RTL Digital Systems with VHDL

Index

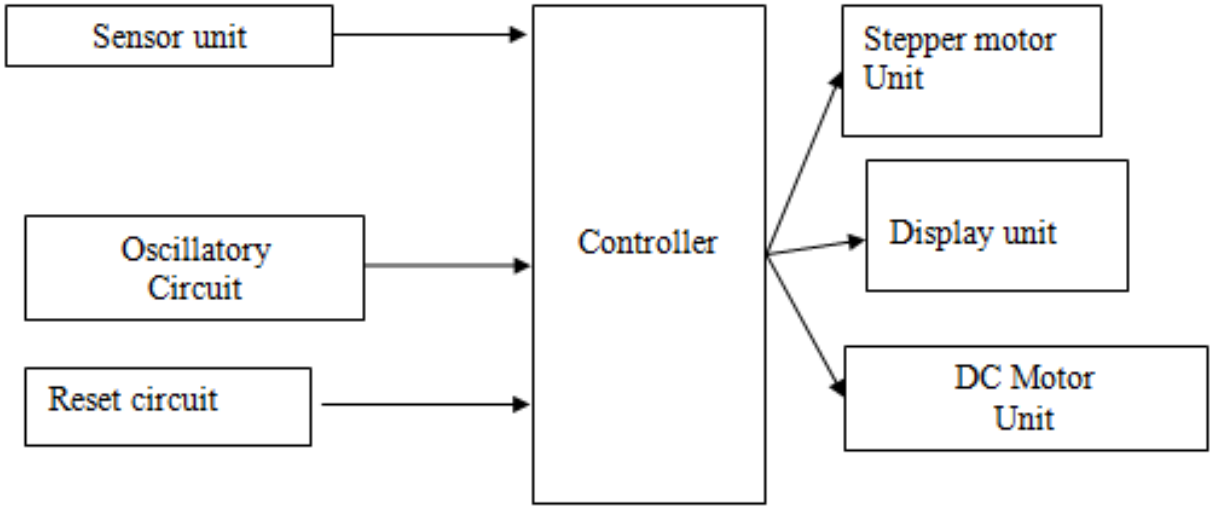
- Abstraction levels in microprocessor-based digital systems
- Introduction to RTL digital systems
  - Combinational Systems. Examples and VHDL description
  - Sequential Systems. Examples and VHDL description

Abstraction levels

Microprocessor-based systems can be described at different abstraction levels. Most commonly:

System level

Controller, sensor, actuator…



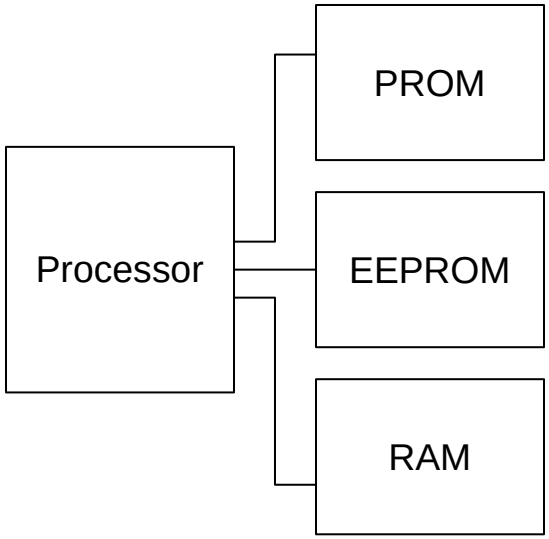
Abstraction levels

Microprocessor-based systems can be described at different abstraction levels. Most commonly:

System level

Chip Level

Processor, main memory, cache…

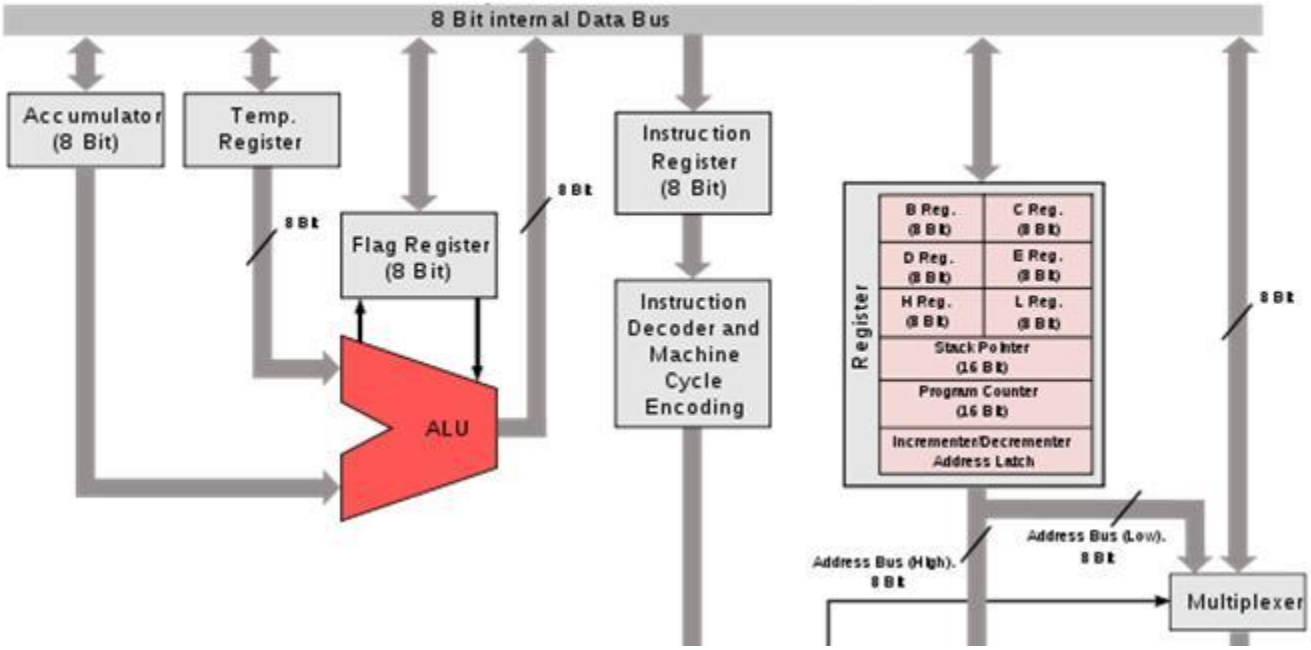


Abstraction levels

Microprocessor-based systems can be described at different abstraction levels. Most commonly:

System level Chip Level Register Transfer Level (RTL)

Register, decoder, multiplexer…



Introduction to RTL digital systems

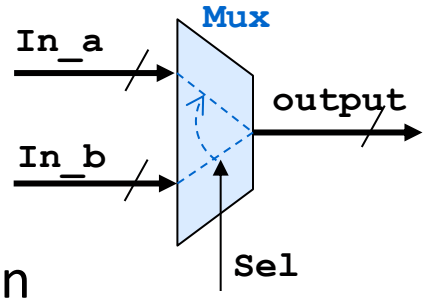
- We will focus on the synthesis of RTL digital systems with VHDL
- It is important to understand and to be capable of designing RTL circuits because they are essential in a large number of basic functions of a microprocessor-based hardware architecture
- RTL systems can be classified into the same categories as digital logic systems
  - Combinational systems. Output is a function of only present inputs
  - Sequential systems. Output is a function of present and past inputs.

    Sequential systems have *memory* and therefore possible *states*

Combinational Systems

- Output is a function of only present inputs
- Output signals change as inputs change, only taking into consideration the delay due to the propagation through the logic gates and wires
- Most common combinational RTL digital systems
  - Multiplexer
  - Encoder
  - Decoder
  - Comparator
  - Tri-state buffer
  - ALU (Arithmetic Logic Unit)

Multiplexer I: 2 inputs



- Function

  output = (sel AND in_a) OR ($\overline{\text{sel}}$ AND in_b)

- Utility: data-path circuit that allows signal selection

- Example of VHDL code

```
--Multiplexer of 2 binary inputs
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

--entity circuit declaration
ENTITY mux2to1 IS
   PORT (in_a:   IN  std_logic;
         in_b:   IN  std_logic;
         sel:    IN  std_logic;
         output: OUT std_logic);
END mux2to1;
```

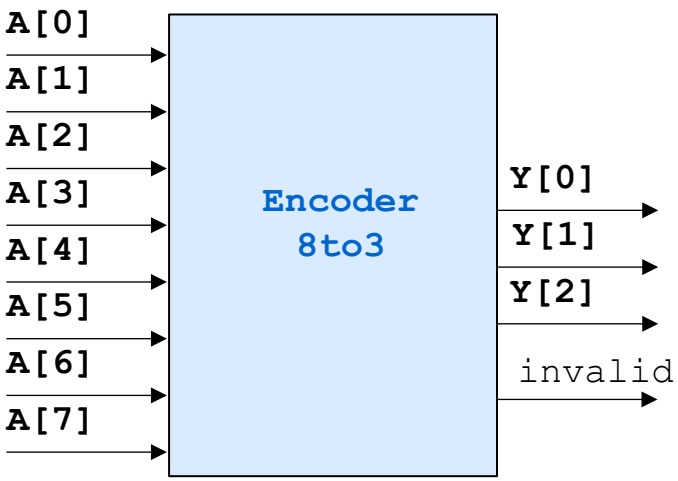
```
ARCHITECTURE Behavioral OF mux2to1 IS
BEGIN
   PROCESS (in_a, in_b, sel)
     BEGIN
        IF (sel='1') THEN
          output<=in_a;
        ELSE
          output<=in_b;
        END IF;
   END PROCESS;

END Behavioral;
```

Multiplexer II: More than 2 inputs

```
--Multiplexer of 3 binary inputs
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
--entity circuit declaration
ENTITY mux3to1 IS
   PORT (in_A:   IN  std_logic;
         in_B:   IN  std_logic;
         in_C:   IN  std_logic;
         SEL:    IN  std_logic_vector(1 downto 0);
         output: OUT std_logic);
END mux3to1;

ARCHITECTURE Behavioral OF mux3to1 IS
BEGIN
   PROCESS (in_A, in_B, in_C, SEL)
     BEGIN
       CASE SEL IS
         WHEN "00"   => output <= in_A;
         WHEN "01"   => output <= in_B;
         WHEN others => output <= in_C;
       END CASE;
   END PROCESS;
END Behavioral;
```

Encoder

- Utility: to encode information (to reduce the number of bits)
- Example of VHDL code




```
-- 8 to 3 encoder
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY enc_8_3 IS
  PORT ( A: IN std_logic_vector(7 DOWNTO 0);
         Y: OUT std_logic_vector(2 DOWNTO 0);
   invalid: OUT std_logic);
END enc_8_3;

ARCHITECTURE Behavioral OF enc_8_3 IS

BEGIN
```

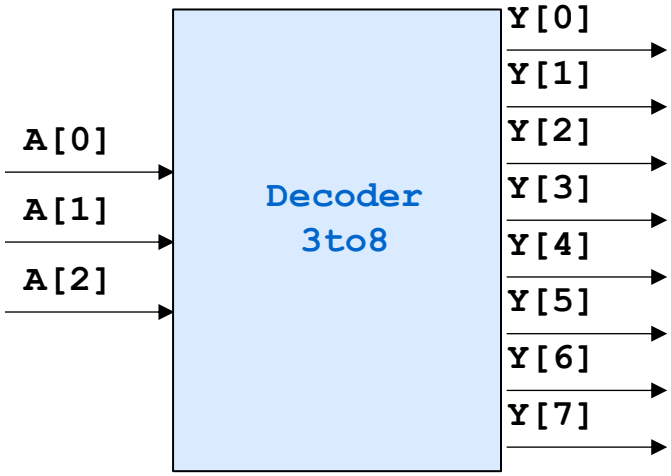
```
PROCESS(A)
 BEGIN
  invalid <= '0';
  CASE A IS
    WHEN "00000001" => Y <= "000";
    WHEN "00000010" => Y <= "001";
    WHEN "00000100" => Y <= "010";
    WHEN "00001000" => Y <= "011";
    WHEN "00010000" => Y <= "100";
    WHEN "00100000" => Y <= "101";
    WHEN "01000000" => Y <= "110";
    WHEN "10000000" => Y <= "111";
    WHEN others     =>
         Y <="000";
         invalid <= '1';
   END CASE;
 END PROCESS;
END Behavioral;
```

Decoder

- Utility: to decode information

  Example. To enable the proper device from the (codified) Address Bus

- Example of VHDL code

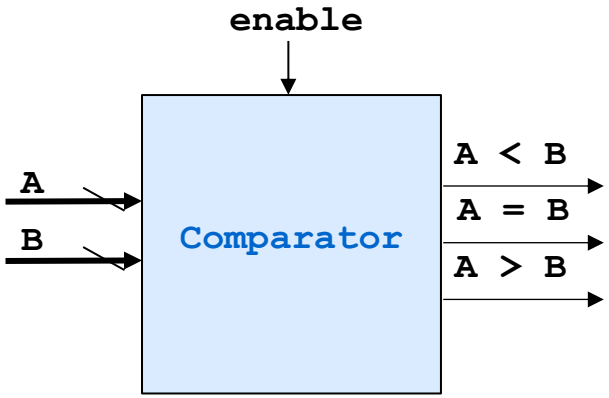



```
-- 3 to 8 decoder
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY dec_3_8 IS
  PORT ( A: IN std_logic_vector(2 DOWNTO 0);
         Y: OUT std_logic_vector(7 DOWNTO 0));
END dec_3_8;

ARCHITECTURE Behavioral OF dec_3_8 IS

BEGIN
```

```
PROCESS(A)
 BEGIN
  CASE A IS
   WHEN "000" => Y <= "00000001";
   WHEN "001" => Y <= "00000010";
   WHEN "010" => Y <= "00000100";
   WHEN "011" => Y <= "00001000";
   WHEN "100" => Y <= "00010000";
   WHEN "101" => Y <= "00100000";
   WHEN "110" => Y <= "01000000";
   WHEN others=> Y <= "10000000";
  END CASE;
 END PROCESS;

END Behavioral;
```

Comparator

- Utility: to compare multiple values
- Example of VHDL code

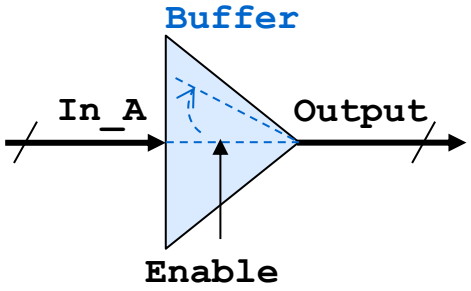



```
-- 2 words comparator
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY comparator IS
  PORT ( A: IN std_logic_vector(7 DOWNTO 0);
         B: IN std_logic_vector(7 DOWNTO 0);
         enable:    IN  std_logic;
         AlessB:    OUT std_logic;
         AequalB:   OUT std_logic;
         AgreaterB: OUT std_logic);
END comparator ;

ARCHITECTURE Behavioral OF comparator IS

BEGIN
```

```
PROCESS(A, B, enable)
BEGIN
  IF (enable = '0') THEN
    AlessB<='0'; AequalB<= '0'; AgreaterB <= '0';
  ELSE
    IF ( A < B ) THEN --A less than B
      AlessB <= '1';
    ELSE
     AlessB <= '0';
    END IF;
    IF ( A > B ) THEN --A greater than B
      AgreaterB <= '1';
    ELSE
      AgreaterB <= '0';
    END IF;
    IF ( A = B ) THEN --A equals B
      AequalB <= '1';
    ELSE
      AequalB <= '0';
    END IF;
  END IF;
END PROCESS;
END Behavioral;
```

Tri-state buffer

- Utility: to connect and disconnect signals. Example. When several signals must access a common line, to prevent short-circuits
- Example of VHDL code

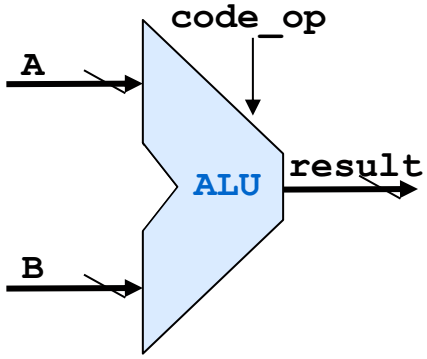



```
-- tri-state binary buffer
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY tristate IS
 PORT(enable: IN std_logic;
      A: IN  std_logic_vector(7 DOWNTO 0);
      Y: OUT std_logic_vector(7 DOWNTO 0));
END tristate;
```

```
ARCHITECTURE Behavioral OF tristate
IS
BEGIN
PROCESS(A,enable)

 BEGIN
   IF (enable='1') THEN
      Y <= A;
   ELSE
      Y <= (others=>'Z');
  END IF;

 END PROCESS;
END Behavioral;
```

ALU (Arithmetic Logic Unit)

- Utility: to perform arithmetic and logic operations
- Example of VHDL code




```
-- ALU (AND, OR, +, -)
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.NUMERIC_STD.ALL;
ENTITY ALU IS
  PORT ( A: IN std_logic_vector(7 DOWNTO 0);
         B: IN std_logic_vector(7 DOWNTO 0);
   code_op: IN std_logic_vector(1 DOWNTO 0);
   result: OUT std_logic_vector(7 DOWNTO 0));
END ALU;

ARCHITECTURE Behavioral OF ALU IS
BEGIN
 PROCESS(A, B, code_op)
 BEGIN
  CASE code_op IS
    -- logic operations
    WHEN "00" => result <= A AND B;
    WHEN "01" => result <= A OR B;
    -- arithmetic operations
    WHEN "10" => result <= std_logic_vector(signed(A) + signed(B));
    WHEN others => result <= std_logic_vector(signed(A) - signed(B));
  END CASE;
 END PROCESS;
END Behavioral;
```

Sequential Systems

Output is a function of present and past inputs.

- Sequential systems have *memory* and therefore possible *states*
- Sequential systems are mostly ruled by the *clock* signal
- Signals can be
  - Synchronous or clock-dependent
  - Asynchronous or clock-independent (typically only the asynchronous reset)

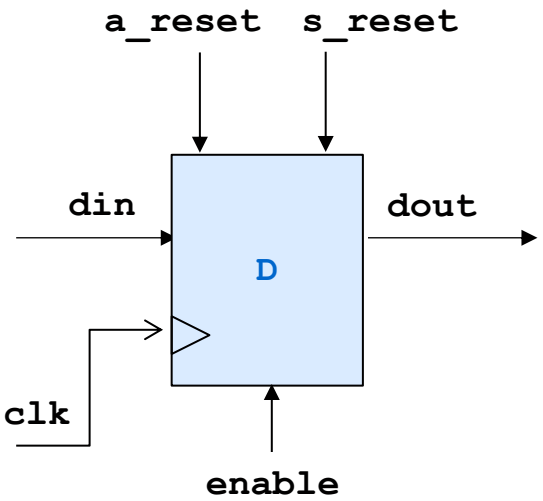
Most common RTL sequential digital systems

- D flip-flop
- Data registers
- Shift registers
- Counters

D flip-flop

- Utility: to store a bit
- Flip-flops have two possible states:

  0 or 1
- Example of VHDL code

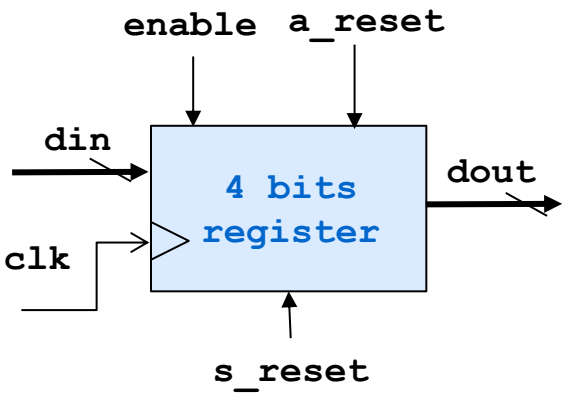



```
-- D-type flip-flop with asynchronous
 -- reset, enable and synchronous reset
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY Dff IS
  PORT ( clk:     IN  std_logic;
         a_reset: IN  std_logic;
         enable:  IN  std_logic;
         s_reset: IN  std_logic;
         din:     IN  std_logic;
         dout:    OUT std_logic);
END Dff;

ARCHITECTURE Behavioral OF Dff IS

BEGIN
```

```
PROCESS(a_reset, clk)

 BEGIN
   IF (a_reset='1') THEN
      dout <= '0';
   ELSIF (clk'EVENT AND clk = '1') THEN
      IF (enable='1') THEN
        IF (s_reset='1') THEN
           dout <= '0';
        ELSE
           dout <= din;
         END IF;
      END IF;
   END IF;
 END PROCESS;
END Behavioral;
```

Data register

- Utility: to store a word
- VHDL code

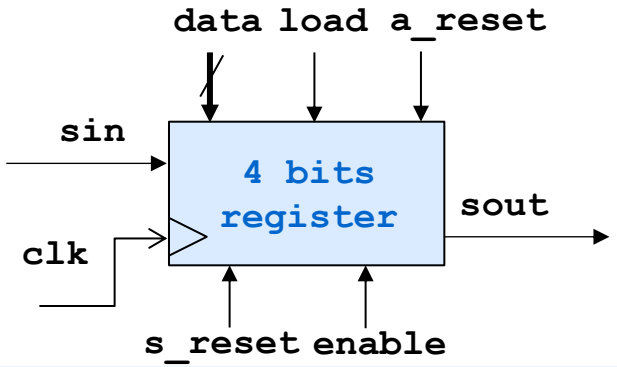



```
-- 4-bits data register, parallel input/output,
asynchronous reset, enable and synchronous reset

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY dataregister IS
  PORT( clk     : IN  std_logic;
        a_reset: IN  std_logic;
        s_reset: IN  std_logic;
        enable : IN  std_logic;
        din    : IN  std_logic_vector(3 DOWNTO 0);
        dout   : OUT std_logic_vector(3 DOWNTO 0));
END dataregister;
```

```
ARCHITECTURE Behavioral OF dataregister IS
BEGIN
PROCESS (a_reset, clk)
  BEGIN
    IF (a_reset='1') THEN
      dout <= (OTHERS=>'0');
    ELSIF (clk'EVENT AND clk='1') THEN
      IF (enable='1') THEN
        IF (s_reset='1') THEN
          dout <= (OTHERS=>'0');
        ELSE
          dout <= din;
        END IF;
      END IF;
    END IF;
  END PROCESS;
END Behavioral;
```

Shift register

- Utility: to store and shift a word
- VHDL code



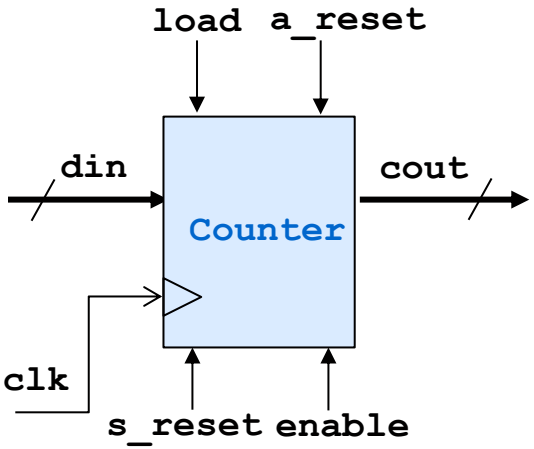

```
-- 4-bits left shift data register, serial
-- input/output, parallel load, asynchronous reset
-- enable and synchronous reset
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY shiftregister IS
  PORT( clk   :  IN  std_logic;
        a_reset:IN  std_logic;
        s_reset:IN  std_logic;
        enable: IN  std_logic;
        load :  IN  std_logic;
        data :  IN  std_logic_vector(3 DOWNTO 0);
        sin  :  IN  std_logic;
        sout :  OUT std_logic);
END shiftregister;

ARCHITECTURE Behavioral OF shiftregister IS
  SIGNAL reg: std_logic_vector(3 DOWNTO 0);

BEGIN
```

```
PROCESS (a_reset, clk)
  BEGIN
    IF (a_reset='1') THEN
      reg <= (OTHERS=>'0');
    ELSIF (clk'EVENT AND clk='1') THEN
     IF (enable='1') THEN
       IF (s_reset='1') THEN
          reg <= (OTHERS=>'0');
        ELSIF (load='1') THEN
           reg <= data;
        ELSE
           -- left shift
           reg <= reg(2 DOWNTO 0) & sin;
        END IF;
      END IF;
    END IF;
  END PROCESS;

 sout <= reg(3); -- serial output

END Behavioral;
```

Counter

- Utility: to count up and/or down

  Example: loops, iterated executions
- VHDL code




```
-- rising counter with asynchronous reset
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.NUMERIC_STD.ALL;


ENTITY counter IS
  PORT (clk:      IN std_logic;
        load:     IN std_logic;
        a_reset:  IN std_logic;
        s_reset:  IN std_logic;
        enable:   IN std_logic;
        din:      IN std_logic_vector(2 downto 0);
        count:   OUT std_logic_vector(2 downto 0));
END counter;

ARCHITECTURE Behavioral OF counter IS
   SIGNAL count_aux: unsigned(2 downto 0);
BEGIN
```

```
PROCESS (clk, a_reset)
BEGIN
   IF (a_reset='1') THEN
      count_aux <= (OTHERS => '0');
   ELSIF (clk='1' AND clk'EVENT) THEN
     IF (enable='1') THEN
       IF (s_reset='1') THEN
          count_aux <= (OTHERS => '0');
       ELSIF (load='1') THEN
          count_aux <= unsigned(din);
       ELSE
          count_aux <= count_aux + 1;
       END IF;
     END IF;
   END IF;
END PROCESS;

count <= std_logic_vector(count_aux);
END Behavioral;
```