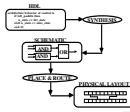
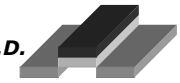
DESIGNING FPGAS & ASICS

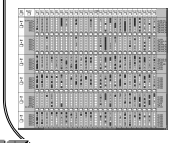
HDL-Synthesis

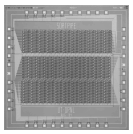




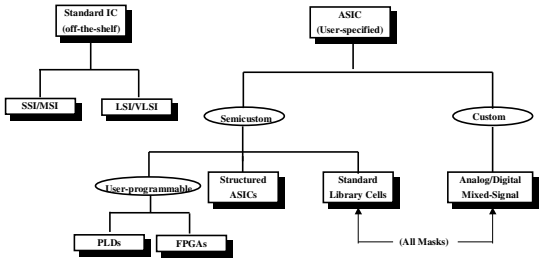
Prof. Don Bouldin, Ph.D.

Electrical & Computer Engineering
University of Tennessee
TEL: (865)-974-5444
FAX: (865)-974-5483
dbouldin@tennessee.edu



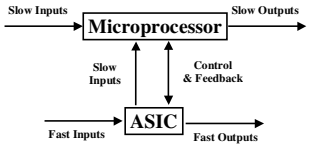


APPLICATIONS MAY USE STANDARD ICs or FPGAs/ASICs



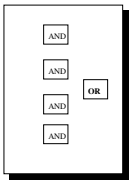
SYSTEM FUNCTIONS ARE OFTEN SPLIT BETWEEN THE CPU AND AN ASIC

- **The most economical means of implementing logic functions is to use a microprocessor.**
- **When the microprocessor is too slow or too busy to handle some fast inputs and outputs, an ASIC can be used to implement "random" logic.**



PROGRAMMABLE LOGIC DEVICES ARE BEST FOR SMALL DESIGNS WITH I/O

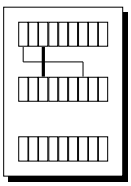
PLDs



- **Vendor prefabricates multiple sets of ANDs and ORs with programmable connections**
- **User specifies connections to implement desired logic functions**
- **Replaces 200 to 8,000 gates with single package of 20-84 pins**
- **Electrically programmable (and erasable) by the user one at a time within minutes**

FPGAS ARE BEST FOR ADAPTABLE SITUATIONS

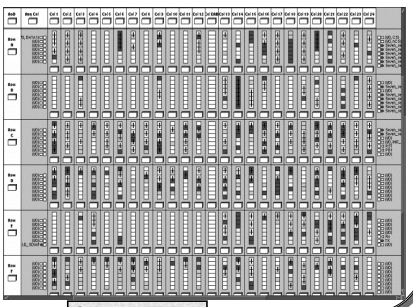
FPGAs



- **Vendor prefabricates parts with rows of gates and programmable connections**
- **User specifies connections to implement logic functions**
- **Replaces 8,000 to 200,000 gates (or more)**
- **Electrically programmable (and erasable) by the user one at a time within minutes**
- **Production quantity < 200,000**
- **PC-based development system costs $10K**

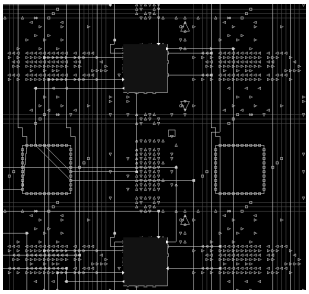
ALTERA FPGA LAYOUT

Vendor prefabricates parts with rows of gates (look-up tables) and programmable connections (not shown).



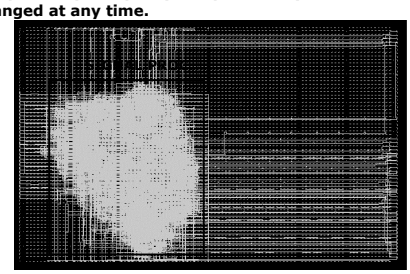
DETAILED LAYOUT OF XILINX FPGA

Programmable switches (*puddles* which have RC delay) determine which wiring segments (short, medium, long) are connected.

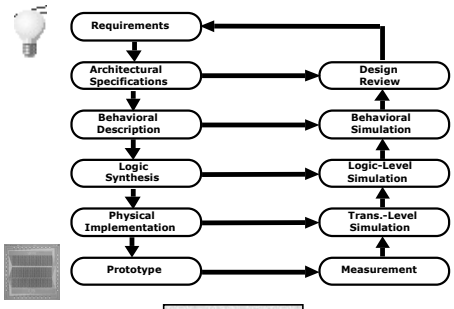


RECONFIGURABLE COMPONENTS ARE ADAPTABLE

The internal logic and interconnect of a reconfigurable component (FPGA) may be specified by the user and changed at any time.



MICROELECTRONIC SYSTEM DESIGN CONSISTS OF ITERATIVE REFINEMENTS OF SYNTHESIS AND VERIFICATION



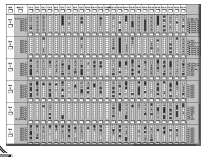
SEMI-CUSTOM DESIGN FLOW OF DIGITAL FPGAS/ASICS

1—HDL

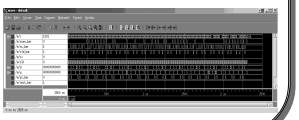
```
CASE w IS
  WHEN "00" => y <= "1000" ;
  WHEN "01" => y <= "0100" ;
  WHEN "10" => y <= "0010" ;
  WHEN OTHERS => y <= "0001";
END CASE ;
```

2--PRE-SYNTHESIS SIMULATION

3—SYNTHESIS/AUTO LAYOUT



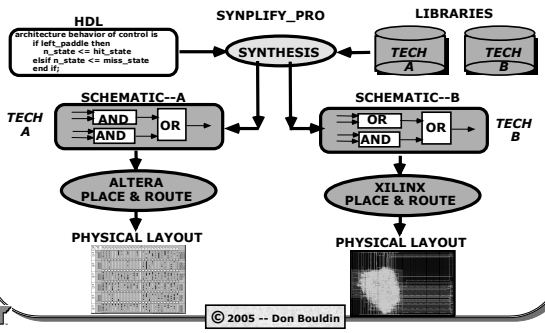
4--POST-LAYOUT SIMULATION



A HARDWARE DESCRIPTION LANGUAGE CAN BE SYNTHESIZED

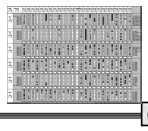
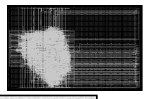
- The desired functionality and timing may be described using a *hardware description language* such as VHDL or Verilog and then *synthesized* into the structural level for a specified device.
- *Synthesis* involves:
  (1) translation into Boolean equations,
  (2) optimization for area/delay, and then
  (3) mapping to a FPGA or ASIC process (library).
- The physical level is then implemented automatically using a placement and routing program.

HDL DESIGNS CAN BE TARGETED TO MULTIPLE LAYOUTS

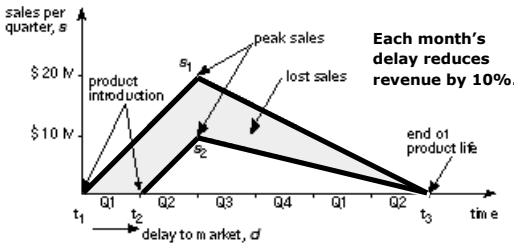


SYNTHESIS AND FPGAS CAN REDUCE TIME-TO-MARKET

- Synthesis can reduce the design time required to achieve and verify a given functionality since many candidate solutions can be constructed quickly and accurately.
- Simulation is required for verification and risk reduction but is time-consuming and may not be entirely representative of the full system environment.
- Prototyping with FPGAs can speed verification and reduce risk.
- Synthesis facilitates retargeting a design from one FPGA to another or to an ASIC when the requirements are frozen and the production quantity is sufficient.


TIME-TO-MARKET AFFECTS PRODUCT REVENUE



Each month's delay reduces revenue by 10%.

Fig. 1.13, Page 24 ASICs by M. Smith © 1997 A-D-W, Inc. Used by permission.

REDUCTION IN TIME-TO-MARKET INCREASES PROFITS

METHODOLOGY	Design Capture	Design Verification	Production and Profit
TRADITIONAL:	CAP	VER	$$$
SYNTHESIS:	CAP	VER	$$$$$$$$$
SYNTHESIS AND FPGAs:	CAP	VER	$$$$$$$$$$$$$$$$

DRIVING THE DESIGN FLOW

- Graphical User Interface (GUI)

  Presents only valid choices and can guide user.
  May improve productivity by reducing errors.
  May degrade productivity by taking longer to enter.

- Command Line Interface (CLI)

  Command lines (script) are generally faster.
  Can use old script as a guide.
  Best for iterations.

vsim -coverage Seq_TestBench -do stim.do

PRINCIPLES OF SYSTEM OPTIMIZATION

- A global figure of merit for the entire system should be determined and optimized.
- This figure of merit involves multiple dimensions including cost, area, speed, power, design time, risk, etc.
- Optimization of a particular level or component of a system may not constitute a good return on investment.
- Decisions made at the higher levels of the design are often more significant than at the lower levels.
- Designers should pinpoint and concentrate on sensitive components for which small changes yield big payoffs.

THE ORIGINAL AND PRESENT USES OF VHDL

- VHDL = VHSIC HDL = Very High-Speed Integrated Circuit Hardware Description Language
- In 1981 VHDL was developed by the US Dept. of Defense to standardize documentation for maintenance or possible redesign.
- In 1987 IEEE approved a VHDL standard.
- Since then, CAE companies have been using VHDL with enhancements for synthesis.
- In 1992, a new IEEE standard with many of these synthesis enhancements was approved.

VERILOG

- **VERILOG is a hardware description language originally developed by Gateway Automation (Cadence) for verification of logic.**
- **Cadence and other CAE companies have been using Verilog with enhancements for synthesis.**
- **Most users say Verilog is easier to learn than VHDL.**
- **VERILOG is no longer proprietary.**

TRUTH TABLE OR EQUATIONS USING VHDL

TRUTH TABLE

Row number	x_1	x_2	x_3	f
0	0	0	0	1
1	0	0	1	0
2	0	1	0	1
3	0	1	1	0
4	1	0	0	1
5	1	0	1	1
6	1	1	0	1
7	1	1	1	0

The function $f(x_1, x_2, x_3) = \sum m(0, 2, 4, 5, 6)$.

EQUATION

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
ENTITY func3 IS
      PORT ( x1, x2, x3 : IN          STD_LOGIC ;
                f   : OUT     STD_LOGIC ) ;
END func3 ;
ARCHITECTURE LogicFunc OF func3 IS
BEGIN
      f <= (NOT x1 AND NOT x2 AND NOT x3) OR
           (NOT x1 AND x2 AND NOT x3) OR
           (x1 AND NOT x2 AND NOT x3) OR
           (x1 AND NOT x2 AND x3) OR
           (x1 AND x2 AND NOT x3) ;
END LogicFunc ;
```


IEEE STD_LOGIC_1164 PACKAGE

```
library IEEE; use IEEE.std_logic_1164. all ; -- to use the IEEE package
package Part_STD_LOGIC_1164 is
type STD_ULOGIC is
( '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);
type STD_ULOGIC_VECTOR is array (NATURAL range <>) of STD_ULOGIC;
function resolved (s : STD_ULOGIC_VECTOR) return STD_ULOGIC;
subtype STD_LOGIC is resolved STD_ULOGIC;
type STD_LOGIC_VECTOR is array (NATURAL range <>) of STD_LOGIC;
subtype X01   is resolved STD_ULOGIC range 'X' to '1';
subtype X01Z  is resolved STD_ULOGIC range 'X' to 'Z';
subtype UX01  is resolved STD_ULOGIC range 'U' to '1';
subtype UX01Z is resolved STD_ULOGIC range 'U' to 'Z';

-- Vectorized overloaded logical operators:
function "and" (L : STD_ULOGIC; R : STD_ULOGIC) return UX01;
-- Logical operators not, and, nand, or, nor, xor, xnor (VHDL-93),
-- overloaded for STD_ULOGIC STD_ULOGIC_VECTOR
STD_LOGIC_VECTOR.
-- Strength strippers and type conversion functions:
-- function To_T (X : F) return T;
-- defined for types, T and F, where
-- F=BIT BIT_VECTOR STD_ULOGIC STD_ULOGIC_VECTOR
STD_LOGIC_VECTOR
-- T=types F plus types X01 X01Z UX01 (but not type UX01Z)
-- Exclude _'s in T in name: TO_STDULOGIC not TO_STD_ULOGIC
-- To_X01 : L->0, H->1 others->X
-- To_X01Z: Z->Z, others as To_X01
-- To_UX01: U->U, others as To_X01
-- Edge detection functions:
function rising_edge  ( signal s: STD_ULOGIC) return BOOLEAN;
function falling_edge ( signal s: STD_ULOGIC) return BOOLEAN;
-- Unknown detection (returns true if s = U, X, Z, W):
-- function Is_X (s : T) return BOOLEAN;
-- defined for T = STD_ULOGIC STD_ULOGIC_VECTOR
STD_LOGIC_VECTOR.
end Part_STD_LOGIC_1164;
```

Page 401
ASICs by M. Smith
© 1997 A-W-L, Inc.
Used by permission.

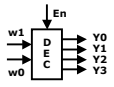
MUX USING IF-THEN-ELSE IN VHDL

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
ENTITY mux2to1 IS
      PORT (w0, w1, s : IN
            STD_LOGIC ;
            f       : OUT     STD_LOGIC ) ;
END mux2to1 ;
ARCHITECTURE Behavior OF mux2to1 IS
BEGIN
      PROCESS ( w0, w1, s )
      BEGIN
       IF s = '0' THEN
            f <= w0 ;
       ELSE
            f <= w1 ;
       END IF ;
      END PROCESS ;
END Behavior ;
```


```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
ENTITY mux2to1 IS
      PORT (w0, w1, s : IN
            STD_LOGIC ;
            f       : OUT     STD_LOGIC ) ;
END mux2to1 ;
ARCHITECTURE Behavior OF mux2to1 IS
BEGIN
      PROCESS ( w0, w1, s )
      BEGIN
       f <= w0 ;
       IF s = '1' THEN
            f <= w1 ;
       END IF ;
      END PROCESS ;
END Behavior ;
```


DEC2TO4 USING CASE IN VHDL

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
ENTITY dec2to4 IS
PORT
(w: IN
   STD_LOGIC_VECTOR(1 DOWNTO 0);
En: IN
   STD_LOGIC;
y: OUT
   STD_LOGIC_VECTOR(0 TO 3) ) ;
END dec2to4 ;
```




```
ARCHITECTURE Behavior OF dec2to4 IS
BEGIN
   PROCESS ( w, En )
   BEGIN
    IF En = '1' THEN
         CASE w IS
          WHEN "00" =>  y <= "1000" ;
          WHEN "01" =>  y <= "0100" ;
          WHEN "10" =>  y <= "0010" ;
          WHEN OTHERS =>  y <= "0001";
         END CASE ;
        ELSE      y <= "0000" ;
    END IF ;
   END PROCESS ;
END Behavior ;
```


BCD-TO-7SEG CONVERTER USING VHDL

(a) Code converter (b) 7-segment display

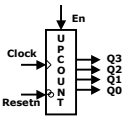
w_3	w_2	w_1	w_0	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1

(c) Truth table

```
ARCHITECTURE Behavior OF seg7 IS
BEGIN
      PROCESS ( bcd )
      BEGIN
       CASE bcd IS              --          abcdefg
            WHEN "0000"         => leds     <= "1111110" ;
            WHEN "0001"         => leds     <= "0110000" ;
            WHEN "0010"         => leds     <= "1101101" ;
            WHEN "0011"         => leds     <= "1111001" ;
            WHEN "0100"         => leds     <= "0110011" ;
            WHEN "0101"         => leds     <= "1011011" ;
            WHEN "0110"         => leds     <= "1011111" ;
            WHEN "0111"         => leds     <= "1110000" ;
            WHEN "1000"         => leds     <= "1111111" ;
            WHEN "1001"         => leds     <= "1110011" ;
            WHEN OTHERS         => leds     <= "-------" ;
       END CASE ;
      END PROCESS ;
END Behavior ;
```

BINARY UP-COUNTER USING VHDL

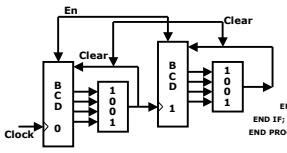
```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE ieee.std_logic_unsigned.all ;

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                                   RISING EDGE
         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 ;
```

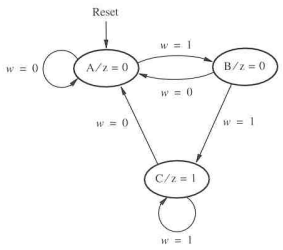

TWO-DIGIT BCD COUNTER

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE ieee.std_logic_unsigned.all ;
ENTITY BCDcount IS
PORT (Clock : IN STD_LOGIC ;
      Clear, E : IN STD_LOGIC ;
      BCD1, BCD0       : BUFFER
        STD_LOGIC_VECTOR(3 DOWNTO 0) ) ;
END BCDcount ;
```




```
ARCHITECTURE Behavior OF BCDcount IS
BEGIN
        PROCESS ( Clock )
        BEGIN
         IF Clock'EVENT AND Clock = '1' THEN
                IF Clear = '1' THEN
                        BCD1 <= "0000" ; BCD0 <= "0000" ;
                ELSIF E = '1' THEN
                        IF BCD0 = "1001" THEN
                                BCD0 <= "0000" ;
                                IF BCD1 = "1001" THEN
                                    BCD1 <= "0000";
                                ELSE
                                    BCD1 <= BCD1 + '1' ;
                                END IF ;
                        ELSE
                                BCD0 <= BCD0 + '1' ;
                        END IF ;
                END IF ;
         END IF;
        END PROCESS;

END Behavior ;
```


STATE DIAGRAM AND STATE TABLE WITH BINARY ASSIGNMENT



Present state	Next state $w=0$	Next state $w=1$	Output z
A	A	B	0
B	A	C	0
C	A	C	1

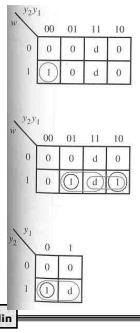
	Present state y_2y_1	Next state $w=0$ Y_2Y_1	Next state $w=1$ Y_2Y_1	Output z
A	00	00	01	0
B	01	00	10	0
C	10	00	10	1
	11	*dd*	*dd*	*d*

LOGIC MINIMIZATION USING K-MAPS

$Y_1 = W\,Y1'\,Y2'$

$Y_2 = WY1 \text{ OR } WY2$
$= W\,(Y1 \text{ OR } Y2)$

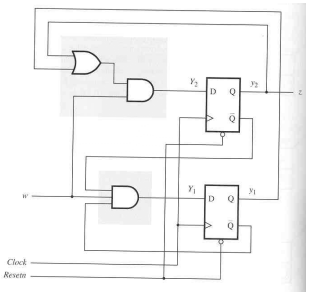
$Z = Y2$



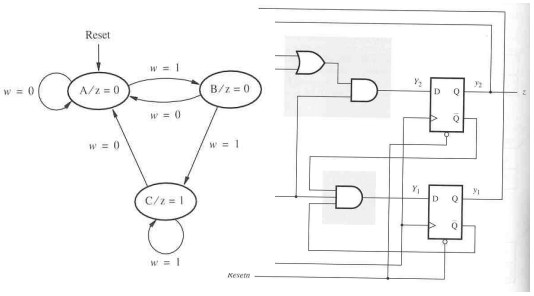
LOGIC EQUATIONS AND SCHEMATIC

$Y_2 = W\,(Y1 \text{ OR } Y2)$

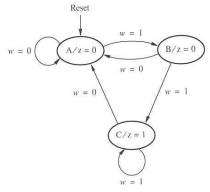
$Y_1 = W\,Y1'\,Y2'$



STATE DIAGRAM AND FINAL SCHEMATIC

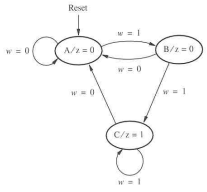


STATE DIAGRAM USING VHDL (part 1)



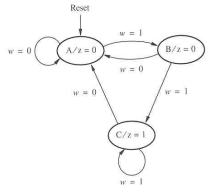

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY simple IS
        PORT (Clock, Resetn, w          : IN
        STD_LOGIC ;
                z                       : OUTSTD_LOGIC ) ;
END simple ;

ARCHITECTURE Behavior OF simple IS
        TYPE State_type IS (A, B, C) ;
        SIGNAL y_present, y_next : State_type ;
```


STATE DIAGRAM USING VHDL (part 2)




```
BEGIN
        PROCESS ( w, y_present )
        BEGIN
         CASE y_present IS
                WHEN A =>
                        IF w = '0' THEN  y_next <= A ;
                        ELSE  y_next <= B ;
                        END IF ;
                WHEN B =>
                        IF w = '0' THEN  y_next <= A ;
                        ELSE  y_next <= C ;
                        END IF ;
                WHEN C =>
                        IF w = '0' THEN  y_next <= A ;
                        ELSE  y_next <= C ;
                        END IF ;
         END CASE ;
        END PROCESS ;
END Behavior ;
```

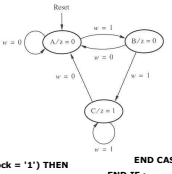

STATE DIAGRAM USING VHDL (part 3)




```
        PROCESS (Clock, Resetn)
        BEGIN
         IF Resetn = '0' THEN
              y_present <= A ;
         ELSIF (Clock'EVENT AND Clock = '1') THEN
              y_present <= y_next ;
         END IF ;
        END PROCESS ;

        z <= '1' WHEN y_present = C ELSE '0' ;
END Behavior ;
```

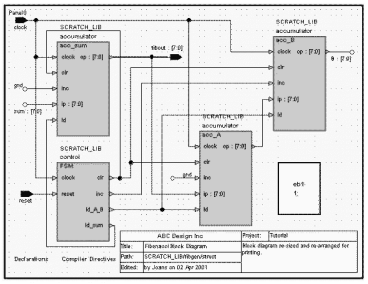

STATE DIAGRAM USING ONE VHDL PROCESS

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
ENTITY simple IS
        PORT (Clock, Resetn, w : IN
                z                       : OU
END simple ;
ARCHITECTURE Behavior OF simple I
        TYPE State_type IS (A, B, C)
        SIGNAL y : State_type ;
BEGIN
        PROCESS ( Resetn, Clock )
        BEGIN
         IF Resetn = '0' THEN
              y <= A ;
         ELSIF (Clock'EVENT AND Clock = '1') THEN
                CASE y IS
                        WHEN A =>
                                IF w = '0' THEN  y <= A ;
                                ELSE  y <= B ;
                                END IF ;
                        WHEN B =>
                                IF w = '0' THEN  y <= A ;
                                ELSE  y <= C ;
                                END IF ;
                        WHEN C =>
                                IF w = '0' THEN  y <= A ;
                                ELSE  y <= C ;
                                END IF ;
                END CASE ;
         END IF ;
        END PROCESS ;
        z <= '1' WHEN y = C ELSE '0' ;
                                END Behavior ;
```

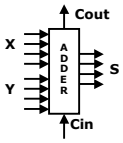


GRAPHICS CAN SPECIFY STRUCTURE

Components and their interconnections



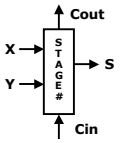
VHDL CAN SPECIFY STRUCTURE (part 1)




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

ENTITY adder IS
      PORT (    Cin     : IN     STD_LOGIC ;
           X, Y  : IN     STD_LOGIC_VECTOR(3 DOWNTO 0) ;
           S             : OUT    STD_LOGIC_VECTOR(3 DOWNTO 0) ;
           Cout  : OUT    STD_LOGIC ) ;
END adder ;
```

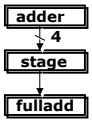
VHDL CAN SPECIFY STRUCTURE (part 2)

```
ARCHITECTURE Structure OF adder IS
    SIGNAL C : STD_LOGIC_VECTOR(1 TO 3) ;
    COMPONENT fulladd
     PORT (    Cin, x, y  : IN     STD_LOGIC ;
               s, Cout    : OUT    STD_LOGIC) ;
    END COMPONENT ;
BEGIN
    stage0: fulladd PORT MAP ( Cin , X(0), Y(0), S(0), C(1) ) ;
    stage1: fulladd PORT MAP ( C(1), X(1), Y(1), S(1), C(2) ) ;
    stage2: fulladd PORT MAP ( C(2), X(2), Y(2), S(2), C(3) ) ;
    stage3: fulladd PORT MAP (
     x => X(3), y => Y(3), Cin => C(3), s => S(3), Cout => Cout ) ;
END Structure ;
```



VHDL CAN SPECIFY STRUCTURE (part 3)

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
ENTITY fulladd IS
    PORT (  Cin, x, y      : IN    STD_LOGIC ;
        s, Cout      : OUT  STD_LOGIC ) ;
END fulladd ;
ARCHITECTURE LogicFunc OF fulladd IS
BEGIN
    s <= x XOR y XOR Cin ;
    Cout <= (x AND y) OR (x AND Cin) OR (y AND
    Cin) ;
END LogicFunc ;
```



USE HDL AND GRAPHICS WISELY

- **Use HDL:**
  - Most straightforward transcription of control (if-then).
  - Can be technology-independent.
  - Can be optimized and retargeted by synthesis.
  - Can be used to describe structure if needed.
- **Use Graphics:**
  - Most straightforward transcription of structure/flow.
  - Best for visualization and even animation.
  - May be slower to enter/modify.
  - May be more difficult to manage large designs.
- **Capture design using either and then convert to the other.**