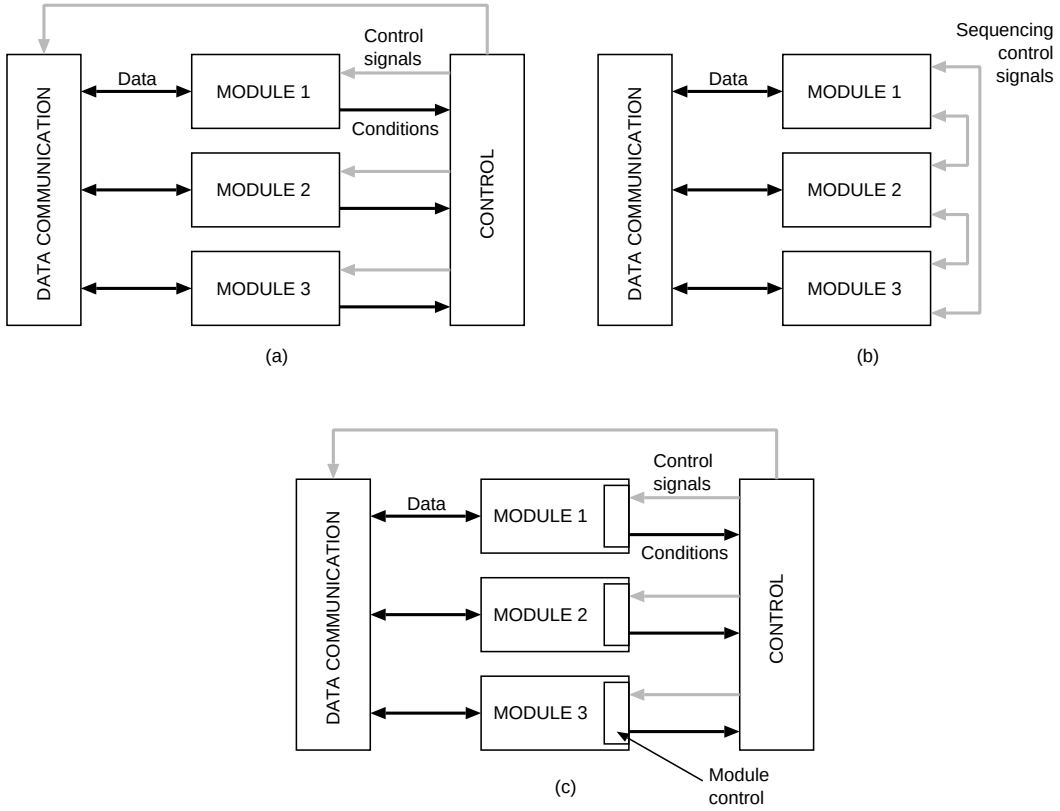


Figure 13.4: CONTROL STRUCTURES: a) CENTRALIZED; b) DECENTRALIZED; c) SEMICENTRALIZED.

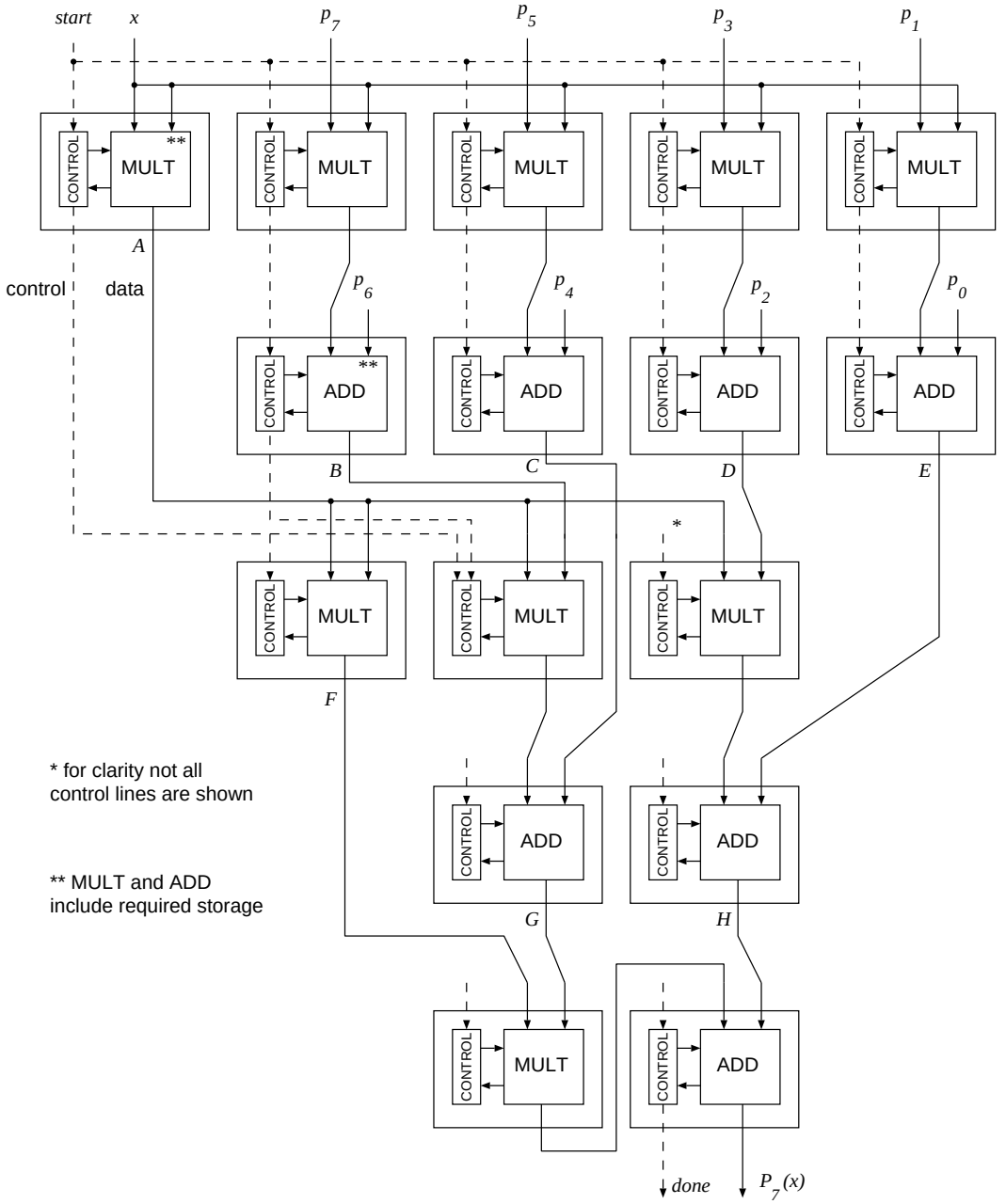


Figure 13.5: NON-SHARING DATA/DECENTRALIZED CONTROL IMPLEMENTATION FOR $P_7(x)$.

SHARING DATA SUBSYSTEM AND CENTRALIZED CONTROL

- MODULE OPERATIONS:

$$\textbf{if} \quad c1 = 1 \quad \textbf{then} \quad O_1 = I_{11} \times I_{12} + I_{13}$$
$$\textbf{if} \quad c2 = 1 \quad \textbf{then} \quad O_2 = I_{21} \times I_{22} + I_{23}$$

I_{ij} corresponds to the i-th input of set j

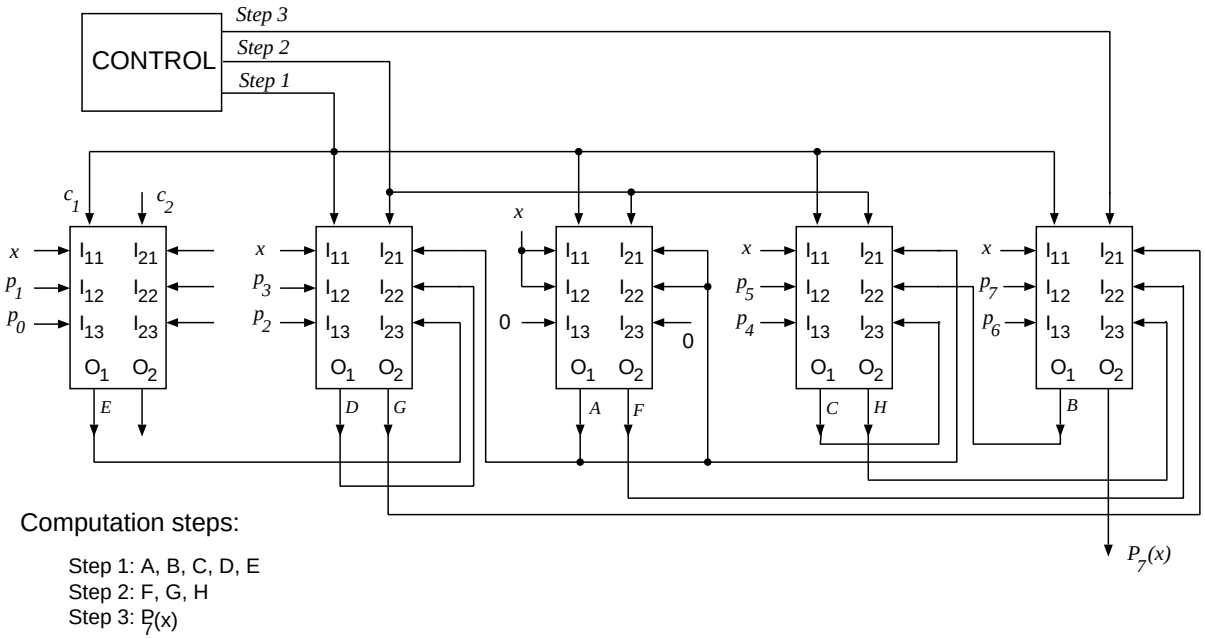


Figure 13.6: SHARING DATA SUBSYSTEM FOR $P_7(x)$.

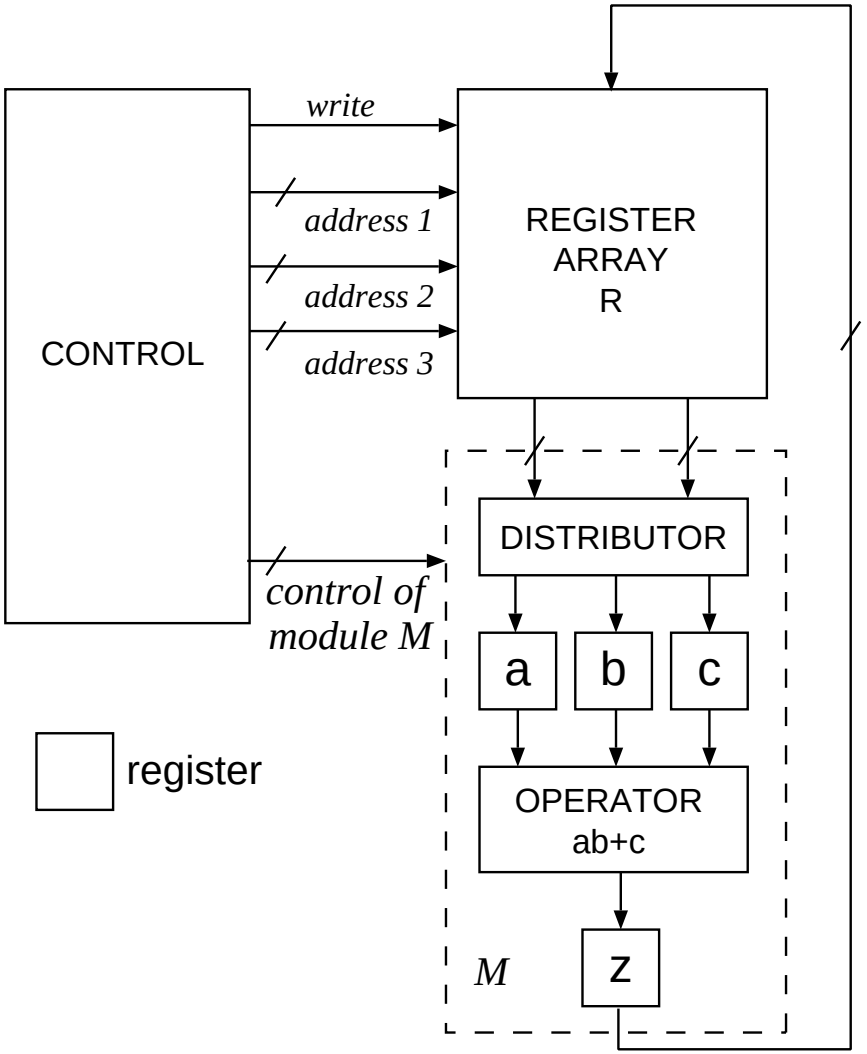


Figure 13.7: UNIMODULE DATA SUBSYSTEM FOR $P_7(x)$.

UNIMODULE DATA SUBSYSTEM

- A POSSIBLE MAPPING:

1: $a \leftarrow p_7,\ b \leftarrow x$
2: $c \leftarrow p_6$
3: $z \leftarrow p_7 x + p_6$
4: $T \leftarrow z$
5: $a \leftarrow T,\ c \leftarrow p_5$
6: $z \leftarrow Tx + p_5$
and so on.

ORGANIZATION OF RTL SYSTEM

- DATA SUBSYSTEM: implements data storage, data movement and data transfers

- CONTROL SUBSYSTEM: controls the operations in the data subsystem and their sequencing

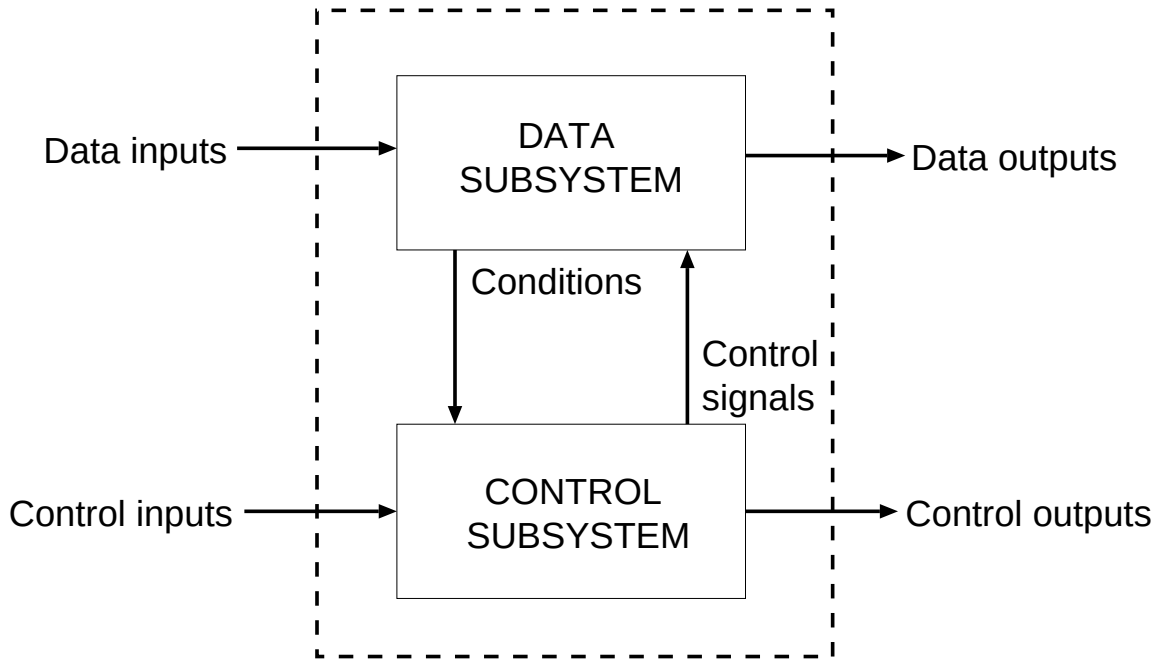


Figure 13.8: STRUCTURE OF A RTL SYSTEM.

SPECIFICATION OF RTL SYSTEMS USING μVHDL

Example 13.6

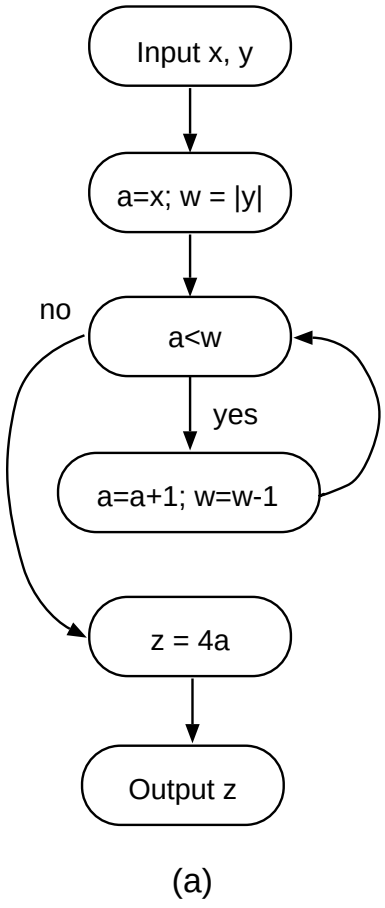
- A SYSTEM WHICH PERFORMS THE FOLLOWING COMPUTATION WITHOUT USING A TWO-OPERAND ADDER:

INPUTS: $x, y \in \{-128, \ldots, 127\}$
OUTPUT: $z \in \{-256, \ldots, 508\}$

FUNCTION: $$z = \begin{cases} 4\lceil (x + |y|)/2 \rceil & \textbf{if} \quad x < |y| \\ 4x & \textbf{otherwise} \end{cases}$$

- RTL SEQUENCE FOR THIS COMPUTATION IS BASED ON

$$a = (x + |y|)/2 = x + (|y| - x)/2$$



(a)

```
-- This is a high-level description; it is not intended
-- for synthesis. (The WHILE statement might not be
-- supported by a synthesis tool)

  PACKAGE inc_dec_pkg IS
    SUBTYPE SignDataT IS INTEGER RANGE -128 TO 127;
    SUBTYPE PosDataT  IS INTEGER RANGE    0 TO 128;
    SUBTYPE DataoutT  IS INTEGER RANGE -256 TO 508;
  END inc_dec_pkg;

  USE WORK.inc_dec_pkg.ALL;
  ENTITY inc_dec IS
    PORT(x_in,y_in: IN  SignDataT;
         z_out    : OUT DataoutT;
         clk      : IN  BIT);
  END inc_dec;

  ARCHITECTURE high_level OF inc_dec IS
  BEGIN
    PROCESS (clk)
      VARIABLE a : SignDataT;
      VARIABLE w : PosDataT;
    BEGIN
      IF (clk'EVENT AND clk = '1') THEN
        a:= x_in;              -- initialize variables
        w:= ABS(y_in);         -- compute abs(y_in)
        WHILE (a < w) LOOP
          a:= a+1;             -- if x < y compute
          w:= w-1;             -- by incr. x and decr. y
        END LOOP;
        z_out <= (a * 4);
      END IF;
    END PROCESS;
  END high_level;
```

(b)

Figure 13.9: SYSTEM IN EXAMPLE 13.6: a) EXECUTION GRAPH; b) HIGH-LEVEL DESCRIPTION.

PROCEDURE TO OBTAIN SPECIFICATION

1. DRAW AN EXECUTION GRAPH FOR THE COMPUTATION

2. WRITE THE μVHDL DESCRIPTION OF THE SYSTEM, BASED ON THE EXECUTION GRAPH

Example 13.7

- NEWTON-RAPHSON RECURRENCE FOR COMPUTING APPROXIMATION z TO RECIPROCAL OF $1/2 \leq x < 1$

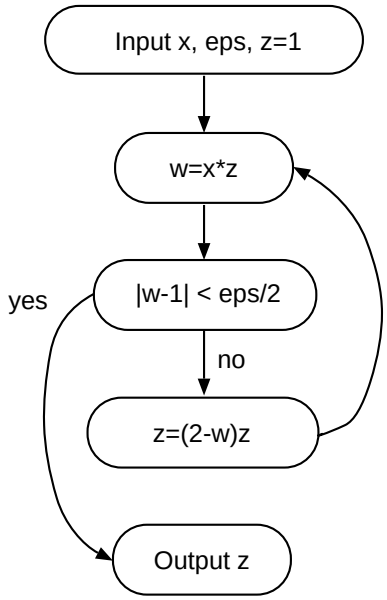
- RTL SEQUENCE DERIVED FROM

$$z_{i+1} = z_i(2 - xz_i), \quad z_0 = 1$$

- TERMINATES WHEN $x \times z_k - 1 < 0.5 \times \varepsilon$

- QUADRATIC CONVERGENCE

$$|xz_k - 1| = (xz_{k-1} - 1)^2$$



(a)

```
-- This is a high-level description; it is not intended
-- for synthesis. (The WHILE statement and REAL TYPE might
-- not be supported by a synthesis tool)

  PACKAGE recip_pkg IS
    SUBTYPE DatainT  IS REAL RANGE 0.5 TO 1.0;
    SUBTYPE DataoutT IS REAL RANGE 1.0 TO 2.0;
    SUBTYPE EpsT     IS REAL RANGE 0.0 TO 0.1;
  END recip_pkg;

  USE WORK.recip_pkg.ALL;
  ENTITY reciprocal IS
    PORT(x_in  : IN  DatainT ;
         eps_in: IN  EpsT    ;
         z_out : OUT DataoutT;
         clk   : IN  BIT     );
  END reciprocal;

  ARCHITECTURE high_level OF reciprocal IS
  BEGIN
    PROCESS (clk)
      VARIABLE x,w: DatainT ;
      VARIABLE z  : DataoutT;
    BEGIN
      IF (clk'EVENT AND clk = '1') THEN
        z := 1.0; x := x_in;
        w := x * z;
        WHILE (ABS(w - 1.0) > eps_in/2.0) LOOP
          z := (2.0 - w) * z;
          w := x * z;
        END LOOP;
        z_out <= z;
      END IF;
    END PROCESS;
  END high_level;
```

(b)

Figure 13.10: SYSTEM IN EXAMPLE 13.7: a) EXECUTION GRAPH; b) HIGH-LEVEL SPECIFICATION.

IMPLEMENTATION OF RTL SYSTEMS

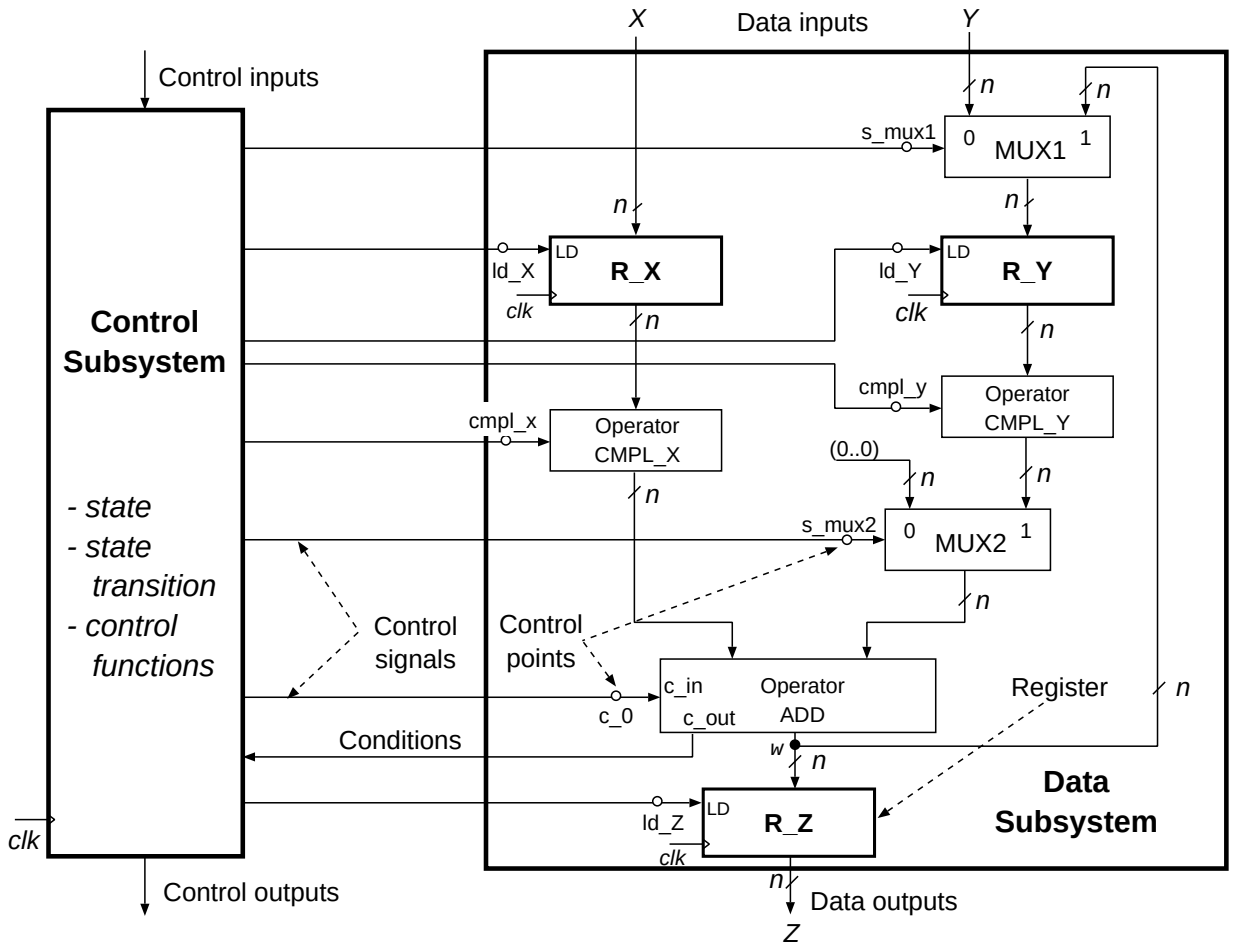


Figure 13.11: RTL SYSTEM.

```
ARCHITECTURE behavioral OF data_sub IS
  SIGNAL R_X,R_Y,R_Z: BIT_VECTOR(n-1 DOWNTO 0);
BEGIN
  PROCESS (clk)
    VARIABLE zero_n: BIT_VECTOR(n-1 DOWNTO 0):= (OTHERS => '0');
    VARIABLE w     : BIT_VECTOR(n-1 DOWNTO 0);
    VARIABLE selec : BIT_VECTOR(  2 DOWNTO 0);
  BEGIN
  selec:= cmpl_x & cmpl_y & smux_2;
  CASE selec IS
    WHEN "000" => w := add(R_X,zero_n,c_0);
    WHEN "001" => w := add(R_X,R_Y,c_0)   ;
    WHEN "010" => w := add(R_X,zero_n,c_0);
    WHEN "011" => w := add(R_X,cmpl(R_Y),c_0)   ;
    WHEN "100" => w := add(cmpl(R_X),zero_n,c_0);
    WHEN "101" => w := add(cmpl(R_X),R_Y,c_0)   ;
    WHEN "110" => w := add(cmpl(R_X),zero_n,c_0);
    WHEN "111" => w := add(cmpl(R_X),cmpl(R_Y),c_0);
  END CASE;
  IF (clk'EVENT AND clk = '1') THEN
    IF (ld_X = '1') THEN R_X <= X; END IF;             -- R_X
    IF (ld_Z = '1') THEN R_Z <= w; END IF;             -- R_Z
    IF (ld_Y = '1') THEN
      IF (smux_1 = '0') THEN R_Y <= Y;                 -- R_Y
      ELSE                   R_Y <= w;
      END IF;
    END IF;
  END IF;
  END PROCESS;
END behavioral;
```

Figure 13.12: Register transfers in Figure 13.11

ANALYSIS OF RTL SYSTEMS

```
ENTITY rtl_system IS
  GENERIC (n: NATURAL:=8);        -- bit-vectors width
  PORT(data_in : IN  BIT_VECTOR ; -- input data
       data_out: OUT BIT_VECTOR ; -- output data
       ctrl_in : IN  BIT_VECTOR ; -- input controls
       ctrl_out: OUT BIT_VECTOR ; -- output controls
       clk     : IN  BIT);
END rtl_system;

ARCHITECTURE general OF rtl_system IS
  TYPE stateT IS (S0,S1,...,Sk);
  SIGNAL state: stateT:= S0;                    -- state register
  SIGNAL reg_A,...: BIT_VECTOR(n-1 DOWNTO 0);   -- data registers

  CONSTANT c1: NATURAL:= ...;     --length of controls bit-vector
  CONSTANT c2: NATURAL:= ...;     --length of conds. bit-vector
  SIGNAL data_ctrls : BIT_VECTOR(c1-1 DOWNTO 0);  -- controls
  SIGNAL data_conds : BIT_VECTOR(c2-1 DOWNTO 0);  -- conds.
BEGIN
  PROCESS(clk)                    -- data subsystem  -----------------+
  BEGIN                           --                                  |
    IF (clk = '1') THEN           --                                  |
      CASE data_ctrls IS          --                                  |
        WHEN ... => ...;          -- register transfer group 0        |
        WHEN ... => ...;          --                                  |
        WHEN ... => ...;          -- register transfer group k        |
        WHEN OTHERS => NULL;      --                                  |
      END CASE;                   --                                  |
    END IF;                       --                                  |
  END PROCESS;                    -- ---------------------------------+
```

ANALYSIS OF RTL SYSTEM (cont.)

```
  PROCESS (clk)                    -- control subsystem, --------------+
  BEGIN                            --   transition function            |
    IF (clk = '1') THEN            --   transitions might depend on    |
      CASE state IS                --   data conditions, described by  |
        WHEN S0 => state <= ...;   --   IF statements in each state    |
        WHEN ....              ;   --                                  |
        WHEN Sk => state <= ...;   --                                  |
      END CASE;                    --                                  |
    END IF;                        --                                  |
  END PROCESS;                     --                                  |
                                   --                                  |
  PROCESS (state,ctrl_in)          -- control subsystem,               |
  BEGIN                            --      output function             |
    CASE state IS                  --                                  |
        WHEN S0 => data_ctrls <= ...; ctrl_out <= ...;                 |
        WHEN ....                  --                                  |
        WHEN Sk => data_ctrls <= ...; ctrl_out <= ...;                 |
    END CASE;                      --                                  |
  END PROCESS;                     -- ----------------------------------+
END general;
```

EXAMPLE 13.9: DATA OPERATORS

$$\text{MULTF}(m, rec) = m \times rec$$

$$\text{SUB2F}(w) = 2 - w$$

$$\text{SUB1F}(w) = 1 - w$$

$$\text{COMP}(b, eps) = \begin{cases} 1 \ \textbf{if} \ \ b > eps \\ 0 \ \textbf{otherwise} \end{cases}$$

EXAMPLE 13.9: RTL SYSTEM

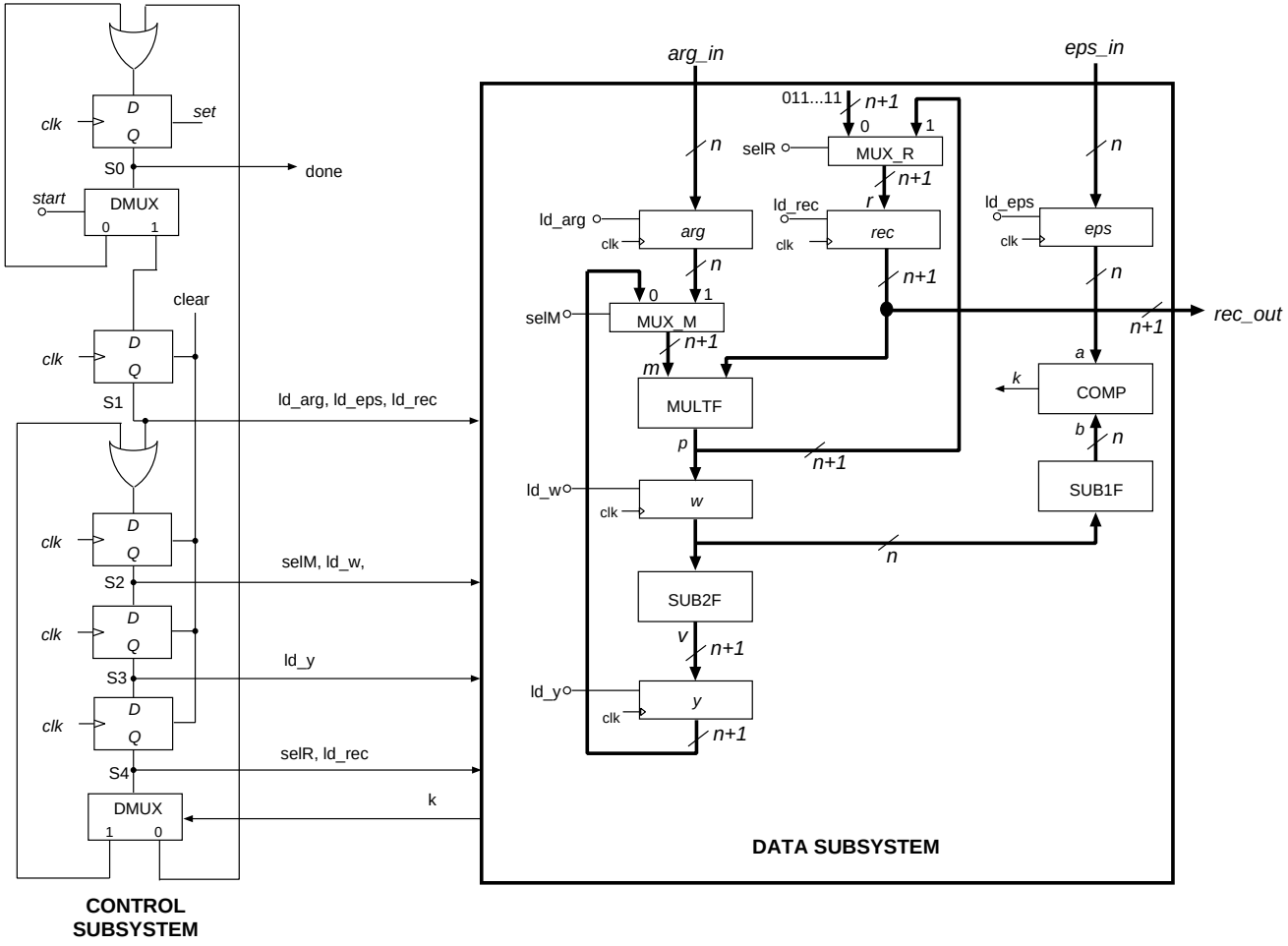


Figure 13.14: RTL SYSTEM FOR Example 13.9.

Example 13.9 (cont.)

```
ENTITY example IS
  GENERIC(n: NATURAL:= 16);  -- bit-vectors length
  PORT(start         : IN  BIT      ;
       arg_in,eps_in: IN  UNSIGNED(n-1 DOWNTO 0);
       rec_out       : OUT UNSIGNED(n DOWNTO 0)  ;
       done          : OUT BIT      ;
       clk           : IN  BIT     );
END example;
```

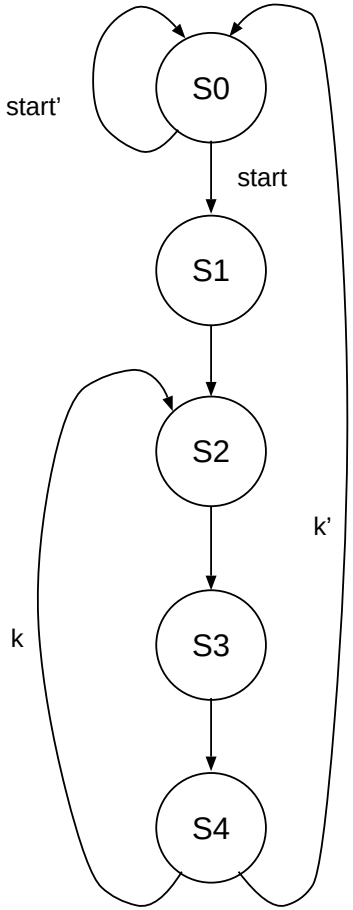
Example 13.9 (cont.)

```
ARCHITECTURE direct OF example IS
 TYPE   stateT is (s0, s1, s2, s3, s4);
  SIGNAL state    : stateT:= s0;
  SIGNAL arg,eps,w: UNSIGNED(n-1 DOWNTO 0);  -- n-bit registers
  SIGNAL rec,y    : UNSIGNED(n DOWNTO 0)  ;  -- n+1 bit register
  SIGNAL k        : BIT                   ;  -- condition
  SIGNAL ld_arg,ld_rec,ld_eps,ld_w,ld_y,selR,selM: BIT;  -- controls
BEGIN
  PROCESS (clk)                              -- data subsystem ---------+
    VARIABLE b: UNSIGNED(n-1 DOWNTO 0);      -- internal data elements  |
    VARIABLE r,m,p,v: UNSIGNED(n DOWNTO 0);  --                         |
  BEGIN                                      -- combinational modules   |
    IF (selR = '0') THEN r:= (OTHERS => '1');r(n):='0';                 |
                    ELSE r:= p; END IF;                          --|
    IF (selM = '0') THEN m:= y; ELSE m:= '0' & arg; END IF;           --|
    p:= multf(m,rec); b:= sub1f(w); v:= sub2f(w); k <= compf(eps,b);  --|
    IF (clk = '1') THEN                      -- register modules        |
      IF (ld_arg = '1') THEN arg <= arg_in; END IF;                   --|
      IF (ld_rec = '1') THEN rec <= r     ; END IF;                   --|
      IF (ld_eps = '1') THEN eps <= eps_in; END IF;                   --|
      IF (ld_w   = '1') THEN w   <= p(n-1 DOWNTO 0); END IF;          --|
      IF (ld_y   = '1') THEN y   <= v     ; END IF;                   --|
    END IF;                                                           --|
    rec_out <= rec;                                                   --|
  END PROCESS;                               -- ------------------------+
```

```
                                        -- control subsystem ------+
PROCESS (clk)                           -- transition function     |
BEGIN                                                              |
  IF (clk = '1') THEN                                              |
    CASE state IS                                                  |
      WHEN s0 => IF (start = '1') THEN state <= s1;                --|
                     ELSE             state <= s0; END IF;         --|
      WHEN s1 => state  <= s2;                                     --|
      WHEN s2 => state  <= s3;                                     --|
      WHEN s3 => state  <= s4;                                     --|
      WHEN s4 => IF (k = '1') THEN state <= s2;                      |
                     ELSE         state <= s0; END IF;             --|
    END CASE;                                                      --|
  END IF;                                                          --|
END PROCESS;                            -- -----------------------+
```

```
  PROCESS (state)                              -- output function       --|
  BEGIN                                                                 --|
    CASE state IS                                                       --|
      WHEN s0 => done   <= '1'; ld_rec <= '0';                          --|
      WHEN s1 => ld_arg <= '1'; ld_eps <= '1';                          --|
                 selR   <= '0'; ld_rec <= '1'; done   <= '0';           --|
      WHEN s2 => selM   <= '1'; ld_w   <= '1';                          --|
                 ld_arg <= '0'; ld_eps <= '0'; ld_rec <= '0';           --|
      WHEN s3 => ld_y   <= '1'; ld_w   <= '0';                          --|
      WHEN s4 => selM   <= '0'; selR   <= '1'; ld_rec <= '1';           --|
      WHEN others  => NULL;                                             --|
    END CASE;                                                           --|
  END PROCESS;                                 -- ----------------------+
END direct;
```

STATE DIAGRAM



Outputs (Control Signals)

	done	selM	selR	ld_arg	ld_eps	ld_rec	ld_y	ld_w
S0	1	0	0	0	0	0	0	0
S1	0	0	0	1	1	1	0	0
S2	0	1	0	0	0	0	0	1
S3	0	0	0	0	0	0	1	0
S4	0	0	1	0	0	1	0	0

Figure 13.17: STATE DIAGRAM OF SYSTEM IN EXAMPLE 13.9.

Example 13.9 (cont.)

```
S0: done <= '1';
S1: done <= '0'; arg <= arg_in; rec <= "01...1";
    eps <= eps_in;
S2: w    <= MULTF(arg,rec);
S3: y    <= SUB2F(w)
S4: rec  <= MULTF(y,rec);
    k    <= COMP(eps,SUB1F(w));
```

Example 13.9 (cont.)

```
ARCHITECTURE behavioral OF example IS
  TYPE   stateT IS (S0,S1,S2,S3,S4);
  SIGNAL state     : stateT:= S0;
  SIGNAL arg,eps,w : UNSIGNED(n-1 DOWNTO 0);
  SIGNAL rec,y     : UNSIGNED(n DOWNTO 0)  ;
BEGIN
  PROCESS (clk)
  BEGIN
    IF (clk'EVENT AND clk = '1') THEN
      CASE state IS
        WHEN S0 => done <= '1';
                   IF (start = '1') THEN state <= S1;
                   ELSE                  state <= S0; END IF;
        WHEN S1 => arg   <= arg_in;
                   rec   <= (OTHERS => '1'); rec(n) <= '0';
                   eps   <= eps_in; done <= '0';
                   state <= S2;
        WHEN S2 => w     <= arg * rec;
                   state <= S3   ;
        WHEN S3 => y     <= 2 - w;
                   state <= S4 ;
        WHEN S4 => rec   <= y * rec;
                   IF (1-w > eps) THEN state <= S2;
                   ELSE                state <= S0; END IF;
      END CASE; END IF; END PROCESS; END behavioral;
```

FUNCTION OF THE SYSTEM

```
WHILE (1-arg * rec >  eps) LOOP
  rec := (2-arg * rec) * rec;
END WHILE;
```

Example 13.9: POSSIBLE REGISTER TRANSFERS

```
IF (ld_arg = '1') THEN  arg <= arg_in;      END IF;
IF (ld_eps = '1') THEN  eps <= eps_in;      END IF;
IF (ld_w   = '1') AND (selM = '0') THEN w <= MULTF(y,rec);
   END IF;
IF (ld_w   = '1') AND (selM = '1') THEN w <= MULTF(arg,rec);
   END IF;
IF (ld_y   = '1') THEN  y   <= SUB2F(w);  END IF;
IF (ld_rec = '1') AND (selR = '0') THEN rec <= '1' ; END IF;
IF (ld_rec = '1') AND (selR = '1') AND (selM = '0') THEN
                                  rec <= MULTF(y,rec); END IF;
IF (ld_rec = '1') AND (selR = '1') AND (selM = '1') THEN
                                  rec <= MULTF(y,arg); END IF;
```

DESIGN OF RTL SYSTEMS

DESIGN OF DATA SUBSYSTEM:

1. DETERMINE THE OPERATORS (functional units)

   Two operations can be assigned to the same functional unit if they form part of different groups

2. DETERMINE THE REGISTERS REQUIRED TO STORE OPERANDS, RESULTS, AND INTERMEDIATE VARIABLES

   Two variables can be assigned to the same register if they are active in disjoint time intervals

3. CONNECT THE COMPONENTS BY DATAPATHS (wires and multiplexers) as required by the transfers in the sequence

4. DETERMINE THE CONTROL SIGNALS AND CONDITIONS required by the sequence

5. DESCRIBE THE STRUCTURE OF THE DATA SECTION by a logic diagram, a net list, or a μVHDL structural description

GENERALIZED BEHAVIORAL DESCRIPTION

```
ARCHITECTURE generalized OF
          data_subsystem IS
BEGIN
  PROCESS (clk)
    SIGNAL reg_A,reg_B: BIT_VECTOR;
  BEGIN
    IF (clk = '1') THEN
      IF (ctl0 = '1') THEN ... END IF;
      IF (ctl1 = '1') THEN ... END IF;
          .....
      IF (ctlj = '1') THEN ... END IF;
    END IF;
  END PROCESS;
END generalized;
```

(a)

```
ARCHITECTURE generalized OF
          control_subsystem IS
BEGIN
  PROCESS (clk)
    TYPE stateT is (s0, s1, sk);
    SIGNAL state: stateT:= s0;
  BEGIN
    IF (clk = '1') THEN
      CASE state IS
        WHEN s0 => ....;
        WHEN s1 => ....;
        WHEN sk => ....;
      END CASE;
    END IF;
  END PROCESS;
END generalized;
```

(b)

INTERFACE BETWEEN DATA AND CONTROL SUBSYSTEMS

```
ENTITY group_seq_system IS
  PORT    (data_in : IN  BIT_VECTOR; -- input data
           data_out: OUT BIT_VECTOR; -- output data
           ctrl_in : IN  BIT_VECTOR; -- input conditions
           ctrl_out: OUT BIT_VECTOR; -- output conditions
           clk     : IN  BIT      );
END group_seq_system;

ARCHITECTURE generalized OF group_seq_system IS
  SIGNAL controls : BIT_VECTOR; -- control signals
                                -- to data subsystem
  SIGNAL conds    : BIT_VECTOR; -- condition signals
                                -- from data subsystem
BEGIN
  U1: ENTITY data_subsystem
             PORT MAP (data_in,data_out,controls,conds,clk);
  U2: ENTITY control_subsystem
             PORT MAP (ctrl_in,ctrl_out,conds,controls,clk);
END generalized;
```

DESIGN OF CONTROL SUBSYSTEM

1. DETERMINE THE REGISTER-TRANSFER SEQUENCE

2. ASSIGN ONE STATE TO EACH `RT-group`

3. DETERMINE STATE-TRANSITION AND OUTPUT FUNCTIONS

4. IMPLEMENT THE CORRESPONDING SEQUENTIAL SYSTEM

DESIGN EXAMPLE: MULTIPLIER

INPUTS: $x, y \in \{0, 1, \ldots, 2^n - 1\}$
OUTPUT: $z \in \{0, 1, \ldots, 2^{2n} - 2^{n+1} + 1\}$
FUNCTION: $z = x \times y$

- RECURRENCE:

$$z[i+1] = \left(\frac{1}{2}\right)(z[i] + (x \times 2^n) \times y_i)$$

$z[0] = 0 \; z = z[n]$

OPERATIONS IN ONE ITERATION OF THE RECURRENCE:

- Multiplication of x by 2^n.
- Multiplication of $(x \times 2^n)$ by y_i, the i-th bit of $\underline{y}$ (value 0 or $x \times 2^n$)
- Addition of $z[i]$ and $(x \times 2^n) \times y_i$.
- Multiplication of the sum by $\frac{1}{2}$ (one-bit right shift)

EXECUTION GRAPH OF MULTiPlIER

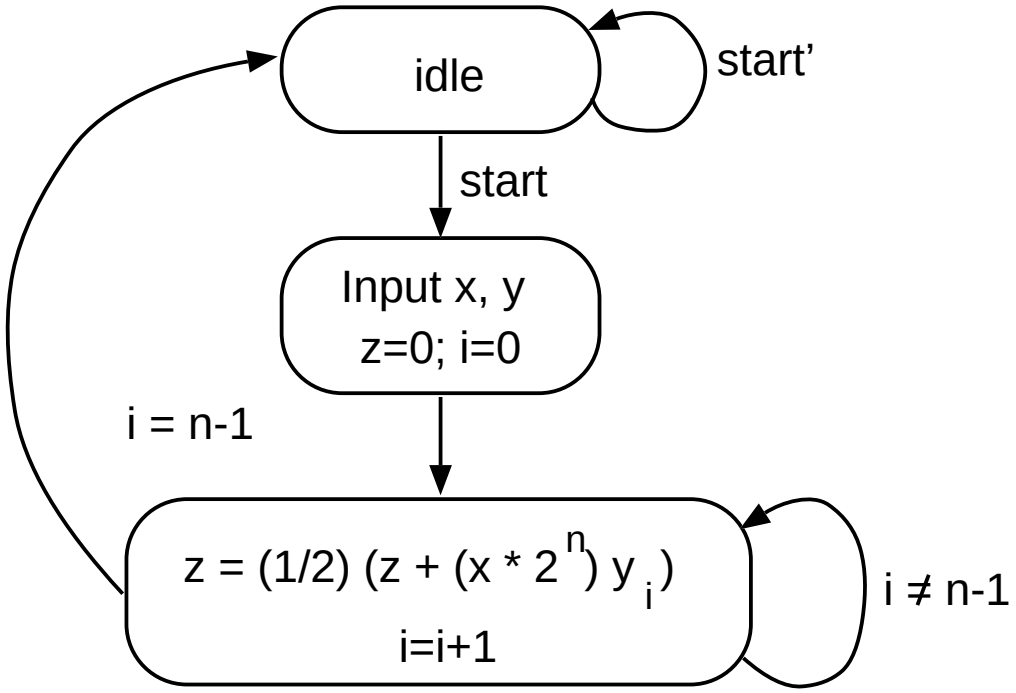


Figure 13.22: EXECUTION GRAPH OF SERIAL-PARALLEL MULTIPLIER.

Example 13.10

$$x = 188 \ , \ \underline{x} = 10111100$$
$$y = 203 \ , \ \underline{y} = 11001011$$
$$z = z[8] = 1001010100010100 = 38164$$

$z[0]$	$=$	00000000 00000000
$(x \times 2^8) \times y_0$	$=$	10111100 00000000
$z[0] + (x \times 2^8) \times y_0$	$=$	010111100 00000000
$z[1]$	$=$	01011110 00000000
$(x \times 2^8) \times y_1$	$=$	10111100 00000000
$z[1] + (x \times 2^8) \times y_1$	$=$	100011010 00000000
$z[2]$	$=$	10001101 00000000
$(x \times 2^8) \times y_2$	$=$	00000000 00000000
$z[2] + (x \times 2^8) \times y_2$	$=$	010001101 00000000
$z[3]$	$=$	01000110 10000000
$(x \times 2^8) \times y_3$	$=$	10111100 00000000
$z[3] + (x \times 2^8) \times y_3$	$=$	100000010 10000000
$z[4]$	$=$	10000001 01000000
$(x \times 2^8) \times y_4$	$=$	00000000 00000000
$z[4] + (x \times 2^8) \times y_4$	$=$	010000001 01000000
$z[5]$	$=$	01000000 10100000
$(x \times 2^8) \times y_5$	$=$	00000000 00000000
$z[5] + (x \times 2^8) \times y_5$	$=$	001000000 10100000
$z[6]$	$=$	00100000 01010000
$(x \times 2^8) \times y_6$	$=$	10111100 00000000
$z[6] + (x \times 2^8) \times y_6$	$=$	011011100 01010000
$z[7]$	$=$	01101110 00101000
$(x \times 2^8) \times y_7$	$=$	10111100 00000000
$z[7] + (x \times 2^8) \times y_7$	$=$	100101010 00101000
$z[8]$	$=$	10010101 00010100

$$z = z[8] = 1001010100010100 = 38164$$

VHDL SPECIFICATION

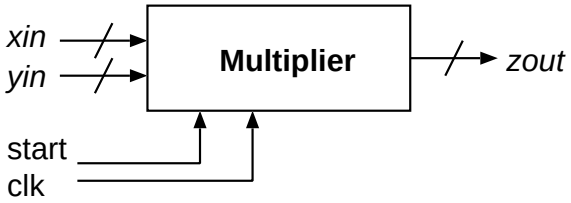


Figure 13.23: BLOCK DIAGRAM

```
ENTITY multiplier IS
  GENERIC(n : NATURAL:= 16);  -- number of bits in operands
  PORT   (start  : IN  BIT ;
          xin,yin: IN  UNSIGNED(n-1 DOWNTO 0);
          clk    : IN  BIT ;
          zout   : OUT UNSIGNED(2*n-1 DOWNTO 0);
          done   : OUT BIT);
END multiplier;
```

```
ARCHITECTURE behavioral OF multiplier IS
  TYPE   stateT  IS (idle,setup,active)  ;
  SIGNAL state : stateT := idle          ;
  SIGNAL x,y   : BIT_VECTOR(n-1 DOWNTO 0); -- operand registers
  SIGNAL z     : BIT_VECTOR(2*n-1 DOWNTO 0);
  SIGNAL count : NATURAL RANGE 0 TO n-1  ;
BEGIN
  PROCESS (clk)
    VARIABLE zero_2n : UNSIGNED(2*n-1 DOWNTO 0); -- constant zero
    VARIABLE scale   : UNSIGNED(n-1 DOWNTO 0)  ; -- aligning vector
    VARIABLE add_out : UNSIGNED(2*n DOWNTO 0)  ;
  BEGIN
    zero_2n:= (OTHERS => '0');
    scale  := (OTHERS => '0');
    IF (clk'EVENT AND clk = '1') THEN
      CASE state IS
        WHEN idle   => done <= '1';
                       IF (start = '1') THEN state <= setup;
                       ELSE                  state <= idle;
                       END IF;
```

```
      WHEN setup  => x <= xin; y <= yin; z <= zero_2n;
                     count <= 0;
                     zout <= zero_2n; done <= '0';
                     state <= active;
      WHEN active => IF (y(count)) = '0') THEN
                        add_out :='0' & z;
                     ELSE
                        add_out := ('0' & z) + ('0' & x & scale);
                     END IF
                     z    <= add_out(2*n DOWNTO 1);
                     zout <= add_out(2*n DOWNTO 1);
                     IF (count /= (n-1)) THEN
                       state <= active;
                       count <= count+1;
                     ELSE
                       state <= idle;
                       done  <= '1' ;
                     END IF;
    END CASE;
  END IF;
END PROCESS;
END behavioral;
```

IMPLEMENTATION - DATA SUBSYSTEM

- n-bit register x to store x and a $2n$-bit register z to store z
- n-bit register y to store y
- A module to generate $x \times 2^n$
- A module to generate $(x \times 2^n) \times y_i$
- A module to perform addition

$(x \times 2^8) \times y_i$	x	x	x	x	x	x	x	x	0	0	0	0	0	0	0	0
$z[i]$	z	z	z	z	z	z	z	z	z	z	z	z	z	z	z	z

- A module to perform multiplication by 1/2 and load the result in z

DATA SUBSYSTEM: CONTROL SIGNALS

`ldX,ldY,ldZ`	for loading `x`,`y`,`z`, respectively
`shY`	for right-shifting `y`
`clrZ`	for loading 0 into `z`

DATA SUBSYSTEM: BEHAVIORAL DESCRIPTION

```
ENTITY multdata_bhv IS
  GENERIC(n : NATURAL :=  16);                        -- number of bits
  PORT   (xin,yin     : IN  UNSIGNED(n-1 DOWNTO 0); -- data inputs
          ldX,ldY,ldZ : IN  BIT;                    -- control signals
          shY,clrZ    : IN  BIT;                    -- control signals
          zout        : OUT UNSIGNED(2*n-1 DOWNTO 0);   -- data output
          clk         : IN BIT);
END multdata_bhv;
```

```
ARCHITECTURE behavioral OF multdata_bhv IS
  SIGNAL x,y : UNSIGNED(n-1 DOWNTO 0)  ;        -- registers
  SIGNAL z   : UNSIGNED(2*n-1 DOWNTO 0);
BEGIN
  PROCESS(clk)
    VARIABLE zero_2n : UNSIGNED(2*n-1 DOWNTO 0); -- vector constant 0
    VARIABLE scale   : UNSIGNED(n-1 DOWNTO 0);
    VARIABLE add_out : UNSIGNED(2*n DOWNTO 0);
  BEGIN
     zero_2n:= (OTHERS =>'0');
     scale  := (OTHERS =>'0');
     IF (clk'EVENT AND clk = '1') THEN
       IF (ldX = '1') THEN x <= xin;  END IF;
       IF (ldY = '1') THEN y <= yin;  END IF;
       IF (y(0) ='0') THEN
              add_out := '0' & z;
       ELSE
              add_out := ('0' & z) + ('0' & x & scale);
       END IF;
       IF (ldZ = '1') THEN
            z    <= add_out(2*n DOWNTO 1);
            zout <= add_out(2*n DOWNTO 1);
       END IF;
       IF (clrZ= '1') THEN z <= zero_2n;  END IF;
       IF (shY = '1') THEN y <= '0' & y(n-1 DOWNTO 1);  END IF;
    END IF; END PROCESS; END behavioral;
```

DATA SUBSYSTEM: STRUCTURAL DESCRIPTION

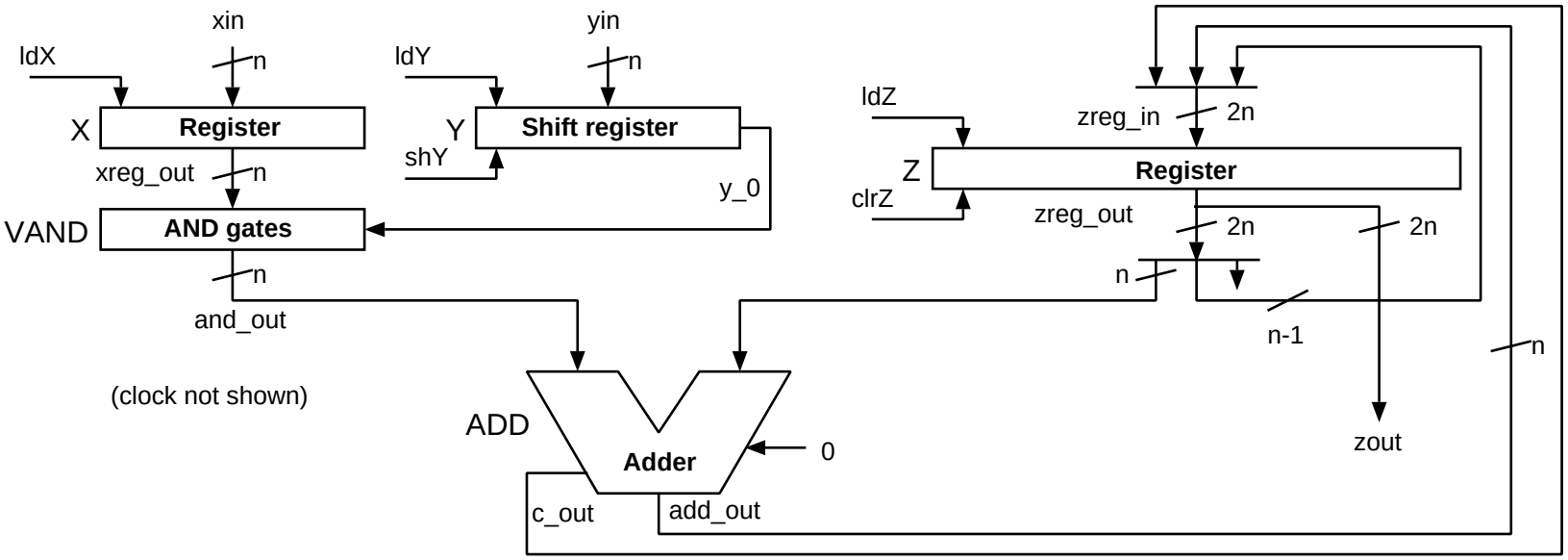


Figure 13.25: DATA SUBSYSTEM

```
ENTITY multdata_str IS
  GENERIC(n : NATURAL :=  16);                          -- number of bits
  PORT   (xin,yin     : IN  UNSIGNED(n-1 DOWNTO 0);     -- data inputs
          ldX,ldY,ldZ : IN  BIT;                        -- control signals
          shY,clrZ    : IN  BIT;                        -- control signals
          zout        : OUT UNSIGNED(2*n-1 DOWNTO 0);   -- data output
          clk         : IN BIT);
END multdata_str;

ARCHITECTURE structural OF multdata_str IS
 SIGNAL add_out, xreg_out, and_out: BIT_VECTOR(n-1 DOWNTO 0);
 SIGNAL c_out, y_0                : BIT;
 SIGNAL zreg_out                  : BIT_VECTOR(2*n-1 DOWNTO 0);
 SIGNAL zreg_in                   : BIT_VECTOR(2*n-1 DOWNTO 0);
 SIGNAL clr                       : BIT;

BEGIN
       zreg_in <= c_out & add_out & zreg_out(n-1 DOWNTO 1);
  X : ENTITY reg       PORT MAP (xin,ldX,xreg_out,clk);
  Y : ENTITY shiftreg  PORT MAP (yin,ldY,shY,y_0,clk);
  Z : ENTITY regclr    PORT MAP (zreg_in,ldZ,clrZ,zreg_out,clk);
VAND: ENTITY vectorand PORT MAP (xreg_out,y_0,and_out);
 ADD: ENTITY adder_pos PORT MAP (zreg_out(2*n-1 DOWNTO n),
                                 and_out,'0',add_out,c_out);
END structural;
```

STRUCTURAL DESCRIPTION OF MULTIPLIER

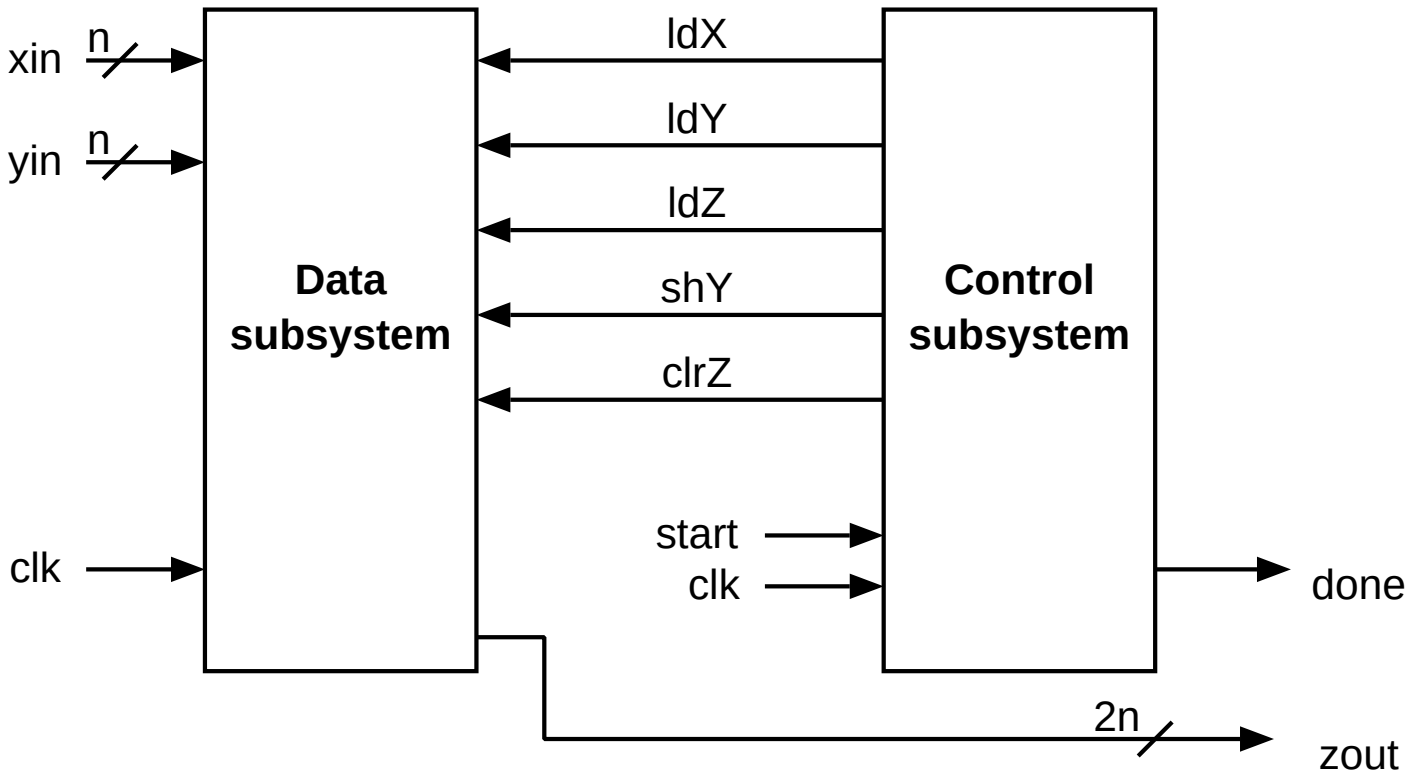


Figure 13.26: MULTIPLIER SCHEME

```
ENTITY multiplier IS
  GENERIC(n : NATURAL:= 16);  -- number of bits in operands
  PORT   (start  : IN  BIT ;
          xin,yin: IN  UNSIGNED(n-1 DOWNTO 0);
          clk    : IN  BIT ;
          zout   : OUT UNSIGNED(2*n-1 DOWNTO 0);
          done   : OUT bit);
END multiplier;
```

```
ARCHITECTURE structural OF multiplier IS
  SIGNAL ldX,ldY,ldZ,clrZ,shY: BIT;
BEGIN
  U1: ENTITY multdata_bhv
            PORT MAP (xin,yin,ldX,ldY,ldZ,shY,clrZ,zout,clk);
  U2: ENTITY multctrl
            PORT MAP (start,ldX,ldY,ldZ,shY,clrZ,done,clk);
END structural;
```

CONTROL SUBSYSTEM

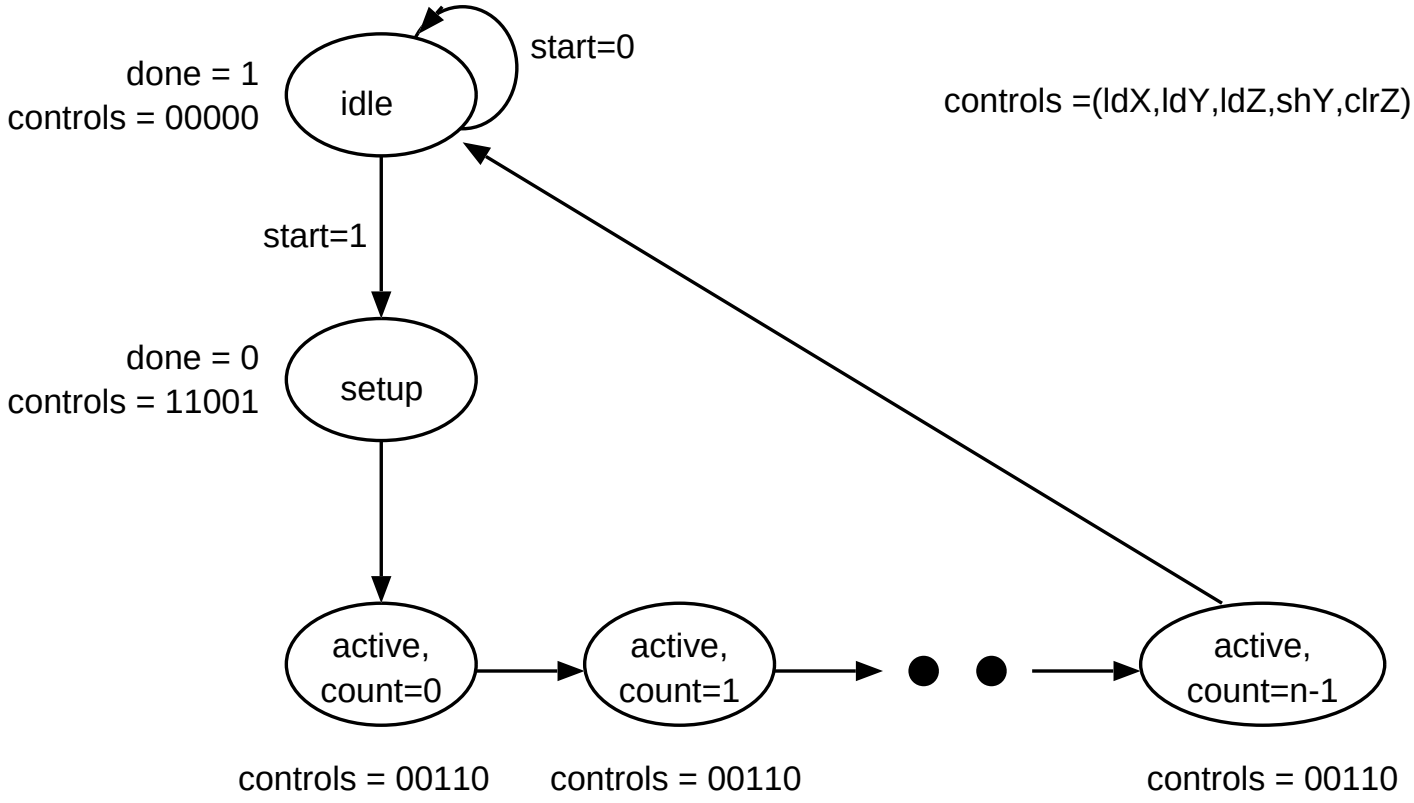


Figure 13.27: STATE DIAGRAM FOR MULTIPLIER CONTROL SUBSYSTEM.

CONTROL SUBSYSTEM: BEHAVIORAL DESCRIPTION

```
ENTITY multctrl IS
  GENERIC(n: NATURAL :=  16);   -- number of bits
  PORT   (start      : IN  BIT; -- control input
          ldX,ldY,ldZ: OUT BIT; -- control signals
          shY, clrZ  : OUT BIT; -- control signals
          done       : OUT BIT; -- control output
          clk        : IN  BIT);
END multctrl;
```

```
ARCHITECTURE behavioral OF multctrl IS
  TYPE   stateT IS (idle,setup,active);
  SIGNAL state   : stateT:= idle;
  SIGNAL count   : NATURAL RANGE 0 TO n-1;
BEGIN
  PROCESS (clk)                        -- transition function
  BEGIN
    IF (clk'EVENT AND clk = '1') THEN
      CASE state IS
        WHEN idle   => IF (start = '1') THEN state <= setup;
                       ELSE                 state <= idle ;
                       END IF;
        WHEN setup  => state <= active; count <= 0;
        WHEN active => IF (count = (n-1)) THEN
                         count <= 0; state <= idle  ;
                       ELSE
                         count <= count+1; state <= active;
                       END IF;
      END CASE;
    END IF;
  END PROCESS;
```

```
  PROCESS (state,count)                -- output function
    VARIABLE controls: BitVector(5 DOWNTO 0);
                          -- code = (ldX,ldY,ldZ,shY,clrZ)
  BEGIN
    CASE state IS
      WHEN idle   =>  controls := "100000";
      WHEN setup  =>  controls := "011001";
      WHEN active =>  controls := "000110";
    END CASE;
    done <= controls(5);
    ldX <= controls(4); ldY <= controls(3); ldZ <= controls(2);
    shY <= controls(1); clrZ<= controls(0);
  END PROCESS;
END behavioral;
```

CYCLE TIME AND MULTIPLICATION TIME

- CYCLE TIME

  t_r : the delay of the registers to produce stable outputs. This includes the setup delay and the propagation delay.

  t_{buf} : the delay of the buffer required by the load on y_0.

  t_{and} : the delay of the AND gates.

  t_{add} : the delay of the adder.

- EXECUTION TIME: $n + 2$ cycles:
  one cycle in the idle state,
  one cycle in the setup state,
  n cycles for the iterations.