Chapter 2
RTL Design with VHDL

RTL Design with VHDL

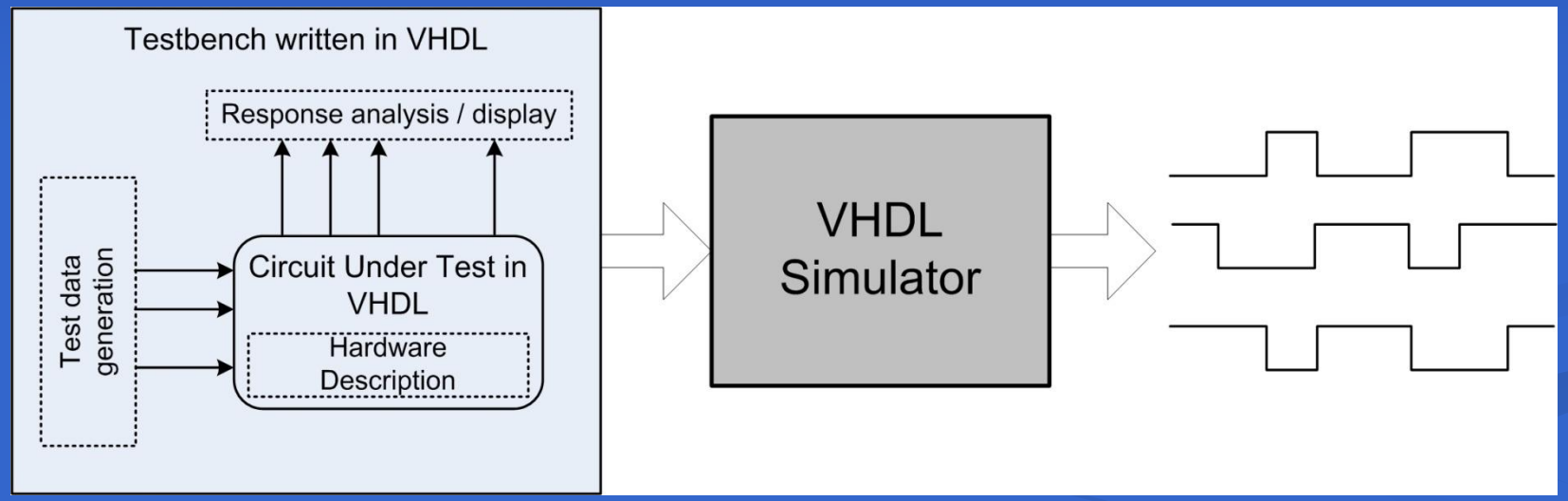
2.1 Basic Structures of VHDL
- 2.1.1 Entities and Architectures
- 2.1.2 Entity-Architecture Outline
- 2.1.3 Entity Ports
- 2.1.4 Signals and Variables
- 2.1.5 Logic Value System
- 2.1.6 Resolutions

2.2 Combinational Circuits
- 2.2.1 Gate Level Combinational Circuits
- 2.2.2 Gate Level Synthesis
- 2.2.3 Descriptions by Use of Equations
- 2.2.4 Instantiating Other Modules
- 2.2.5 Synthesis of Assignment Statements
- 2.2.6 Descriptions with Sequential Flow
- 2.2.7 Combinational Rules

RTL Design with VHDL

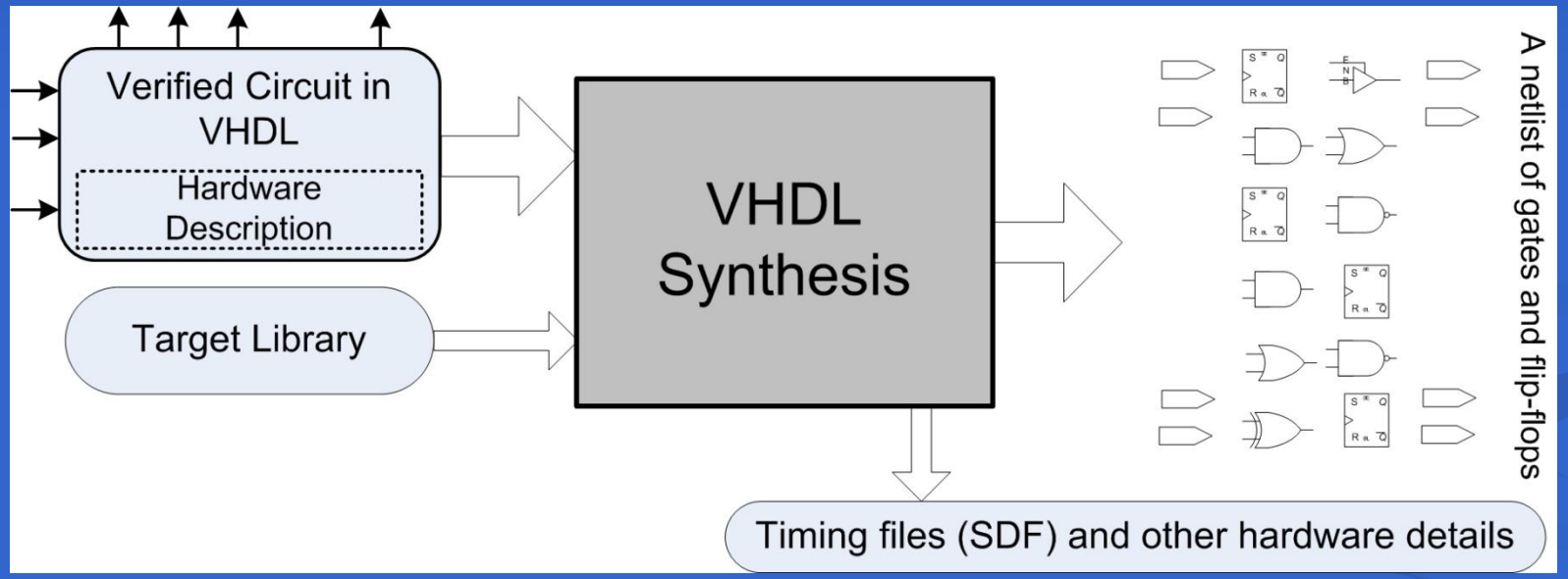
RTL Design with VHDL

Basic Structures of VHDL



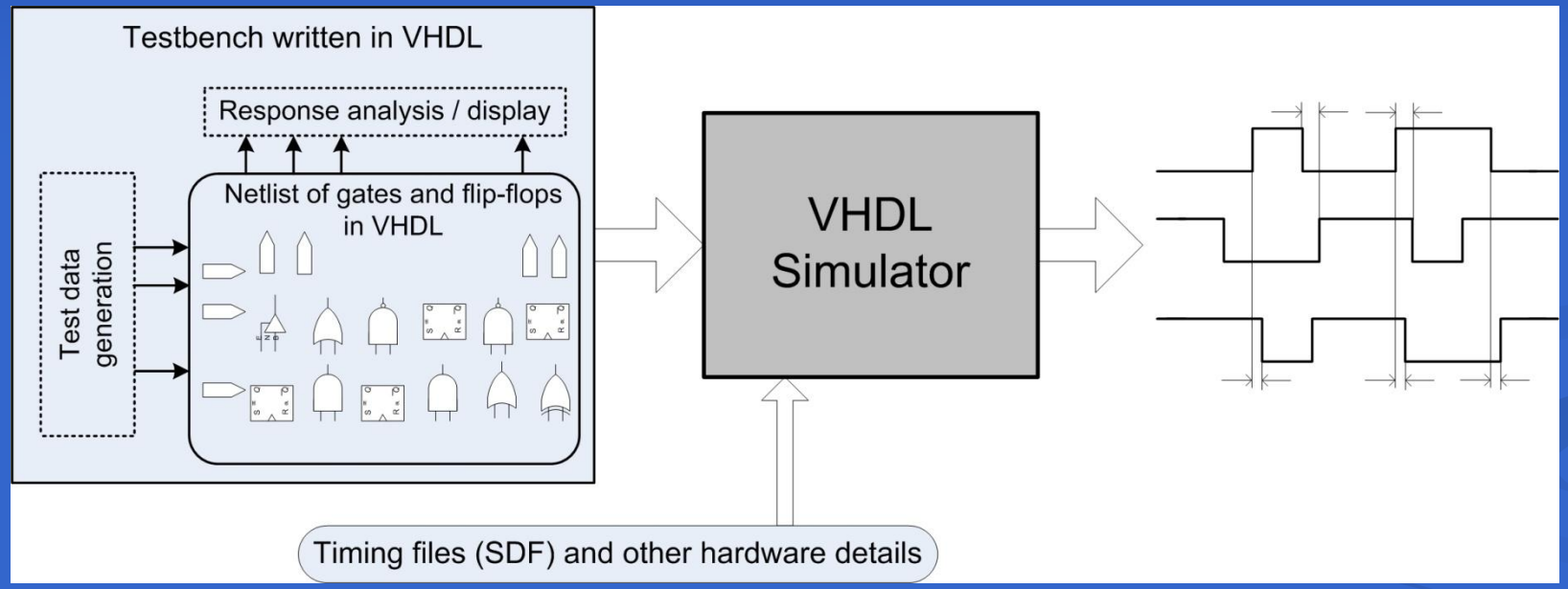
- **Simulation in VHDL**

Basic Structures of VHDL



- **Synthesis of a VHDL Design**

Basic Structures of VHDL

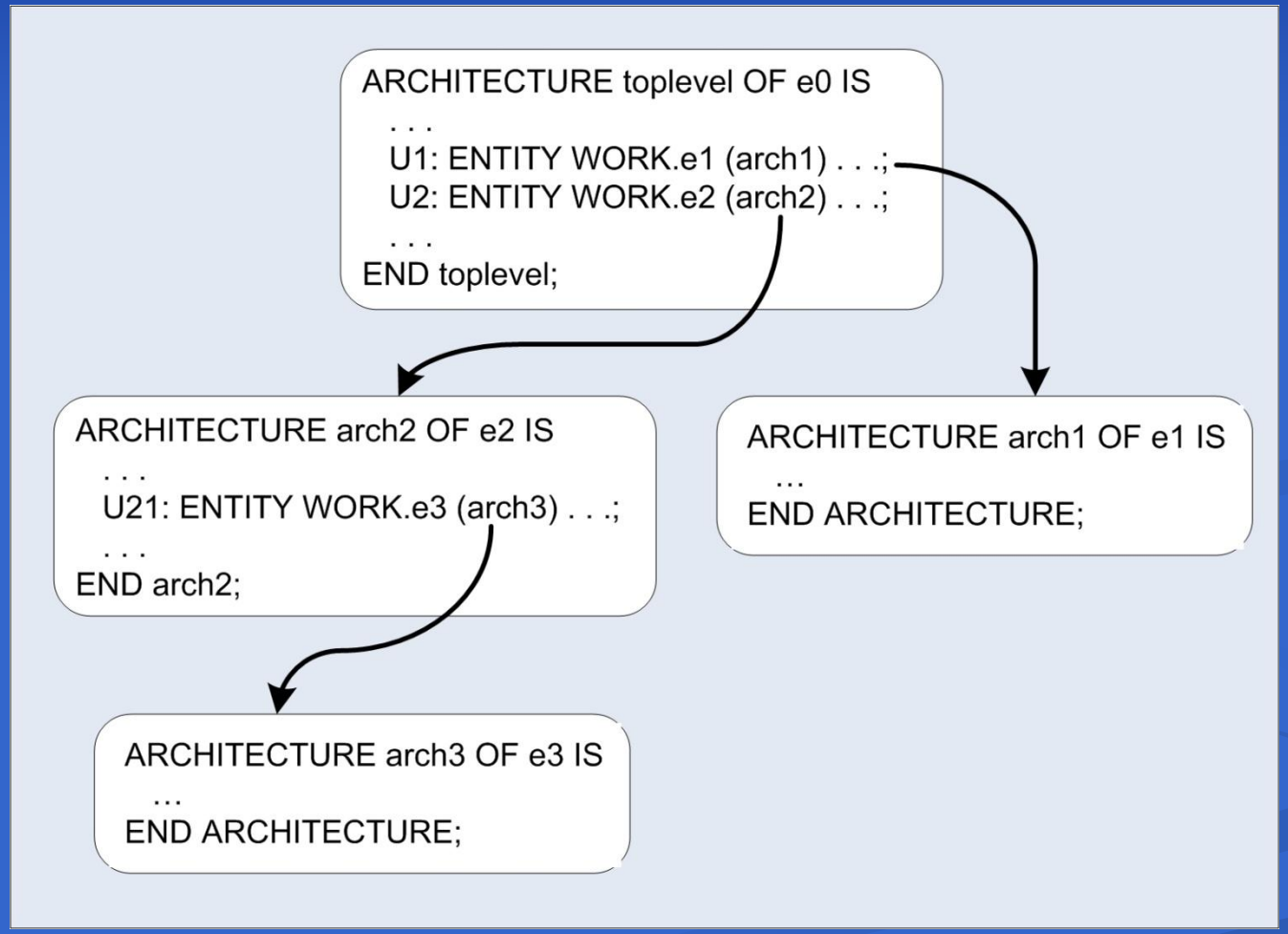


- **Post-synthesis Simulation in VHDL**

Entities and Architectures

```
ENTITY entity_name IS
   input and output ports
END ENTITY entity_name;
ARCHITECTURE identifier OF entity_name IS
   declarative part
BEGIN
   statement part
END ARCHITECTURE identifier;
```

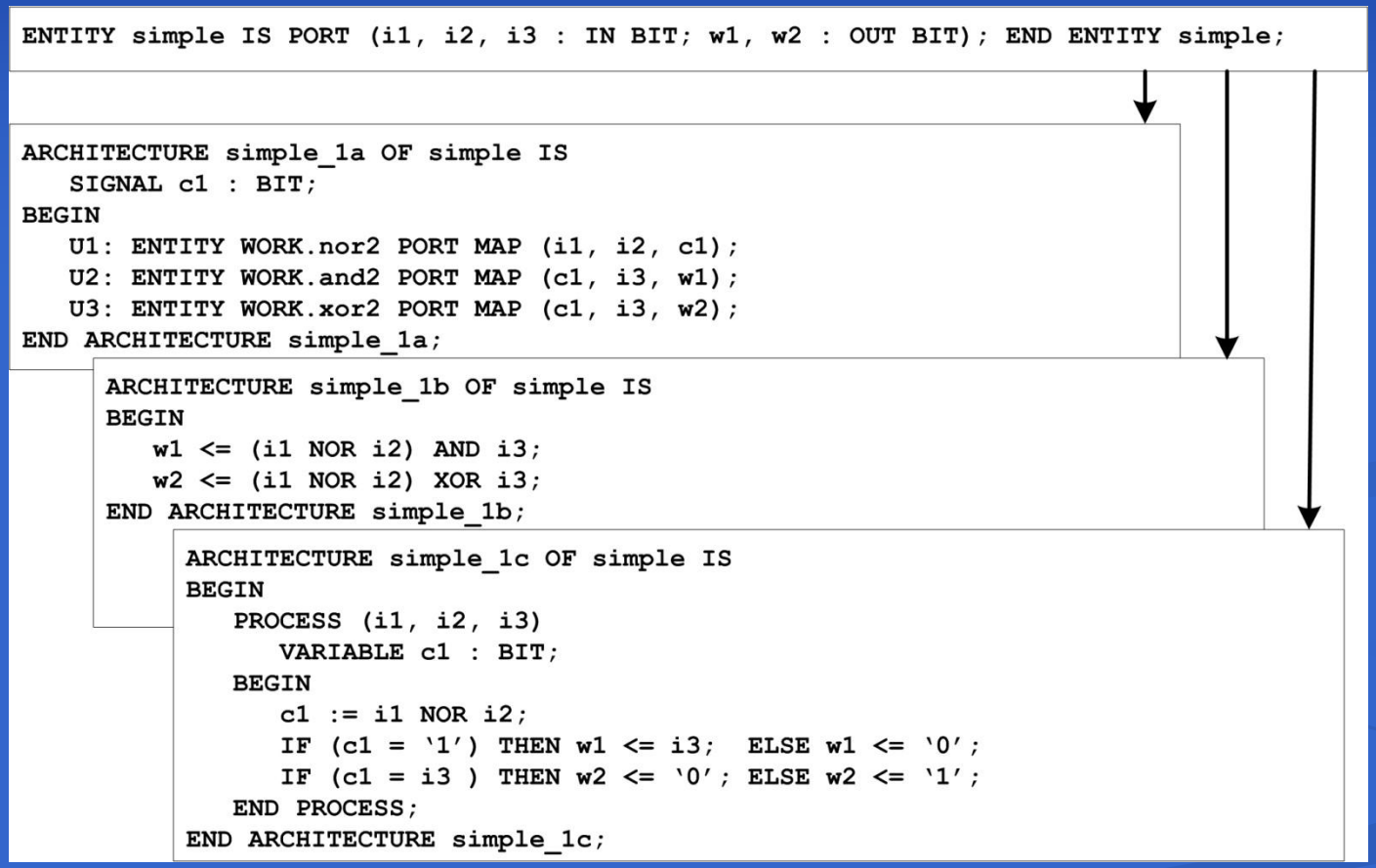
Entities and Architectures



Entity-Architecture Outline

```
ENTITY entity1 IS PORT (i1, i2 : IN BIT; w1 : OUT
BIT);
END ENTITY entity1;
ARCHITECTURE simple1 OF entity1 IS
   SIGNAL s1 : BIT;
BEGIN
   statement1;
   statement2;
   statement3;
END ARCHITECTURE simple1;
```

Entity-Architecture Outline



```
ENTITY simple IS PORT (i1, i2, i3 : IN BIT; w1, w2 : OUT BIT); END ENTITY simple;

ARCHITECTURE simple_1a OF simple IS
   SIGNAL c1 : BIT;
BEGIN
   U1: ENTITY WORK.nor2 PORT MAP (i1, i2, c1);
   U2: ENTITY WORK.and2 PORT MAP (c1, i3, w1);
   U3: ENTITY WORK.xor2 PORT MAP (c1, i3, w2);
END ARCHITECTURE simple_1a;

ARCHITECTURE simple_1b OF simple IS
BEGIN
   w1 <= (i1 NOR i2) AND i3;
   w2 <= (i1 NOR i2) XOR i3;
END ARCHITECTURE simple_1b;

ARCHITECTURE simple_1c OF simple IS
BEGIN
   PROCESS (i1, i2, i3)
      VARIABLE c1 : BIT;
   BEGIN
      c1 := i1 NOR i2;
      IF (c1 = '1') THEN w1 <= i3;  ELSE w1 <= '0';
      IF (c1 = i3 ) THEN w2 <= '0'; ELSE w2 <= '1';
   END PROCESS;
END ARCHITECTURE simple_1c;
```

- Architecture Definition Alternatives

Entity Ports

```
ENTITY aCircuit IS
   PORT (a, b : IN BIT;
         c : INOUT BIT;
         av, bv : IN BIT_VECTOR (7 DOWNTO 0);
         cv : INOUT BIT_VECTOR (7 DOWNTO 0);
         w : OUT BIT;
         wv : OUT BIT_VECTOR (7 DOWNTO 0));
END ENTITY aCircuit;
```

Signals and Variables

```
ARCHITECTURE two_processes OF aCircuit IS
   SIGNAL d : BIT;
   SIGNAL dv : BIT_VECTOR (7 DOWNTO 0);
BEGIN
   p1: PROCESS (a, b, cv)
      VARIABLE e : BIT;
      VARIABLE ev : BIT_VECTOR (7 DOWNTO 0);
   BEGIN
      -- Can see all of aCircuit, plus d, dv, e, and ev.
      . . .
   END PROCESS;
   p2: PROCESS (av, bv, c)
      VARIABLE f : BIT;
      VARIABLE fv : BIT_VECTOR (7 DOWNTO 0);
   BEGIN
      -- Can see all of aCircuit, plus d, dv, f, and fv.
      . . .
   END PROCESS;
END ARCHITECTURE two_processes;
```

- Signal and Variable Declaration

Data Part

```
ARCHITECTURE four_assignments OF aCircuit IS
   SIGNAL d : BIT;
   SIGNAL iv, jv, kv : BIT_VECTOR (7 DOWNTO 0);
BEGIN
   iv <= av AND cv;
   jv <= bv AND cv;
   kv <= av NOR bv;
   wv <= iv XOR jv WHEN c = '1' ELSE iv NAND kv;
END ARCHITECTURE four_assignments;
```

- Using Signals

Data Part

```
ARCHITECTURE mixed_processes_assignments OF aCircuit IS
   SIGNAL d : BIT;
   SIGNAL dv : BIT_VECTOR (7 DOWNTO 0);
BEGIN
   p1: PROCESS (a, b, cv)
      VARIABLE e : BIT;
      VARIABLE ev : BIT_VECTOR (7 DOWNTO 0);
   BEGIN
      IF (a = b) THEN ev := av; ELSE ev := bv;
      IF (a = '1') THEN wv <= av; ELSE wv <= "1000111";
      d <= e;
   END PROCESS;

   dv <= av XOR bv;
   w <= d AND a;
END ARCHITECTURE mixed_processes_assignments;
```

- Using Signals and Variables

Data Part

```
ARCHITECTURE indexing_slicing OF aCircuit IS
   SIGNAL d : BIT;
   SIGNAL dv : BIT_VECTOR (7 DOWNTO 0);
BEGIN
   wv (3 DOWNTO 0) <= av (7 DOWNTO 4) AND
                                        cv (7 DOWNTO 4);
   w <= cv (4);
   cv (7) <= av (0);
END ARCHITECTURE indexing_slicing;
```

- Using Indexing and Slicing

Logic Value System

Value	Representing
'U'	Uninitialized
'X'	Forcing Unknown
'0'	Forcing 0
'1'	Forcing 1
'Z'	High Impedance
'W'	Weak Unknown
'L'	Weak 0
'H'	Weak 1
'-'	Don't care

Resolutions

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
ENTIRY selector IS
   PORT (av, bv: IN std_logic_vector (7 DOWNTO 0),
         as, bs: IN std_logic;
         yv: OUT std_logic_vector (7 DOWNTO 0));
END ENTITY selector;
ARCHITECTURE multiple_drivers OF selector IS
BEGIN
   yv <= av WHEN as = '1' ELSE "ZZZZZZZZ";
   yv <= bv WHEN bs = '1' ELSE "ZZZZZZZZ";
END ARCHITECTURE multiple_drivers;
```

- Multiple Assignments to a Resolved Signal

Resolutions

U	U	X	0	1	Z
U	**U**	**U**	**U**	**U**	**U**
X	**U**	**X**	**X**	**X**	**X**
0	**U**	**X**	**0**	**X**	**0**
1	**U**	**X**	**X**	**1**	**1**
Z	**U**	**X**	**0**	**1**	**Z**

- **Partial *std_logic resolved* Function**

Gate Level Combinational Circuits

```
ENTITY AND2 IS PORT (i1, i2 : IN std_logic; o1: OUT std_logic);
END ENTITY AND2;

ARCHITECTURE example OF AND2 IS
BEGIN
   o1 <= i1 AND i2 AFTER 3 NS;
END example;
```

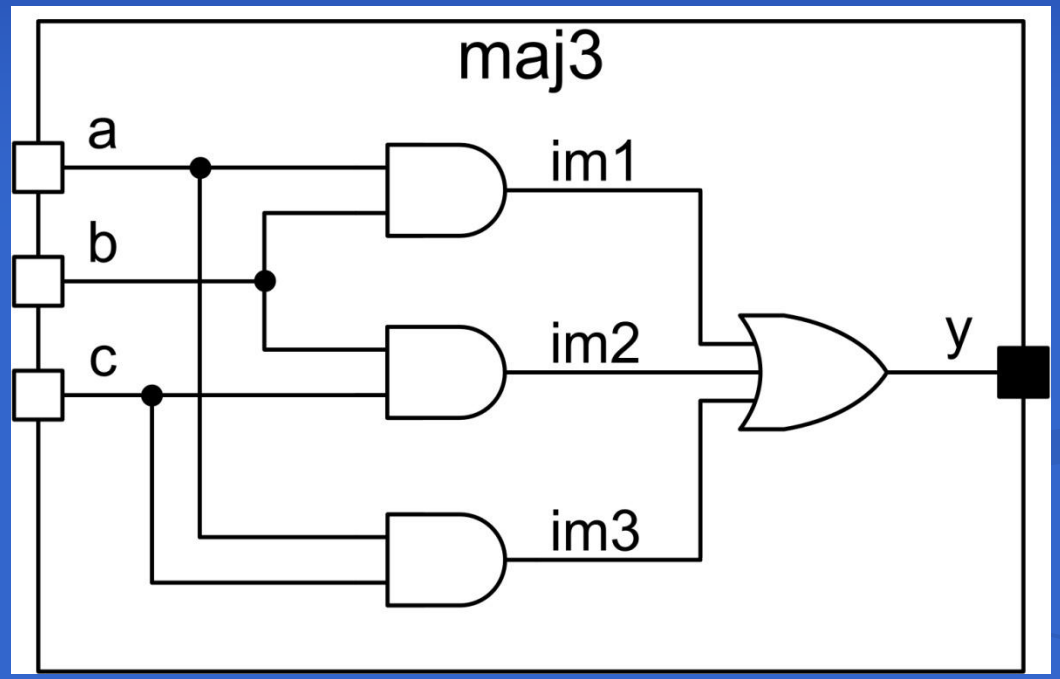
```
ENTITY OR3 IS PORT (i1, i2, i3 : IN std_logic; o1: OUT std_logic);
END ENTITY OR3;

ARCHITECTURE example OF OR3 IS
BEGIN
   o1 <= i1 OR i2 OR i3 AFTER 6 NS;
END example;
```

```
ENTITY BUFIF1 IS PORT (i1, en : IN std_logic; o1: OUT std_logic);
END ENTITY BUFIF1;

ARCHITECTURE example OF BUFIF1 IS
BEGIN
   o1 <= i1 AFTER 4 NS WHEN en = '1' ELSE 'Z' AFTER 3 NS;
END example;
```

```
ENTITY BUFIF0 IS PORT (i1, en : IN std_logic; o1: OUT std_logic);
END ENTITY BUFIF0;

ARCHITECTURE example OF BUFIF0 IS
BEGIN
   o1 <= i1 AFTER 4 NS WHEN en = '0' ELSE 'Z' AFTER 3 NS;
END example;
```

- **Basic Primitives Described in VHDL**

Majority Example

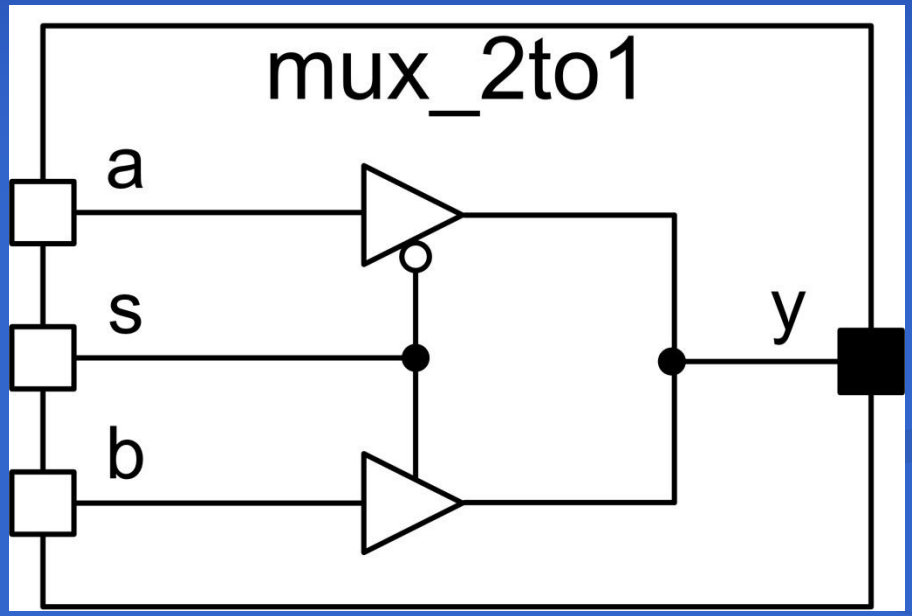


- A Majority Circuit

Majority Example

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
ENTITY maj3 IS
   PORT (a, b, c : IN std_logic;    y  : OUT
              std_logic);
END maj3;

ARCHITECTURE gate_level OF maj3 IS
   SIGNAL im1, im2, im3 : std_logic;
BEGIN
   ANDa: ENTITY WORK.AND2 PORT MAP (a, b, im1);
   ANDb: ENTITY WORK.AND2 PORT MAP (b, c, im2);
   ANDc: ENTITY WORK.AND2 PORT MAP (a, c, im3);
   ORa : ENTITY WORK.OR3  PORT MAP (im1, im2, im3, y);
END ARCHITECTURE gate_level;
```

Multiplexer Example

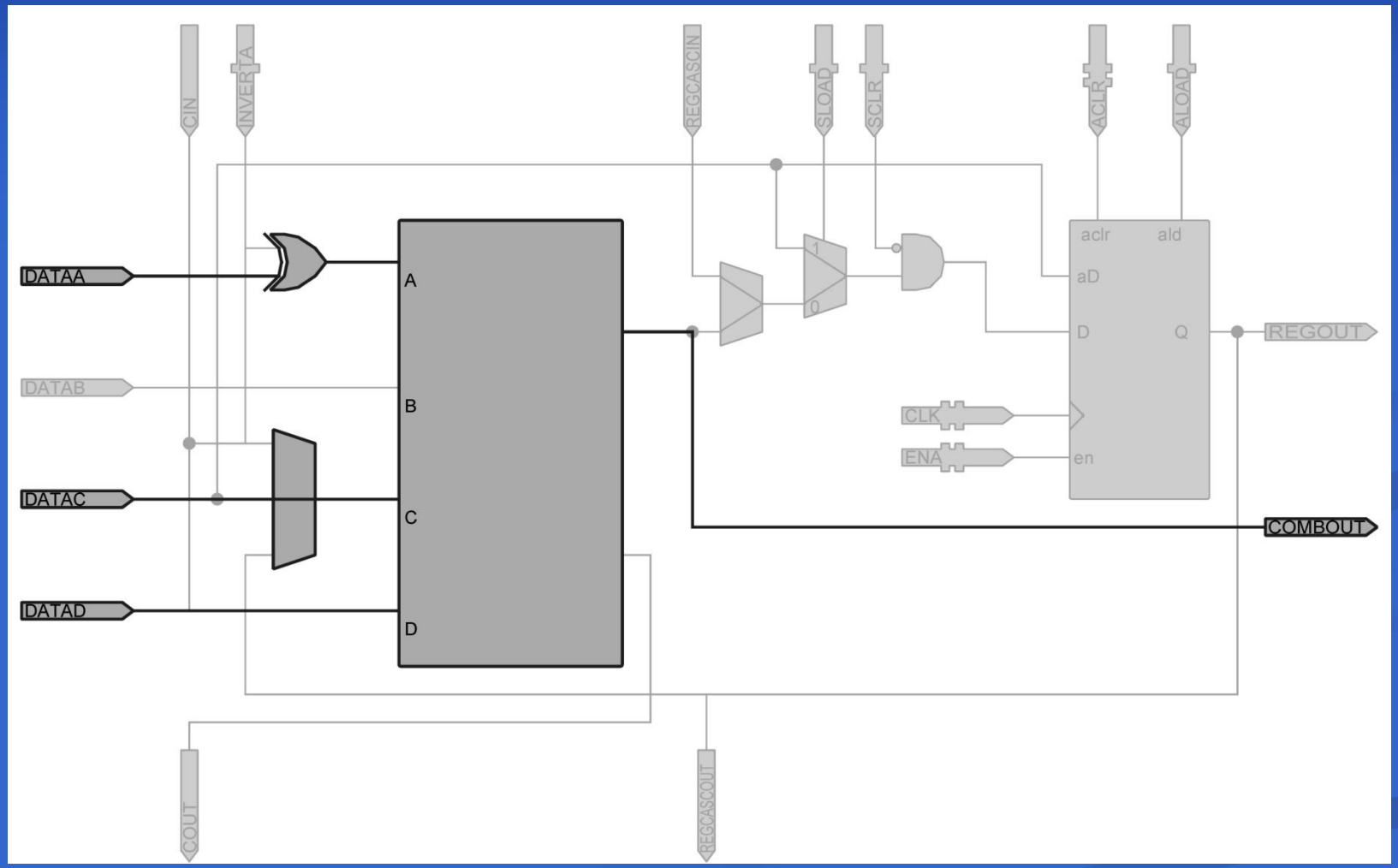


- Multiplexer Using Three-state Gates

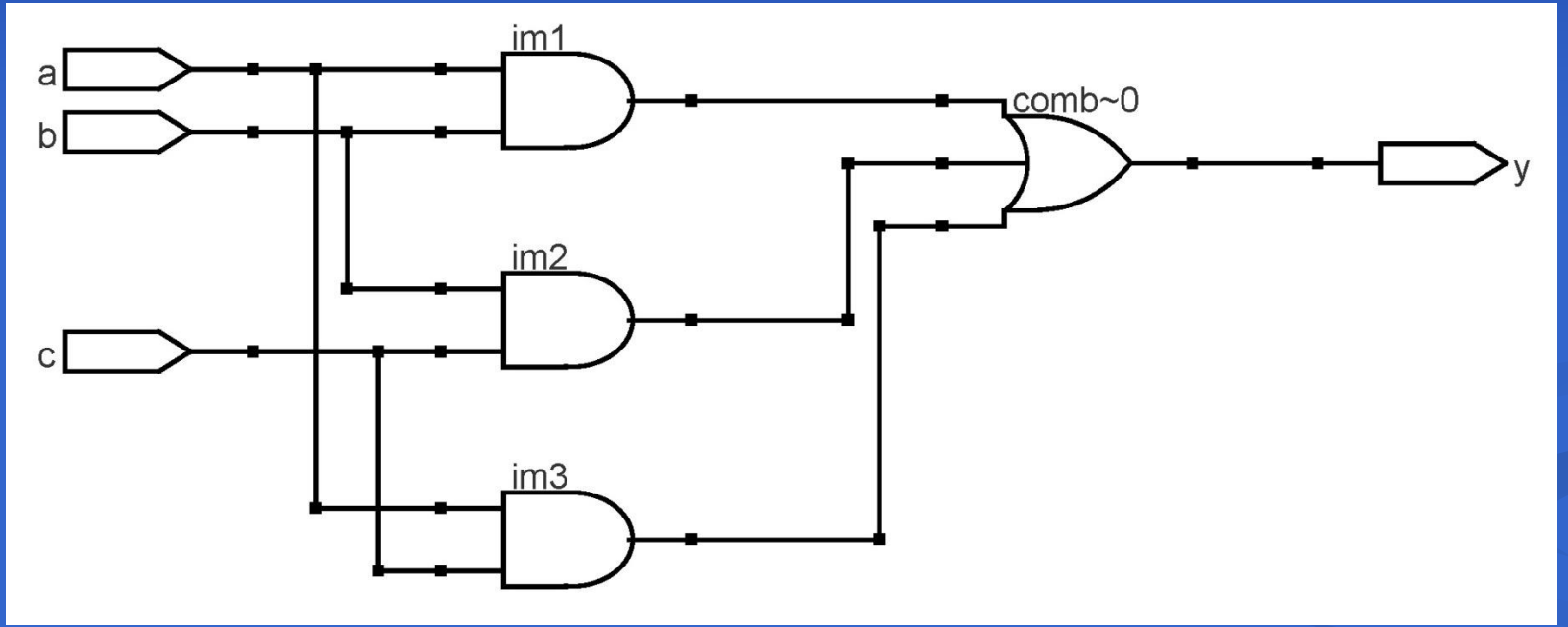
Multiplexer Example

```
ENTITY mux_2to1 IS
                PORT (a, b, s: IN std_logic; y: OUT std_logic);
END ENTITY mux_2to1;
ARCHITECTURE gate_level OF mux_2to1 IS BEGIN
   BUFIF1a: ENTITY WORK.BUFIF1(example) PORT MAP (b, s, y);
   BUFIF1b: ENTITY WORK.BUFIF0(example) PORT MAP (a, s, y);
END ARCHITECTURE gate_level;
```

Gate Level Synthesis



Gate Level Synthesis



- **RTL (logical) View of Synthesized *maj3***

Descriptions by Use of Equations

Boolean Operators	NOT	AND	OR	NAND	NOR	XOR	XNOR	
Comparison Operators	=	/=	<	<=	>	>=		
Arithmetic Operators	+	-	ABS	MOD	REM	*	/	**
Concat. Operators	&							

- VHDL Operators

XOR Example

```
ENTITY xor2 IS
   PORT (i1, i2: IN std_logic; o1: OUT std_logic);
END ENTITY xor2;
--
ARCHITECTURE expression OF xor2 IS
BEGIN
   o1 <= i1 XOR i2 AFTER 3 NS;
END ARCHITECTURE expression;
```

Full-Adder Example

```
ENTITY full_adder IS
   PORT (a, b, cin : IN std_logic;
         sum, cout : OUT std_logic);
END ENTITY full_adder;
--
ARCHITECTURE expression OF full_adder IS
BEGIN
   sum <= a XOR b XOR cin AFTER 0.3 NS;
   cout <= (a AND b) OR (a AND cin) OR (b AND cin) AFTER 0.2
NS;
END ARCHITECTURE expression;
```

- Assign Statement and Boolean

Comparator Example

```
ENTITY comp_4bit IS PORT (
in1, in2 : IN std_logic_vector (3 DOWNTO 0);
eq       : OUT std_logic );
END comp_4bit;

ARCHITECTURE functional OF comp_4bit IS
SIGNAL im : std_logic_vector (3 DOWNTO 0);
FUNCTION nor_reduce
       (in1: IN std_logic_vector (3 DOWNTO 0))
      RETURN std_logic
IS
      VARIABLE result : std_logic ;
BEGIN
       result:= NOT (in1(3) OR in1(2) OR in1(1) OR
in1(0)) ;
      RETURN result;
END;
BEGIN
im <= in1 XOR in2;
eq <= nor_reduce(im);
END functional;
```

Multiplexer Example

```
ENTITY multiplexer IS
   PORT (a, b : IN std_logic_vector; s : IN
std_logic;
         w : OUT std_logic_vector);
END ENTITY;
ARCHITECTURE expression OF multiplexer IS
BEGIN
   w <= a WHEN s = '0' ELSE b;
END ARCHITECTURE expression;
```

- An Unconstrained 2-to-1 Mux using Condition Operator

Decoder Example

```
ENTITY dcd2to4 IS
   PORT (sel: IN std_logic_vector (1 DOWNTO 0);
         y: OUT std_logic_vector (3 DOWNTO 0) );
END dcd2to4;
ARCHITECTURE structural OF dcd2to4 IS
BEGIN
   WITH sel SELECT
      y <= "0001" WHEN "00",
           "0010" WHEN "01",
           "0100" WHEN "10",
           "1000" WHEN "11",
           "0000" WHEN OTHERS;
END ARCHITECTURE structural;
```

Adder Example

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_unsigned.ALL;
ENTITY adder8 IS PORT (
   a    : IN  std_logic_vector (7 DOWNTO 0);
   b    : IN  std_logic_vector (7 DOWNTO 0);
   ci   : IN  std_logic;
   s    : OUT std_logic_vector (7 DOWNTO 0);
   co   : OUT std_logic );
END ENTITY adder8;
--
ARCHITECTURE equation OF adder8 IS
   SIGNAL mid : std_logic_vector (8 DOWNTO 0);
BEGIN
    mid <= ('0'&a) + ('0'&b) + ci;
    co <= mid (8);
    s  <= mid (7 DOWNTO 0);
END equation;
```

- Adder with Carry-in and Carry-out

ALU Example

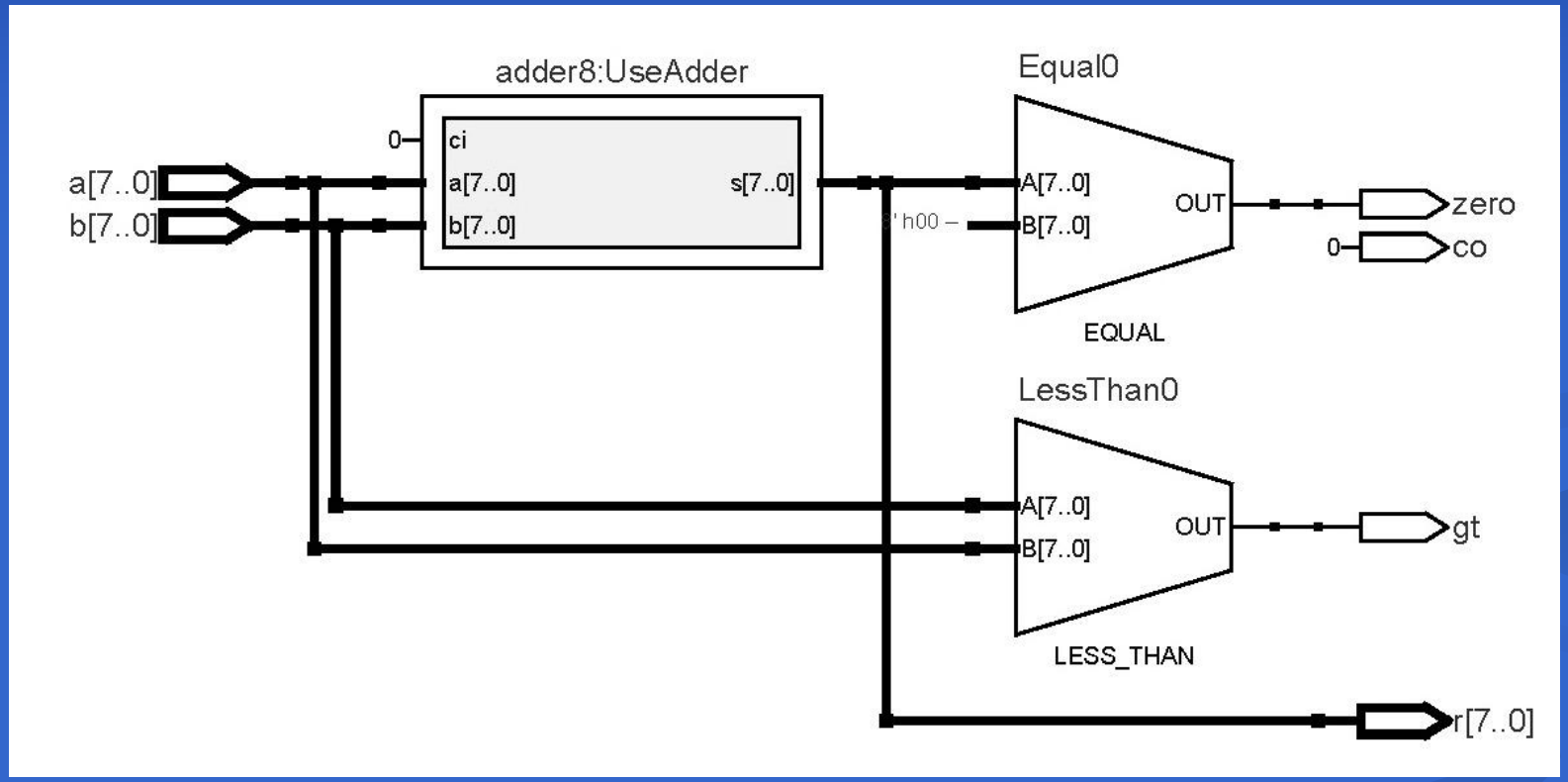
```
ENTITY alu8 IS PORT (
   a, b   : IN  std_logic_vector (7 DOWNTO 0);
   addsub : IN  std_logic;
   gt, zero, co : OUT std_logic;
   r      : OUT std_logic_vector (7 DOWNTO 0));
END ENTITY alu8;
ARCHITECTURE assigns OF alu8 IS
   SIGNAL mid : std_logic_vector (8 DOWNTO 0);
BEGIN
    mid <= ('0'& a) + ('0'& b) WHEN addsub = '1' ELSE ('0'& a)
- ('0'& b);
    co <= mid (8);
    r  <= mid (7 DOWNTO 0);
    gt <= '1' WHEN a > b ELSE '0';
    zero <='1' WHEN mid (7 DOWNTO 0) = "00000000" ELSE '0';
END assigns;
```

ALU Example Using Adder

```
ENTITY alu8add IS PORT (
   a, b   : IN  std_logic_vector (7 DOWNTO 0);
   gt, zero, co : OUT std_logic;
   r      : OUT std_logic_vector (7 DOWNTO 0));
END ENTITY alu8add;
ARCHITECTURE assigns OF alu8add IS
   SIGNAL mid8 : std_logic_vector (7 DOWNTO 0);
   SIGNAL mid1 : std_logic;
BEGIN
   AD: ENTITY WORK.adder8 PORT MAP (a, b, '0', mid8, OPEN);
-- AD: ENTITY WORK.adder8 PORT MAP
--    (a => a, b => b, ci => '0', s => mid8);
   r  <= mid8;
   gt <= '1' WHEN a > b ELSE '0';
   zero <= '1' WHEN mid8 = "00000000" ELSE '0';
END assigns;
```

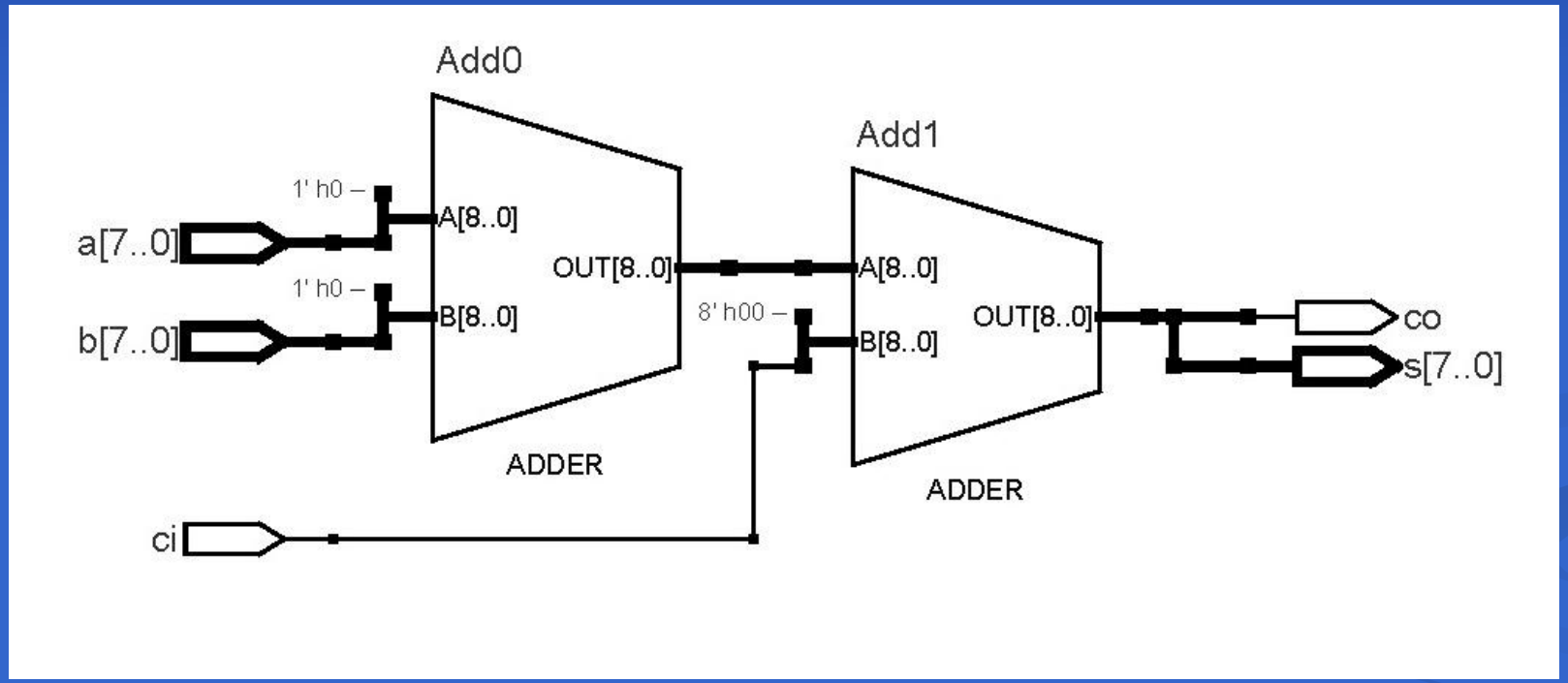
- **ALU VHDL Code Using Instantiating an Adder**

Synthesis of Assignment Statements



- ***ALU_Adder* RTL View after Synthesis**

Synthesis of Assignment Statements



- ***ALU_Adder*** **RTL View after Synthesis**

Descriptions with Sequential Flow

```
ENTITY maj3 IS
   PORT (a, b, c : IN std_logic;
                           y        : OUT std_logic);
END maj3;
ARCHITECTURE sequential OF maj3 IS
BEGIN
   PROCESS (a, b, c)
   BEGIN
      y <= (a AND b) OR (b AND c) OR (a AND c);
   END PROCESS;
END ARCHITECTURE sequential;
```

- Procedural Block Describing a Majority Circuit

Majority Example with Delay

```
ARCHITECTURE sequential_delay OF maj3 IS
BEGIN
   PROCESS (a, b, c)
   BEGIN
      y <= (a AND b) OR (b AND c) OR (a AND c) AFTER 5
NS;
   END PROCESS;
END ARCHITECTURE sequential_delay;
```

Procedural Multiplexer Example

```
ENTITY multiplexer IS
   PORT (a, b, s : IN BIT; w : OUT BIT);
END ENTITY;
--
ARCHITECTURE procedural OF multiplexer IS BEGIN
   PROCESS (a, b, s) BEGIN
      IF (s = '0') THEN w <= a;
      ELSE w <= b;
      END IF;
   END PROCESS;
END ARCHITECTURE procedural;
```

- Sequential Flow Multiplexer

Procedural ALU Example

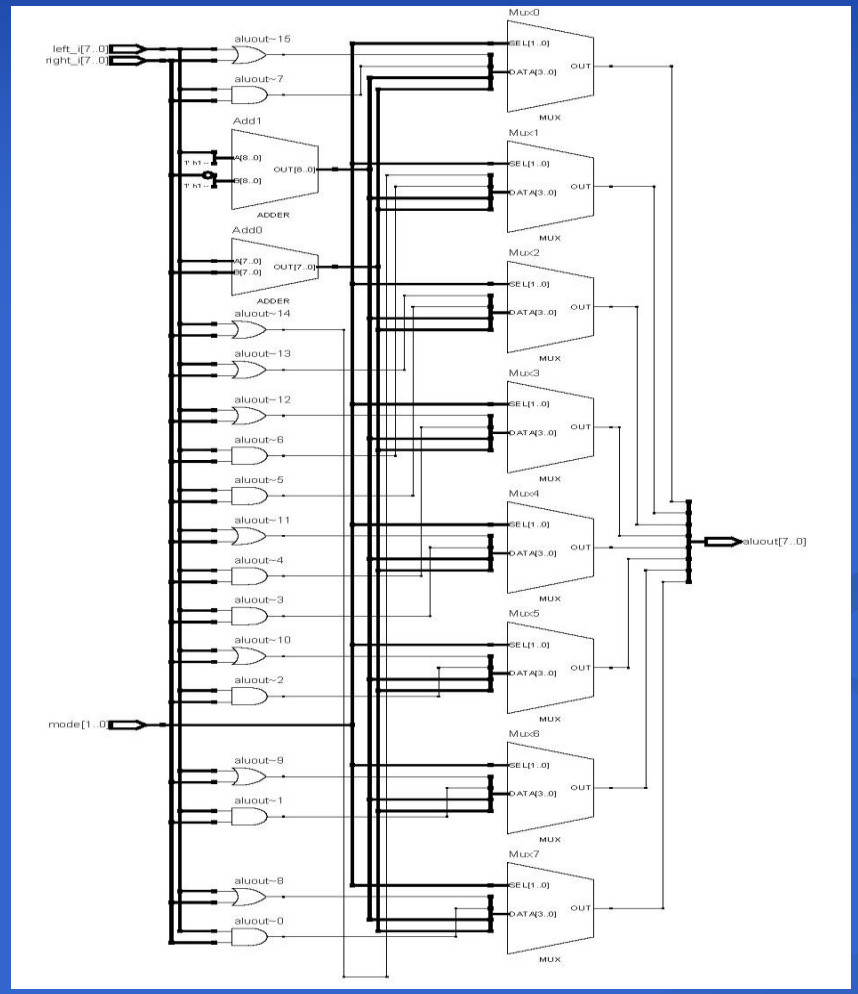
```
ENTITY alu8 IS
   PORT (left_i, right_i: IN std_logic_vector (7 DOWNTO 0);
         mode : IN std_logic_vector (1 DOWNTO 0);
         aluout : OUT std_logic_vector (7 DOWNTO 0));
END ENTITY;
--
ARCHITECTURE procedural OF alu8 IS BEGIN
   PROCESS (left_i, right_i, mode) BEGIN
      CASE mode IS
         WHEN "00" => aluout <= left_i + right_i;
         WHEN "01" => aluout <= left_i - right_i;
         WHEN "10" => aluout <= left_i AND right_i;
         WHEN "11" => aluout <= left_i OR right_i;
         WHEN OTHERS => aluout <= "XXXXXXXX";
      END CASE;
   END PROCESS;
END ARCHITECTURE procedural;
```

Bussing

```
ENTITY bussing IS
   PORT (
      busin1: IN std_logic_vector (3 DOWNTO 0);
      busin2: IN std_logic_vector (3 DOWNTO 0);
      busin3: IN std_logic_vector (3 DOWNTO 0);
      en1: IN  std_logic;
      en2: IN  std_logic;
      en3: IN  std_logic;
      busout: OUT std_logic_vector(3 DOWNTO 0) );
END bussing;
--
ARCHITECTURE structural OF bussing IS
BEGIN
   busout <= busin1 WHEN en1 = '1' ELSE (OTHERS => 'Z');
   busout <= busin2 WHEN en2 = '1' ELSE (OTHERS => 'Z');
   busout <= busin3 WHEN en3 = '1' ELSE (OTHERS => 'Z');
END structural;
```

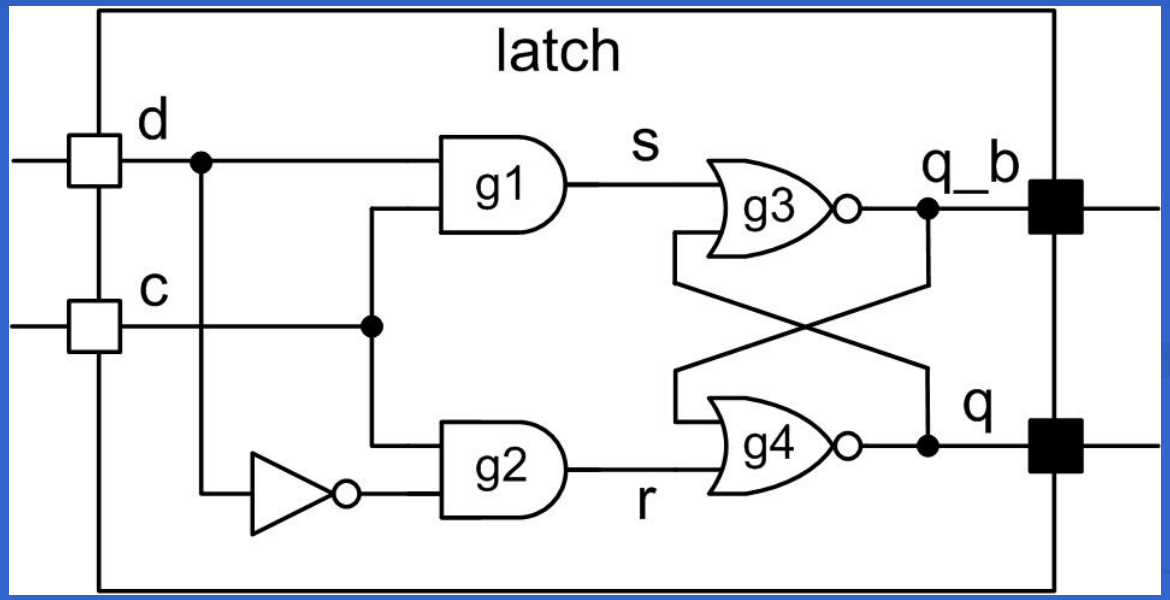
- Three-state Bussing

Synthesizing Procedural Blocks



- Synthesis of Sequential Flow ALU

Basic Memory Elements at the Gate Level



- Clocked D-latch

Basic Memory Elements at the Gate Level

```
ENTITY latch IS
   PORT (d, c: IN std_logic;
         q, q_b : BUFFER std_logic);
END latch;
ARCHITECTURE structural OF latch IS
   SIGNAL s, r : std_logic;
BEGIN
   s   <= c AND d       AFTER 6 ns;
   r   <= c AND (NOT d) AFTER 6 ns;
   q_b <= s NOR q       AFTER 4 ns;
   q   <= r NOR q_b     AFTER 4 ns;
END structural;
```

- 37 VHDL Code for a Clocked D-latch

Basic Memory Elements at the Gate Level

```
ENTITY master_slave IS
   PORT (d, c: IN std_logic;
         q : OUT std_logic);
END master_slave;
ARCHITECTURE dual OF master_slave IS
   SIGNAL qm : std_logic;
BEGIN
   qm <= d WHEN c = '1';
   q  <= qm WHEN c = '0';
END dual;
```

- Master-Slave Flip-Flop

Memory Elements Using Procedural Statements

```
ENTITY latch1 IS
   PORT (d, c: IN std_logic; q: OUT std_logic);
END latch1;
ARCHITECTURE behavioral OF latch1 IS
BEGIN
   PROCESS (d, c)
   BEGIN
      IF c = '1' THEN
         q <= d;
      END IF;
   END PROCESS;
END ARCHITECTURE behavioral;
```

- Procedural Latch

D Flip-Flop

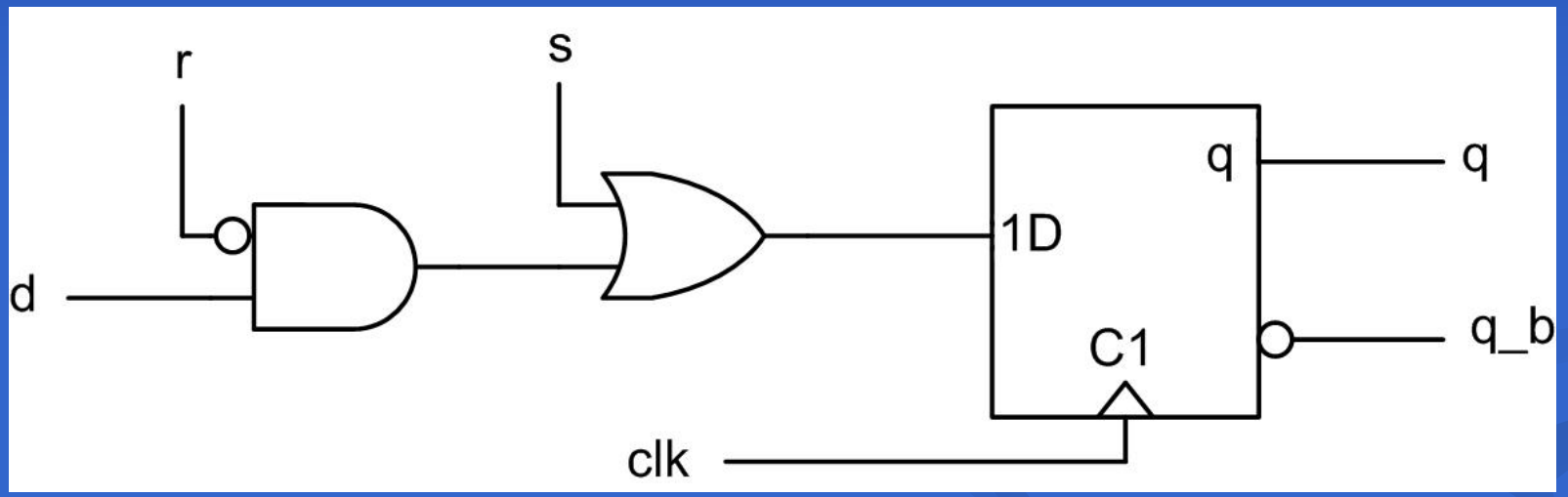
```
ENTITY DFF1 IS
   PORT (d, clk: IN std_logic; q : OUT std_logic);
END DFF1;
--
ARCHITECTURE behavioral OF DFF1 IS
BEGIN
   PROCESS (clk)
   BEGIN
      IF clk = '1' AND clk'EVENT THEN
         q <= d;
      END IF;
   END PROCESS;
END ARCHITECTURE behavioral;
```

- A Positive-Edge D Flip-Flop

Synchronous Control

```
ENTITY DFF1sr IS
   PORT (d, clk, s, r: IN std_logic; q : OUT std_logic);
END DFF1sr;
--
ARCHITECTURE behavioral OF DFF1sr IS
BEGIN
   PROCESS (clk)
   BEGIN
      IF clk = '1' AND clk'EVENT THEN
         IF s = '1' THEN
            q <= '1';
         ELSIF r = '1' THEN
            q <= '0';
         ELSE
            q <= d;
         END IF;
      END IF;
   END PROCESS;
END ARCHITECTURE behavioral;
```

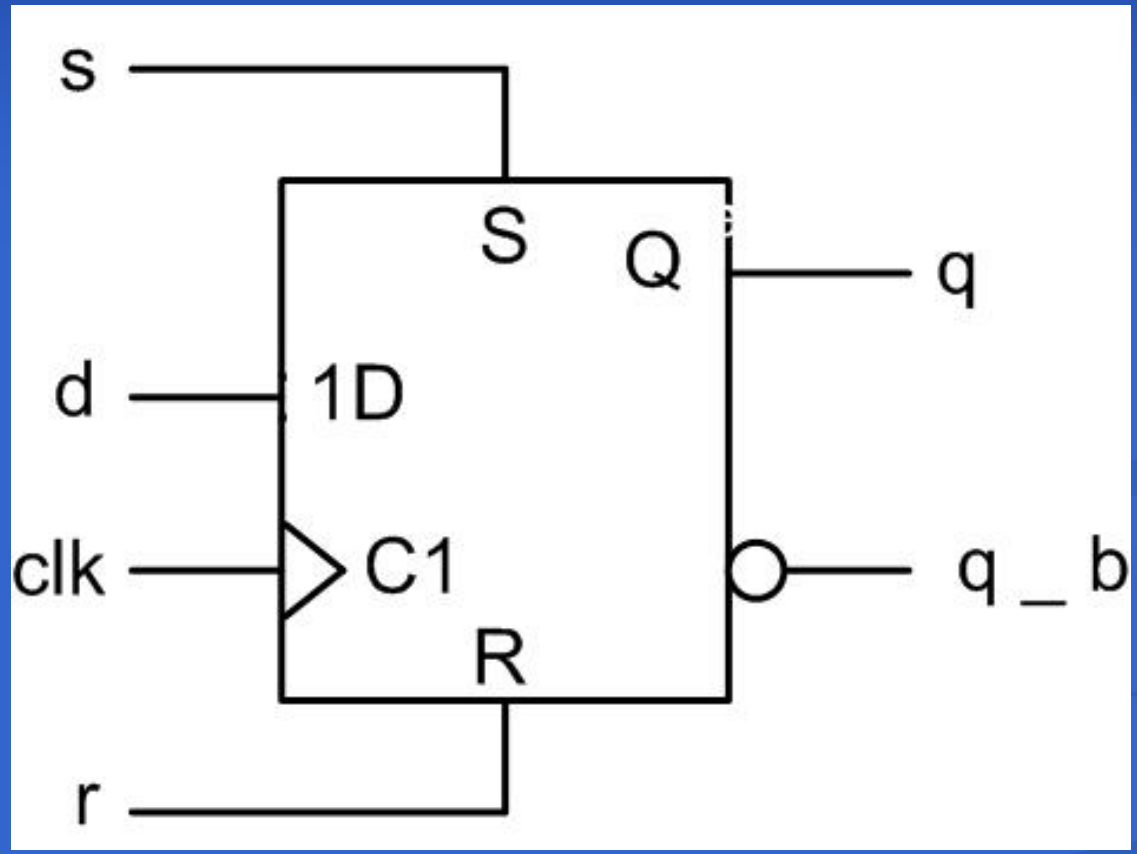
Synchronous Control



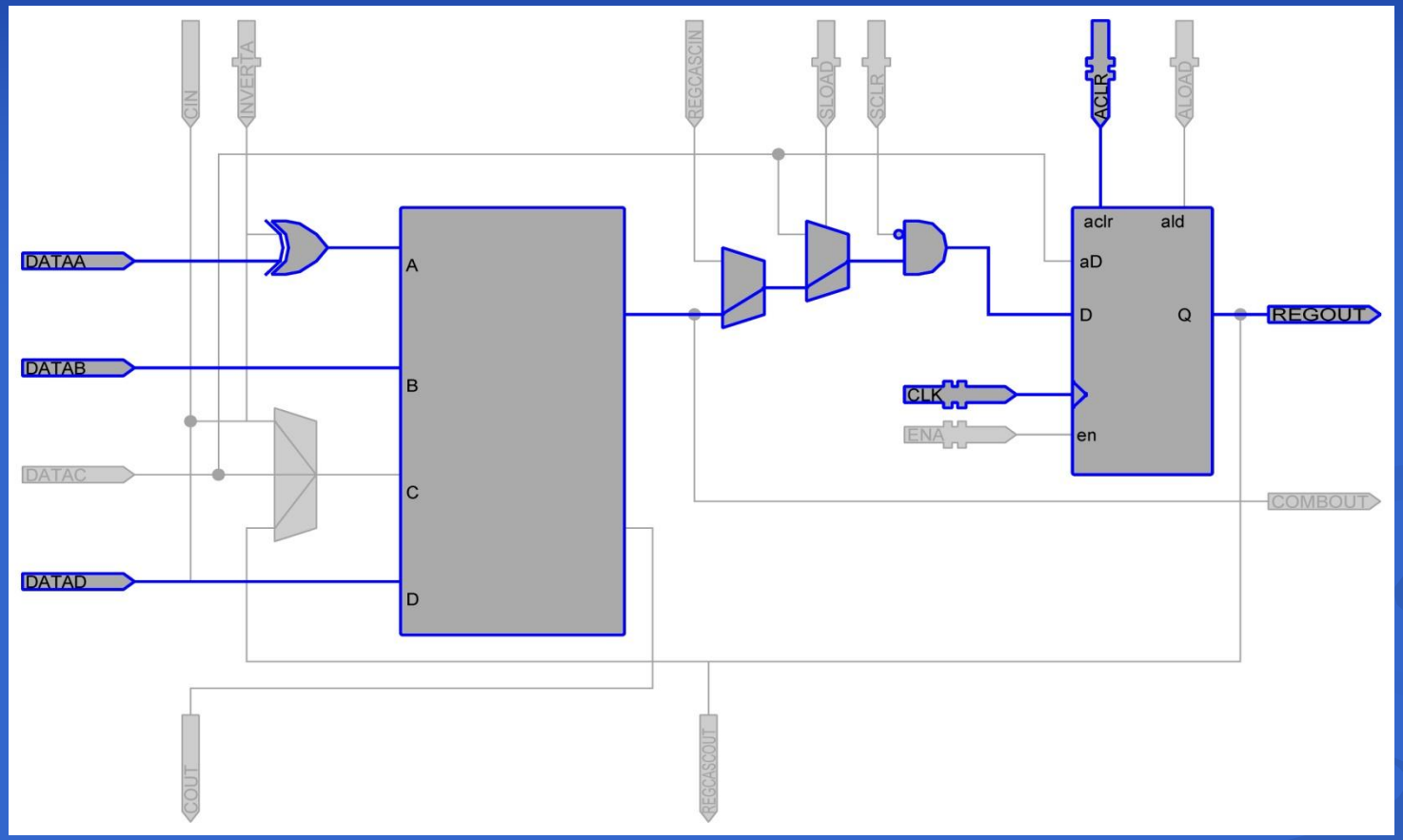
Asynchronous Control

```
ARCHITECTURE asynchronous OF DFF1sr IS
BEGIN
   PROCESS (clk, s, r) BEGIN
      IF s = '1' THEN
         q <= '1';
      ELSIF r = '1' THEN
         q <= '0';
      ELSIF clk = '1' AND clk'EVENT THEN
         q <= d;
      END IF;
   END PROCESS;
END ARCHITECTURE asynchronous;
```

Asynchronous Control

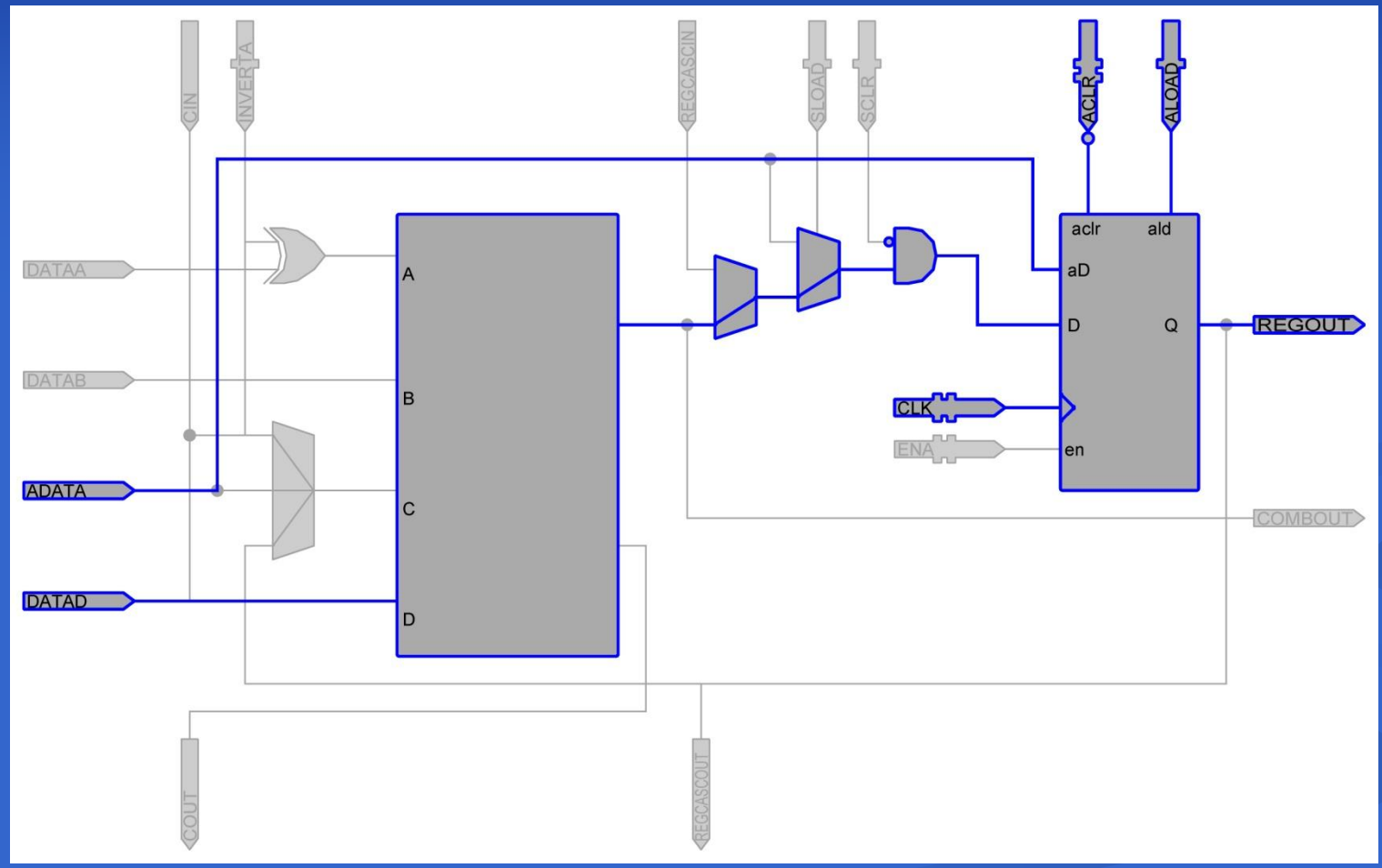


Flip-flop Synthesis



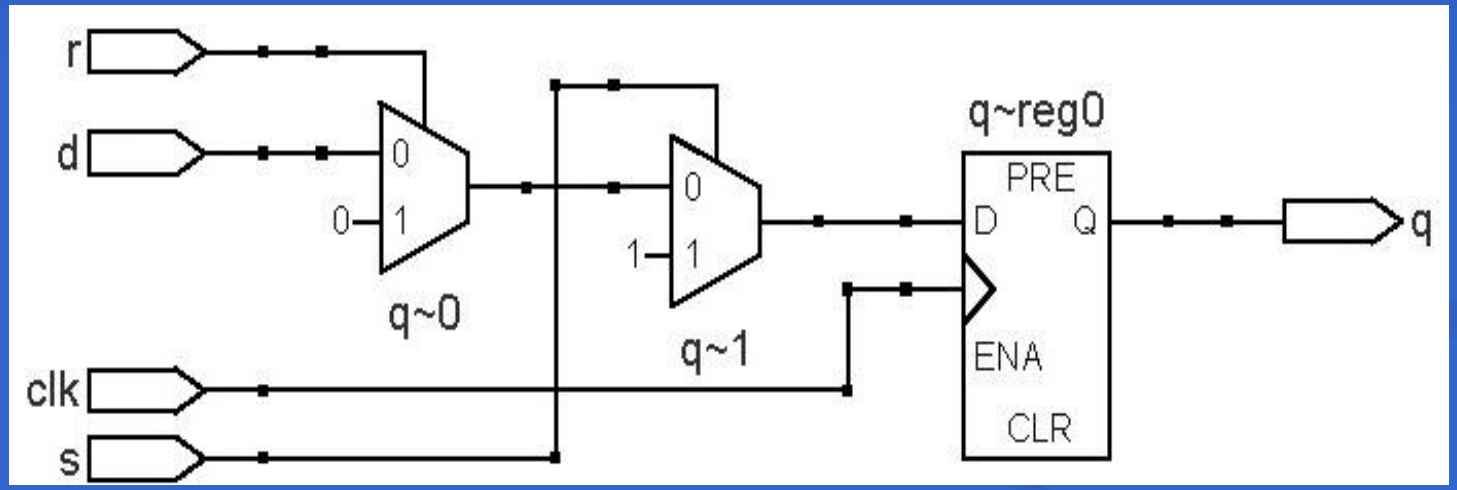
- **Synchronous Flip-Flop Synthesis**

Flip-flop Synthesis

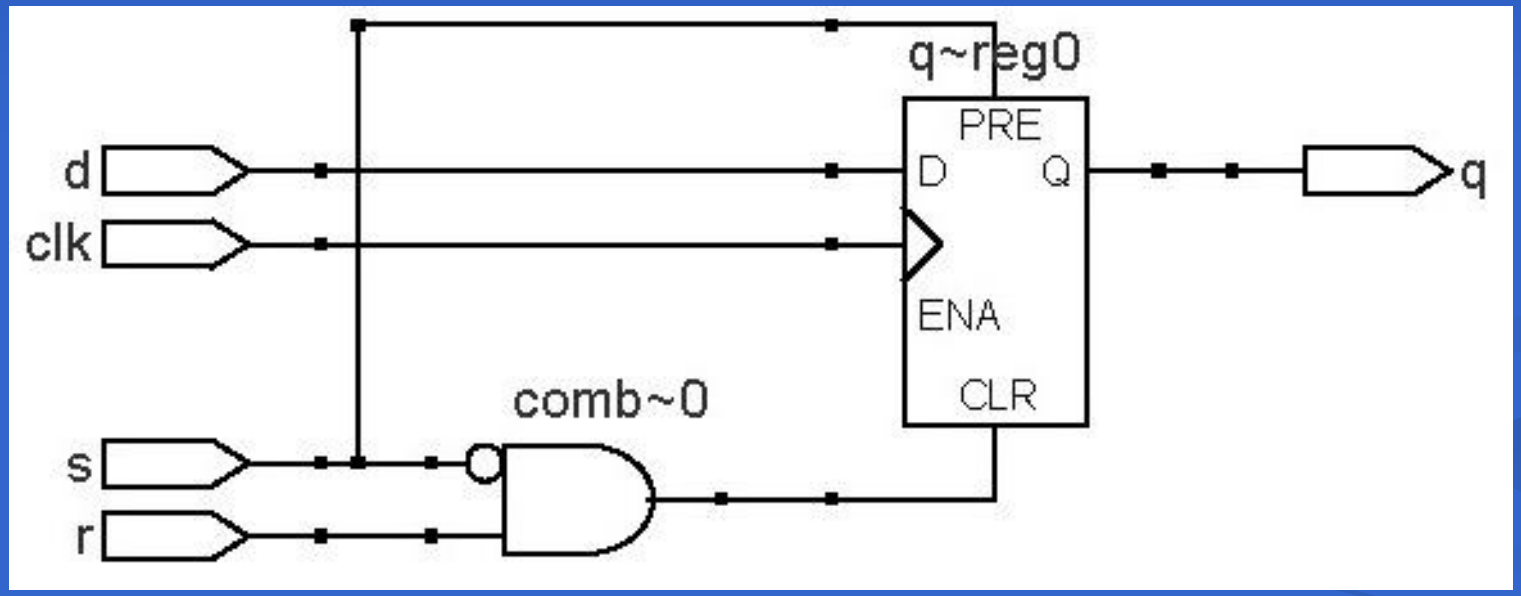


- **Asynchronous Flip-Flop Synthesis**

Flip-flop Synthesis



Flip-flop Synthesis



Registers

```
ENTITY register8 IS
   PORT (
  : IN std_logic_vector (7 DOWNTO 0);              d
  : IN std_logic;              clk, s, r
  : OUT std_logic_vector ( 7 DOWNTO 0));              q
END register8;
--
ARCHITECTURE behavioral OF register8 IS
BEGIN
   PROCESS (clk)
   BEGIN
      IF clk = '1' AND clk'event THEN
         IF s= '1' THEN
            q <= (OTHERS => '1');
         ELSIF r = '1' THEN
            q <= (OTHERS => '0');
         ELSE
            q <= d;
         END IF;
      END IF;
   END PROCESS;
END behavioral;
```

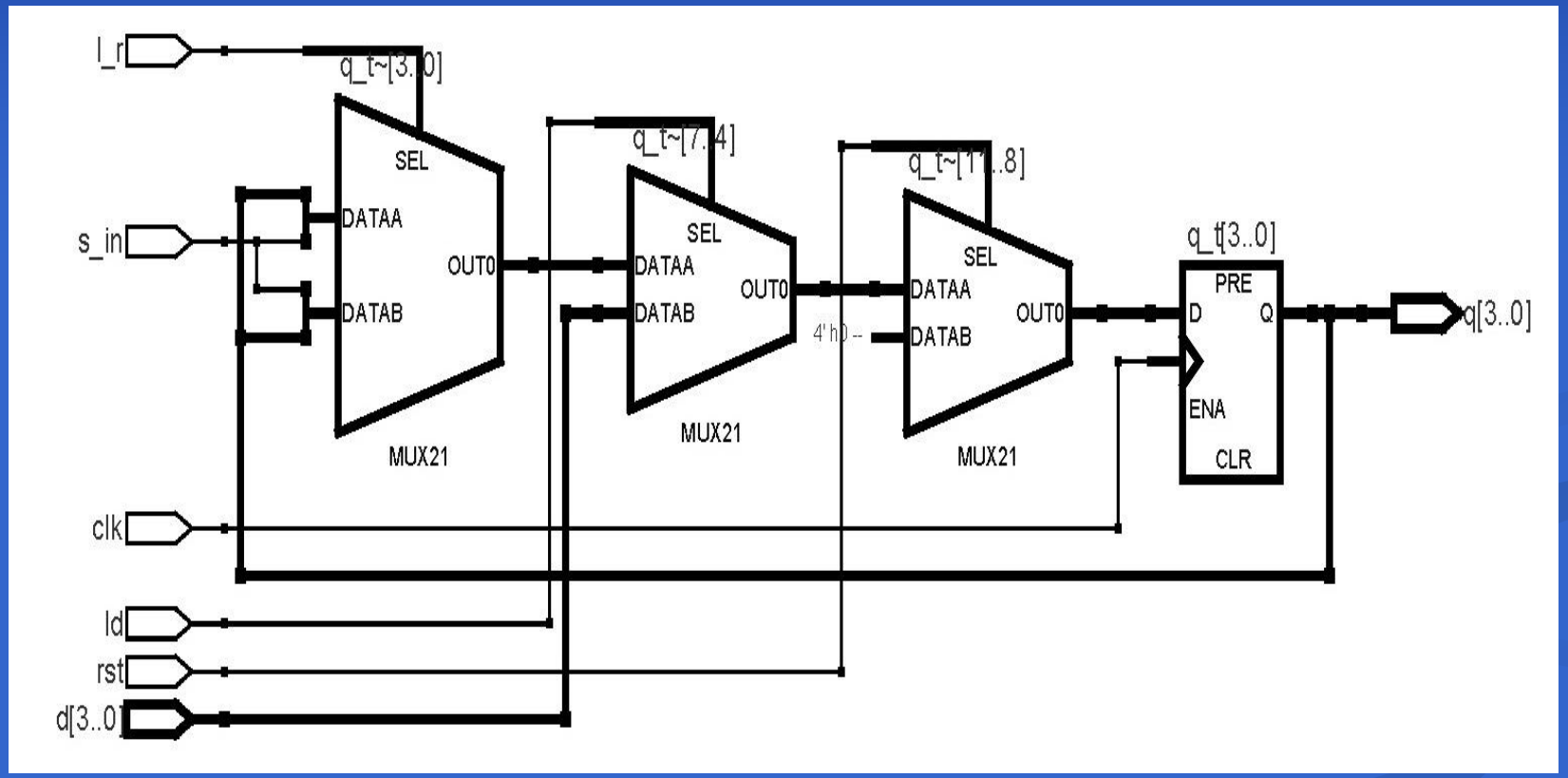
Shift-Registers

```
ENTITY shift_reg4 IS
   PORT (
      d   : IN std_logic_vector (3 DOWNTO 0);
      clk, ld, rst, l_r, s_in : IN std_logic;
      q   : OUT std_logic_vector (3 DOWNTO 0));
END shift_reg4;
ARCHITECTURE behavioral OF shift_reg4 IS
BEGIN
   PROCESS (clk)
      VARIABLE q_t: std_logic_vector (3 DOWNTO 0);
   BEGIN
      IF rising_edge (clk) THEN
         IF rst= '1' THEN
            q_t := (OTHERS => '0');
         ELSIF ld = '1' THEN
            q_t := d;
         ELSIF l_r = '1' THEN
            q_t :=  q_t (2 DOWNTO 0) & s_in ;
         ELSE
                     q_t := s_in & q_t (3 DOWNTO 1);
         END IF;
      END IF;
      q <= q_t;
   END PROCESS;
END behavioral;
```

Counters

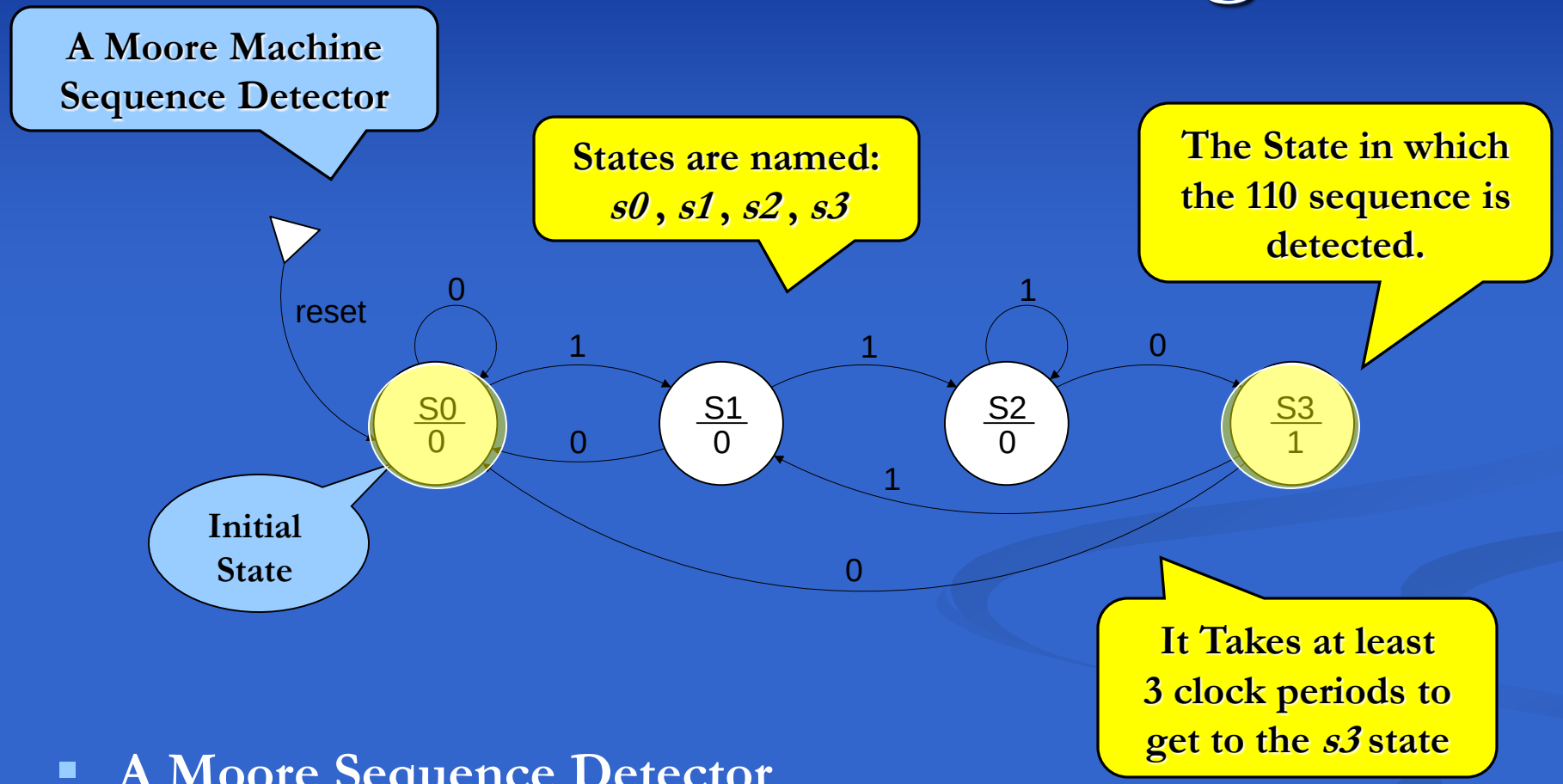
```
ENTITY counter4 IS
   PORT (reset, clk : IN std_logic;
         count : OUT std_logic_vector (3 DOWNTO 0));
END ENTITY;
--
ARCHITECTURE procedural OF counter4 IS
   SIGNAL cnt_reg : std_logic_vector (3 DOWNTO 0);
BEGIN
   PROCESS (clk)
   BEGIN
      IF (clk = '0' AND clk'EVENT) THEN
         IF (reset='1') THEN
            cnt_reg <="0000" AFTER 1.2 NS;
         ELSE
            cnt_reg <= cnt_reg + 1 AFTER 1.2 NS;
         END IF;
      END IF;
   END PROCESS;
   count <= cnt_reg;
END ARCHITECTURE procedural;
```

Synthesis of Shifters and Counters



- **Shift Register Synthesis RTL View**

State Machine Coding



- A Moore Sequence Detector

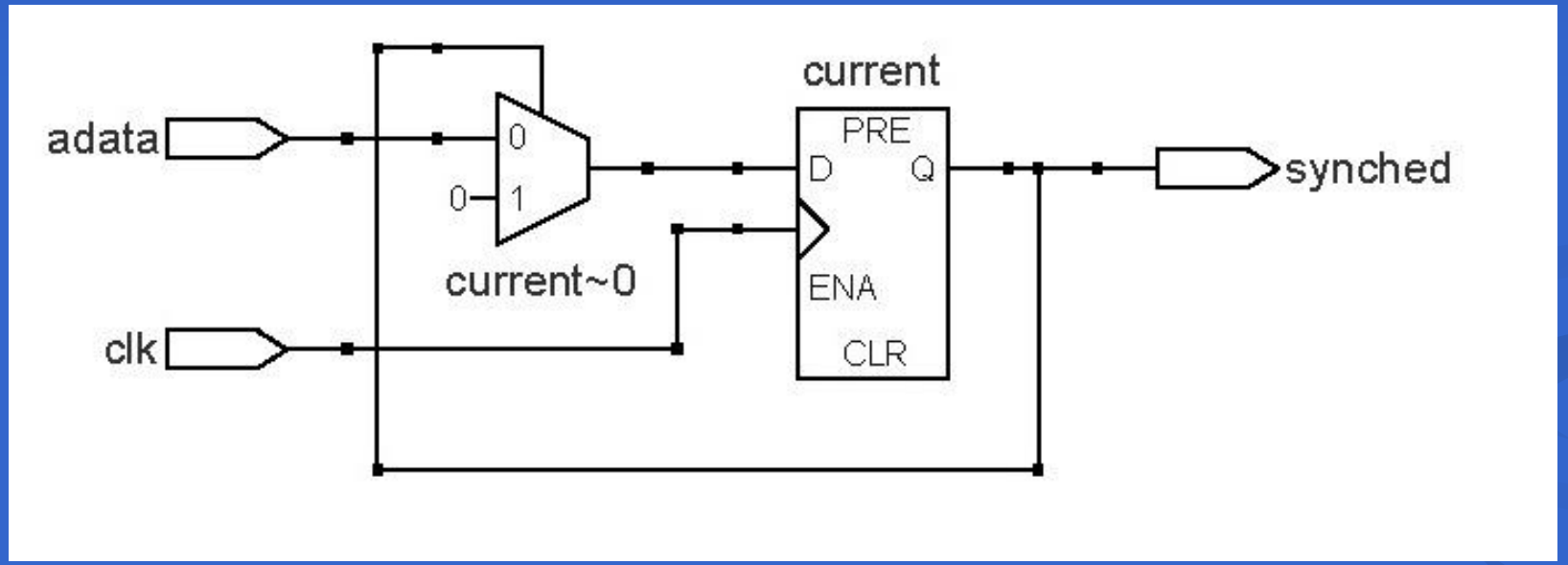
Moore Machine VHDL Code

```
ENTITY detector110 IS
   PORT (a, clk, reset : IN std_logic; w : OUT
std_logic);
END ENTITY;
--
ARCHITECTURE procedural OF detector110 IS
   TYPE state IS (S0, S1, S2, S3);
   SIGNAL current : state := S0;
BEGIN
   PROCESS (clk) BEGIN
      IF (clk = '0' AND clk'EVENT) THEN
         IF reset = '1' THEN current <= S0;
         ELSE
            CASE current IS
               WHEN S0 =>
                  IF a='1' THEN current <= S1;
                  ELSE current <= S0; END IF;
               WHEN S1 =>
                  IF a='1' THEN current <= S2;
                  ELSE current <= S0; END IF;
               WHEN S2 =>
                  IF a='1' THEN current <= S2;
                  ELSE current <= S3; END IF;
               WHEN S3 =>
                  IF a='1' THEN current <= S1;
                  ELSE current <= S0; END IF;
               WHEN OTHERS => current <= S0;
            END CASE;
         END IF;
      END IF;
   END PROCESS;
   w <= '1' WHEN current = S3 ELSE '0';
END ARCHITECTURE procedural;
```

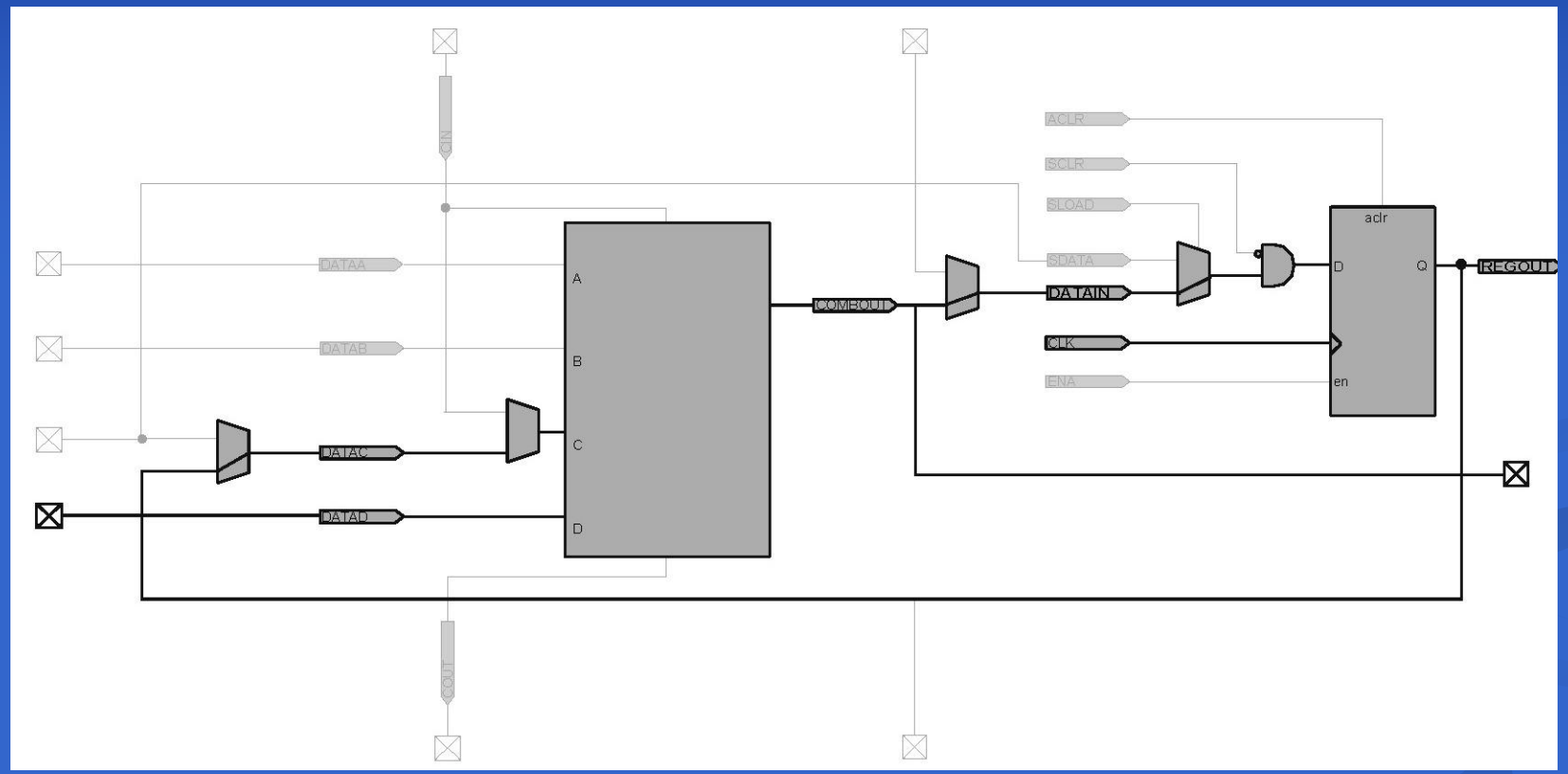
Pulse Synchronizer

```
ENTITY synchronizer IS
   PORT (clk, adata : IN std_logic;
         synched : OUT std_logic);
END ENTITY;
--
ARCHITECTURE procedural OF synchronizer IS
   TYPE state IS (S0, S1);
   SIGNAL current : state;
BEGIN
   PROCESS (clk) BEGIN
      IF (rising_edge(clk)) THEN
         IF current = S0 THEN
            IF adata = '0' THEN
               current <= S0;
            ELSE
               current <= S1;
         END IF;
         ELSE -- current = S1
            current <= S0;
         END IF;
      END IF;
   END PROCESS;
   synched <= '1' WHEN current = S1 ELSE '0';
END ARCHITECTURE procedural;
```

State Machine Synthesis



State Machine Synthesis



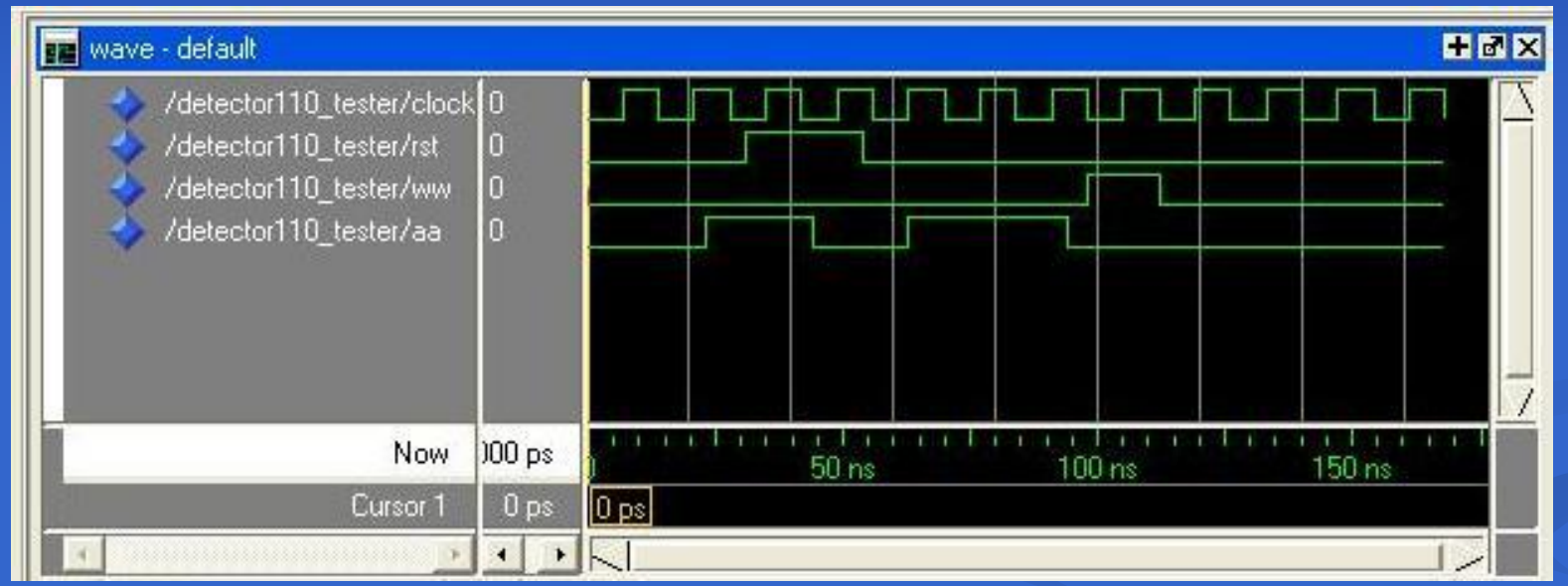
Writing Testbenches

```
ARCHITECTURE . . .
   TYPE memory IS
      ARRAY (INTEGER RANGE <>) OF
      std_logic_vector (7 DOWNTO 0);
   SIGNAL mem: memory(0 to 1023);
  BEGIN
   PROCESS (mem)
      VARIABLE memv: memory(0 to 15);
      VARIABLE data: std_logic_vector(7 DOWNTO 0);
      VARIABLE short_data: std_logic_vector(3 DOWNTO 0);
   BEGIN
      . . .
                                 data := mem(956);
      short_data := mem(931)(6 downto 3);
      memv (12) := mem(189);
      mem (932) <= data ;
      mem (321)(5 DOWNTO 2) <= short_data;
      mem (940) <= "0000" & short_data ;
   END PROCESS;
   . . .
END ARCHITECTURE;
```

Writing Testbenches

```
ENTITY detector110_tester IS END ENTITY;
--
ARCHITECTURE timed OF detector110_tester IS
   SIGNAL aa, clock, rst, ww : std_logic := '0';
BEGIN
   UUT1: ENTITY WORK.detector110 (procedural)
         PORT MAP (aa, clock, rst, ww);
   rst <= '1' AFTER 31 NS, '0' AFTER 54 NS;
   clock <= NOT clock AFTER 7 NS WHEN NOW<=165 NS ELSE '0';
   PROCESS BEGIN
      WAIT FOR 23 NS; aa <= '1';
      WAIT FOR 21 NS; aa <= '0';
      WAIT FOR 19 NS; aa <= '1';
      WAIT FOR 31 NS; aa <= '0';
      WAIT;
   END PROCESS;
   PROCESS (ww) BEGIN
      REPORT "Signal w changed to:"& std_logic'IMAGE(ww)&
             " at " & TIME'IMAGE(NOW)
      SEVERITY NOTE;
   END PROCESS;
END ARCHITECTURE timed;
```

Writing Testbenches



- **Testbench Waveform Results**

Summary

- This chapter presented:
  - RT level description in the VHDL HDL language
  - examples of synthesizable one-to-one hardware correspondence
  - introducesing some VHDL terminologies that are needed for understanding the linguistics of VHDL
  - How testbenches could be developed in VHDL and new constructs of it in this part