**Dice Game Implementation**

- Why was dice game implemented in three 22V10 PLDs?
- What are the resources needed by the Dice Game?
  - Outputs: 6 for dice values (3 bits each dice), win, loss
  - Inputs: Ra, Rh, Reset, Clk
  - 6 FFs for dice, minimum 3 FFs for FSM, 4 FFs for Point register (13 total)
- 22V10 has
  - 12 input-only pins, 10 input/output pins
  - 10 FFs (1 FF per input/output pin)
- Need at least TWO 22V10s just for FFs

**Partitioning of Design**

- Splitting a design over multiple PLDs is called “partitioning”
- Number of inputs will not be a concern
  - Will be limited by # of FFs, # of outputs
- Could we have used just two PLDs? (see next page)
  - PLD_A: 3 FFs for Cntr, 3 FFs for FSM, 7 outputs (Cntb[2:0], Win, Lose, Ena, Sp). Could even use one-hot encoding for FSM (3 more FFs, three more outputs).
  - PLD_B: 3 FFs for Cntr, 4 FFs for Point Register, 7 outputs (Cnta[2:0], Eq, D7, D11, D2312). But 4 FFs for Cntr consumes 4 outputs so 11 outputs total!!!
  - This division of the logic will not work.
- Some other partition of the logic could possibly be found.
  Three PLDs gives good observation of internal signal values.

**Could we have used 2 PLDs? - NO**

PLD_B needs 11 outputs!
VHDL For Three PLD Solution
• Up to this time, have been using VHDL to specify boolean equations.
• VHDL has high-level statements that allow more natural specification of a problem
• Example: What is the boolean equation for signal “D7”? (=1 when dicesum = 7).
• VHDL Boolean equation:
  D7 <= (not dsum(3)) and dsum(2) and dsum(1) and dsum(0);
• High level VHDL Statement:
  D7 <= '1' when (dsum = "0111") else '0';

“dpathb” Details

“sp” (save point) loads value of dicesum into register.
VHDL for “dpathb”

entity dpathb is
port ( clk, reset: in std_logic;
    diceSum: in std_logic_vector(3 downto 0);
    sp: in std_logic;
    point: out std_logic_vector(3 downto 0);
    eq: out std_logic;
    d7_i: out std_logic;
    d11_i: out std_logic;
    d2312_i: out std_logic);
end dpathb;

architecture a of dpathb is
signal q, d: std_logic_vector(3 downto 0);
begin
    point <= q;  -- point register output
    -- Flip flops for point register
    state_ff: process (clk, reset)
    begin
        if (reset = '1') then
            q <= "0000";
        elsif (clk'event and clk = '1') then
            q <= d;
        end if;
    end process state_ff;
    -- equations for D inputs of point register
    d <= diceSum when (sp = '1') else q;
end architecture a;

Note that ‘D’ is equal to diceSum when sp is asserted else keeps its same state - this is a register!!

Boolean equations vs. High Level
I could have specified the boolean equations for each ‘d’ FF input of the point register as:

- d(0) <= (sp and diceSum(0)) or ((not sp) and (q(0)));
- d(1) <= (sp and diceSum(1)) or ((not sp) and (q(1)));
- d(2) <= (sp and diceSum(2)) or ((not sp) and (q(2)));
- d(3) <= (sp and diceSum(3)) or ((not sp) and (q(3)));

However, it is much easier (and clearer!) to simply write:

d <= diceSum when (sp = '1') else q;
This statement DOES REPRESENT the above boolean equations; it is simply expressed differently.
VHDL for "dpathb" (cont)

--- other equations

\[
eq a \leftarrow '1' \text{ when } \text{dicesum} = q \text{ else } '0';
\]

\[
d7_i \leftarrow '1' \text{ when } \text{dicesum} = "0111" \text{ else } '0';
\]

\[
d11_i \leftarrow '1' \text{ when } \text{dicesum} = "1011" \text{ else } '0';\]

\[
d2312_i \leftarrow '1' \text{ when } (\text{dicesum} = "0010") \text{ or } (\text{dicesum} = "0011") \text{ or } (\text{dicesum} = "1100") \text{ else } '0';
\]

end a;

Again, we could have written boolean equations, but this is clearer.

---

"dpatha" Details

Counts count 1,2,3,4,5,6,1,2, etc

Ena = 1 when Cntb = 6 and Roll = 1

---

VHDL for dpatha

ntity dpatha is

port ( clk, reset: in std_logic;
roll : in std_logic;
dicesum: out std_logic_vector(3 downto 0);
douta: out std_logic_vector(2 downto 0);
doutb: out std_logic_vector(2 downto 0)
);

---
---

### Architecture of dpatha

```vhdl
architecture a of dpatha is
    signal cnta, cntb : std_logic_vector(2 downto 0);
    signal ena : std_logic; -- enable for counter A
    signal sum : std_logic_vector(3 downto 0);
    signal c1,c2,c3 : std_logic; -- carry signals

begin
    -- State Flip Flops
    process (clk, reset)
    begin
        if (reset = '1') then
            cnta <= "001"; -- initialize both counters to '1'
            cntb <= "001";
            end if;
        if (clk 'event and clk = '1') then
            case cntb is
            when "001" => cntb <= "010";
            when "010" => cntb <= "011";
            when "011" => cntb <= "100";
            when "100" => cntb <= "101";
            when "101" => cntb <= "110";
            when "110" => cntb <= "001";
            when others => cntb <= "001";
            end case;
        end if;
    end process stateff;

    -- Sum equations to add cnta + cntb
    -- sum = a XOR b XOR c
    -- cout = (a and b) or (c(a or b))

    -- bit 0, no carry in
    sum(0) <= cnta(0) xor cntb(0);
    c1 <= cnta(0) and cntb(0);

    -- bit 1, c1 is carry in
    sum(1) <= cnta(1) xor cntb(1) xor c1;
    c2 <= (cnta(1) and cntb(1)) or (c1 and (cnta(1) or cntb(1)));

    -- bit 2, c2 is carry in
    sum(2) <= cnta(2) xor cntb(2) xor c2;
    c3 <= (cnta(2) and cntb(2)) or (c2 and (cnta(2) or cntb(2)));

    -- bit 3 is carry3 since no counter bits
    sum(3) <= c3;
end a;
```

---

### Dpatha adder equations.

We could have simply written:

```
sum <= cnta + cntb;
```

This is a valid operation in VHDL!

---

### Dice Game

**ASM Chart**

![Dice Game ASM Chart](image-url)
VHDL for control

entity control is
port ( clk reset: in std_logic;
    d7_i: in std_logic;
    d2312_i: in std_logic;
    ra: in std_logic;
    rb: in std_logic;
    eq: in std_logic;
    sp: out std_logic;
    win: out std_logic;
    lose: out std_logic;
    q0: out std_logic;
    q1: out std_logic;
    q4: out std_logic;
    q5: out std_logic
);
end control;

end entity control;

architecture a of control is
begin
-- FFs for Finite State Machine
signal q: std_logic_vector(5 downto 0);
-- State Flip Flops
state: process(clk,reset)
begin
if (reset = '1') then
    q <= "000001";
elsif (clk event and clk='1') then
    q <= d;
end if;
end process state;

-- FF equations
d(0) <= q(0) and (not rb);
d(1) <= q(1) and ra and d7_i or d11_i;
d(2) <= q(2) or (q(1) and ra and (not d7_i) and (not d11_i) or d2312_i or (q(5) and ra and eq));
d(3) <= q(3) or (q(1) and ra and (not d7_i) and (not d11_i) or d2312_i or (q(5) and ra and eq) or (q(4) and (not rb)));
d(4) <= (q(1) and ra and (not d7_i) and (not d11_i) or d2312_i or (q(5) and ra and eq) or (q(4) and (not rb)));
d(5) <= (q(4) and rb) or (q(5) and not ra);
end arch a;

Is there an easier way to do VHDL for a FSM?

- There is an easier way to write the VHDL for the finite state machine code
- Will use a "case" statement for specifying the FSM action
  - Will generate the same boolean equations
  - Will be more readable
- Will also use symbolic names for states (S0, S1, etc)
  - Can change state encoding very easily.
architecture a of control_all is

-- FFs for Finite State Machine

signal q, d : std_logic_vector5 downto 0);
constant S0 : std_logic_vector5 downto 0) := "00000";
constant S1 : std_logic_vector5 downto 0) := "00010";
constant S2 : std_logic_vector5 downto 0) := "00000";
constant S3 : std_logic_vector5 downto 0) := "00000";
constant S4 : std_logic_vector5 downto 0) := "00000";
constant S5 : std_logic_vector5 downto 0) := "00000"

begin

-- State Flip Flops
stateff : process (clk,reset)
begin
if (reset = '1') then
q <= S0;
elsif (clk'event and clk='1') then
q <= d;
end if;
end process stateff;

Will define symbolic names for the states.
To use new state encoding, only have to change definition of symbolic names!
This uses one-hot encoding.

-- q is present state, d is next state.
logic: process (q, ra, rb, d7_i, d11_i, d2312_i)
begin

-- defaults
win <= '0'; lose <= '0';
d <= q; -- default is to stay in same state.
case q is
when 0 =>
when S0 => if (rb = '1') then d <= S1; end if;
when S1 =>
if (ra = '1') then
if (d7_i = '1' or d11_i = '1') then
  d <= S3;
else
  if (d2312_i = '1') then
    d <= S2;
  else
    d <= S4;
  end if;
else
  if (eq = '1') then
    d <= S3;
  else
    d <= S2;
  end if;
else
  if (d7_i = '1') then
    d <= S3;
  else
    d <= S2;
  end if;
end if;
end case;
end process logic;

Symboolic names changed to define a binary counting order encoding for States! No other changes necessary to code!!!
Summary

• High level VHDL can let you describe digital systems easier and faster. These descriptions are more understandable to an external reader.

• Still MUST KNOW implications of a high level VHDL statement -- ie. What gates get generated?
  - Sum <= Cnta + Cntb. Easy to write, but what kind of adder gets synthesized? There are many different ways to build an adder, and each one has a different tradeoff in terms of speed and gate count!

• Take EE 4743/EE 6743 to find out more about Digital System design!