Finanziato dall'Unione europea NextGenerationEU
Ministero dell'Università e della Ricerca
Italiadomani PIANO NAZIONALE DI RIPRESA E RESILIENZA
Fondazione ICSC Centro Nazionale di Ricerca in HPC, Big Data and Quantum Computing

VHDL by Examples

Andrea Triossi – University of Padova – INFN Padova

Hardware description

- Drawing a diagram of the hardware design (schematics)
- Textual description
  - Programming language that include explicitly the notion of time
  - Concurrent language
  - Implement the Register Transfer Level (RTL) of a circuit (not dependent on hardware technology but only dataflow between registers and logical operations)
  - Most widely used
    - Verilog
    - VHDL [VHSIC (Very High Speed Integrated Circuit) Hardware Description Language]

VHDL basic structures

- entity: it is a black box where only the interface signals (ports) are described
- architecture: it describes the content of the box in terms of functionalities and/or structures of the circuits

```
entity And2 is
   port (x,y: in BIT; z: out BIT);
end entity And2;
```

```
architecture ex1 of And2 is
begin
 z <= x and y;
end architecture ex1;
```

```
architecture ex2 of And2 is
begin
 z <= '1' when x & y = "11" else
      '0';
end architecture ex2;
```

VHDL instantiation

- Instead of directly describe the functionality, a more structured (hierarchical) description can be given
- Sub-blocks instantiation

```
entity comb_function is
 port (a, b, c : in BIT; z: out BIT);
end entity comb_function;
architecture expression of comb_function is
begin
 z <= (not a and b) or (a and c);
end architecture expression;
```

VHDL instantiation

- Instead of directly describe the functionality, a more structured (hierarchical) description can be given
- Sub-blocks instantiation

```
entity comb_function is
 port (a, b, c : in BIT; z: out BIT);
end entity comb_function;
architecture netlist of comb_function is
 signal p, q, r : BIT;
begin
 g1: entity WORK.Not1(ex1) port map (a, p);
 g2: entity WORK.And2(ex1) port map (p, b, q);
 g3: entity WORK.And2(ex1) port map (a, c, r);
 g4: entity WORK.Or2(ex1) port map (q, r, z);
end architecture netlist;
```

```
entity Or2 is
 port (x, y : in BIT; z: out BIT);
end entity Or2;
architecture ex1 of Or2 is
begin
 z <= x or y;
end architecture ex1;
```

```
entity Not1 is
 port (x : in BIT; z: out BIT);
end entity Not1;
architecture ex1 of Not1 is
begin
 z <= not x;
end architecture ex1;
```

Standard logic

'U' Uninitialized
'X' Forcing (i.e. strong) unknown
'0' Forcing 0
'1' Forcing 1
'Z' High impedance
'W' Weak unknown
'L' Weak 0
'H' Weak 1
'–' Don't care

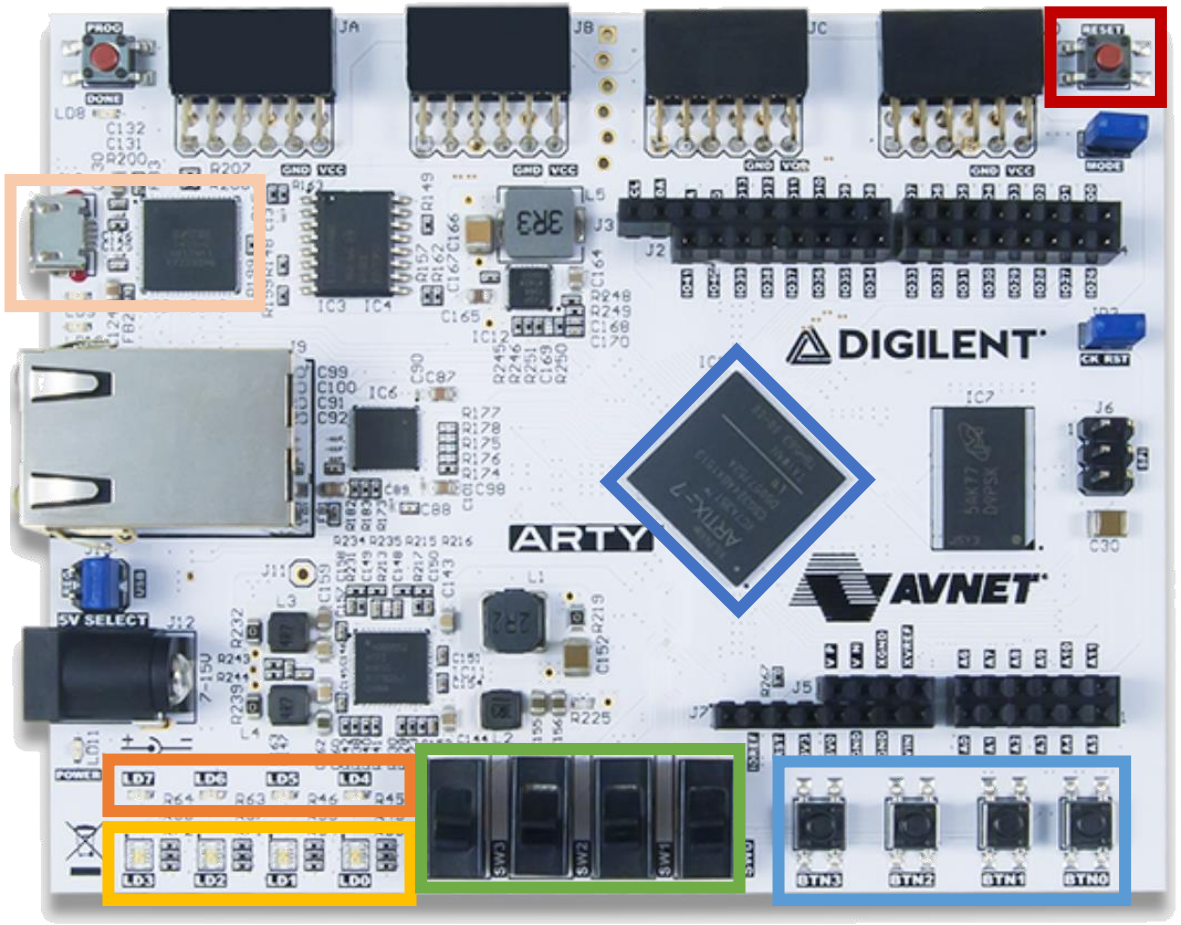
	U	**X**	**0**	**1**	**Z**	**W**	**L**	**H**	**–**
U	U	U	U	U	U	U	U	U	U
X	U	X	X	X	X	X	X	X	X
0	U	X	0	X	0	0	0	0	X
1	U	X	X	1	1	1	1	1	X
Z	U	X	0	1	Z	W	L	H	X
W	U	X	0	1	W	W	W	W	X
L	U	X	0	1	L	W	L	W	X
H	U	X	0	1	H	W	W	H	X
–	U	X	X	X	X	X	X	X	X

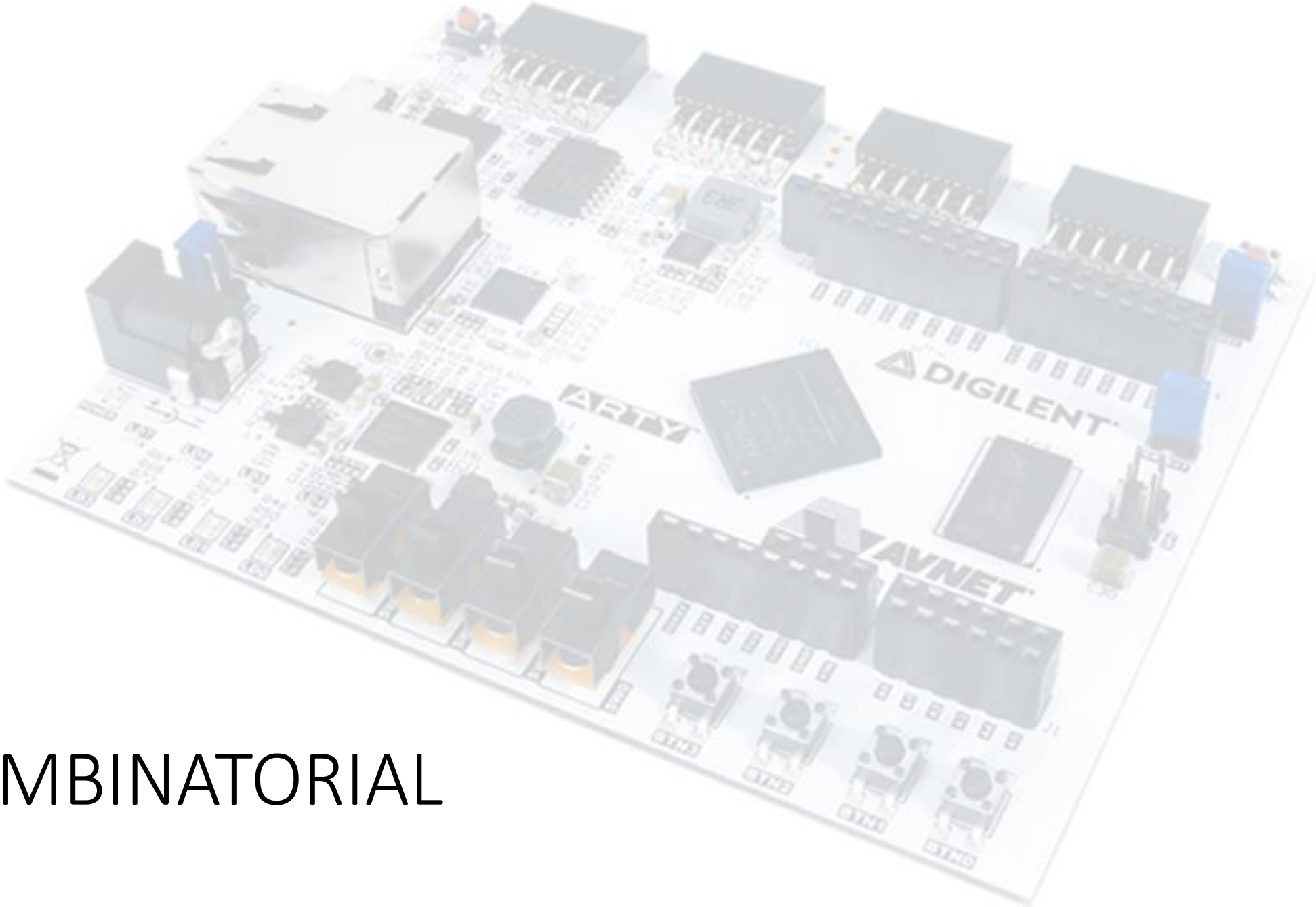
ARTY A7

- Artix-7 FPGA
  - XC7A100TCSG324-1
  - 15,850 slices
  - 4,860 Kbits BRAM
  - 6 CMTs
  - 240 DSP

- ARTY A7 board
  - 1 USB-UART Bridge
  - 4 Switches
  - 4 Buttons
  - 1 Reset Button
  - 4 LEDs
  - 4 RGB LEDs

- Reference manual
- Constraints file
- Schematics





LAB COMBINATORIAL

Adder

- Elementary cell for adding two bits (binary digit)
- Extension to two n-bits numbers

A_i	B_i	C_i	S_i	C_{i+1}
0	0	0	0	0
1	0	0	1	0
0	1	0	1	0
1	1	0	0	1
0	0	1	1	0
1	0	1	0	1
0	1	1	0	1
1	1	1	1	1

$$S_i = \left(A\,\overline{B}\,\overline{C} + \overline{A}\,B\,\overline{C} + \overline{A}\,\overline{B}\,C + A\,B\,C\right)_i$$

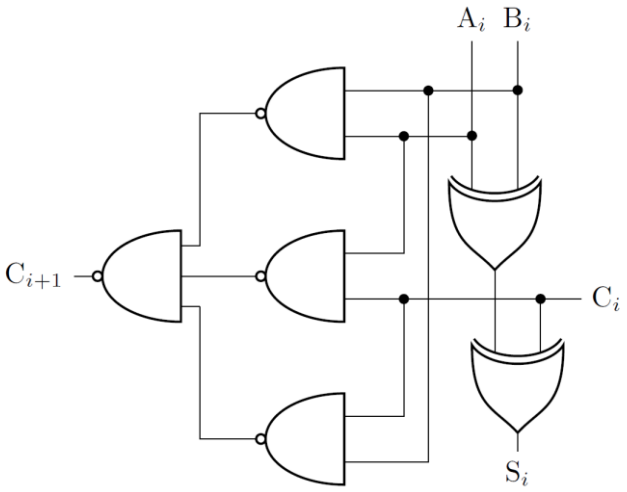
$$C_{i+1} = \left(A\,B\,\overline{C} + A\,\overline{B}\,C + \overline{A}\,B\,C + A\,B\,C\right)_i$$

$$S_i = \left(A\,\overline{B} + \overline{A}\,B\right)_i \overline{C_i} + \left(\overline{A}\,\overline{B} + A\,B\right)_i C_i = (A \oplus B)_i \oplus C_i$$

$$C_{i+1} = (A\,B + A\,C + B\,C)_i$$

Adder

- Elementary cell for adding two bits (binary digit)
- Extension to two n-bits numbers



$$S_i = \left(A\,\overline{B}\,\overline{C} + \overline{A}\,B\,\overline{C} + \overline{A}\,\overline{B}\,C + A\,B\,C\right)_i$$

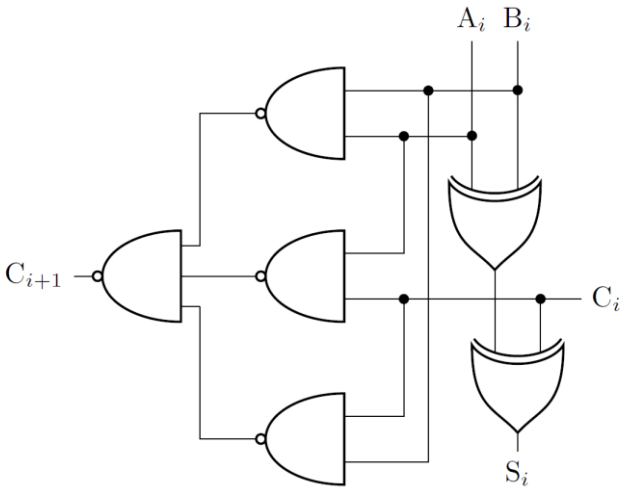
$$C_{i+1} = \left(A\,B\,\overline{C} + A\,\overline{B}\,C + \overline{A}\,B\,C + A\,B\,C\right)_i$$

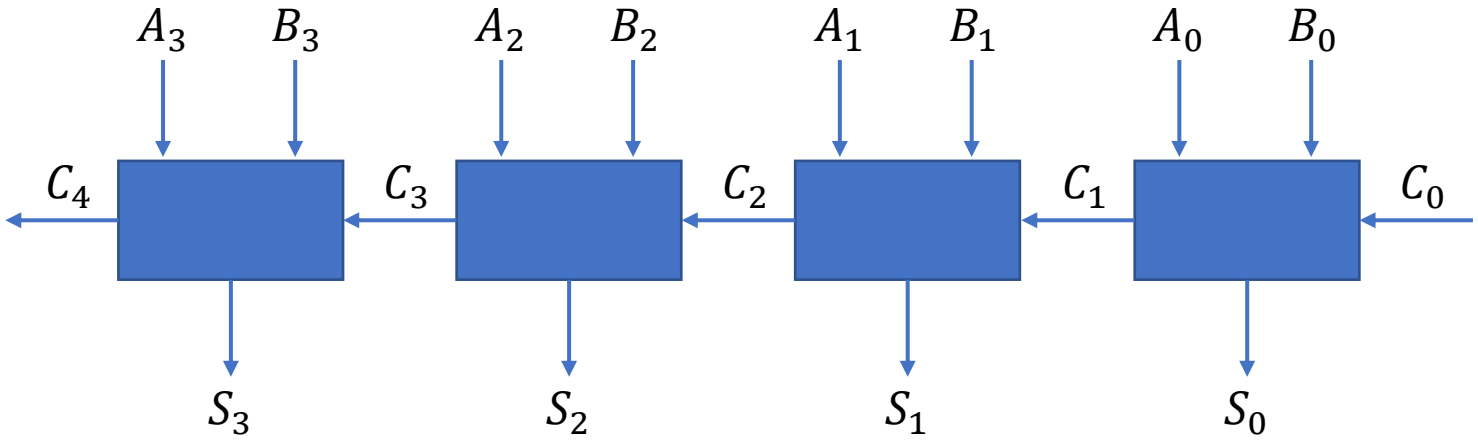
$$S_i = \left(A\,\overline{B} + \overline{A}\,B\right)_i \overline{C_i} + \left(\overline{A}\,\overline{B} + A\,B\right)_i C_i = (A \oplus B)_i \oplus C_i$$

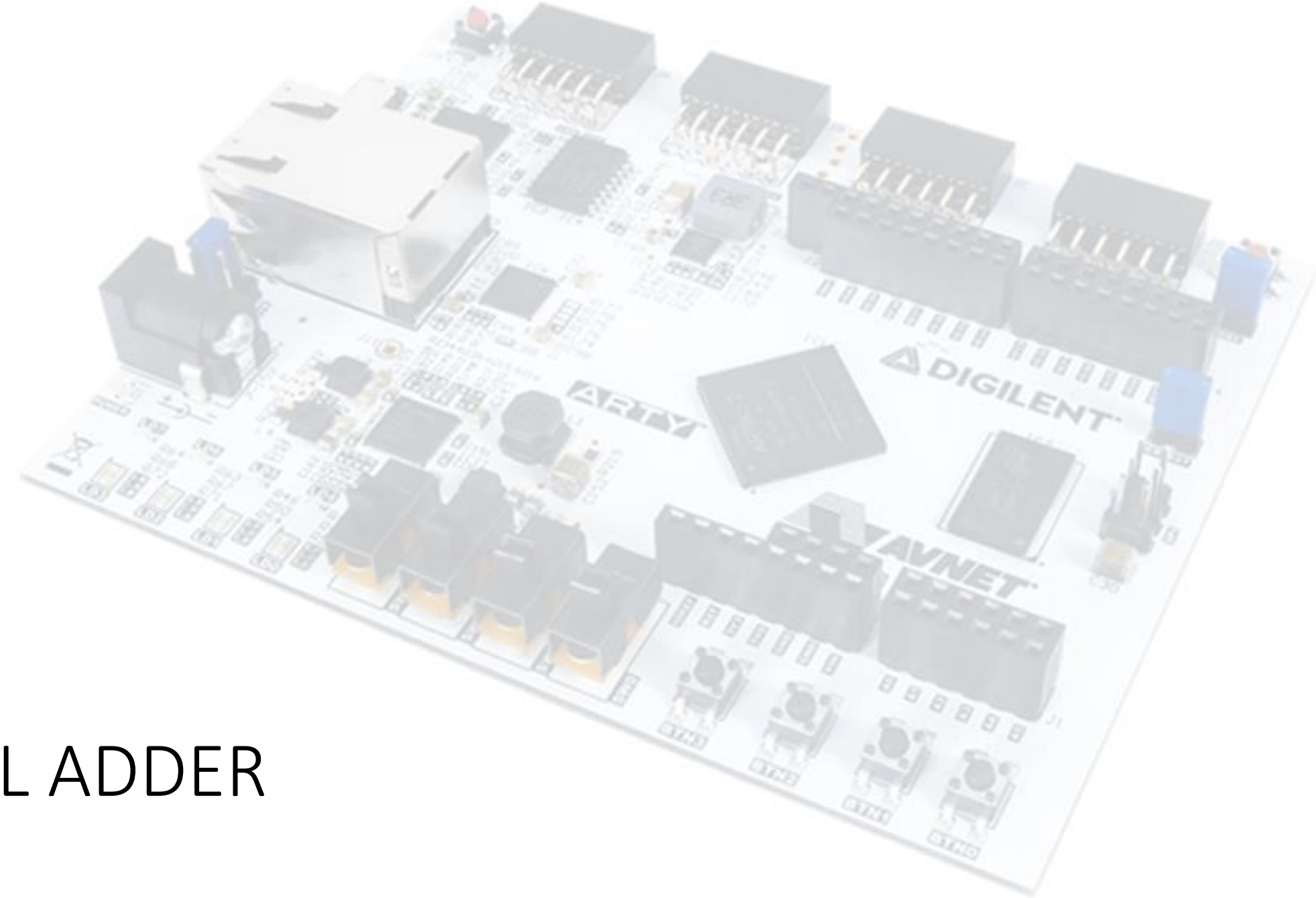
$$C_{i+1} = (A\,B + A\,C + B\,C)_i$$

Adder

- Elementary cell for adding two bits (binary digit)
- Extension to two n-bits numbers

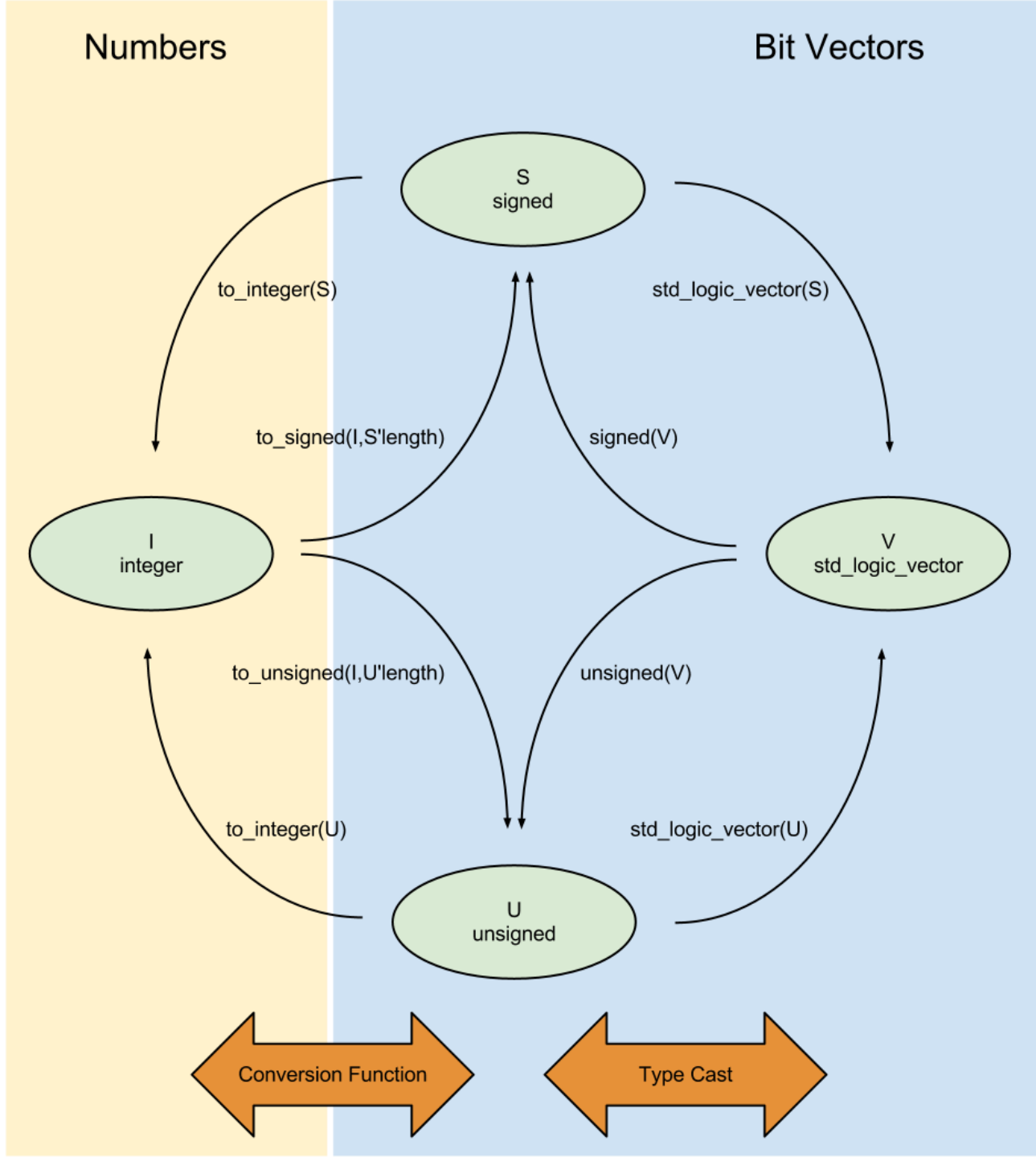






LAB RTL ADDER

Type conversions



Adder

- Adder in VHDL (with unsigned arithmetic)

```
entity NBitAdder is
  generic (n: NATURAL :=4);
  port (A, B: in std_logic_vector(n-1 downto 0);
  Cin : in std_logic;
  Sum : out std_logic_vector(n-1 downto 0);
  Cout: out std_logic);
end entity NBitAdder;
```

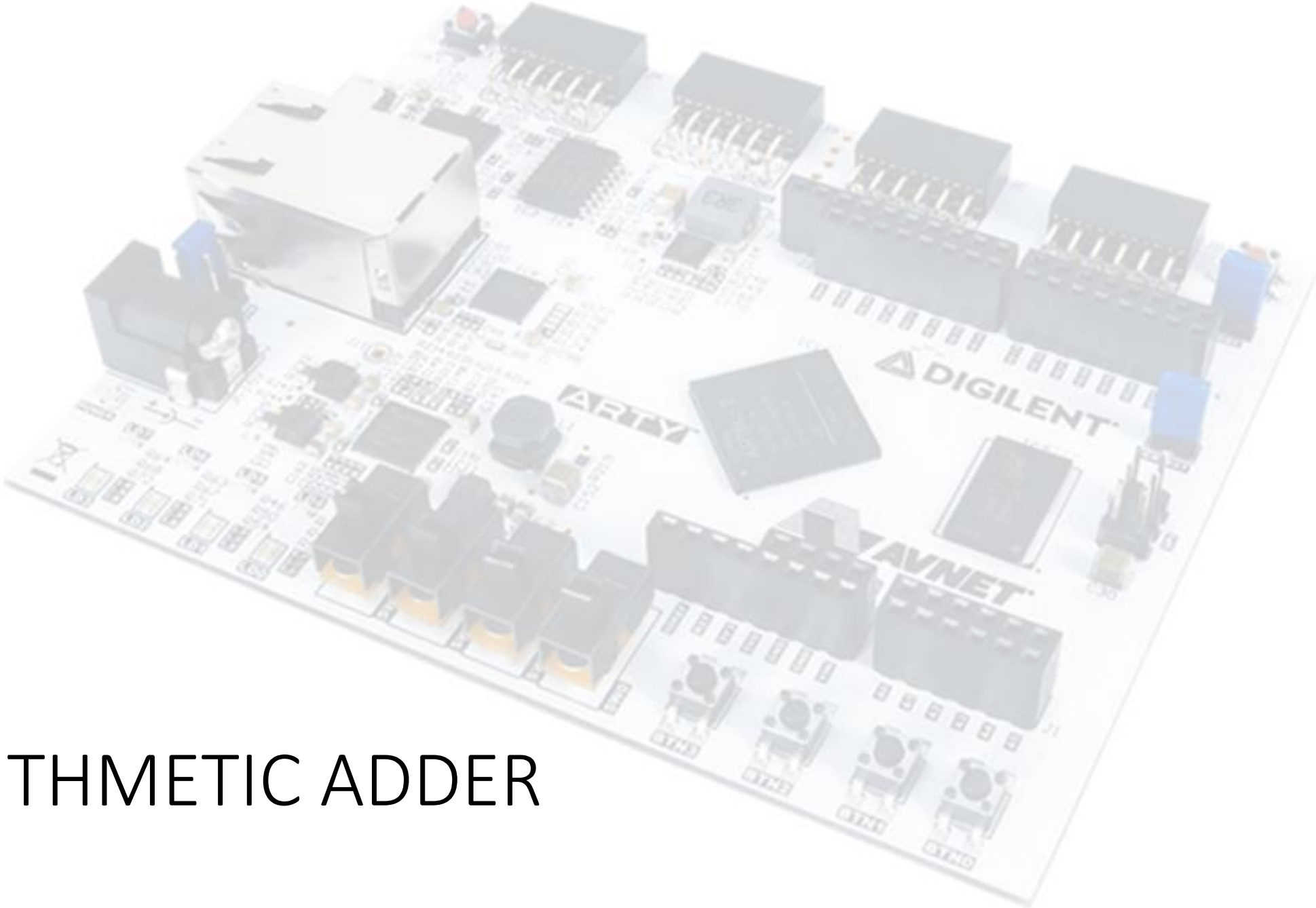
```
architecture unsigned of NBitAdder is
  signal result : unsigned(n downto 0);
  signal carry : unsigned(n downto 0);
  constant zeros : unsigned(n-1 downto 0) := (others => '0');
begin
  carry <= (zeros & Cin);
  result <= ('0' & unsigned(A)) + ('0' & unsigned(B)) + carry;
  Sum <= std_logic_vector(result(n-1 downto 0));
  Cout <= result(n);
end architecture unsigned;
```

Adder

- Adder in VHDL (with signed arithmetic)

```
entity NBitAdder is
  generic (n: NATURAL :=4);
  port (A, B: in std_logic_vector(n-1 downto 0);
  Cin : in std_logic;
  Sum : out std_logic_vector(n-1 downto 0);
  Cout: out std_logic);
end entity NBitAdder;
```

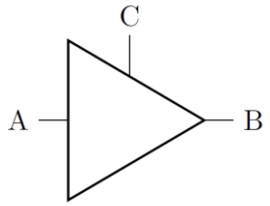
```
architecture signed of NBitAdder is
  signal result : signed(n downto 0);
  signal carry : signed(n downto 0);
  constant zeros : signed(n-1 downto 0) := (others => '0');
begin
  carry <= (zeros & Cin);
  result <= (A(n-1) & signed(A)) + (B(n-1) & signed(B)) + carry;
  Sum <= std_logic_vector(result(n-1 downto 0));
  Cout <= result(n);
end architecture signed;
```



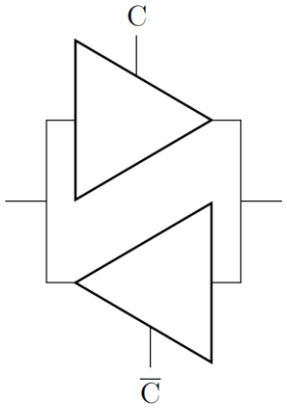
LAB ARITHMETIC ADDER

Tristate

- It is used for developing bidirectional connections
- A control line is used for putting the output in high-impedance
- Many data buses are usually tristate because they are used to link devices that can be both source and sink of information



A	*C*	*B*
0	0	Z
1	0	Z
0	1	0
1	1	1

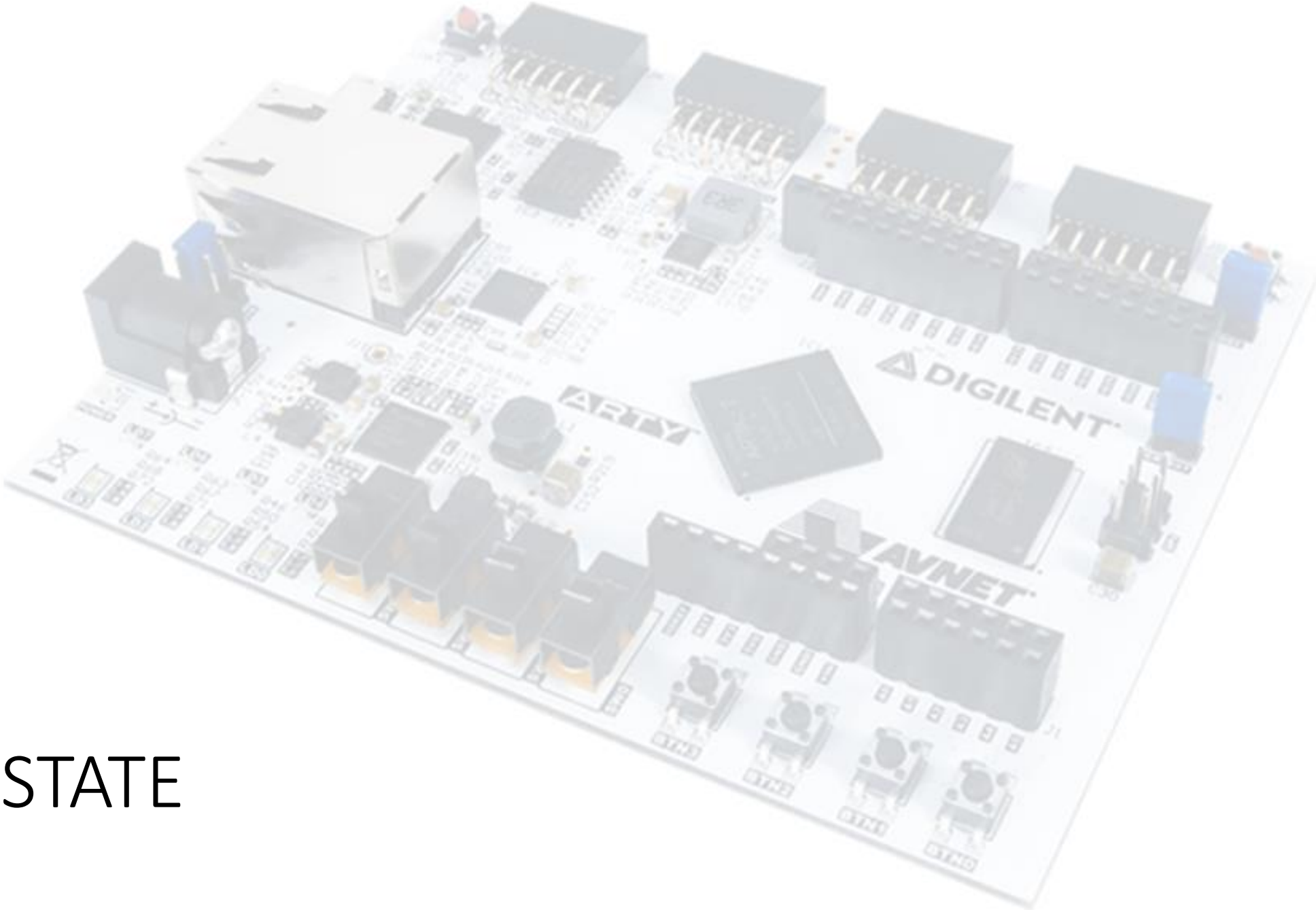


Tristate

- Tristate in VHDL

```
entity tri_state_buffer_top is
  Port ( I    : in  STD_LOGIC;
  T   : in  STD_LOGIC;
  O   : out STD_LOGIC;
end tri_state_buffer_top;

architecture Behavioral of tri_state_buffer_top is
begin
  O <= I when (T = '0') else 'Z';
end Behavioral;
```

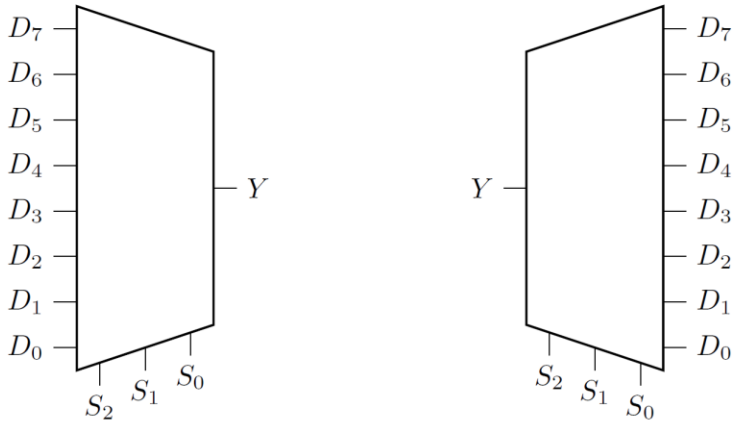
```
architecture Primitive of tri_state_buffer_top is
begin
 OBUFT_inst : OBUFT
   generic map (
     DRIVE => 12,
     IOSTANDARD => "DEFAULT",
     SLEW => "SLOW")
   port map (
     O => O,     -- Buffer output (connect directly to top-level port)
     I => I,     -- Buffer input
     T => T      -- 3-state enable input
   );
end Primitive;
```



LAB TRISTATE

Multiplexer

- A Mux selects one signal among 2^n (D_i) thanks to n address lines (S_j)
- A Demux applies the inverse operation

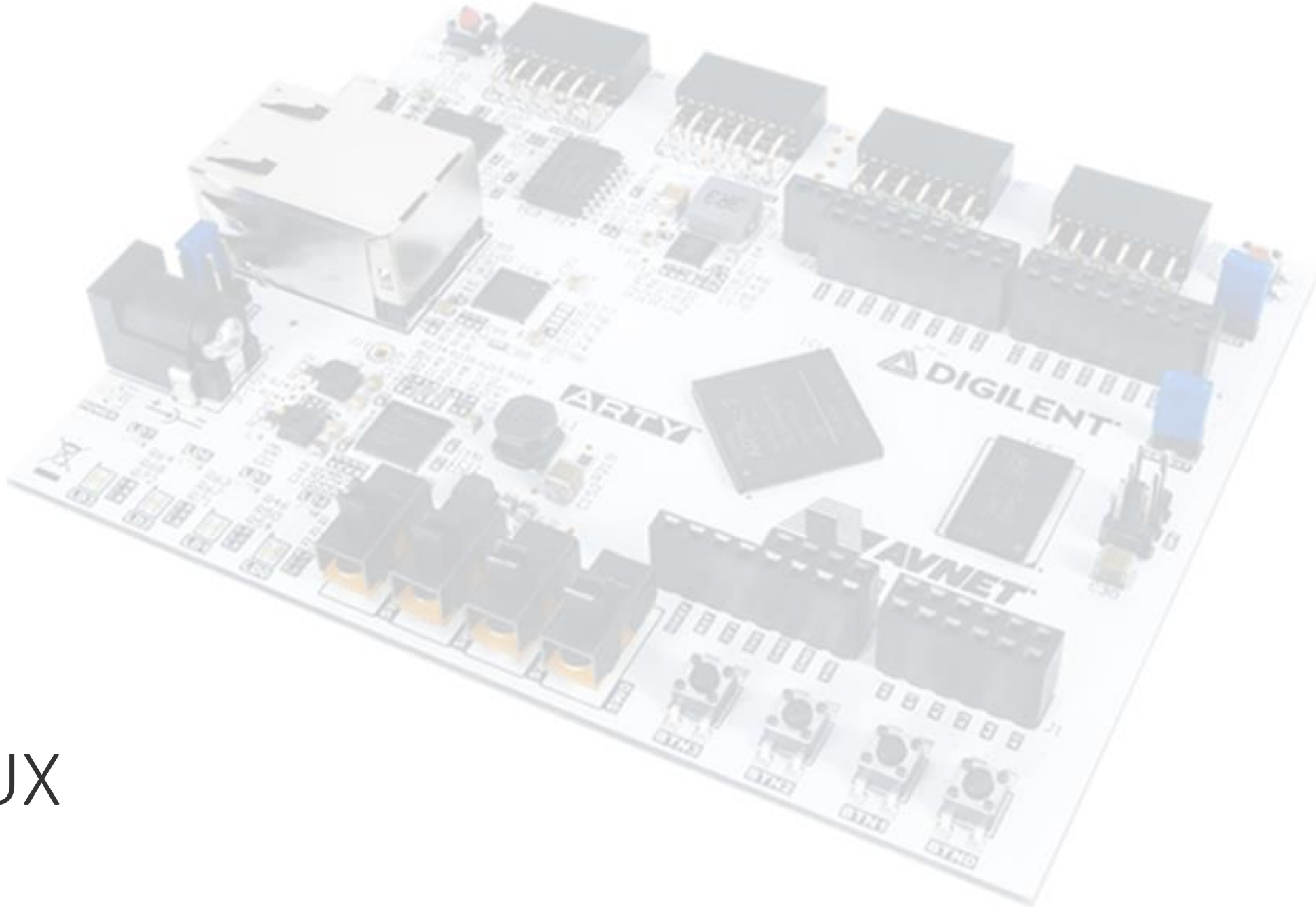


Multiplexer

- 4 to 1 multiplexer in VHDL

```
entity mux is
  port (a, b, c, d: in std_logic;
  s: in std_logic_vector(1 downto 0);
  y: out std_logic;
end entity mux;

architecture Behavioural of mux is
begin
  y <=  a when s = "00" else
        b when s = "01" else
        c when s = "10" else
        d when s = "11" else
        'X';
end architecture Behavioural;
```

LAB MUX

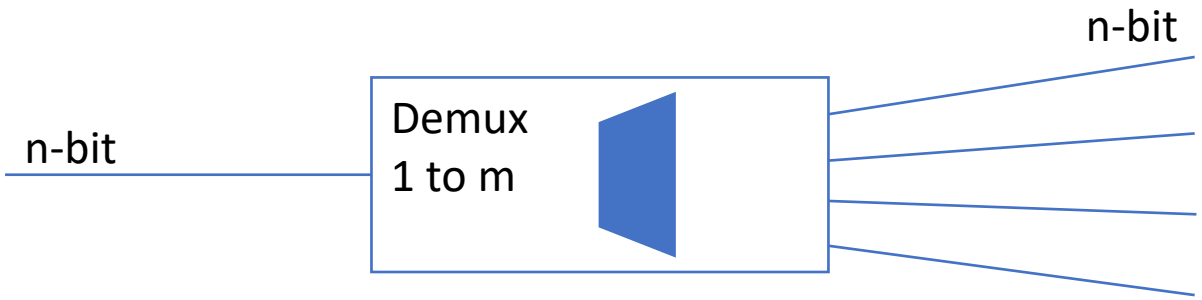


Exercise

- Write a n-bit 1 to 4 demultiplexer of an n-bit input

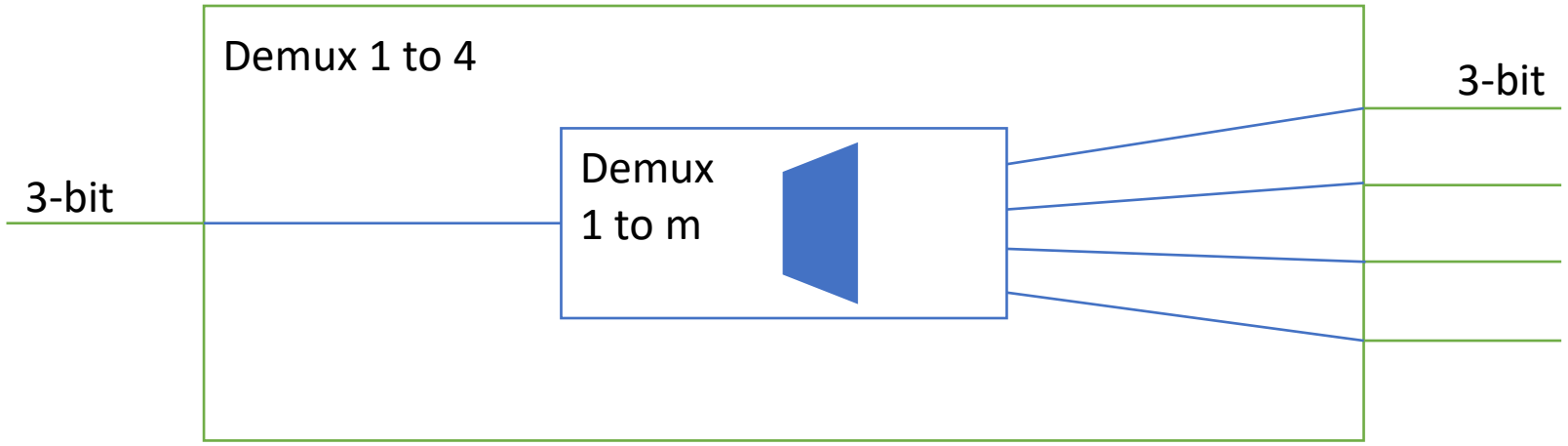
Exercise

- Write a n-bit 1 to 4 demultiplexer of an n-bit input
- Extend the demultiplexer to a generic 1 to m demultiplexer

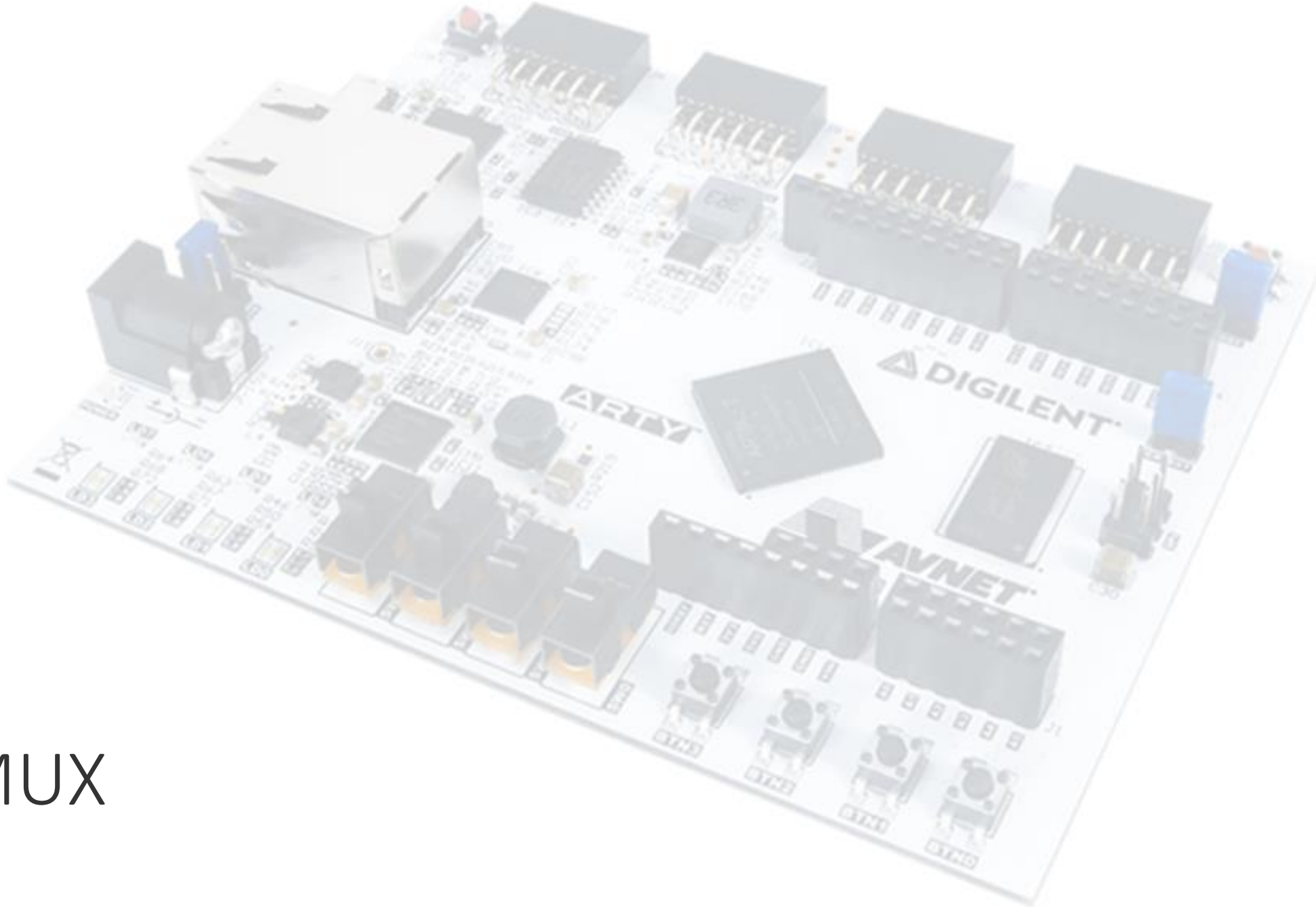


Exercise

- Write a n-bit 1 to 4 demultiplexer of an n-bit input
- Extend the demultiplexer to a generic 1 to m demultiplexer
- Instantiate it in synthesizable top module (n=3, m=4) and simulate it

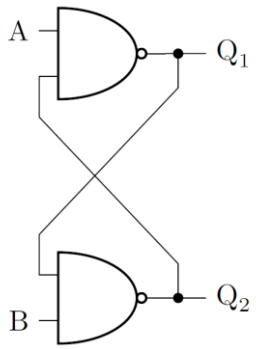


EX DEMUX



Memory elements

- If both inputs are at 1, the circuit keep memory of the previous state

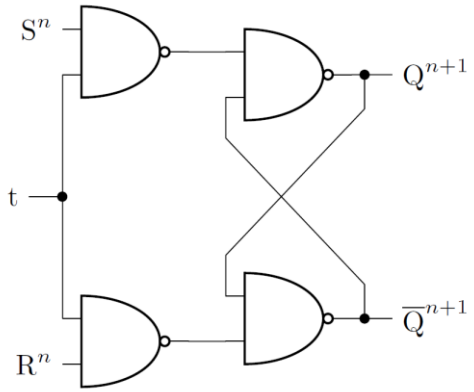


A	B	Q_2	Q_1
0	0	1	1
1	0	1	0
0	1	0	1
1	1	X	$\overline{X}$

- Latch

Memory elements

- Flip-flop set-reset (S-R)
  - A synchronization input t is added
  - If t is not present the output doesn't change



S^n	R^n	Q^{n+1}	$\overline{Q}^{n+1}$
0	0	Q^n	$\overline{Q}^n$
1	0	1	0
0	1	0	1
1	1	-	-

$$Q^{n+1} = \left(Q\,\overline{S}\,\overline{R} + S\,\overline{R}\right)^n = \left(S + Q\,\overline{R}\right)^n$$
$$S\,R = 0$$

Memory elements

- Flip-flop J-K
  - It removes the forbidden state

$$Q^{n+1} = \left(Q\,\overline{J}\;\overline{K} + J\,\overline{K} + \overline{Q}\,J\,K\right)^n = \left(Q\,\overline{K} + \overline{Q}\,J\right)^n$$

J^n	K^n	Q^{n+1}	$\overline{Q}^{n+1}$
0	0	Q^n	$\overline{Q}^n$
1	0	1	0
0	1	0	1
1	1	$\overline{Q}^n$	Q^n

- Flip-flop D
  - Only one input

$$Q^{n+1} = D^n$$

D^n	Q^{n+1}	$\overline{Q}^{n+1}$
0	0	1
1	1	0

Memory elements

- D Flip-flop in VHDL

```
entity RisingEdge_DFlipFlop is
  port(
    Q : out std_logic;
    Clk :in std_logic;
    D :in  std_logic
  );
end RisingEdge_DFlipFlop;
```

```
architecture Behavioral of RisingEdge_DFlipFlop is
begin
  process(Clk)
  begin
    if(rising_edge(Clk)) then
      Q <= D;
    end if;
  end process;
end Behavioral;
```

VHDL sequential

- Usually, all the assignments and instantiations in VHDL are concurrent
- Sequential statements can be used in sub-program (procedure and function) or processes

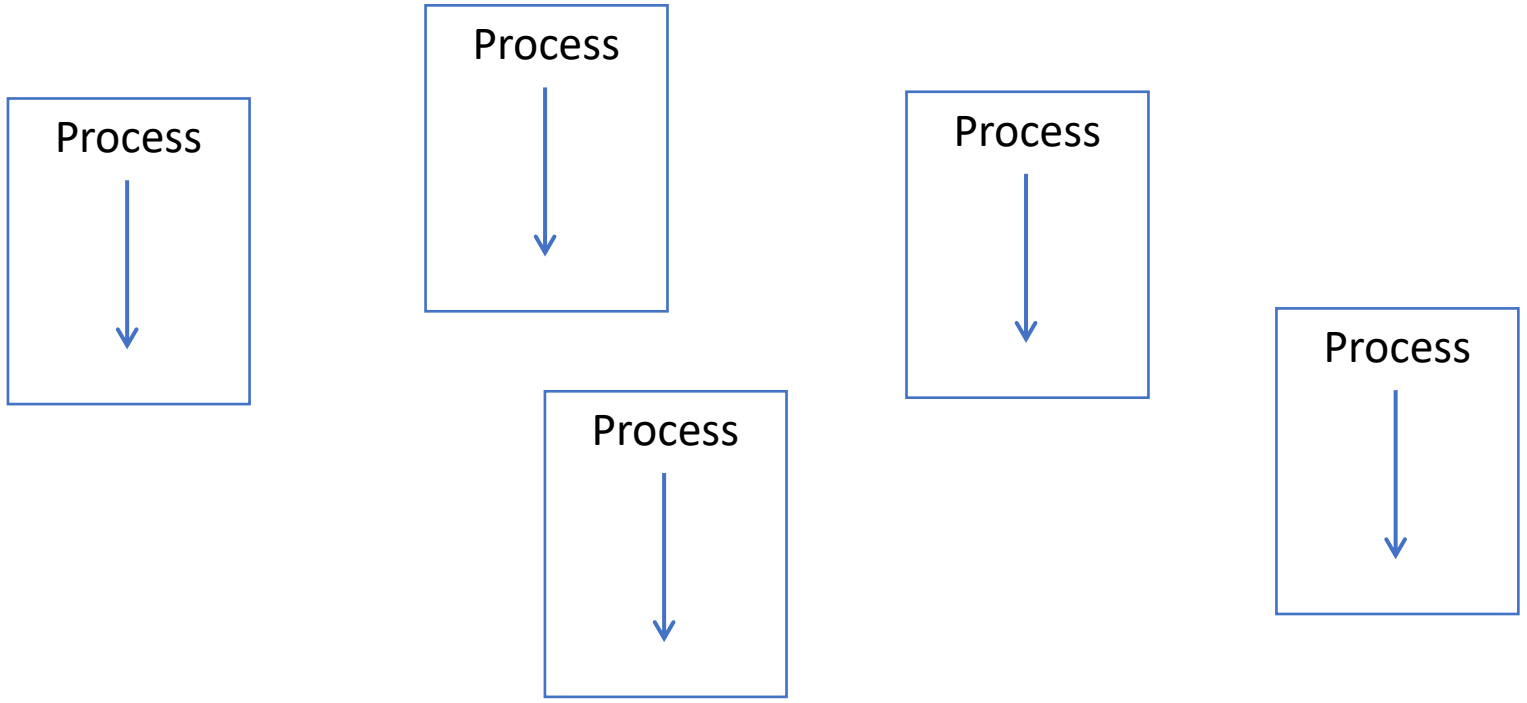
```
entity priority is
   port (a: in std_logic_vector(3 downto 0);
      y: out std_logic_vector(1 downto 0);
      valid: out std_logic);
end entity priority;

architecture Concurrent of priority is
begin
 y <= "11" when a(3) = '1' else
      "10" when a(2) = '1' else
      "01" when a(1) = '1' else
      "00" when a(0) = '1' else
      "00";
 valid <= '1' when a(0) = '1' or a(1) = '1'
          or a(2) = '1' or a(3) = '1' else
          '0';
end architecture Concurrent;
```

```
architecture Sequential of
priority is
begin
 process (a) is
 begin
  if a(3) = '1' then
  y <= "11";
  valid <= '1';
  elsif a(2) = '1' then
  y <= "10";
  valid <= '1';
  elsif a(1) = '1' then
  y <= "01";
  valid <= '1';
  elsif a(0) = '1' then
  y <= "00";
  valid <= '1';
  else
  y <= "00";
  valid <= '0';
  end if;
 end process;
end architecture Sequential;
```

```
architecture Sequential2 of
priority is
begin
 process (a) is
 begin
 valid <= '1';
  if a(3) = '1' then
  y <= "11";
  elsif a(2) = '1' then
  y <= "10";
  elsif a(1) = '1' then
  y <= "01";
  elsif a(0) = '1' then
  y <= "00";
  else
  valid <= '0';
  y <= "00";
  end if;
 end process;
end architecture Sequential2;
```

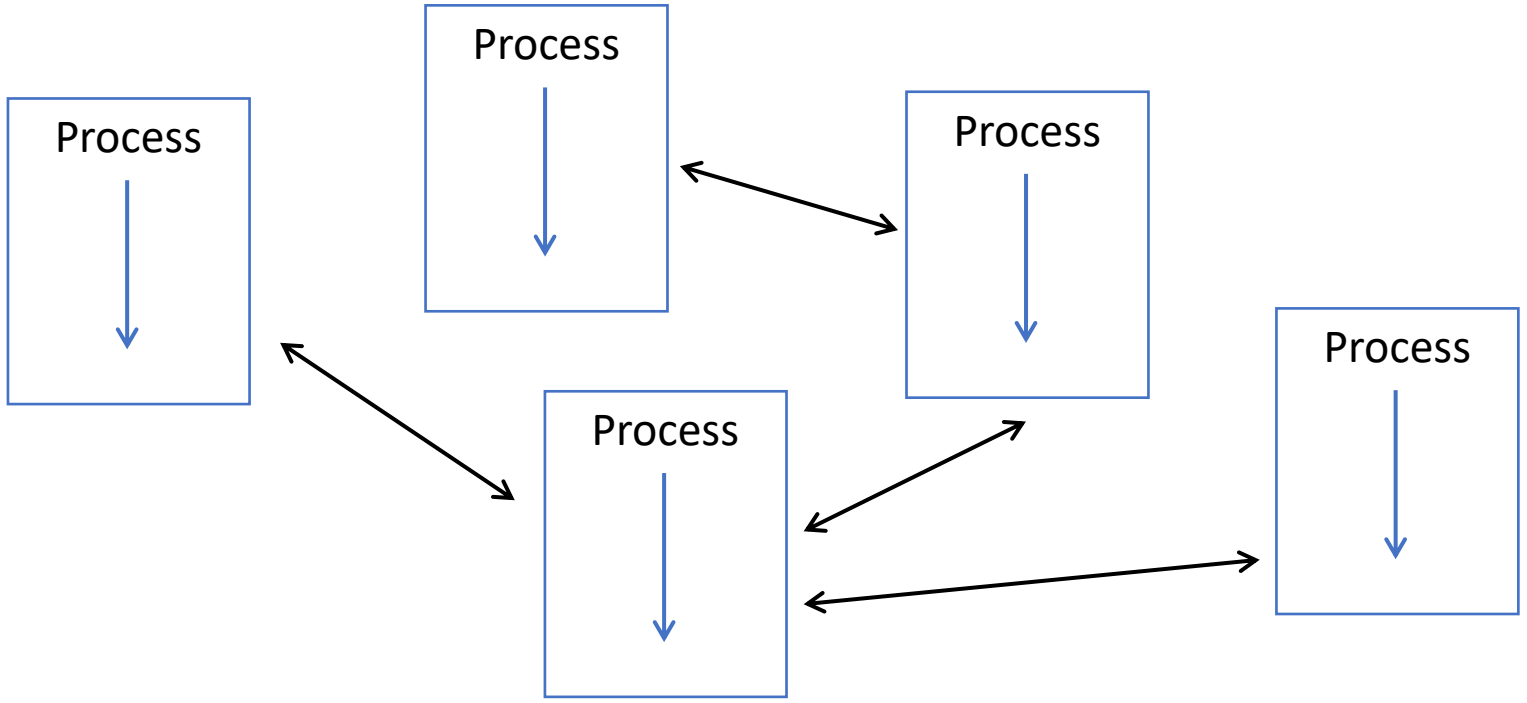
VHDL sequential

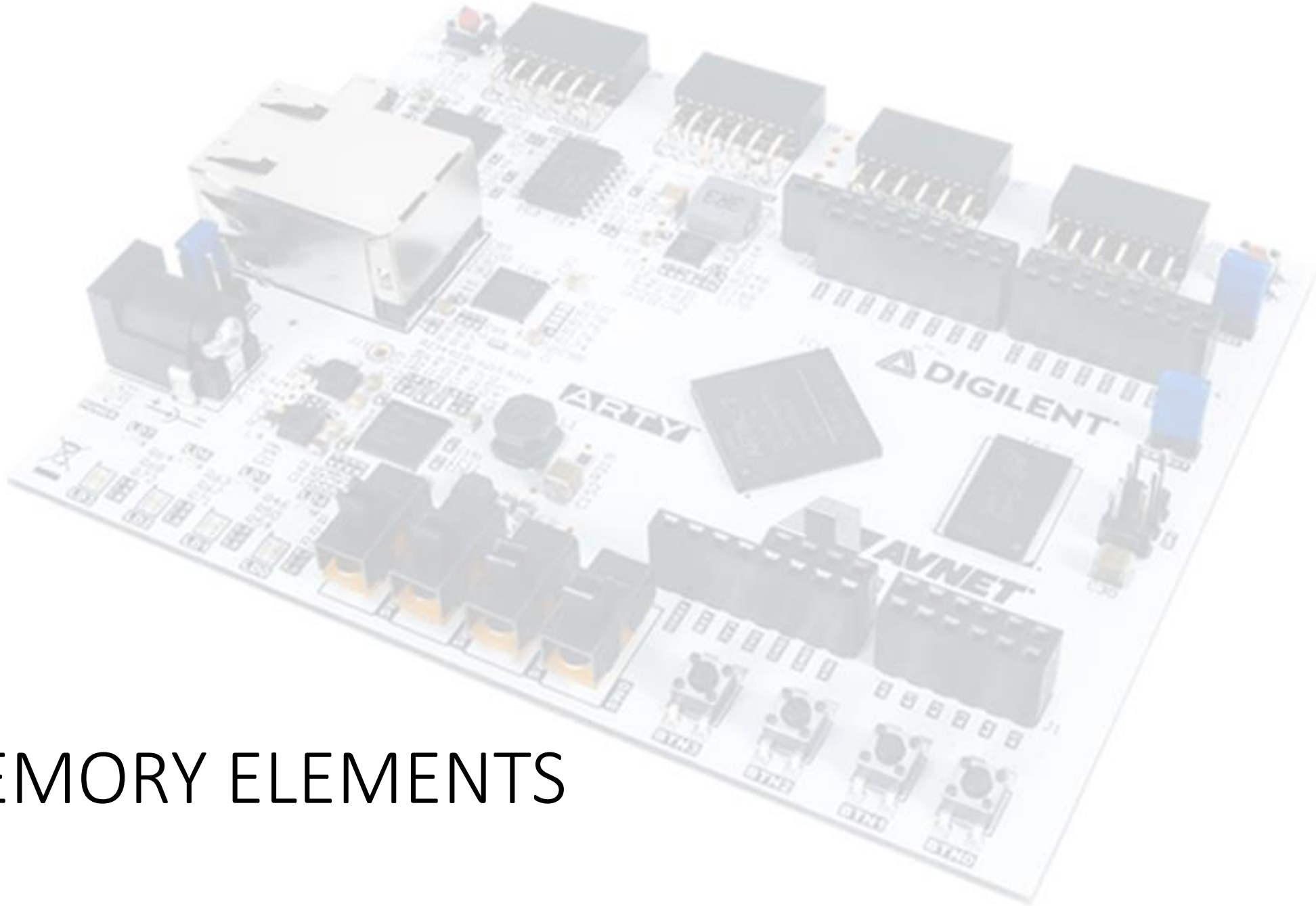
- Usually, all the assignments and instantiations in VHDL are concurrent
- Sequential statements can be used in sub-program (procedure and function) or processes



VHDL sequential

- Usually, all the assignments and instantiations in VHDL are concurrent
- Sequential statements can be used in sub-program (procedure and function) or processes

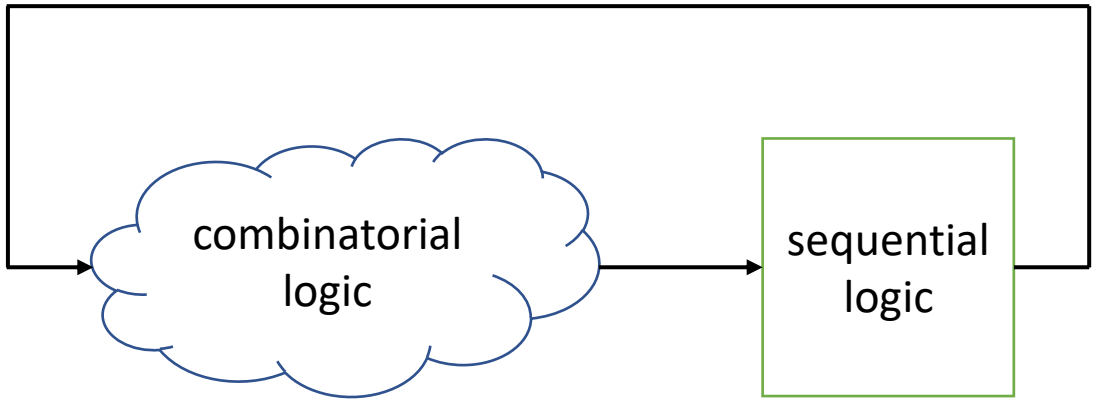




LAB MEMORY ELEMENTS

Sequential circuits

- Usually, memory elements are controlled by logic functions



- Examples are counters, shift registers…

Counter

- Design a 4-bit counter
- $A^{n+1} = \left[\overline{A}\right]^{n}$
- $B^{n+1} = \left[A\overline{B} + \overline{A}B\right]^{n}$
- $C^{n+1} = \left[C\left(\overline{A} + \overline{B}\right) + \overline{C}AB\right]^{n}$
- $D^{n+1} = \left[D\left(\overline{A} + \overline{B} + \overline{C}\right) + \overline{D}ABC\right]^{n}$
- 4 FFs and 4 LUTs

D^n	C^n	B^n	A^n	D^{n+1}	C^{n+1}	B^{n+1}	A^{n+1}
0	0	0	0	0	0	0	1
0	0	0	1	0	0	1	0
0	0	1	0	0	0	1	1
0	0	1	1	0	1	0	0
0	1	0	0	0	1	0	1
0	1	0	1	0	1	1	0
0	1	1	0	0	1	1	1
0	1	1	1	1	0	0	0
1	0	0	0	1	0	0	1
1	0	0	1	1	0	1	0
1	0	1	0	1	0	1	1
1	0	1	1	1	1	0	0
1	1	0	0	1	1	0	1
1	1	0	1	1	1	1	0
1	1	1	0	1	1	1	1
1	1	1	1	0	0	0	0

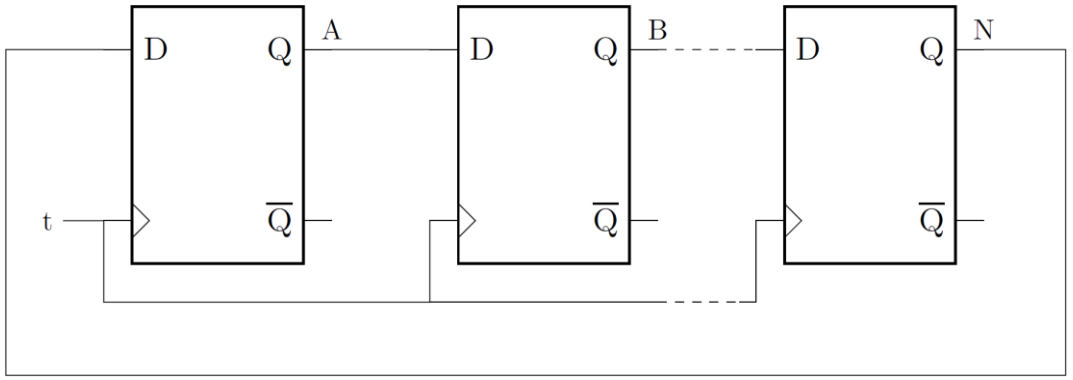
Counter

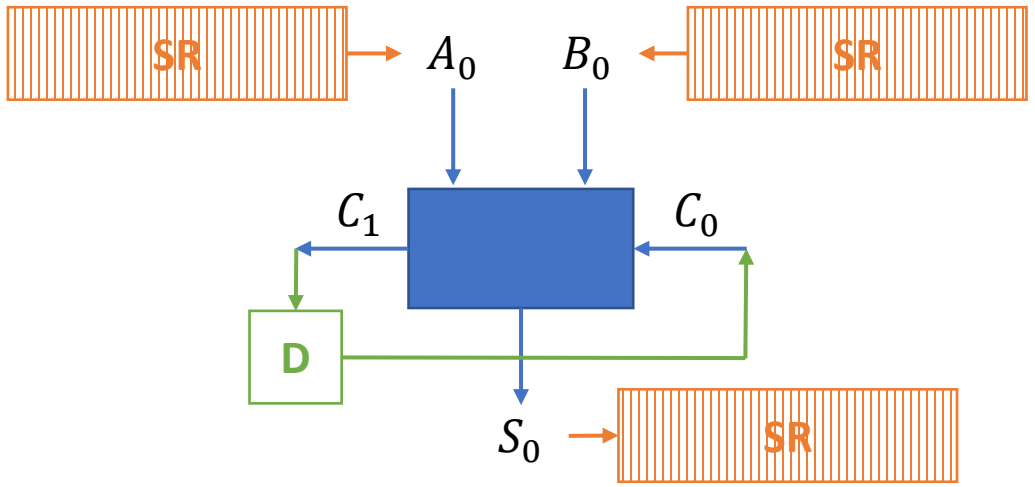
- Counter in VHDL

```
architecture Behavioral of Counter is
signal count : unsigned(3 downto 0) :=(others => '0');
begin
  process(Clk, count)
  begin
    if(rising_edge(Clk)) then
      count <= count + 1;
    end if;
  end process;
  O <= std_logic_vector(count);
end Behavioral;
```

Shift register

- It is used to shift a signal
- Its design can be obtained by the sequential table
- Without feedback can be used to access in parallel to a serial data or to serialize a parallel data
- Each cell can have a multiplexer to choose if shifting the bit or loading a new bit





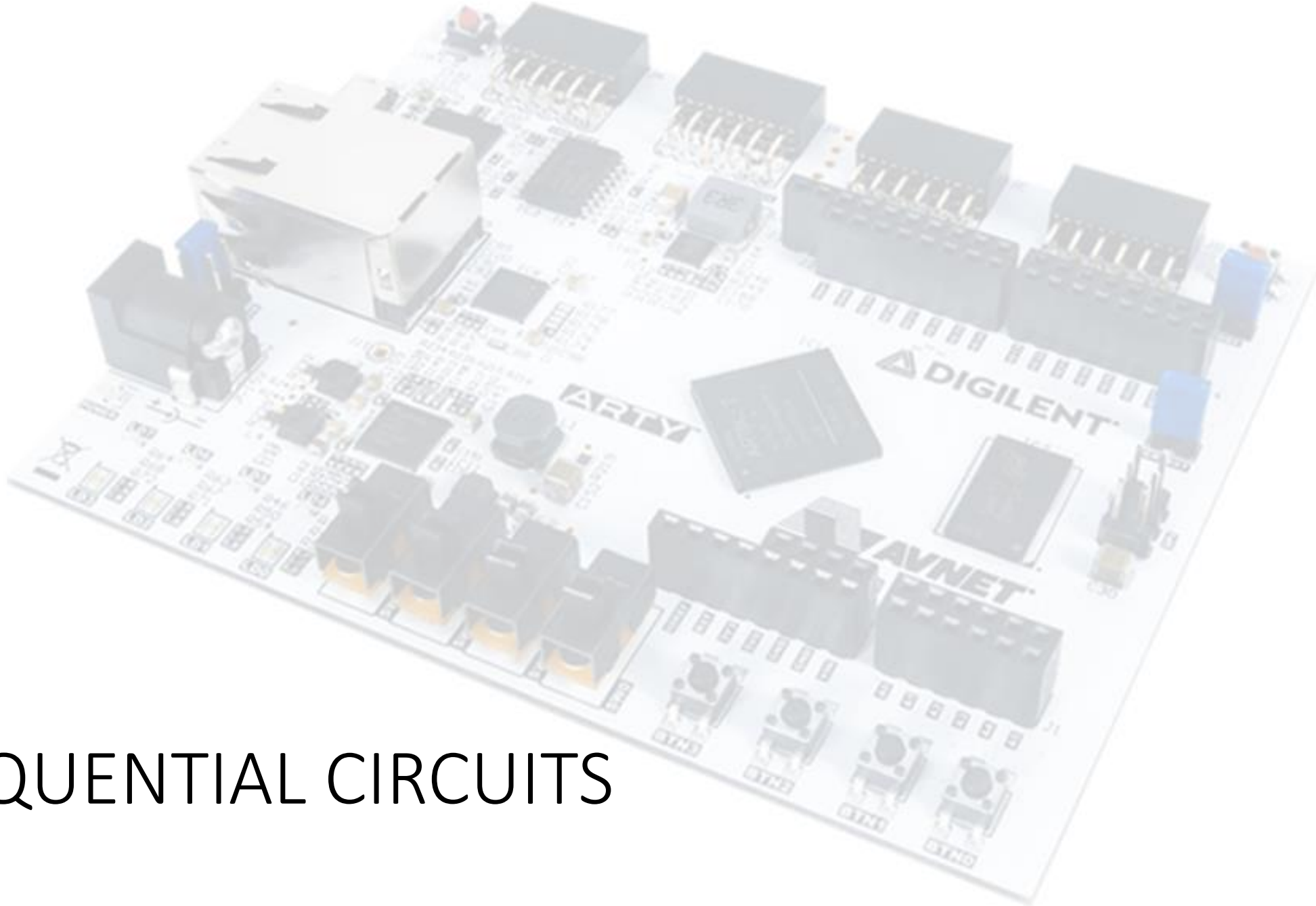
Shift register

- Shift register in VHDL

```
sreg : process(clk, rst)
begin
  if (rst='1') then
    r_data    <= (others=>(others=>'0'));
  elsif (rising_edge(clk)) then
    r_data(0)    <=    i_data;
    for i in 1 to r_data'length-1 loop
      r_data(i)    <=    r_data(i-1) ;
    end loop;
  end if;
end process;
```

Attributes

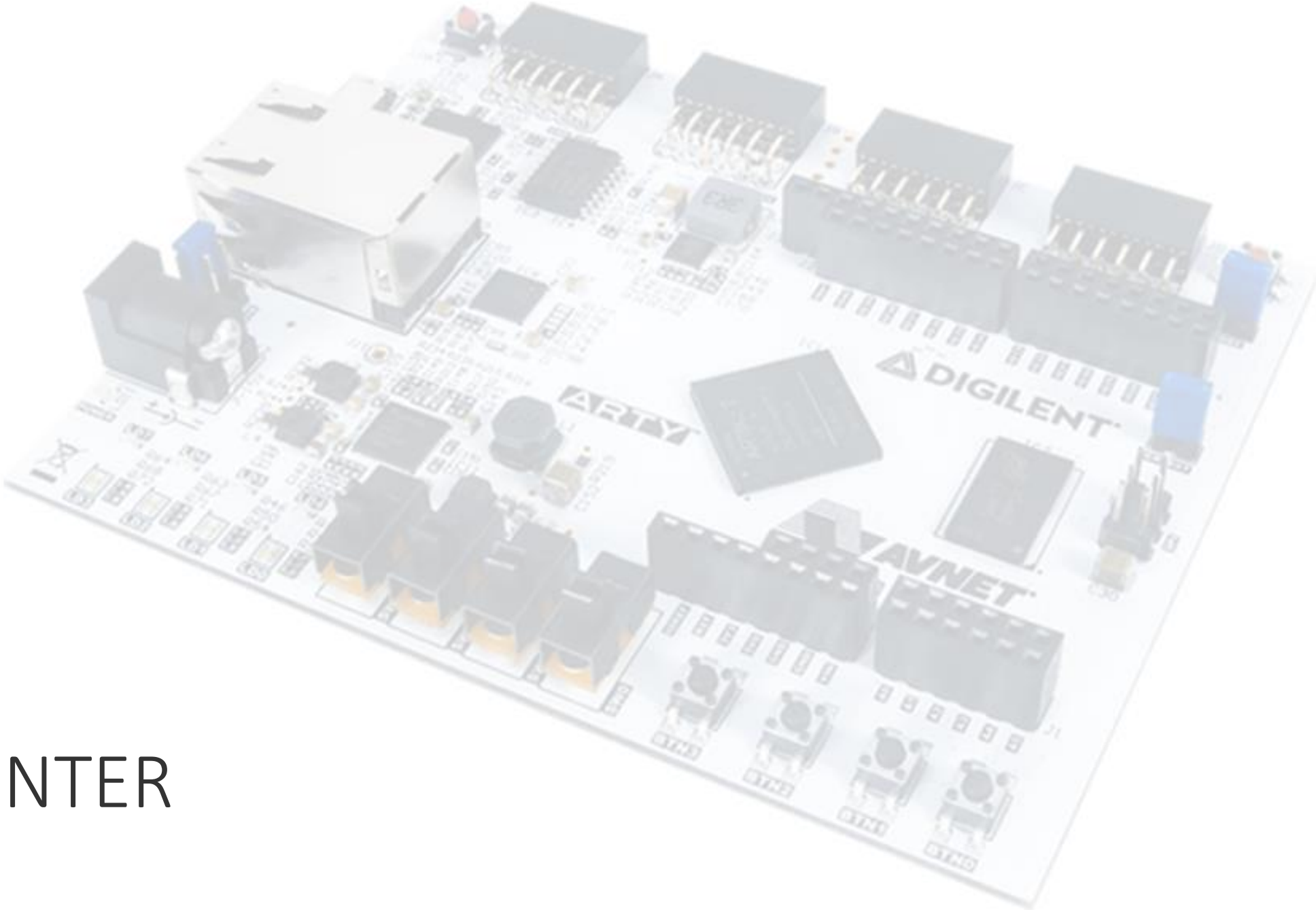
```
T'BASE        is the base type of the type T
T'LEFT        is the leftmost value of type T. (Largest if downto)
T'RIGHT       is the rightmost value of type T. (Smallest if downto)
T'HIGH        is the highest value of type T.
T'LOW         is the lowest value of type T.
T'ASCENDING   is boolean true if range of T defined with to .
T'IMAGE(X)    is a string representation of X that is of type T.
T'VALUE(X)    is a value of type T converted from the string X.
T'POS(X)      is the integer position of X in the discrete type T.
T'VAL(X)      is the value of discrete type T at integer position X.
T'SUCC(X)     is the value of discrete type T that is the successor of X.
T'PRED(X)     is the value of discrete type T that is the predecessor of X.
T'LEFTOF(X)   is the value of discrete type T that is left of X.
T'RIGHTOF(X)  is the value of discrete type T that is right of X.
A'LEFT        is the leftmost subscript of array A or constrained array type.
A'LEFT(N)     is the leftmost subscript of dimension N of array A.
A'RIGHT       is the rightmost subscript of array A or constrained array type.
A'RIGHT(N)    is the rightmost subscript of dimension N of array A.
A'HIGH        is the highest subscript of array A or constrained array type.
A'HIGH(N)     is the highest subscript of dimension N of array A.
A'LOW         is the lowest subscript of array A or constrained array type.
A'LOW(N)      is the lowest subscript of dimension N of array A.
A'RANGE       is the range  A'LEFT to A'RIGHT  or  A'LEFT downto A'RIGHT .
A'RANGE(N)    is the range of dimension N of A.
A'REVERSE_RANGE  is the range of A with to and downto reversed.
A'REVERSE_RANGE(N)  is the REVERSE_RANGE of dimension N of array A.
A'LENGTH      is the integer value of the number of elements in array A.
A'LENGTH(N)   is the number of elements of dimension N of array A.
A'ASCENDING   is boolean true if range of A defined with to .
A'ASCENDING(N)  is boolean true if dimension N of array A defined with to .
S'DELAYED(t)  is the signal value of S at time now - t .
S'STABLE      is true if no event is occurring on signal S.
S'STABLE(t)   is true if no even has occurred on signal S for t units of time.
S'QUIET       is true if signal S is quiet. (no event this simulation cycle)
S'QUIET(t)    is true if signal S has been quiet for t units of time.
S'TRANSACTION  is a bit signal, the inverse of previous value each cycle S is active.
S'EVENT       is true if signal S has had an event this simulation cycle.
S'ACTIVE      is true if signal S is active during current simulation cycle.
S'LAST_EVENT  is the time since the last event on signal S.
S'LAST_ACTIVE  is the time since signal S was last active.
S'LAST_VALUE  is the previous value of signal S.
S'DRIVING     is false only if the current driver of S is a null transaction.
S'DRIVING_VALUE  is the current driving value of signal S.
E'SIMPLE_NAME  is a string containing the name of entity E.
E'INSTANCE_NAME  is a string containing the design hierarchy including E.
E'PATH_NAME  is a string containing the design hierarchy of E to design root.
```



LAB SEQUENTIAL CIRCUITS

Extended counter

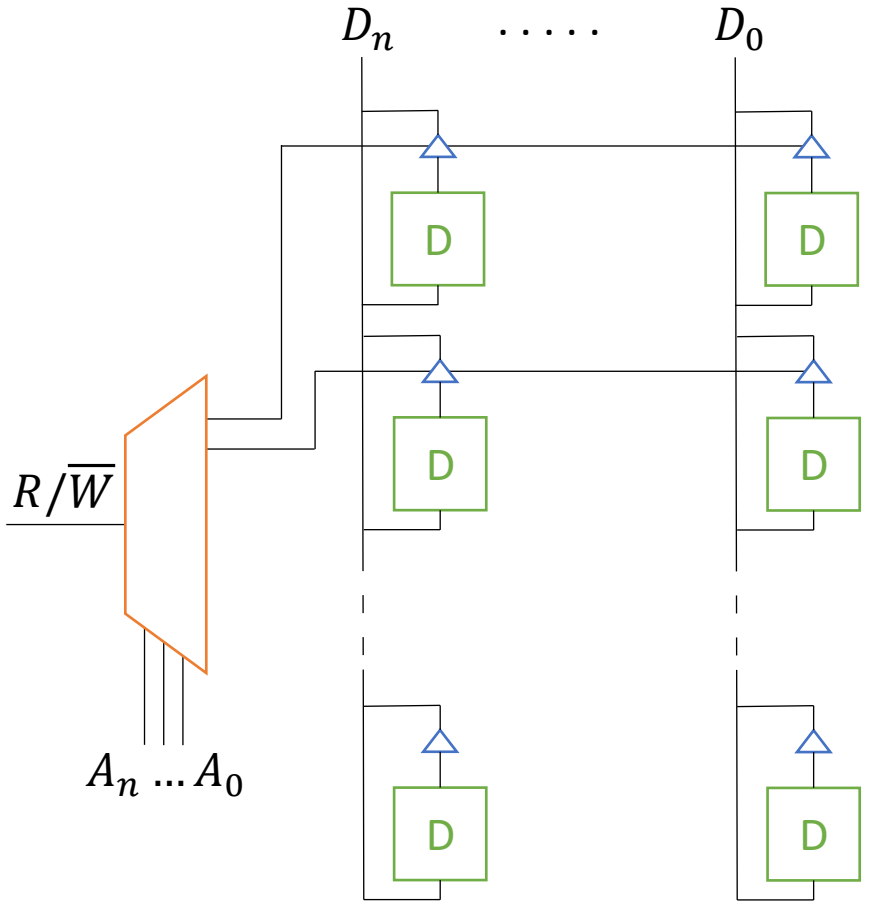
- Goal: make a counter able to increase by one/three or decrease by one/three depending on a control
  - +1/+3/-1/-3 modes
  - Use the leds as counter display
  - Generate internally a control that periodically change
- Simulate
- Implement
  - Upload the bitstream here http://fpgatrio.zapto.org



EX COUNTER

Memory

- It can be seen as an array of FF
- A decoder is used for addressing the data raw
- A tristate allows read/write operation
- Memory size = data width x $2^{\text{address width}}$
- Random Access Memory (RAM)
  - If Read Only is called ROM
  - It can contain a truth table -> It implements any combinatorial function
- Synchronous RAM can be easily synthetized in FPGA

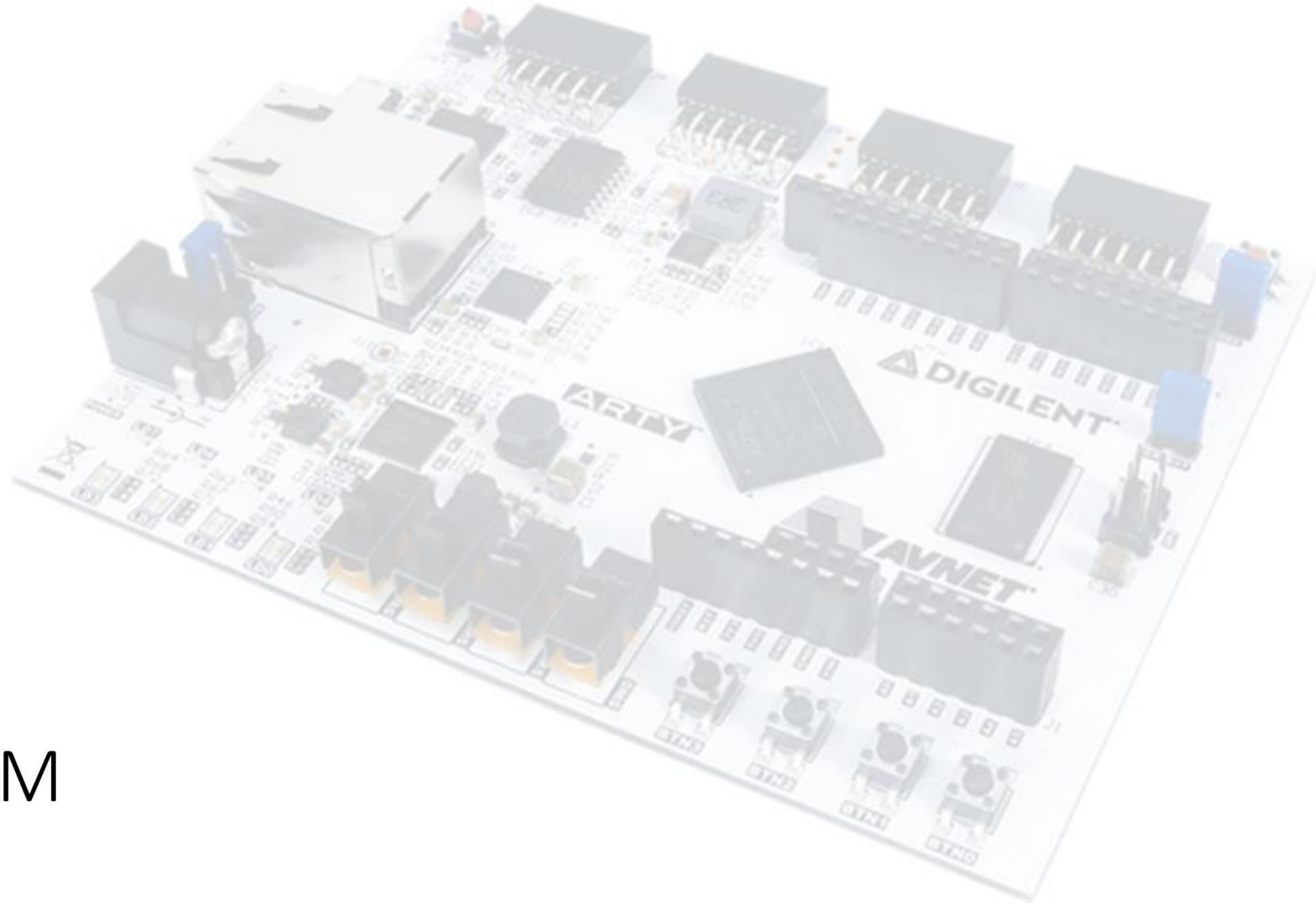


Memory

- RAM in VHDL

```
architecture Behavioral of ram_ent is
  type ram_type is array (31 downto 0)
       of std_logic_vector (3 downto 0);
  signal RAM : ram_type;
  signal read_a : std_logic_vector(4 downto 0);
begin
```

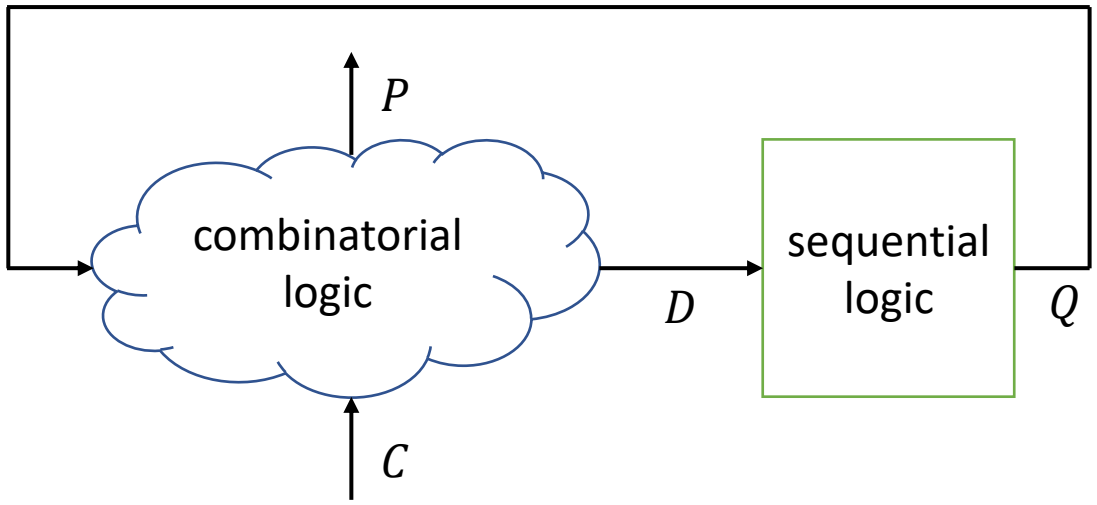
```
 process (clk)
  begin
    if rising_edge(Clk) then
      if (we = '1') then
        RAM(to_integer(unsigned(a))) <= di;
      end if;
        read_a <= a;
    end if;
  end process;
  do <= RAM(to_integer(unsigned(read_a)));
end Behavioral;
```



LAB RAM

State machine

- A general schema for functions that control sequential logic take into account external signals C and produce control signals P
  - Mealey state machine: P is function of C and Q
  - Moore state machine: P is only function of Q
- Moore outputs are synchronous



VHDL state machine

- There are several ways of encoding a state machine
  - One process (clocked process with a case statement)
  - Two processes (clocked process for changing state, combinatorial process for setting outputs)
  - Three processes (clocked process for changing state, combinatorial process for next state choice, combinatorial process for setting outputs)
- Other combinations of processes are also allowed
- Just a coding style

VHDL state machine

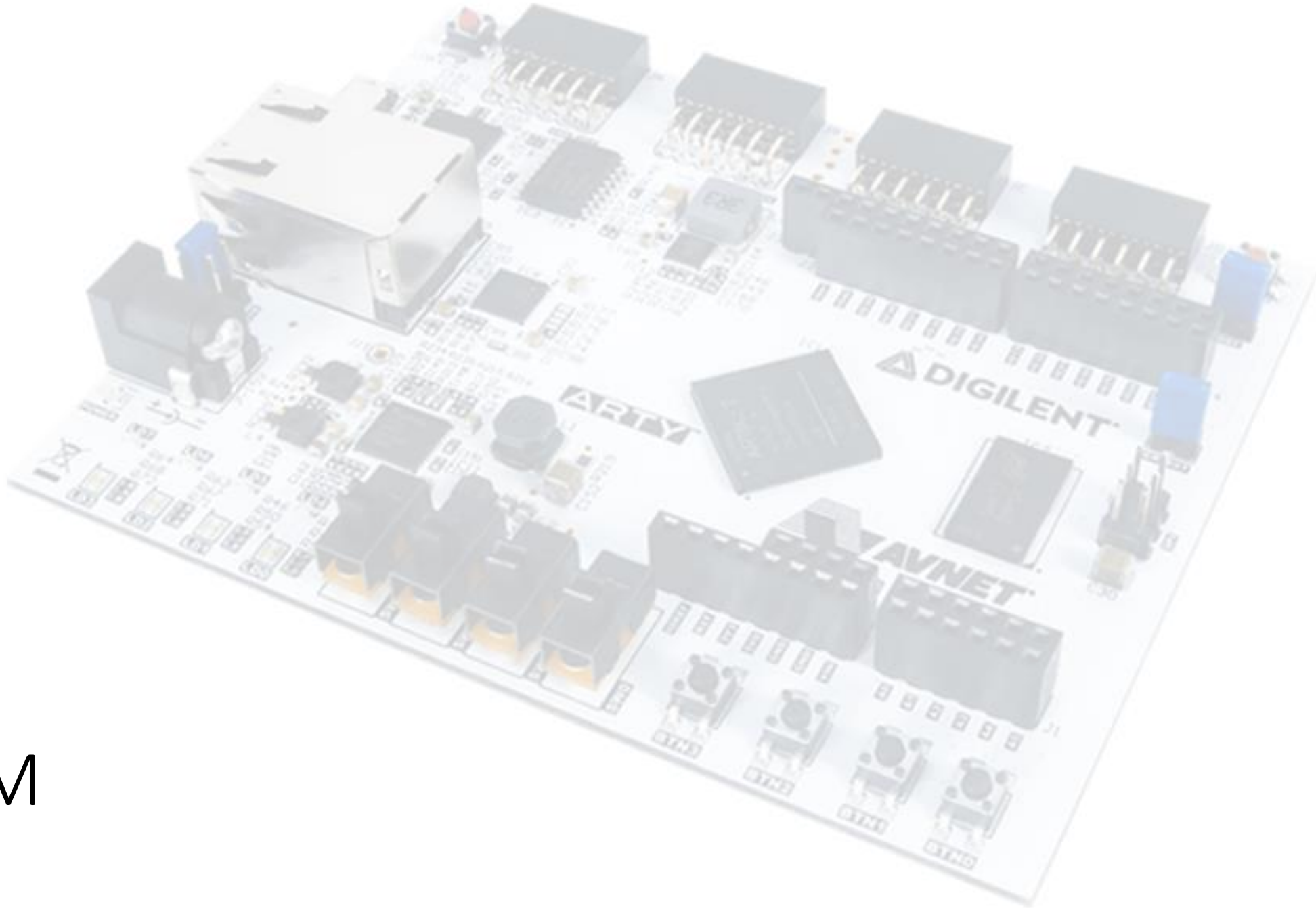
- One process state machine

```
process(Clk) is
begin
  if rising_edge(Clk) then
     if Rst = '1' then
        State <= S0;
        Dout  <= Value0;
     else
        case State is
          when S0 =>   if Condition0 then
                          State <= S1;
                          Dout <= Value1;
                       end if;
          when S1 =>   if Condition1 then
                          State <= S0;
                          Dout <= Value0;
                       end if;
          when others =>  Dout <= Value0;
                          State <= S0;
        end case;
     end if;
  end if;
end process;
```

VHDL state machine

- Two processes state machine

```
process(Clk) is
begin
  if rising_edge(Clk) then
     if Rst = '1' then
        State <= S0;
     else
        State <= NextState;
     end if;
  end if;
end process;

process(State, Condition0, Condition1) is
begin
   NextState <= State;
   case State is
     when S0 =>   Dout <= Value0;
                  if Condition0 then
                    NextState <= S1;
                  end if;
     when S1 =>   Dout <= Value1;
                  if Condition1 then
                    NextState <= S0;
                  end if;
     when others =>  Dout <= Value0;
                     NextState <= S0;
   end case;
  end if;
 end if;
end process;
```



LAB FSM

FSM detecting sequences

- Design two FSMs
  1. Johnson counter
  2. Blink each full sequence of the first
- Simulate
- Implement
  - Upload the bitstream here

    http://fpgatrio.zapto.org

A^n	B^n	C^n	A^{n+1}	B^{n+1}	C^{n+1}
0	0	0	1	0	0
1	0	0	1	1	0
1	1	0	1	1	1
1	1	1	0	1	1
0	1	1	0	0	1
0	0	1	0	0	0

EX FSM