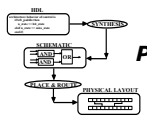
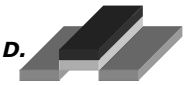
DESIGNING FPGAS & ASICS

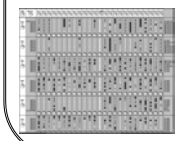


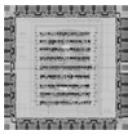
HDL Examples

Prof. Don Bouldin, Ph.D.



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





COURSE OUTLINE

- **Overview of FPGAs and ASICs**
- **Using Synthesis**
- **HDL Examples**
- **Simulation and Testing**
- **Physical Place and Route**
- **Testing ASICs**
- **Component Reuse**

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;
```

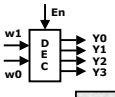

MUX USING IF-THEN-ELSE IN VHDL

```
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 ;
```

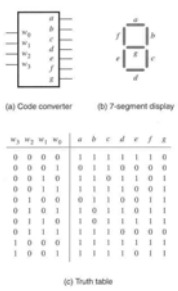
DEC2TO4 USING CASE IN VHDL

```
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 ;
```

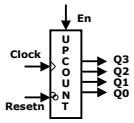
BCD-TO-7SEG CONVERTER USING VHDL




```
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

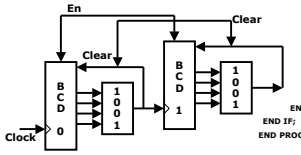
```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE ieee.std_logic_unsigned.all ;

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 ;
```

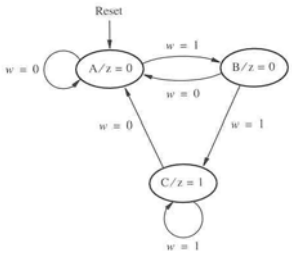

TWO-DIGIT BCD COUNTER

```
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

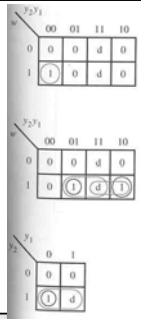
	Present state y_2y_1	Next state $w=0$ Y_2Y_1	Next state $w=1$ Y_2Y_1	Output z
A	00	00	01	0
B	01	00	10	0
C	10	00	10	1
	11	*dd*	*dd*	*d*

LOGIC MINIMIZATION USING K-MAPS

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

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

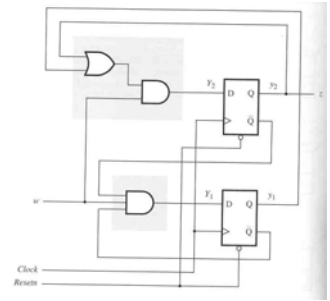
$Z = Y2$



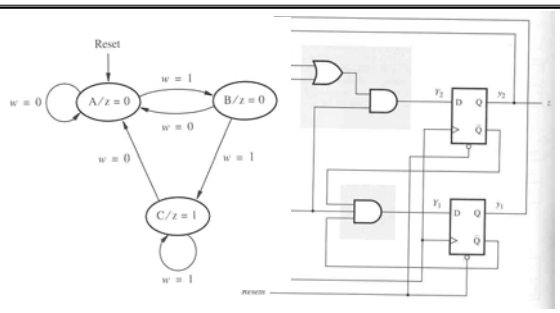
LOGIC EQUATIONS AND SCHEMATIC

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

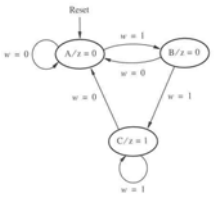
$Y_1 = W\,Y1'\,Y2'$



STATE DIAGRAM AND FINAL SCHEMATIC

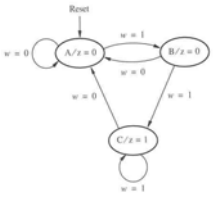


STATE DIAGRAM USING VHDL (part 1)



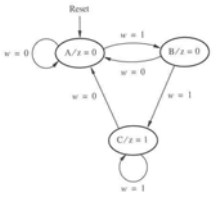

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

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 ;
```

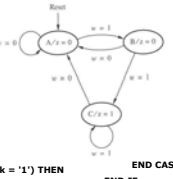

STATE DIAGRAM USING VHDL (part 3)




```
        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

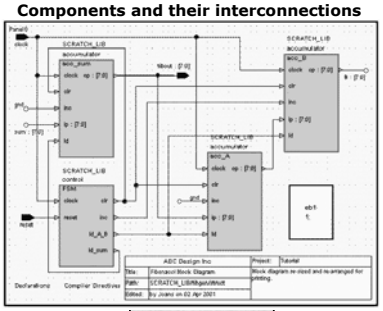
```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
ENTITY simple IS
        PORT (Clock, Resetn, w : IN
              z                : OU
END simple ;
ARCHITECTURE Behavior OF simple IS
        TYPE State_type IS (A, B, C)
        SIGNAL y : State_type ;
BEGIN
        PROCESS ( Resetn, Clock )
     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



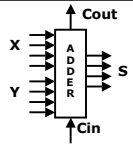

```
          CASE y IS
                WHEN A =>
                        IF w = '0' THEN  y <= A ;
                        ELSE  y <= B ;
                        END IF ;
                WHEN B =>
                        IF w = '0' THEN  y <= A ;
                        ELSE  y <= C ;
                        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

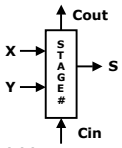


VHDL CAN SPECIFY STRUCTURE (part 1)



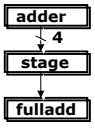

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

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.**