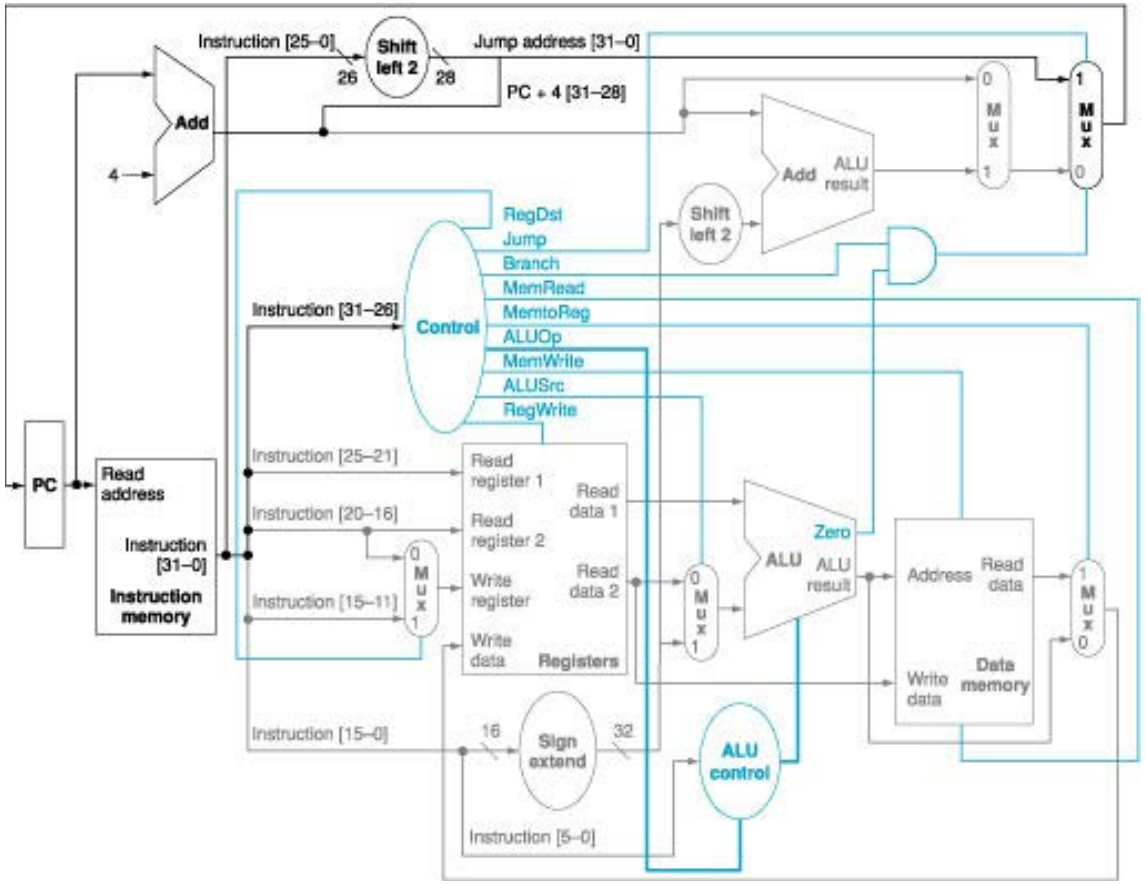
VHDL 4 - Modeling for Synthesis
Register Transfer Level (RTL) Design

References: Roth/John Chapter 2.12-2.14, 4.2,4.8

Register Transfer Language (RTL) Design

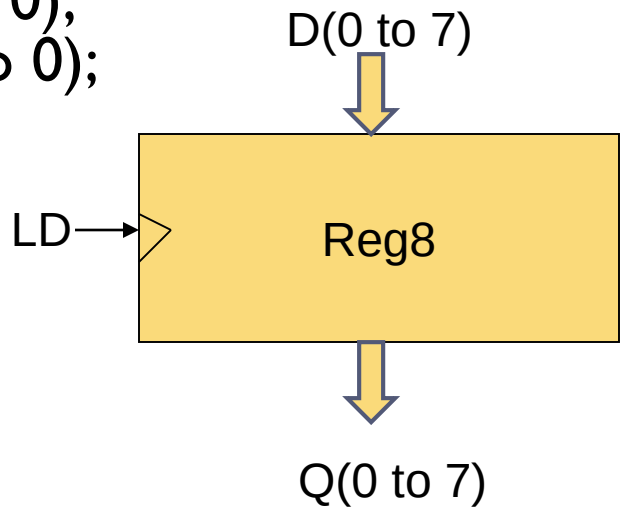
- A **system** is viewed as a structure comprising registers, functions and their control signals
- Show dataflow through the system
- Data = integers, addresses, instruct's
- Functions store and manipulate data



- Not gate/bit level

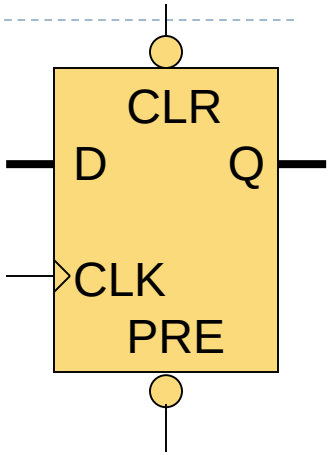
RTL register model

```
-- Model register to hold one datum of some type
-- Individual bits are not manipulated
library ieee;  use ieee.std_logic_1164.all;
entity Reg8 is
  port (D: in std_logic_vector(7 downto 0);
        Q: out std_logic_vector(7 downto 0);
        LD: in std_logic);
end Reg8;
architecture behave of Reg8 is
begin
   process(LD)
   begin
        if (LD'event and LD='1') then
                Q <= D;   -- load data into the register
        end if;
   end process;
end;
```



Asynchronous control inputs

```
library ieee;  use ieee.std_logic_1164.all;
entity Reg8 is
  port (D: in std_logic_vector(7 downto 0);
          CLK,PRE,CLR: in bit;                          --Async PRE/CLR
          Q: out std_logic_vector(7 downto 0));
end Reg8;
architecture behave of Reg8 is
begin
    process(clk,PRE,CLR)
    begin
            if (CLR='0') then                  -- async CLR has precedence
                Q <="00000000";                -- force register to all 0s
            elsif (PRE='0') then               -- async PRE has precedence if CLR='0'
                Q <= (others => '1');          -- force register to all 1s
            elsif rising_edge(clk) then        -- sync operation only if CLR=PRE='1'
                Q <= D;                        -- load D on clock transition
            end if;
    end process;
end;
```



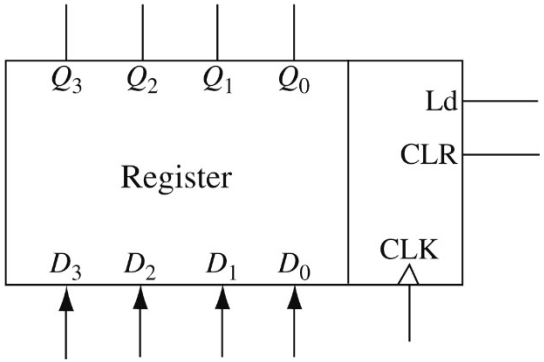
Synchronous reset/set

```
--Reset function triggered by clock edge
process (clk)
begin
      if (clk'event and clk = '1') then
            if reset = '1' then – reset has precedence over load
                  Q <= "00000000" ;
            else
                  Q <= D ;
            end if;
     end if;
end process;
```

Register with clock enable

```
-- “enable” effectively enables/disables clock
process (clk)
begin
      if rising_edge(clk) then  -- detect clock transition
            if enable = ‘1’ then    -- enable load on clock transition
                  Q <= D ;
            end if;
     end if;
end process;
```

Register with synchronous clear and load



Text Figure 2-42

```
process(CLK)
begin
  if CLK'event and CLK = '1' then
    if CLR = '1' then Q <= "0000";   --sync clear
    elsif Ld = '1' then Q <= D;      --sync load
    end if;
  end if;
end process;
```

Left shift register with synchronous clear & load

Text Figure 2-43

```
process(CLK)
begin
  if CLK'event and CLK = '1' then
    if CLR = '1' then Q <= "0000";
      elsif Ld = '1' then Q <= D;
      elsif LS = '1' then Q <= Q(2 downto 0) & Rin;
    end if;
  end if;
end process;
```

shift Q left

Register with parameterized width

```
-- One model of a given function with variable data size
library ieee;  use ieee.std_logic_1164.all;
entity REGN is
   generic (N: integer := 8);                          -- N specified when REG used
   port (    CLK, RST, PRE, CEN: in std_logic;
             DATAIN: in   std_logic_vector (N-1 downto 0);  -- N-bit data in
             DOUT:    out std_logic_vector (N-1 downto 0)   -- N-bit data out
             );
end entity REGN;
architecture RTL of REGN is
begin
process (CLK) begin
   if (CLK'event and CLK = '1') then
        if      (RST = '1') then DOUT <= (others => '0');  --reset to all 0s
        elsif (PRE = '1') then DOUT <= (others => '1');  --preset to all 1s
        elsif (CEN = '1') then DOUT <= DATAIN;           --load data
        end if;
   end if;
end process;
end architecture RTL;
```

Vectors: “100” = (‘1’,’0’,’0’) = (‘1’, others => ‘0’)
Arbitrarily long: “00…0” = (others => ‘0’)

Instantiating the parameterized register

```
library ieee;  use ieee.std_logic_1164.all;
entity TOP is
   port (       CLK,X,Y,A,B,C: in std_logic;
                DIN: in  std_logic_vector(5 downto 0);
                Q1: out std_logic_vector(5 downto 0);
                Q2: out std_logic_vector(4 downto 0);
                Q3: out std_logic_vector(3 downto 0)
                );
end entity TOP;
architecture HIER of TOP is
component REGN is
  generic (N: integer := 8);
  port (       CLK, RST, PRE, CEN: in std_logic;
                DATAIN: in   std_logic_vector (N-1 downto 0);
                DOUT:    out std_logic_vector (N-1 downto 0)
                );
end component REGN;
begin
R1: REGN generic map (6)  port map    --6-bit register
                (CLK, A, B, C, DIN, Q1);
R2: REGN generic map (5)  port map    --5-bit register (low 5 bits of DIN)
                (CLK, Y, X, C, DIN(4 downto 0),Q2);
R3: REGN generic map (4)  port map    --4-bit register (low 4 bits of DIN)
          (CLK=>CLK, RST=>A, PRE=>B, CEN=>C, DATAIN=>DIN(3 downto 0), DOUT=>Q3);
end architecture HIER;
```

Easier to identify signal to port connections.

2-to-1 mux with parameterized data size

```
entity muxN is
    generic (N: integer := 32);  -- data size parameter
    port (  A,B:  in   std_logic_vector(N-1 downto 0);
            Y:    out std_logic_vector(N-1 downto 0);
            Sel:  in   std_logic);
end muxN;
architecture rtl of muxN is
begin
    Y <= A when Sel = '0' else B;    -- A,B,Y same type
end;
-- specify parameter N at instantiation time
M:  muxN generic map (16)
            port map(A=>In1, B=>In2, Y=>Out1);
```

16-bit vectors

Other types of generic parameters

```
entity and02 is
    generic (Tp : time :=  5 ns);    -- gate delay parameter
    port (A,B: in std_logic;
          Y:    out std_logic);
end and02;
architecture eqn of and02 is
begin
    Y <= A and B after Tp;   -- gate with delay Tp
end;
…..
A_tech1:  and02 generic map (2 ns) port map (M,N,P);
A_tech2:  and02 generic map (1 ns) port map (H,K,L);
```

Gates with different delays.

IEEE Std. 1076.3 Synthesis Libraries

- Supports arithmetic models
  - ieee.numeric_std (ieee library package)
    - defines UNSIGNED and SIGNED types as arrays of std_logic

      type SIGNED is array(NATURAL range <>) of STD_LOGIC;

      type UNSIGNED is array(NATURAL range <>) of STD_LOGIC;
    - defines arithmetic/relational operators on these types
  - Supports RTL models of functions

- Lesser-used packages:
  - ieee.numeric_bit
    - same as above except SIGNED/UNSIGNED are arrays of type bit
  - ieee.std_logic_arith *(from Synopsis)*
    - Non-standard predecessor of numeric_std/numeric_bit

NUMERIC_STD package contents

- Arithmetic functions: + - * / rem mod
  - Combinations of operand types for which operators are defined:
    - SIGNED + SIGNED return SIGNED
    - SIGNED + INTEGER return SIGNED
    - INTEGER + SIGNED return SIGNED
    - SIGNED + STD_LOGIC return SIGNED
  - PLUS: above combinations with UNSIGNED and NATURAL

- Other operators for SIGNED/UNSIGNED types:
  - Relational: = /= < > <= >=
  - Shift/rotate: sll, srl, sla, sra, rol, ror
  - Maximum(a,b), Minimum(a,b)

- Convert between types:
  - TO_INTEGER(SIGNED), TO_INTEGER(UNSIGNED)
  - TO_SIGNED(INTEGER,#bits), TO_UNSIGNED(NATURAL,#bits)
  - RESIZE(SIGNED or UNSIGNED,#bits) – changes # bits in the vector

Arithmetic with NUMERIC_STD package

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.NUMERIC_STD.all;
entity Adder4 is
   port ( in1, in2  : in   UNSIGNED(3 downto 0) ;
             mySum : out UNSIGNED(3 downto 0) ) ;
end Adder4;

architecture Behave_B of Adder4 is
begin
    mySum <= in1 + in2; -- overloaded '+‘ operator
end Behave_B;
```

UNSIGNED = UNSIGNED + UNSIGNED

Conversion of "closely-related" types

- STD_LOGIC_VECTOR, SIGNED, UNSIGNED:
  - All arrays of STD_LOGIC elements
  - Example: How would one interpret "1001" ?
    - STD_LOGIC_VECTOR: simple pattern of four bits
    - SIGNED: 4-bit representation of number -7 (2's complement #)
    - UNSIGNED: 4-bit representation of number 9 (unsigned #)
- Vectors of same element types can be "converted" (re-typed/re-cast) from one type to another

```
signal A:  std_logic_vector(3 downto 0) := "1001";
signal B:  signed(3 downto 0);
signal C:  unsigned(3 downto 0);
B <= signed(A);              -- interpret A value "1001" as number -7
C <= unsigned(A);            -- interpret A value "1001" as number 9
A <= std_logic_vector(B);    -- interpret B as bit pattern "1001"
```

Conversion of "closely-related" types

For arrays of same dimension, *having elements of same type*

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.NUMERIC_STD.all;
entity Adder4 is
   port ( in1, in2  : in   STD_LOGIC_VECTOR(3 downto 0) ;
          mySum : out STD_LOGIC_VECTOR(3 downto 0) ) ;
end Adder4;

architecture Behave_B of Adder4 is
begin
    mySum <=
        STD_LOGIC_VECTOR(  SIGNED(in1) + SIGNED(in2)  );
end Behave_B;
```

SIGNED result

Interpret STD_LOGIC_VECTOR as SIGNED

Function: SIGNED = SIGNED + SIGNED

Interpret SIGNED result as STD_LOGIC_VECTOR.

Example – binary counter

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.NUMERIC_STD.all;
ENTITY counter IS
   port( Q:  out std_logic_vector(3 downto 0);
            ….
END counter;

ARCHITECTURE behavior OF counter IS
   signal Qinternal:  unsigned(3 downto 0);
begin

     Qinternal <= Qinternal + 1;          -- UNSIGNED = UNSIGNED + NATURAL
     Q <= std_logic_vector(Qinternal);    -- re-type unsigned as std_logic_vector
```

From NUMERIC_STD package

Using a "variable" to describe sequential behavior within a process

```
-- Assume Din and Dout are std_logic_vector
-- and numeric_std package is included
cnt: process(clk)
        variable count: integer;  -- internal counter state
    begin                          -- valid only within a process
        if clk='1' and clk'event then
          if ld='1' then
                count := to_integer(unsigned(Din));  --update immediately
          elsif cnt='1' then
                count := count + 1;                  --update immediately
          end if;
        end if;
        Dout <= std_logic_vector(to_unsigned(count,32)); --schedule ΔDout
    end process;
```

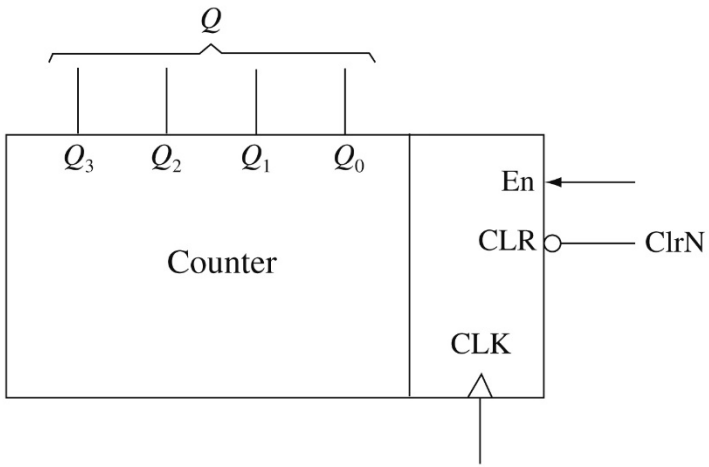
Counting to some max_value (not 2^n)

```
-- full-sized comparator circuit generated to check count = max
process begin
    wait until clk'event and clk='1' ;
    if (count = max_value) then
        count <= 0 ;                --roll over from max_value to 0
    else
        count <= count + 1 ;    --otherwise increment
    end if ;
end process ;
```

Decrementer and comparator

```
-- comparison to 0 easier than a non-zero value (NOR gate)
process begin
    wait until clk'event and clk='1' ;
    if (count = 0) then
        count <= max_value ;  -- roll over from 0 to max_value
    else
        count <= count - 1 ;    -- otherwise decrement
    end if ;
end process ;
```

FIGURE 2-44: VHDL Code for a Simple Synchronous Counter

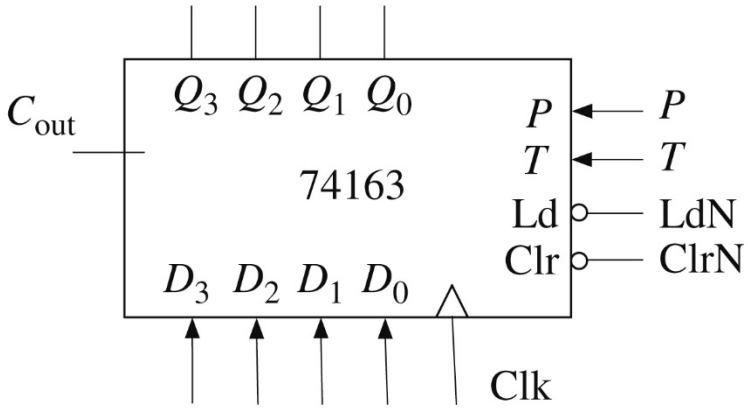


library ieee;
use ieee.numeric_std.all;

```
signal Q: unsigned (3 downto 0);
-----------
process (CLK)
begin
  if CLK'event and CLK = '1' then
    if ClrN = '0' then Q <= "0000";
    elsif En = '1' then Q <= Q + 1;
    end if;
  end if;
end process;
```

unsigned + natural

FIGURE 2-45: 74163 Counter Operation



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	(increment count)
1	1	1	present state + 1				(no change)

FIGURE 2-46: **74163 Counter Model**

```
-- 74163 FULLY SYNCHRONOUS COUNTER

library IEEE;
use IEEE.numeric_bit.all;

entity c74163 is
  port(LdN, ClrN, P, T, Clk: in bit;
       D: in unsigned(3 downto 0);
       Cout: out bit; Qout: out unsigned(3 downto 0));
end c74163;

architecture b74163 of c74163 is
signal Q: unsigned(3 downto 0);   -- Q is the counter register
begin
  Qout <= Q;
  Cout <= Q(3) and Q(2) and Q(1) and Q(0) and T;
  process(Clk)
  begin
    if Clk'event and Clk = '1' then   -- change state on rising edge
      if ClrN = '0' then Q <= "0000";          -- synchronous clear
      elsif LdN = '0' then Q <= D;             -- synchronous load
      elsif (P and T) = '1' then Q <= Q + 1;   -- synchronous count
      end if;
    end if;
  end process;
end b74163;
```

FIGURE 2-47: Two 74163 Counters Cascaded to Form an 8-Bit Counter

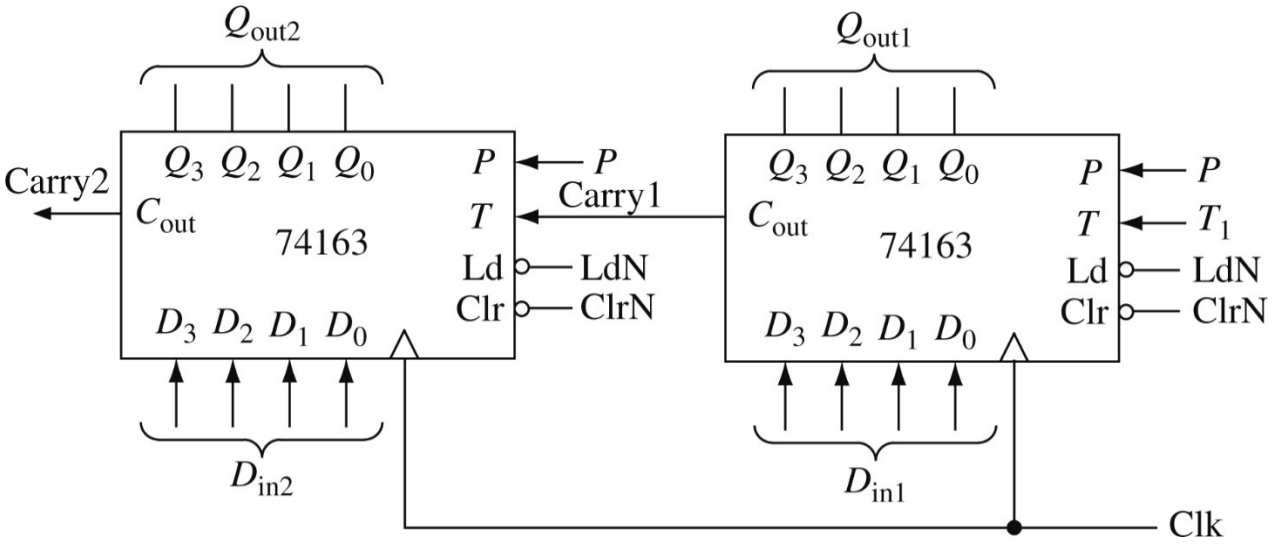


FIGURE 2-48: **VHDL for 8-Bit Counter**

```
--Test module for 74163 counter

library IEEE;
use IEEE.numeric_bit.ALL;

entity eight_bit_counter is
  port(ClrN, LdN, P, T1, Clk: in bit;
       Din1, Din2: in unsigned(3 downto 0);
       Count: out integer range 0 to 255;
       Carry2: out bit);
end eight_bit_counter;

architecture cascaded_counter of eight_bit_counter is
component c74163
  port(LdN, ClrN, P, T, Clk: in bit;
       D: in unsigned(3 downto 0);
       Cout: out bit; Qout: out unsigned(3 downto 0));
end component;

signal Carry1: bit;
signal Qout1, Qout2: unsigned(3 downto 0);
begin
  ct1: c74163 port map (LdN, ClrN, P, T1, Clk, Din1, Carry1, Qout1);
  ct2: c74163 port map (LdN, ClrN, P, Carry1, Clk, Din2, Carry2, Qout2);
  Count <= to_integer(Qout2 & Qout1);
end cascaded_counter;
```

Handling Overflow (Carry)

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.NUMERIC_STD.all;

entity Adder_1 is
   port ( A, B  : in   UNSIGNED(3 downto 0) ;
             C :     out UNSIGNED(4 downto 0) ) ; -- C(4) = carry bit
end Adder_1;

architecture Synthesis_1 of Adder_1  is
begin                                  -- A+B could produce a carry
    C <= ('0' & A) + ('0' & B); -- leading '0' balances # bits
end Behave_B;
```

result bits for "+" is maximum of (#bits in A, #bits in B)
can also use: C <= resize(A,5) + resize(B,5)

FIGURE 2-38: **VHDL Code for 4-Bit Adder Using Unsigned Vectors**

```
library IEEE;
use IEEE.numeric_bit.all;

entity Adder4 is
  port(A, B: in unsigned(3 downto 0); Ci: in bit;  -- Inputs
       S: out unsigned(3 downto 0); Co: out bit);  -- Outputs
end Adder4;

architecture overload of Adder4 is
signal Sum5: unsigned(4 downto 0);
begin
  Sum5 <= '0' & A + B + unsigned'(0=>Ci);
  S <= Sum5(3 downto 0);
  Co <= Sum5(4);
end overload;
```

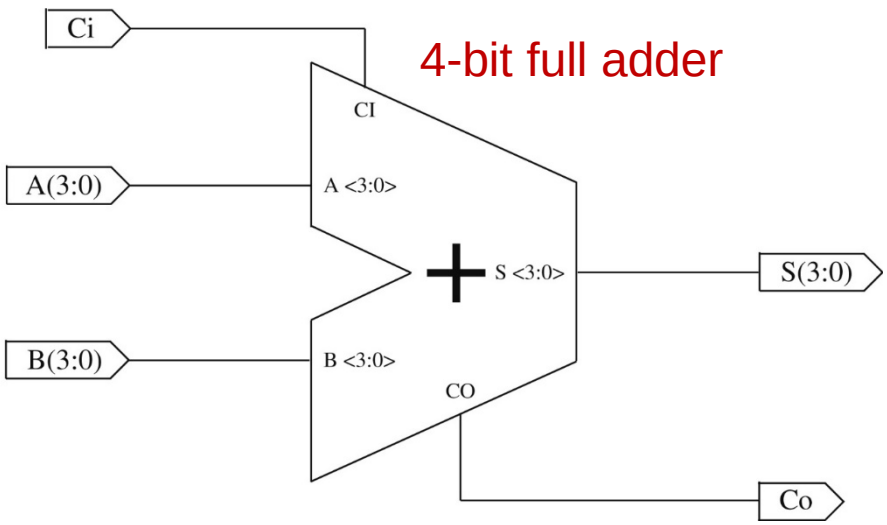
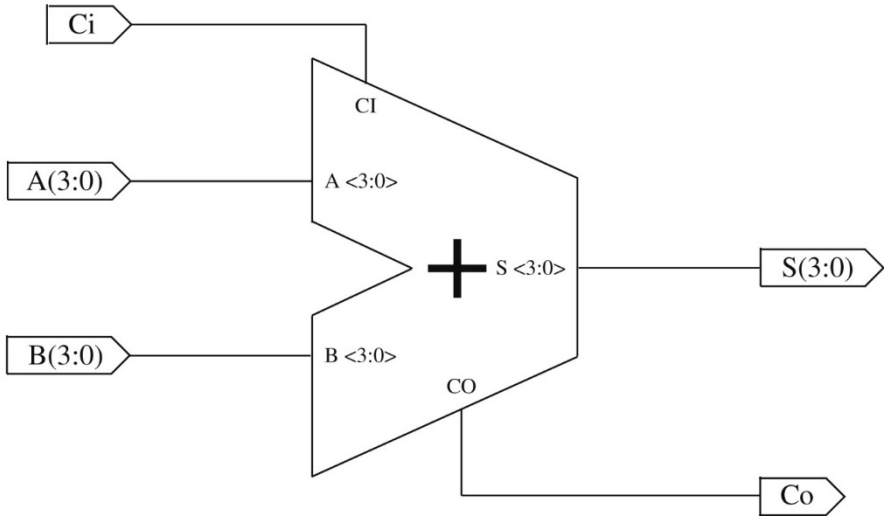


FIGURE 2-40: VHDL Code for 4-Bit Adder Using the std_logic_unsigned Package

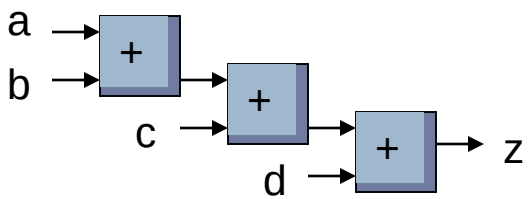
```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
entity Adder4 is
  port(A, B: in std_logic_vector(3 downto 0); Ci: in std_logic; --Inputs
       S: out std_logic_vector(3 downto 0); Co: out std_logic); --Outputs
end Adder4;

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



Multiple adder structures

```
z <= a + b + c + d;
-- 3 adders stacked 3 deep
```




```
z <= (a + b) + (c + d);
-- 3 adders stacked 2 deep
```

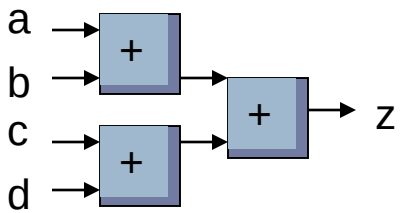
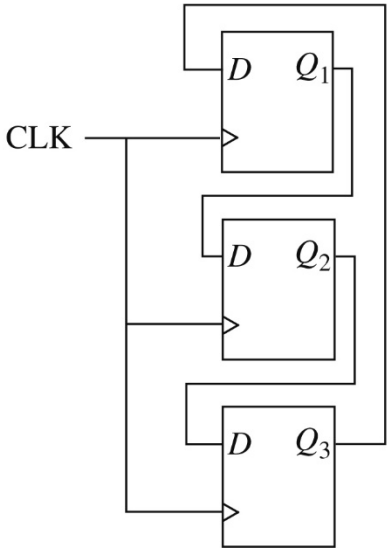


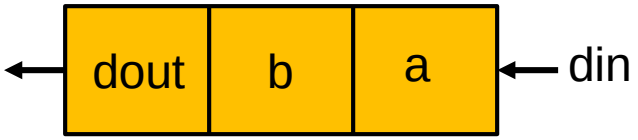
FIGURE 2-41: Cyclic Shift Register




```
process(CLK)
begin
  if CLK'event and CLK = '1' then
    Q1 <= Q3 after 5 ns;
    Q2 <= Q1 after 5 ns;
    Q3 <= Q2 after 5 ns;
  end if;
end process;
```

FFs generated from variables: 3-bit shift register example

```
-- External input/output din/dout
process (clk)
    variable a,b: bit;
begin
      if (clk'event and clk = '1') then
            dout <= b;
            b := a;
            a := din;
     end if;
end process;
-- Note: a,b used before being assigned new values
```



3-bit shift register example
Unexpected resulting structure

```
process (clk)
    variable a,b: bit;
begin
      if (clk'event and clk = '1') then
          a := din;       -- update a value to din
          b := a;         -- update b with new a value (din)
          dout <= b;      -- update dout with new b value (din)
       end if;
end process;
```

a,b updated before used so they effectively become "wires". End result is din -> dout on the clock transition.