VHDL

ELEC 418

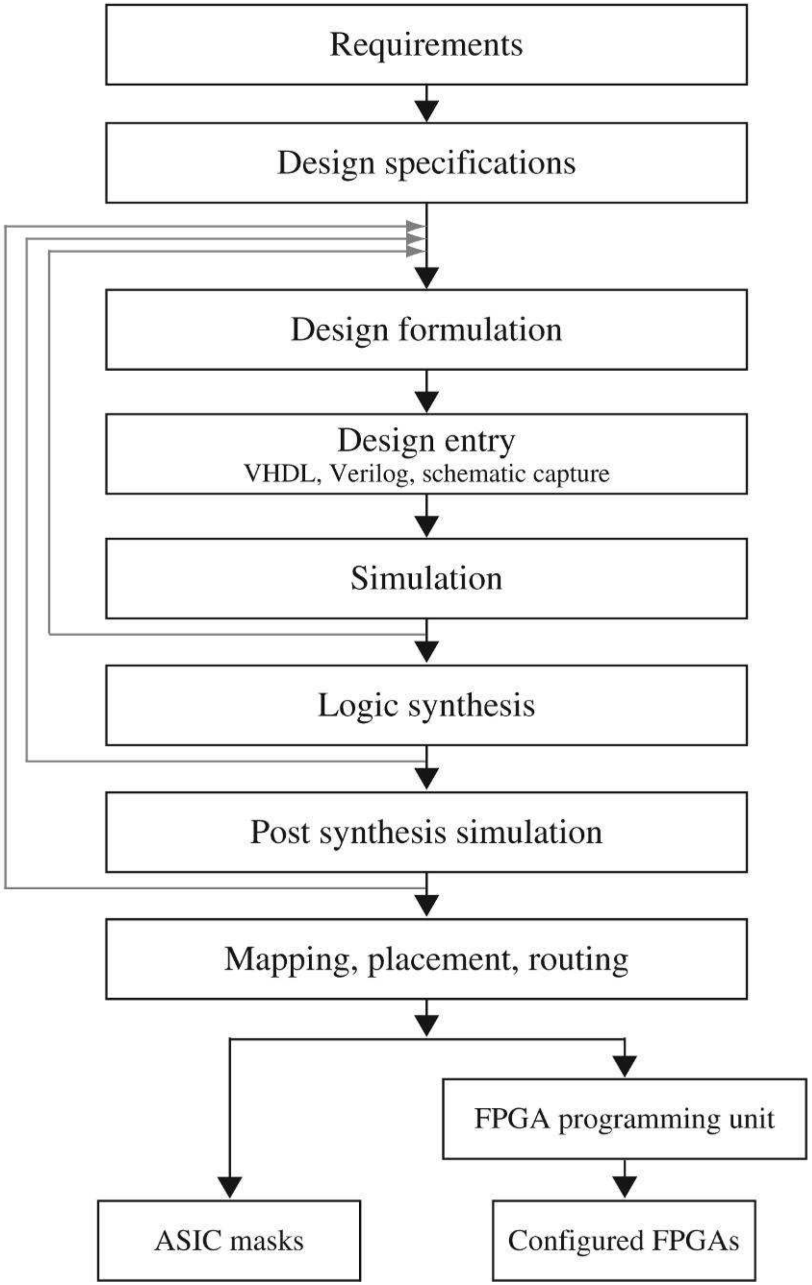
Advanced Digital Systems

Dr. Ron Hayne

Images Courtesy of Thomson Engineering

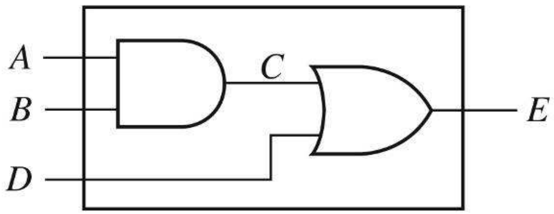
THE CITADEL
DEPARTMENT OF ELECTRICAL & COMPUTER ENGINEERING

Design Flow



Requirements
Design specifications
Design formulation
Design entry
VHDL, Verilog, schematic capture
Simulation
Logic synthesis
Post synthesis simulation
Mapping, placement, routing
FPGA programming unit
ASIC masks
Configured FPGAs

VHDL Modules




```
entity two_gates is
  port(A, B, D: in bit; E: out bit);
end two_gates;

architecture gates of two_gates is
signal C: bit;
begin
  C <= A and B; -- concurrent
  E <= C or D;  -- statements
end gates;
```

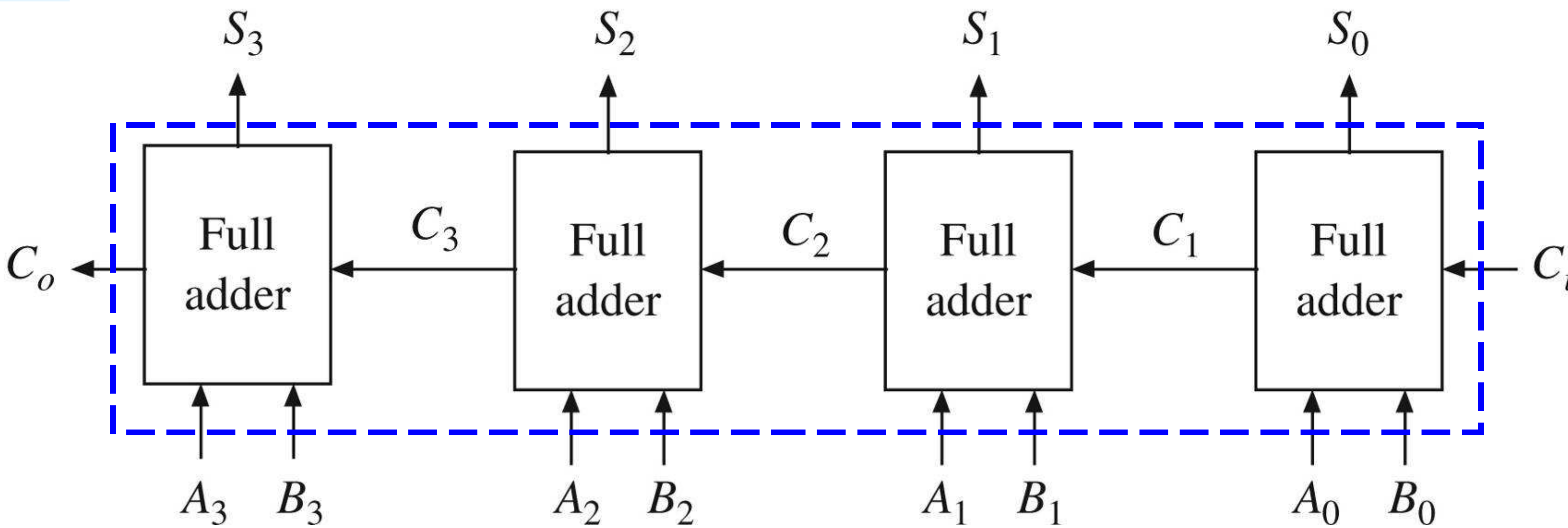
VHDL Libraries

- **`library IEEE;`**
- **`use IEEE.std_logic_1164.all;`**
  - std_logic
    - Single-bit signals
  - std_logic_vector
    - Multi-bit signals

Full Adder (Dataflow)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity FullAdder is
  port(X, Y, Cin: in std_logic;      --Inputs
       Cout, Sum: out std_logic);    --Outputs
end FullAdder;

architecture Dataflow of FullAdder is
begin -- concurrent assignment statements
  Sum  <= X xor Y xor Cin after 2 ns;
  Cout <= (X and Y) or (X and Cin) or (Y and Cin)
          after 2 ns;
end Dataflow;
```

4-bit Ripple-Carry Adder




```
entity Adder4 is
  port(A, B: in std_logic_vector(3 downto 0);
       Ci: in std_logic;
       S: out std_logic_vector(3 downto 0);
       Co: out std_logic);
end Adder4;
```

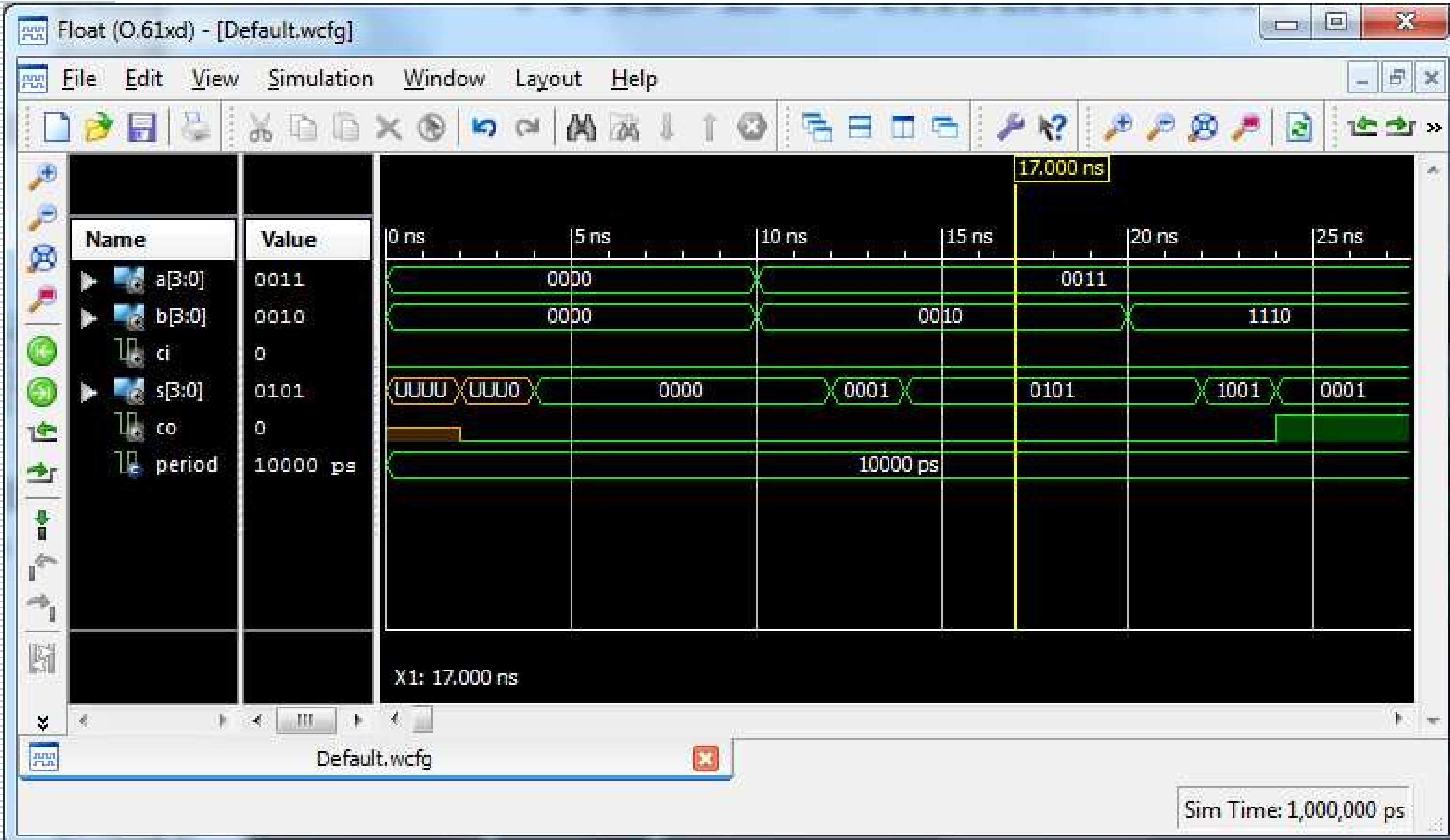
4-bit Adder (Structural)

```
architecture Structure of Adder4 is
  component FullAdder
    port(X, Y, Cin: in std_logic;            -- Inputs
         Cout, Sum: out std_logic);          -- Outputs
  end component;
  signal C: std_logic_vector(3 downto 1); -- internal
begin     --instantiate four copies of the FullAdder
  FA0: FullAdder port map(A(0),B(0),Ci,C(1),S(0));
  FA1: FullAdder port map(A(1),B(1),C(1),C(2),S(1));
  FA2: FullAdder port map(A(2),B(2),C(2),C(3),S(2));
  FA3: FullAdder port map(A(3),B(3),C(3),Co,S(3));
end Structure;
```

VHDL Test Bench

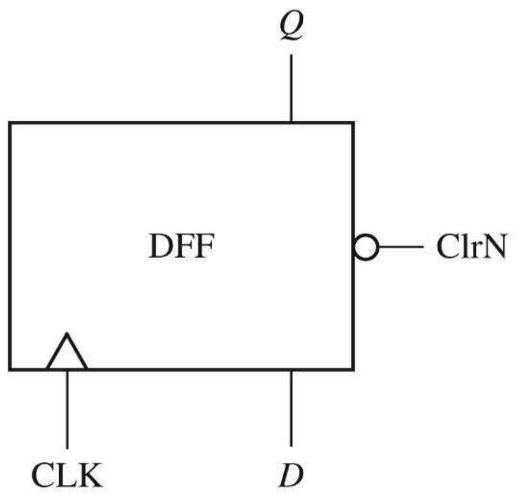
```
  constant PERIOD: time := 10 ns;
BEGIN
stim_proc: process
  begin
    wait for PERIOD;
      A <= "0011";
      B <= "0010";
      Ci <= '0';
    wait for PERIOD;
      B <= "1110";
    wait;
  end process;
END;
```

VHDL Simulation



VHDL Processes (Behavioral)

- **D Flip-Flop with Asyncronous Clear**




```
process(CLK, ClrN)
begin
  if CLRn = '0' then Q <= '0';
  else if CLK'event and CLK = '1'
    then Q <= D;
  end if;
  end if;
end process;
```

VHDL Data Types

- **Bit**
  - '0' or '1'
- **Bit_Vector**
  - "00", "01", "10", ...
- **Boolean**
  - FALSE or TRUE
- **Time**
  - integer with units
  - fs, ps, ns, us, ms, ...
- **Integer**
- **Real**
- **Character**
  - 'a', 'b', '1', '2', ...
- **Enumeration Type**
  - User defined

IEEE 1164 Standard Logic

- **9-Valued Logic System**
  - '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

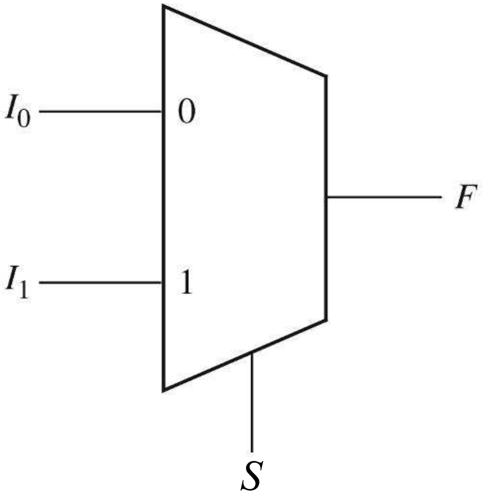
VHDL Operators

- **Logical**
  - and, or, nand, nor, xor
- **Relational**
  - =, /=, <, <=, >, >=
- **Shift**
  - sll, srl, sla, sra, rol, ror
- **Addition**
  - +, -
- **Concatenation**
  - &
- **Unary Sign**
  - +, -
- **Multiplication**
  - *, /, mod, rem
- **Miscellaneous**
  - not, abs, **

VHDL Synthesis Example

```
entity Q3 is
  port(A, B, F, CLK: in std_logic;
       G: out std_logic);
end Q3;
architecture circuit of Q3 is
signal C: std_logic;
begin
  process(CLK)
  begin
    if rising_edge(CLK) then
      C <= A and B; -- statement 1
      G <= C or F; -- statement 2
    end if;
  end process;
end circuit;
```

Multiplexers




```
entity MUX2to1 is
  port(I1, I0, S: in std_logic;
       F: out std_logic);
end MUX2to1;

architecture Dataflow of MUX2to1 is
begin
  F <= I0 when S = '0' else I1;
end Dataflow;
```

Multiplexers

```
entity MUX4to1 is
  port(I: in std_logic_vector(3 downto 0);
       S: in std_logic_vector(1 downto 0);
       F: out std_logic);
end MUX4to1;
architecture Dataflow of MUX4to1 is
begin
  with S select
    F <= I(0) when "00",
         I(1) when "01",
         I(2) when "10",
         I(3) when "11";
end Dataflow;
```

VHDL Libraries

- `library IEEE;`
- `use IEEE.std_logic_1164.all;`
  - Types std_logic and std_logic_vector
- `use IEEE.std_logic_unsigned.all;`
  - Overloaded operators
  - Conversion functions

4-bit Adder (Overload)

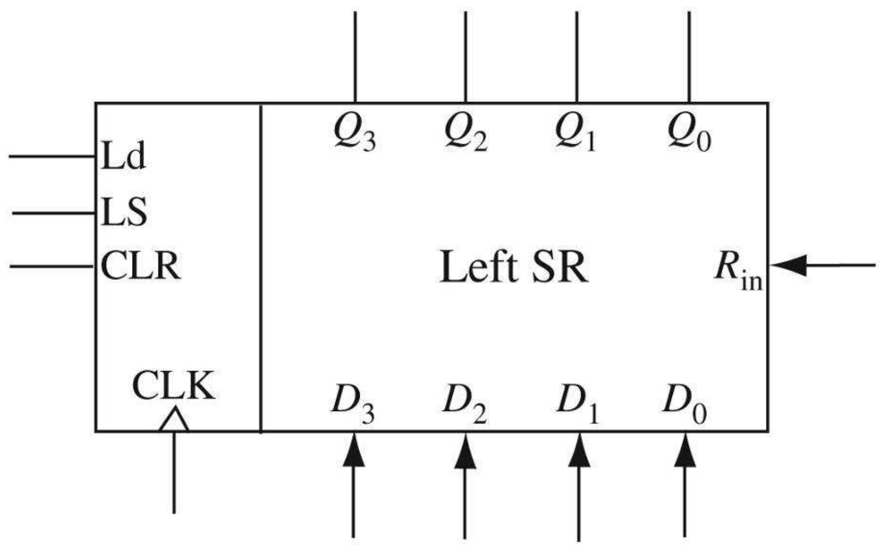
```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

entity Adder4 is
  port(A, B: in std_logic_vector(3 downto 0);
       Ci: in std_logic;
       S: out std_logic_vector(3 downto 0);
       Co: out std_logic);
end Adder4;
```

4-bit Adder (Overload)

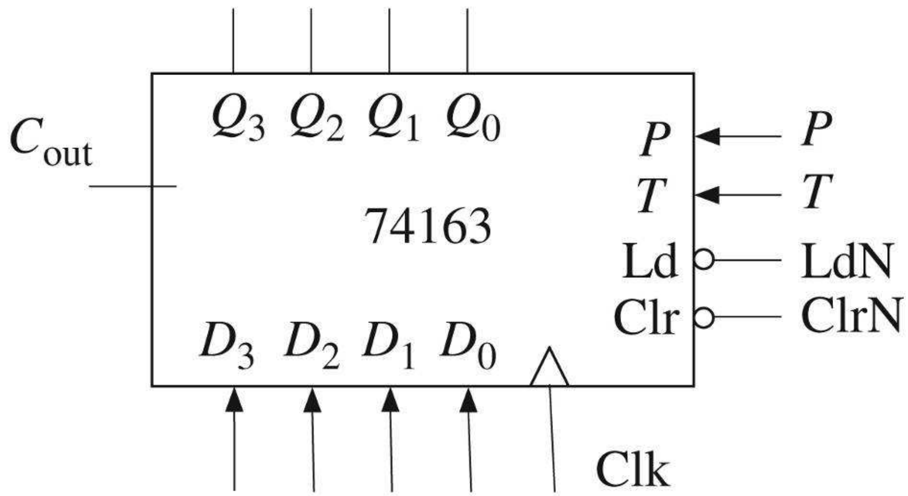
```
architecture Overload of Adder4 is
  signal Sum5: std_logic_vector(4 downto 0);
begin
  Sum5 <= ('0' & A) + ('0' & B) + ("0000" & Ci);
  S <= Sum5(3 downto 0);
  Co <= Sum5(4);
end Overload;
```

Shift Register

- **Left Shift Register**
  - Synchronous Clear and Load



74163 Binary Counter



Control Signals			Next State				
ClrN	LdN	PT	Q_3^+	Q_2^+	Q_1^+	Q_0^+	
0	X	X	0	0	0	0	(clear)
1	0	X	D_3	D_2	D_1	D_0	(parallel load)
1	1	0	Q_3	Q_2	Q_1	Q_0	(no change)
1	1	1	present state + 1				(increment count)

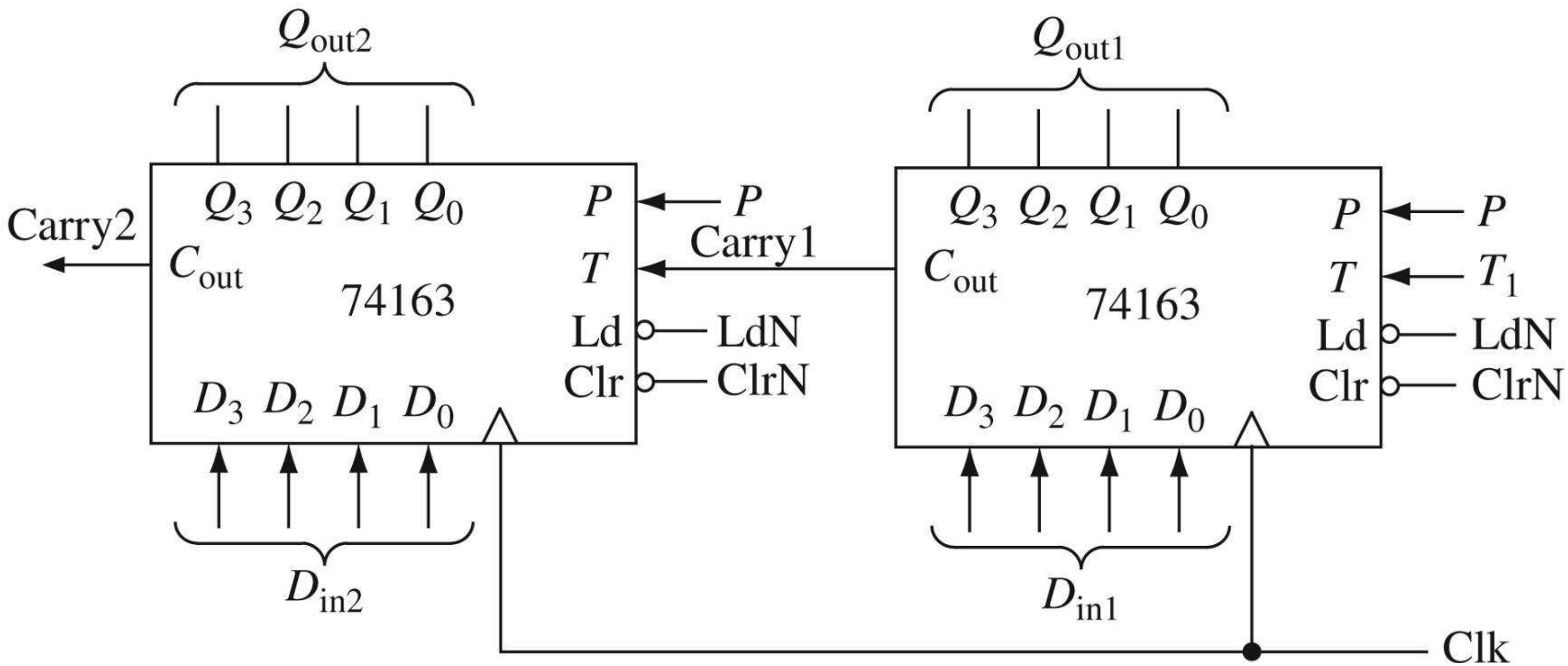
VHDL Counter (Behavioral)

```
-- 74163 FULLY SYNCHRONOUS COUNTER

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

entity Counter74163 is
  port(LdN, ClrN, P, T, Clk: in std_logic;
       D: in std_logic_vector(3 downto 0);
       Cout: out std_logic;
       Qout: out std_logic_vector(3 downto 0));
end Counter74163;
```

VHDL Counter (Behavioral)

```
architecture Behave of Counter74163 is
  signal Q: std_logic_vector(3 downto 0) := "0000";
begin
process(Clk)
  begin
    if rising_edge(Clk) then
      if ClrN = '0' then Q <= "0000";
      elsif LdN = '0' then Q <= D;
      elsif (P and T) = '1' then Q <= Q + 1;
      end if;
    end if;
  end process;
  Qout <= Q;
  Cout <= '1' when Q = "1111" and T = '1' else '0';
end Behave;
```

8-bit Counter



VHDL Counter (Structural)

```
library IEEE;
use IEEE.std_logic_1164.all;

entity Eight_Bit_Counter is
  port(ClrN, LdN, P, T1, Clk: in std_logic;
       Din1, Din2: in std_logic_vector(3 downto 0);
       Qout: out std_logic_vector(7 downto 0);
       Carry2: out std_logic);
end Eight_Bit_Counter;
```

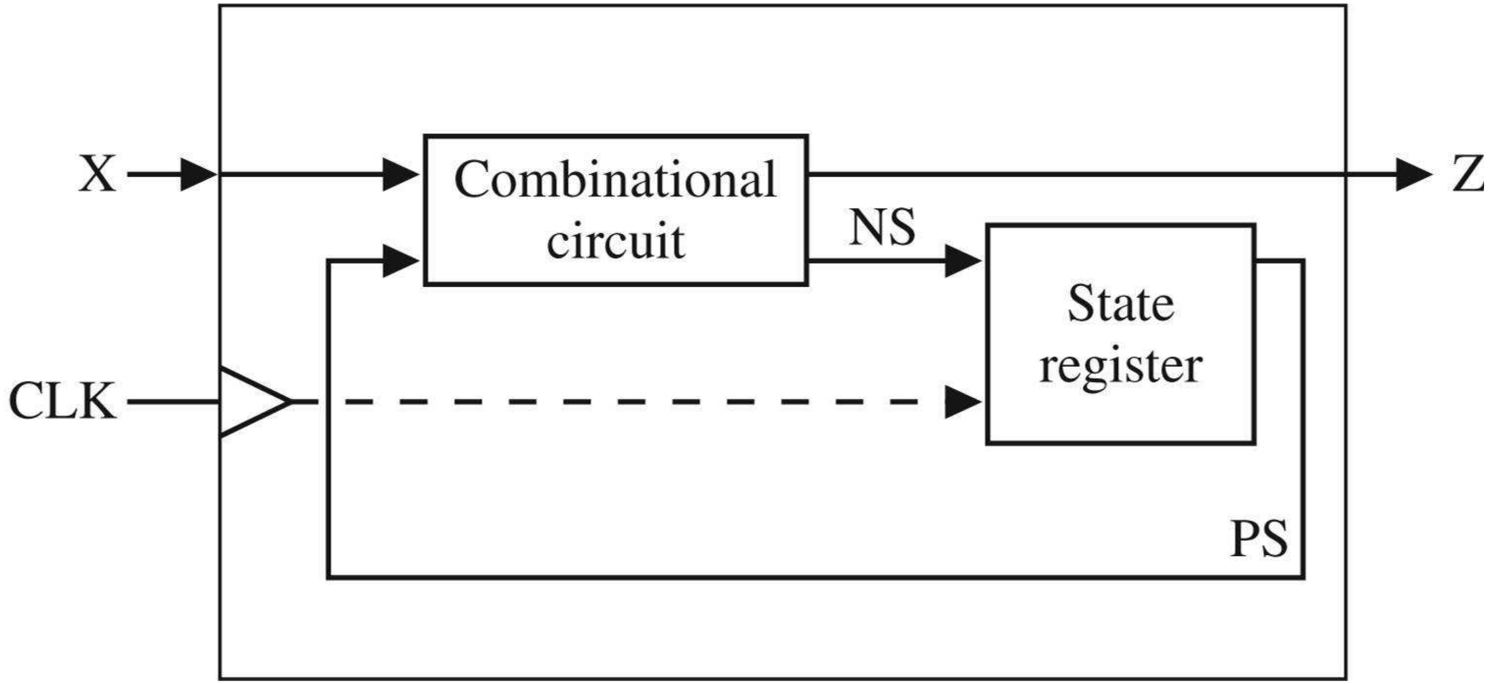
VHDL Counter (Structural)

```
architecture Structure of Eight_Bit_Counter is
  component Counter74163 is
    port(LdN, ClrN, P, T, Clk: in std_logic;
         D: in std_logic_vector(3 downto 0);
         Cout: out std_logic;
         Qout: out std_logic_vector(3 downto 0));
  end component;

  signal Carry1:  std_logic;
  signal Qout1, Qout2:  std_logic_vector(3 downto 0);
```

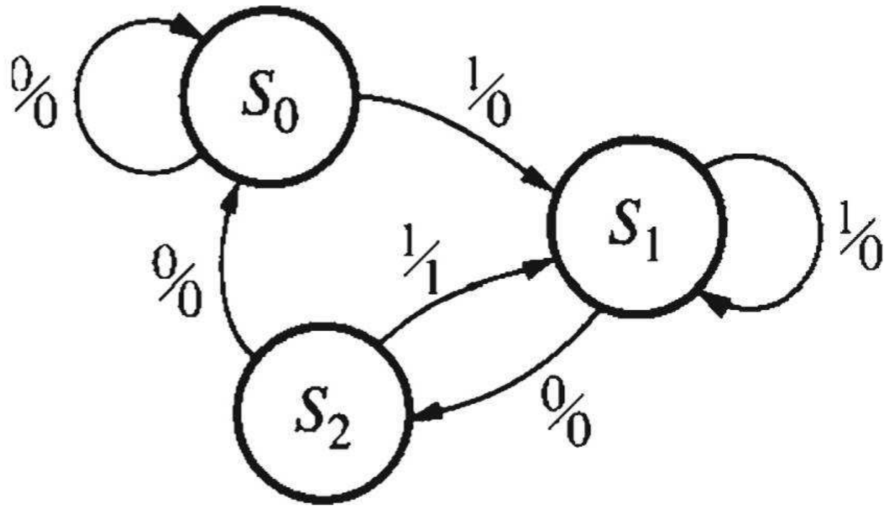
VHDL Counter (Structural)

```
begin

  ct1: Counter74163 port map(LdN,ClrN,P,T1,Clk,
                             Din1,Carry1,Qout1);

  ct2: Counter74163 port map(LdN,ClrN,P,Carry1,Clk,
                             Din2, Carry2, Qout2);

  Qout <= Qout2 & Qout1;

end Structure;
```

Sequential Machine



Sequential Machine



Present State	Next State $X = 0$	Next State $X = 1$	Present Output $X = 0$	Present Output $X = 1$
S_0	S_0	S_1	0	0
S_1	S_2	S_1	0	0
S_2	S_0	S_1	0	1

Behavioral Model

```
entity Sequence_Detector is
  port(X, CLK: in std_logic;
       Z: out std_logic);
end Sequence_Detector;

architecture Behave of Sequence_Detector is
signal State: integer range 0 to 2 := 0;
begin
  process(CLK)
  begin
    if rising_edge(Clk) then
```

Behavioral Model

```
case State is
  when 0 =>
    if X = '0' then
      State <= 0;
    else
      State <= 1;
    end if;
  when 1 =>
    if X = '0' then
      State <= 2;
    else
      State <= 1;
    end if;
```

Behavioral Model

```
        when 2 =>
          if X = '0' then
            State <= 0;
          else
            State <= 1;
          end if;
      end case;
    end if;
  end process;
  Z <= '1' when (State = 2 and X = '1')
       else '0';
end Behave;
```

Additional VHDL

- **Variables, Signals and Constants**
- **Arrays**
- **Loops**
- **Assert and Report Statements**

Look-Up Tables

Input (LUT Address)				Output (LUT Data)				
A	B	C	D	P	Q	R	S	T
0	0	0	0	0	0	0	0	1
0	0	0	1	0	0	0	1	0
0	0	1	0	0	0	1	0	0
0	0	1	1	0	0	1	1	1
0	1	0	0	0	1	0	0	0
0	1	0	1	0	1	0	1	1
0	1	1	0	0	1	1	0	1
0	1	1	1	0	1	1	1	0
1	0	0	0	1	0	0	0	0
1	0	0	1	1	0	0	1	1
1	0	1	0	1	0	1	0	1
1	0	1	1	1	0	1	1	0
1	1	0	0	1	1	0	0	1
1	1	0	1	1	1	0	1	0
1	1	1	0	1	1	1	0	0
1	1	1	1	1	1	1	1	1

Look-Up Tables

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

entity Parity_Gen is
    Port(X: in std_logic_vector(3 downto 0);
         Y: out std_logic_vector(4 downto 0));
end Parity_Gen;
```

Look-Up Tables

```
architecture Table of Parity_Gen is
  type OutTable is array(0 to 15) of std_logic;
  signal ParityBit: std_logic;
  constant OT: OutTable :=
    ('1','0','0','1','0','1','1','0',
     '0','1','1','0','1','0','0','1');
begin
  ParityBit <= OT(conv_integer(X));
  Y <= X & ParityBit;
end Table;
```

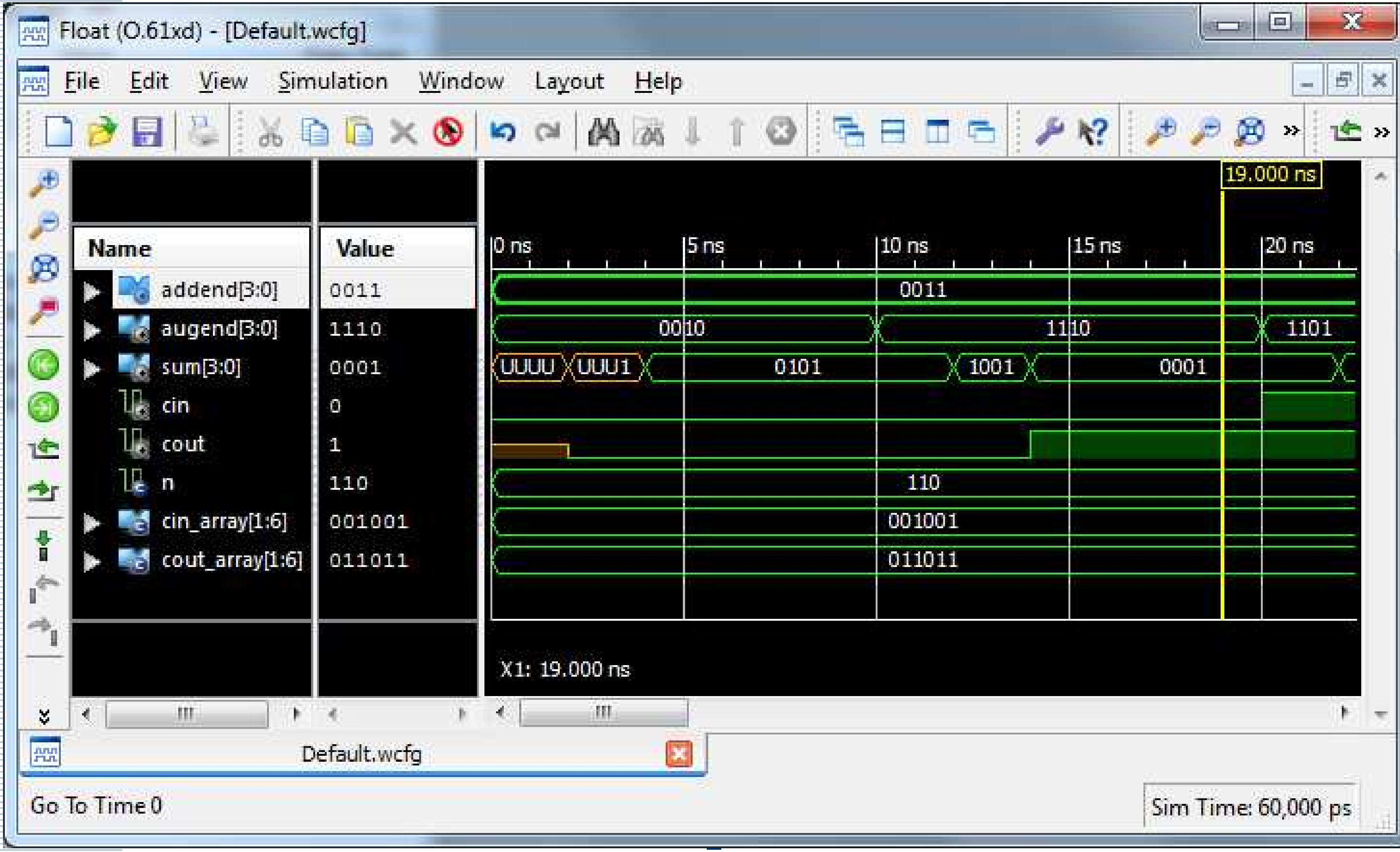
4-bit Adder Test Bench

```
constant PERIOD: time := 10 ns;

-- Test vector arrays
constant N: integer := 4;
type arr1 is array(1 to N) of std_logic;
type arr2 is array(1 to N) of std_logic_vector(3 downto 0);
constant A_array: arr2:= (  "0011", "0011", "0011", "1101");
constant B_array: arr2 := ( "0010", "1110", "1101", "0010");
constant Ci_array: arr1 := (   '0',    '0',    '1',    '0');
constant Co_array: arr1:= ('0',    '1',    '1',    '0'    );
constant S_array: arr2 := ( "0101", "0001", "0001", "1111");
```

4-bit Adder Test Bench

```
-- Stimulus process
stim_proc: process
begin
  for i in 1 to N loop
    A <= A_array(i);
    B <= B_array(i);
    Ci <= Ci_array(i);
    wait for PERIOD;
    assert (S = S_array(i) and Co = Co_array(i))
      report "Wrong Answer"
      severity error;
  end loop;
  report "Test Finished";
end process;
```

VHDL Simulation



VHDL Summary

- **`Entity`**
- **`Architecture`**
  - Dataflow
  - Structural
  - Behavioral
- **`Data Types and Operators`**
- **`Synthesis`**
  - Libraries
- **`Simulation`**
  - Test Benches