# RT-level Functional Blocks

**Paolo PRINETTO**

Director
CINI Cybersecurity
National Laboratory

Paolo.Prinetto@polito.it

Mob. +39 335 227529

CYBER
CHALLENGE
CyberChallenge.IT

cini
Cybersecurity
National Lab

https://cybersecnatlab.it

# License & Disclaimer



## License Information



## Disclaimer

- We disclaim any warranties or representations as to the accuracy or completeness of this material.
- Materials are provided “as is” without warranty of any kind, either express or implied, including without limitation, warranties of merchantability, fitness for a particular purpose, and non-infringement.
- Under no circumstances shall we be liable for any loss, damage, liability or expense incurred or suffered which is claimed to have resulted from use of this material.

# Prerequisites

- Lecture:
  - HW_S_2 – Hardware Systems Representations

# Further readings

- Students interested in making a reference to a textbook on the arguments covered in this lecture can refer, for instance, to:
  - G. Conte, A. Mazzeo, N. Mazzocca, P. Prinetto: “Architettura dei calcolatori”, Città Studi, 2015 (App.D - Componenti funzionali a livello RT) (In Italian)

# Further readings

- M. Morris Mano, C.R.Kime: “Logic and Computer Design Fundamentals,” 5th edition updated Boston : Pearson, [2016]

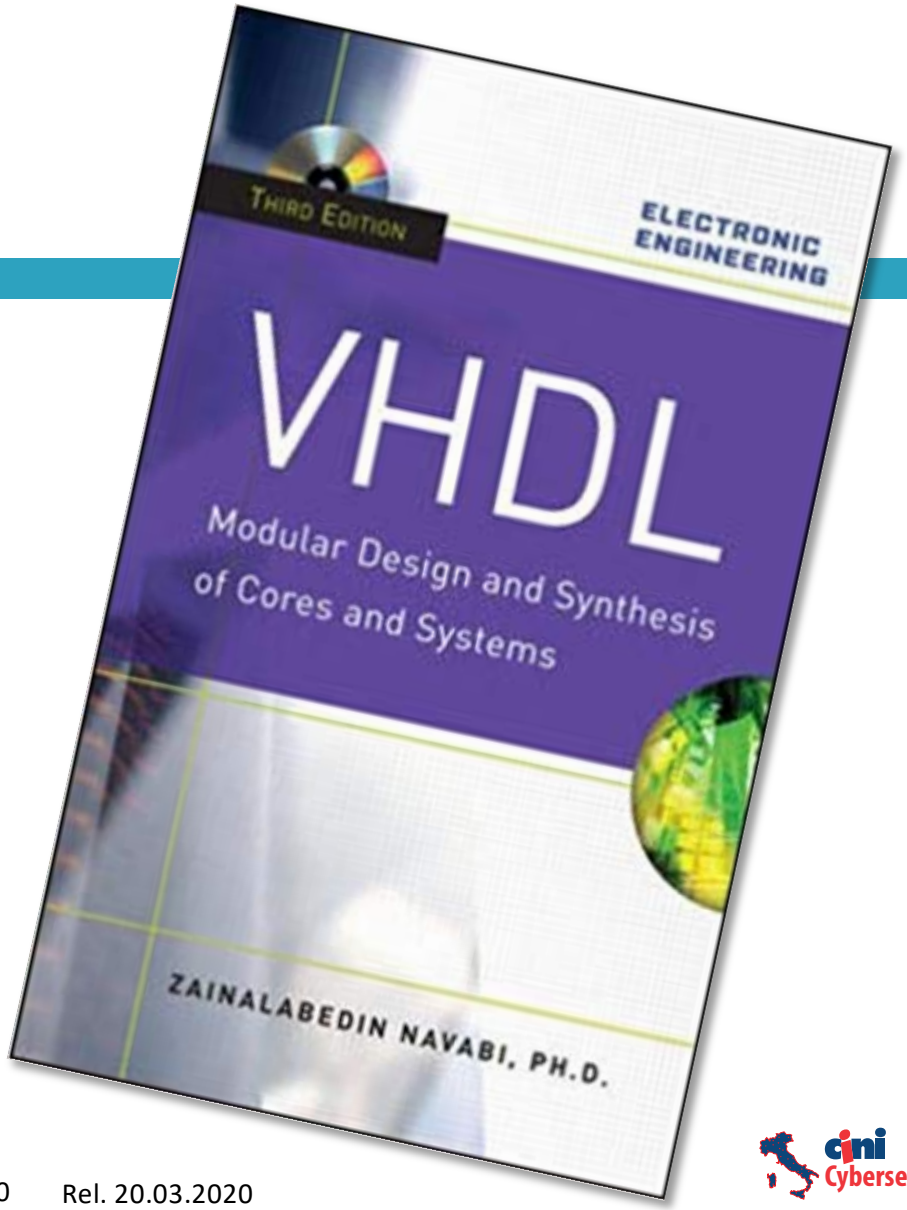
# Remarks



- For most of the presented basic blocks, their VHDL behavioral description is provided.

# Reference book

- Zainalabedin NAVABI:
  *VHDL: Modular Design and Synthesis of Cores and Systems*
  3rd edition
  McGraw Hill, May 2007



# On-line free course



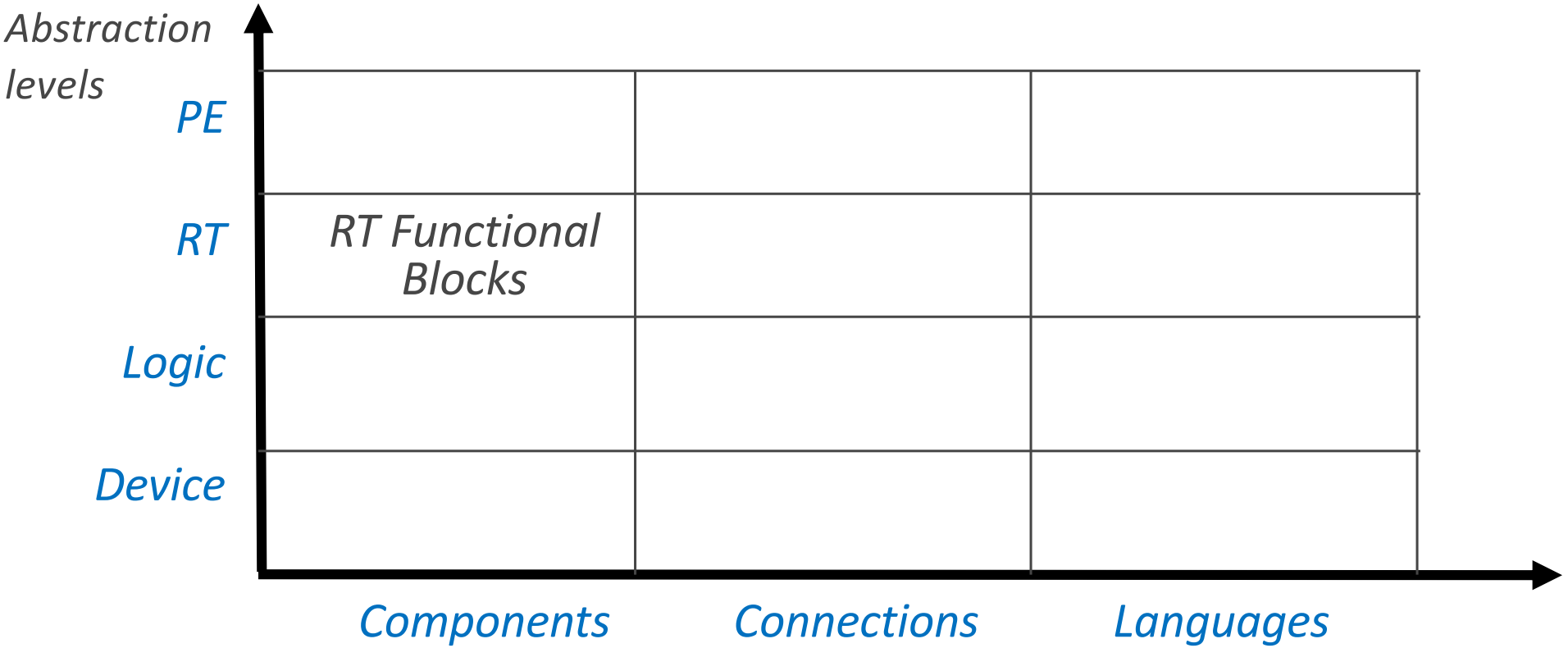
- *VHDL Basics* (OHDL1110) [93 m]
  - https://www.intel.com/content/www/us/en/programmable/support/training/course/ohdl1110.html

# Goal



- This lecture aims at presenting the set of functional blocks that are usually considered to be “elementary” or “basic” in the Structural domain at the RT abstraction level.

# Structural domain – RT level




 Rel. 20.03.2020

# Outline



- Combinational devices
- Sequential devices

# Basic RT-level Functional Blocks

## Combinational

- Adder-Subtracter
- Multiplexer
- Demultiplexer
- Priority Encoder
- Decoder
- Comparator
- ALU

## Sequential

- Registers
- Shift Registers
  - Parallel
  - Serial input
  - Barrel Shifter
- Counters
- Memories

# Basic RT-level Functional Blocks

## Combinational

- Adder-Subtracter
- Multiplexer
- Demultiplexer
- Priority Encoder
- Decoder
- Comparator
- ALU

## Sequential

- Registers
- Shift Registers
  - Parallel
  - Serial input
  - Barrel Shifter
- Counters
- Memories

# Basic RT-level Functional Blocks

## Combinational

- Adder-Subtracter
- Multiplexer
- Demultiplexer
- Priority Encoder
- Decoder
- Comparator
- ALU

## Sequential

- Registers
- Shift Registers
  - Parallel
  - Serial input
  - Barrel Shifter
- Counters
- Memories

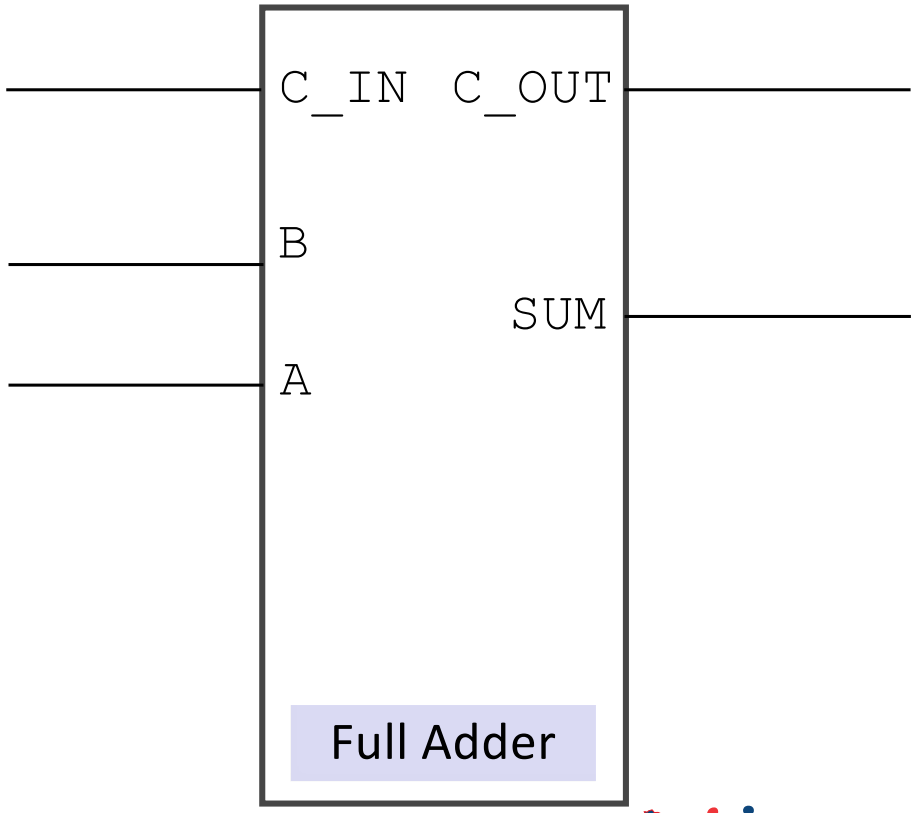
# Adder-Subtracter

- We shall consider:
  - full adder
  - N-bit adder-subtracter

# Full-Adder



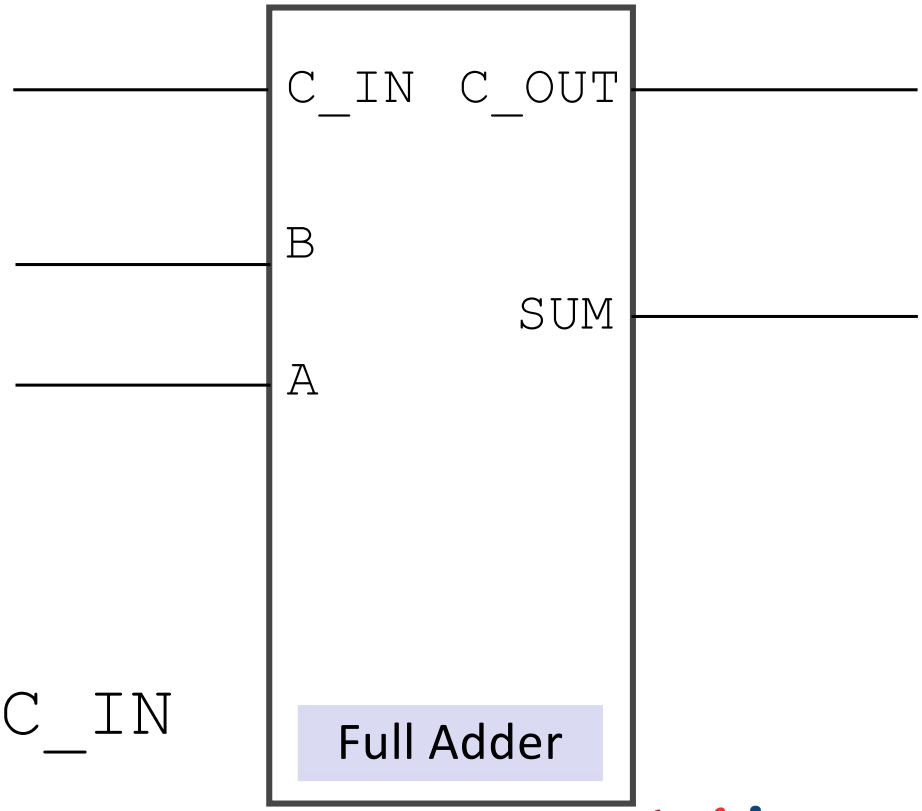
- It computes the binary sum of its 3 input bits (`A`, `B`, and `C_IN`) providing:
  - the sum `SUM`
  - the carry out `C_OUT`



# Full-Adder



- It computes the binary sum of its 3 input bits (A, B, and C_IN) providing:
  - the sum SUM
  - the carry out C_OUT



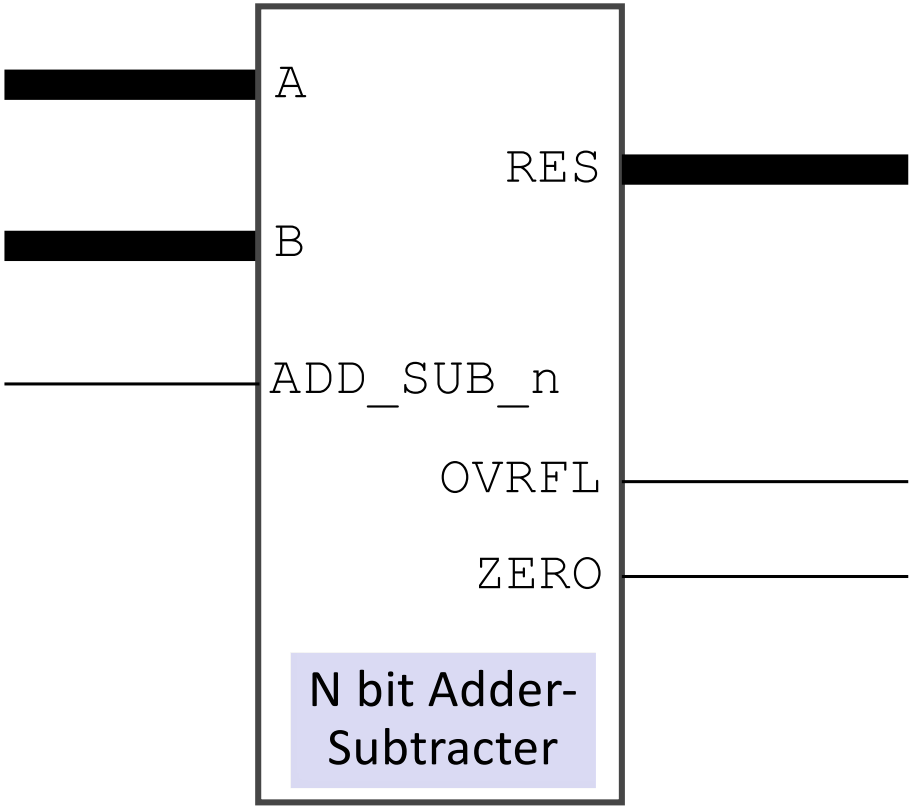

$$SUM = A \oplus B \oplus C\_IN$$

$$C\_OUT = A\,B + A\,C\_IN + B\,C\_IN$$

# N bit Adder-Subtracter



- A combinational block capable of adding/subtracting two n-bits inputs operands, detecting overflow conditions.
- It has
  - 2 n-bits data inputs, both labeled from n-1 to 0
  - 1 control input to select one of the two operations
  - 1 n-bits data output
  - 1 control output asserted when an overflow condition is occurred.



# Basic RT-level Functional Blocks

## Combinational

- Adder-Subtracter
- Multiplexer
- Demultiplexer
- Priority Encoder
- Decoder
- Comparator
- ALU

## Sequential

- Registers
- Shift Registers
  - Parallel
  - Serial input
  - Barrel Shifter
- Counters
- Memories

# Multiplexer - Mux

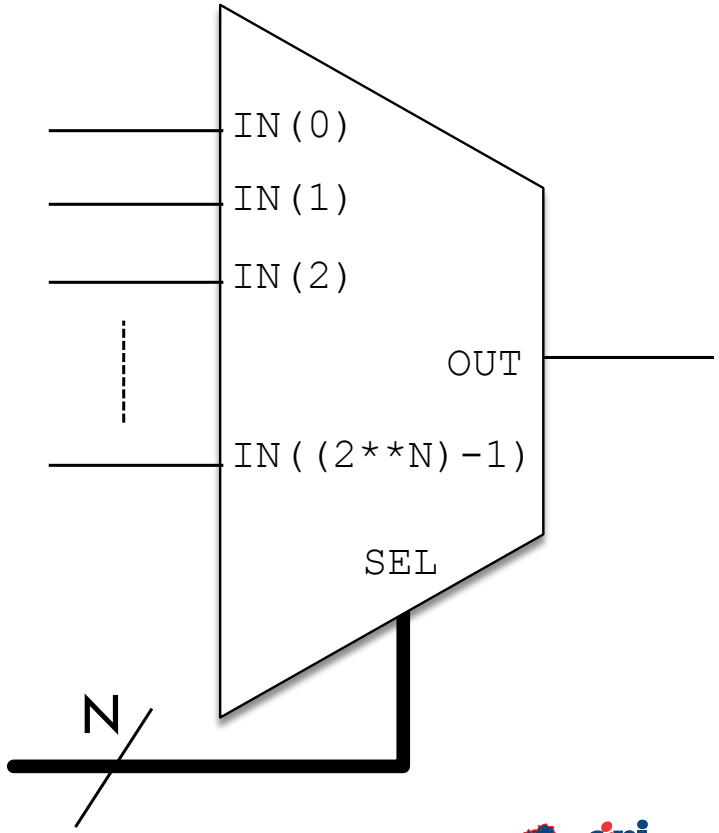


- A multiplexer is a combinational block capable of forcing its output to the current value of one of its data inputs, according to the values of some control signals.
- A Mux is usually used as a sort of “switch” to select one out of its many data inputs.

# Multiplexer



- A multiplexer has:
  - N control inputs
  - $2^N$ data inputs, labeled from $2^N$ -1 to 0
  - 1 data output that gets the value present on the input labeled j ($0 \leq j \leq N-1$), j being the binary number present on N-bit control inputs



# VHDL description

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
entity  mux is
generic(N    : integer := 3);
port(MUX_IN : in std_logic_vector((2**N)-1 downto 0);
     SEL    : in  unsigned(k-1 downto 0);
     MUX_OUT: out std_logic);
end mux;
```

# VHDL description

```
architecture beh of mux is
begin
  P1: process( MUX_IN, SEL )
    begin
       MUX_OUT <= MUX_IN(conv_integer(SEL));
    end process;
end beh;
```

# Basic RT-level Functional Blocks

## Combinational

- Adder-Subtracter
- Multiplexer
- Demultiplexer
- Priority Encoder
- Decoder
- Comparator
- ALU

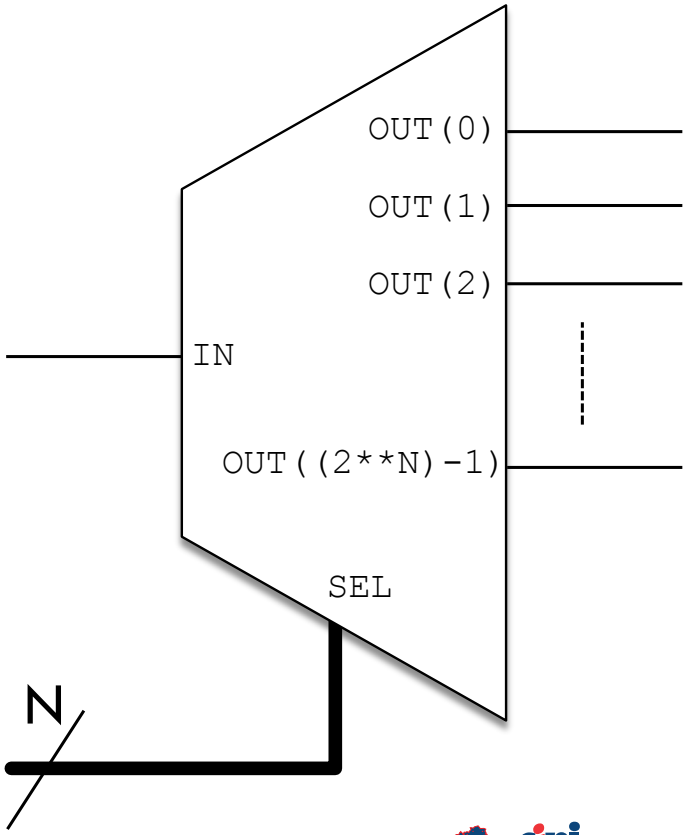
## Sequential

- Registers
- Shift Registers
  - Parallel
  - Serial input
  - Barrel Shifter
- Counters
- Memories

# Demultiplexer



- A multiplexer has:
  - N control inputs
  - 1 data input
  - $2^N$ data outputs, labeled from $2^N$ -1 to 0, such that:
    - the output labeled $2^j$ , j ($0 \leq j \leq N-1$) being the binary number present on N-bit control inputs, gets the value present on the data input
    - The remaining unselected outputs get a value that is implementation dependent



# Demultiplexer

- A demultiplexer practically acts as the “reverse” of a multiplexer.

# Basic RT-level Functional Blocks

## Combinational

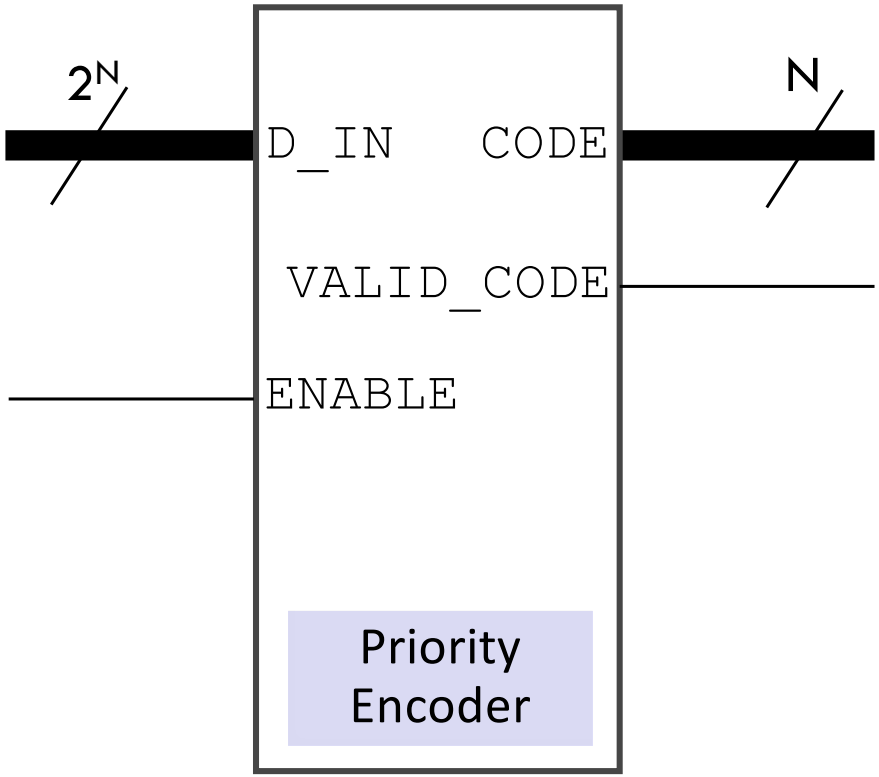
- Adder-Subtracter
- Multiplexer
- Demultiplexer
- Priority Encoder
- Decoder
- Comparator
- ALU

## Sequential

- Registers
- Shift Registers
  - Parallel
  - Serial input
  - Barrel Shifter
- Counters
- Memories

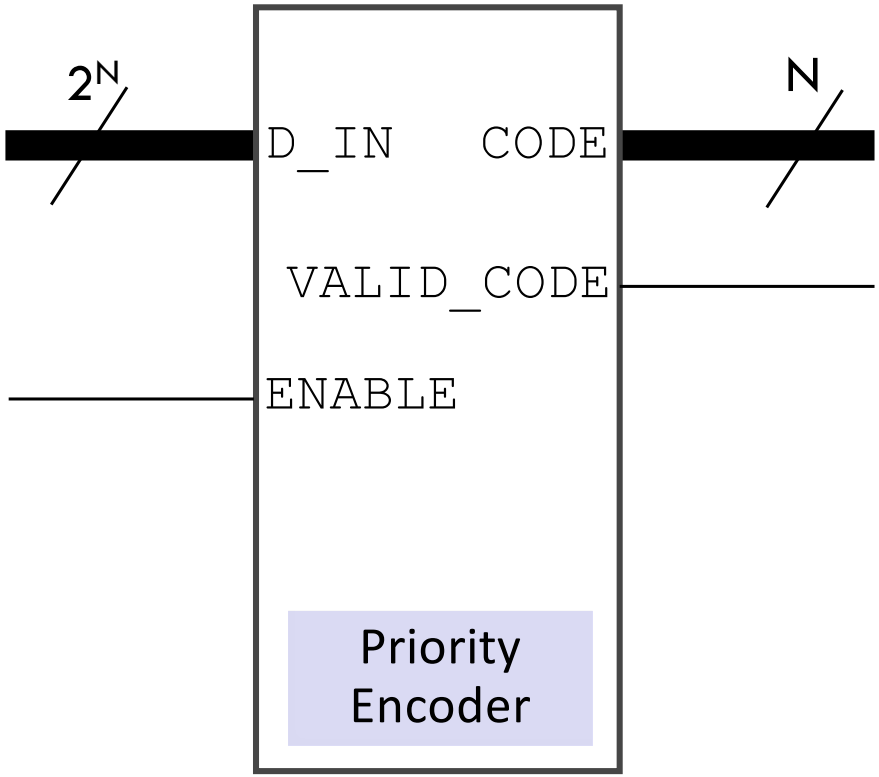
# Priority Encoder

- An Encoder $2^N \rightarrow N$ has:
  - $2^N$ inputs, labeled from $0$ to $2^N-1$ (each input is assigned a fixed priority)
  - $N$ outputs
  - $1$ `VALID_CODE` output
  - $1$ `ENABLE`



# Priority Encoder

- When enabled:
  - the outputs get a value corresponding to the binary encoding of *j* , *j* being the label of the asserted input having the highest priority
  - `VALID_CODE` is asserted.
- When disabled or when no input is asserted:
  - `VALID_CODE` is not asserted.



# VHDL description

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
entity priority_encoder is
generic(N    : integer := 3);
port(M       : in  std_logic_vector( (2**N)-1 downto 0);
     ENABLE  : in std_logic ;
     VALID   : out std_logic;
     CODE    : out unsigned(N-1 downto 0));
end priority_encoder;
```

# VHDL description

```
architecture beh of priority_encoder is
begin
  P1: process( M, ENABLE )
   variable k : integer range 0 to (2**N);
   variable found : boolean;
     begin
        k:= (2**N);  -- while statement's initial conditions
        found := false; -- while statement's exit condition
        while ((k > 0) and (not found))  loop
            k := k-1;
            if  M(k) = '1' then
              found := true ;
            end if;
        end loop;
        CODE <=  conv_unsigned( k, N ) ;
        if found and ENABLE='1' then
            VALID <= '1';
        else
            VALID <= '0';
        end if;
  end process;
end beh;
```

# Basic RT-level Functional Blocks

## Combinational

- Adder-Subtracter
- Multiplexer
- Demultiplexer
- Priority Encoder
- Decoder
- Comparator
- ALU

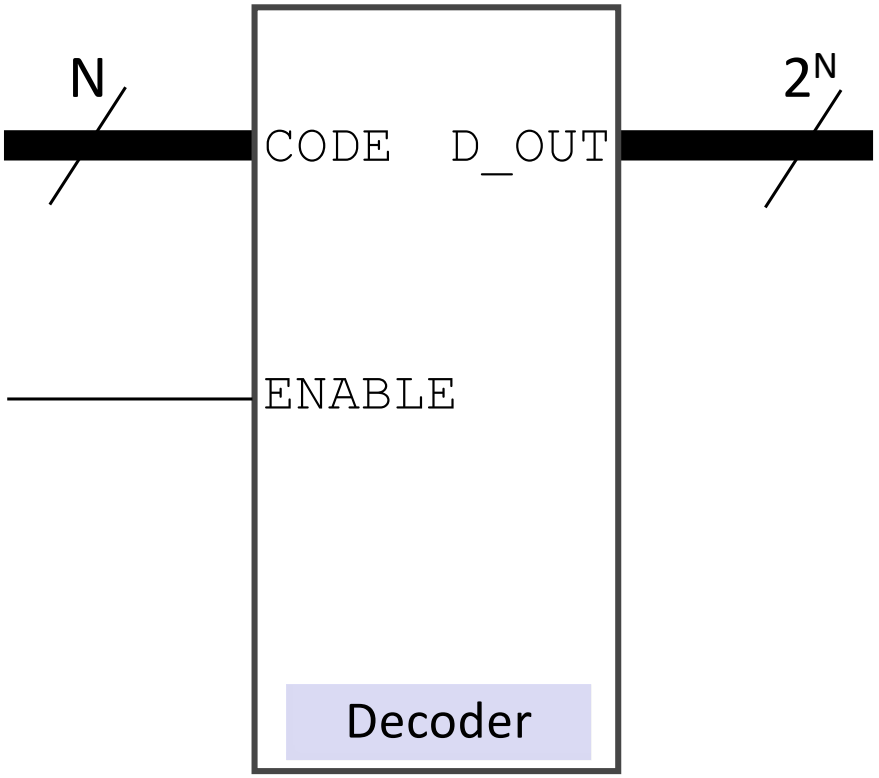
## Sequential

- Registers
- Shift Registers
  - Parallel
  - Serial input
  - Barrel Shifter
- Counters
- Memories

# Decoder



- A *decoder* $N \rightarrow 2^N$ has:
  - *1 N-bit* data input
  - $2^N$ outputs, labeled from *0* to $2^N$-*1*
  - *1* `enable`.
- When enabled, just the output labeled $j = 2^N$-*1* is asserted.
  When disabled, no output is asserted.



# VHDL description

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
entity decoder is
generic( N: integer := 3);
port(
   CODE   : in  unsigned(n-1 downto 0);
   ENABLE : std_logic ;
   DOUT   : out std_logic_vector( (2**N)-1 downto 0)
);
end decoder;
```

# VHDL description

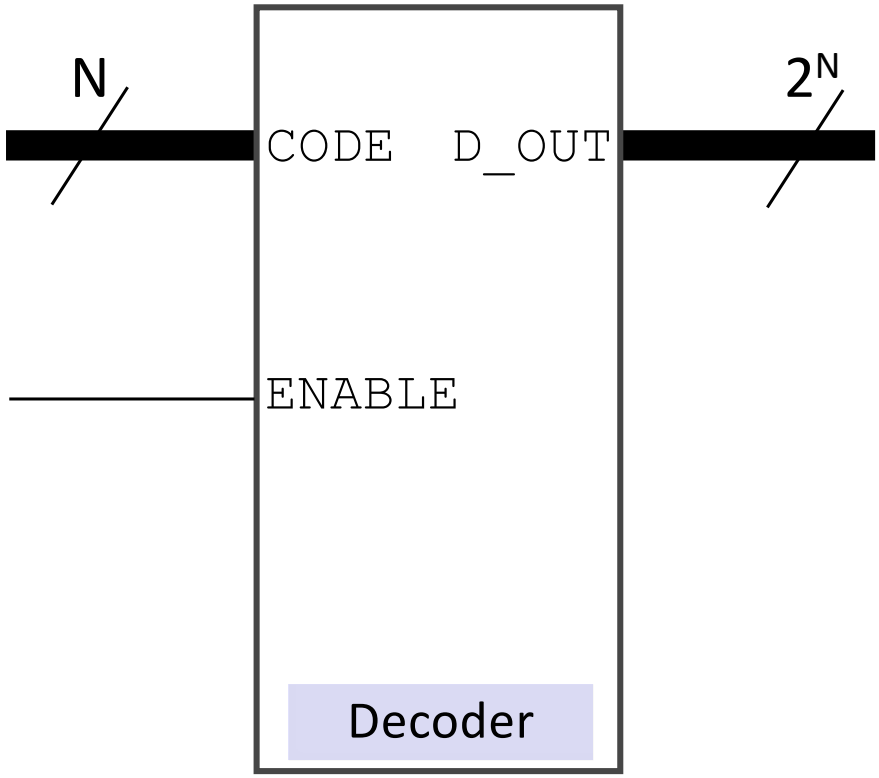
```
architecture beh of decoder is
begin
P1: process( CODE, ENABLE )
  variable sel: integer range 0 to 2**n -1;
    begin
      sel := conv_integer(CODE);
      for k in DOUT 'range loop
        if (ENABLE='1') and (k = sel) then
          DOUT (k) <= '1';
        else
          DOUT (k) <= '0';
        end if;
      end loop;
  end process;
end beh;
```

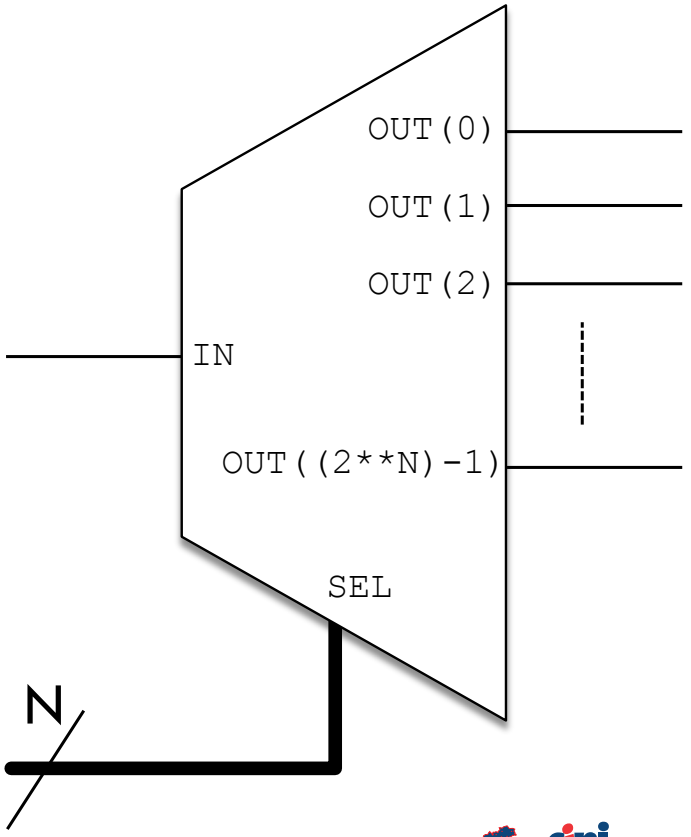
# Decoders vs Demultiplexers



- Decoders and Demultiplexers are conceptually interchangeable, simply swapping data & control signals

# Decoders vs Demultiplexers



# Basic RT-level Functional Blocks

## Combinational

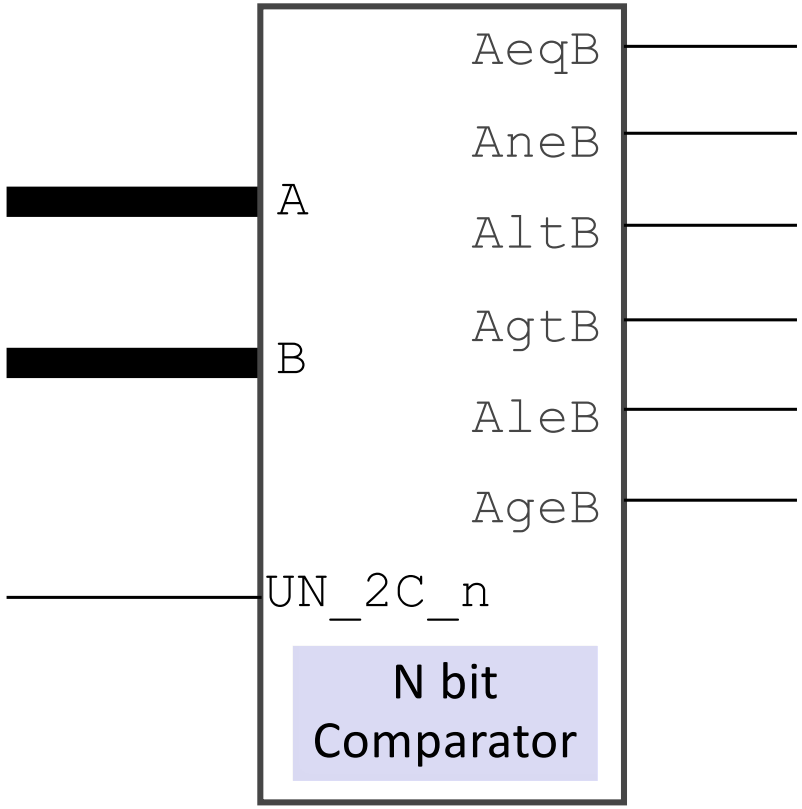
- Adder-Subtracter
- Multiplexer
- Demultiplexer
- Priority Encoder
- Decoder
- Comparator
- ALU

## Sequential

- Registers
- Shift Registers
  - Parallel
  - Serial input
  - Barrel Shifter
- Counters
- Memories

# Comparator

- It gets 2 n-bit binary numbers, `A` and `B`
- It provides in output the result of the comparison between `A` and `B`.
- A control input `UM_2C_n` specifies whether the input operands are unsigned or signed, respectively.



# Basic RT-level Functional Blocks

## Combinational

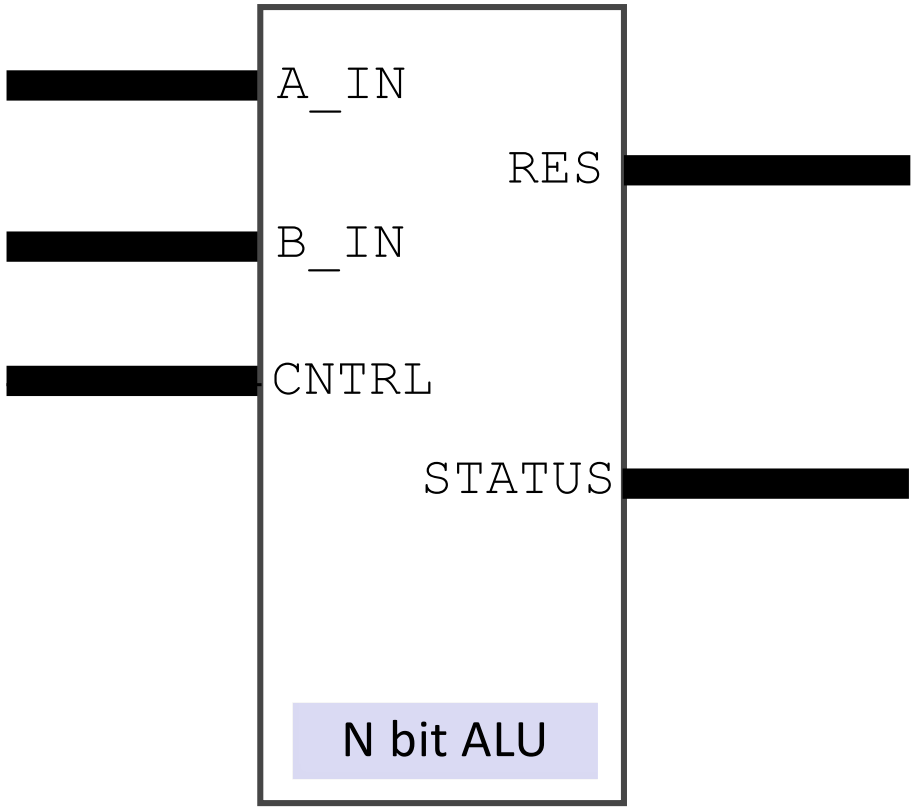
- Adder-Subtracter
- Multiplexer
- Demultiplexer
- Priority Encoder
- Decoder
- Comparator
- ALU

## Sequential

- Registers
- Shift Registers
  - Parallel
  - Serial input
  - Barrel Shifter
- Counters
- Memories

# ALU - Arithmetic Logic Unit

- It gets 2 n-bit binary numbers, `A` and `B`
- It performs logic and/or arithmetic operations on the 2 input operands, under the control of input control signals
- It provides both the result and some status signals



# Example: ( … old times ..)
# Functions implemented by the ALU ’181

| Input | | | | Functions | |
|---|---|---|---|---|---|
| S3 | S2 | S1 | S0 | arithmetic (M=0) | logic (M=1) |
| 0 | 0 | 0 | 0 | F=A minus 1 plus CIN | F=A' |
| 0 | 0 | 0 | 1 | F=AB minus 1 plus CIN | F=A'+ B' |
| 0 | 0 | 1 | 0 | F=AB' minus 1 plus CIN | F=A'+ B |
| 0 | 0 | 1 | 1 | F=1111 plus CIN | F= 1111 |
| 0 | 1 | 0 | 0 | F=A plus (A+B') plus CIN | F=A'B' |
| 0 | 1 | 0 | 1 | F=AB plus (A+B') plus CIN | F=B' |
| 0 | 1 | 1 | 0 | F=A minus B minus 1 plus CIN | F=A ⊕ B' |
| 0 | 1 | 1 | 1 | F=A+B' plus CIN | F=A+B' |
| 1 | 0 | 0 | 0 | F=A plus (A+B) plus CIN | F=A'B |
| 1 | 0 | 0 | 1 | F=A plus B plus CIN | F=A ⊕ B |
| 1 | 0 | 1 | 0 | F=AB' plus (A+B) plus CIN | F=B |
| 1 | 0 | 1 | 1 | F=A+B plus CIN | F=A+B |
| 1 | 1 | 0 | 0 | F=A plus A plus CIN | F=0000 |
| 1 | 1 | 0 | 1 | F=AB plus A plus CIN | F=AB' |
| 1 | 1 | 1 | 0 | F=AB' plus A plus CIN | F=AB |
| 1 | 1 | 1 | 1 | F=A plus CIN | F=A |

CYBER CHALLENGE
CyberChallenge.IT


Rel. 20.03.2020

cini Cybersecurity National Lab

# IUs & FPUs



- Modern CPUs contain very powerful and complex ALUs, often referred to as *Integer Units* (IUs)
- In addition to ALUs, modern CPUs contain *Floating-Point Units* (FPUs) specially designed to carry out operations on floating-point numbers. Typical operations include addition, subtraction, multiplication, division, and square root.

# Basic RT-level Functional Blocks

## Combinational

- Adder-Subtracter
- Multiplexer
- Demultiplexer
- Priority Encoder
- Decoder
- Comparator
- ALU

## Sequential

- Registers
- Shift Registers
  - Parallel
  - Serial input
  - Barrel Shifter
- Counters
- Memories

# General remarks



- All the sequential blocks share some commonalities:
  - *Clock* signal
  - *Asynchronous master reset* signal
  - *Synchronous reset* signal
  - *Enable* signal
  - Several *Control signals*, whose mutual priorities are strictly implementation dependent, having they been defined at the design time.

# Basic RT-level Functional Blocks

## Combinational

- Adder-Subtracter
- Multiplexer
- Demultiplexer
- Priority Encoder
- Decoder
- Comparator
- ALU

## Sequential

- Registers
- Shift Registers
  - Parallel
  - Serial input
  - Barrel Shifter
- Counters
- Memories

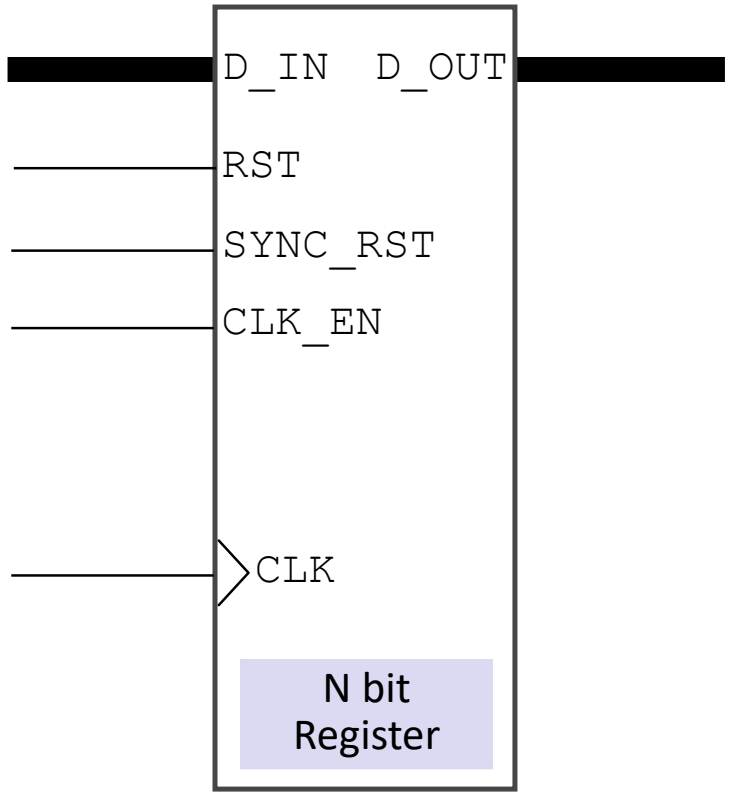
# Register

- An n-bit register is a sequential functional block able to perform the following operations:
  - asynchronous clear
  - synchronous clear
  - n-bit data parallel load
  - stored data  hold

# Register: I/O signals

- Input Data signals: `D_IN(n-1 downto 0)`
- Output Data signals: `D_OUT(n-1 downto 0)`
- Clock signal: `CLK`
- Reset signal: `RST`
- Input Control signals: `CLK_EN`, `SYNC_CLR`
- Output Control signals: none

# Register: I/O signals



# VHDL description

```
library ieee;
use ieee.std_logic_1164.all;

entity pdt_register is
  generic(N:positive :=4);
  port    (d_in : in std_logic_vector(N-1 downto 0);
           clk_en, clk, rst, sync_clr : in std_logic;
           d_out : out std_logic_vector(N-1 downto 0)
          );
 end pdt_register;
```

# VHDL description

```
architecture pdt_register_arch of pdt_register is
  begin
    process(clk, rst)
      begin
        if rst='1' then
          d_out<=(others=>'0’);
        elsif rising_edge(clk) then
          if clk_en='1' then
            if sync_clr='1' then
              d_out<=(others=>'0');
            else
              d_out<=d_in;
           end if;
          end if;
        end if;
     end process;
 end pdt_register_arch;
```

# Basic RT-level Functional Blocks

## Combinational

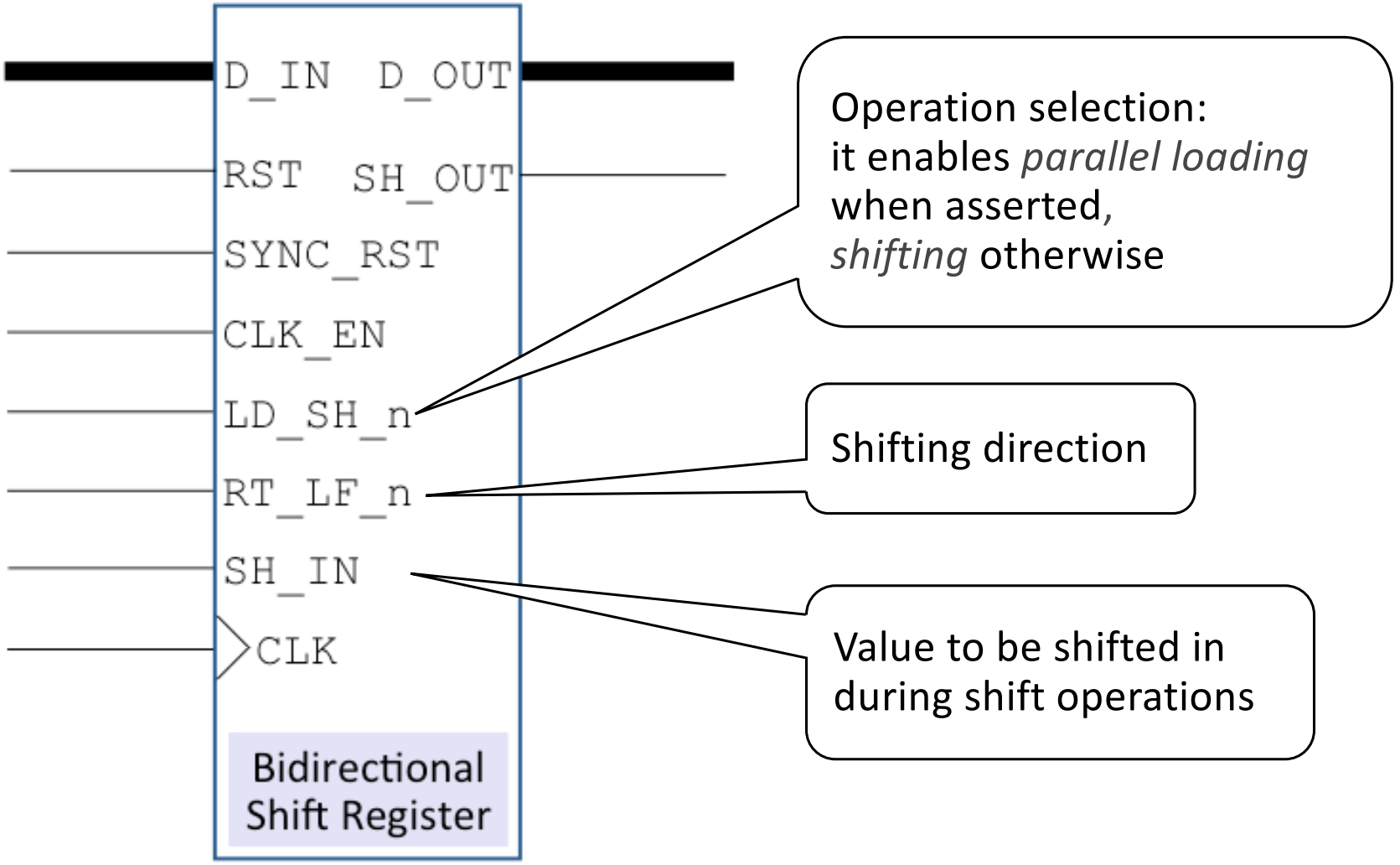
- Adder-Subtracter
- Multiplexer
- Demultiplexer
- Priority Encoder
- Decoder
- Comparator
- ALU

## Sequential

- Registers
- Shift Registers
  - Parallel
  - Serial input
  - Barrel Shifter
- Counters
- Memories

# Shift register

- A shift register is a sequential functional block able to perform the following operations:
  - parallel load of an n-bit data
  - stored data retention
  - asynchronous clear
  - synchronous clear
  - shifting right or left its content of 1 position


D_IN
D_OUT
RST
SH_OUT
SYNC_RST
CLK_EN
LD_SH_n
RT_LF_n
SH_IN
CLK
Bidirectional Shift Register
Operation selection: it enables *parallel loading* when asserted, *shifting* otherwise
Shifting direction
Value to be shifted in during shift operations

# VHDL description

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity pdt_shift_register is
generic(N:positive:=4);
port(d_in: in std_logic_vector(N-1 downto 0);
     rst: in std_logic;
     sync_clr : in std_logic;
     clk_en: in std_logic;
     ld_sh_n: in std_logic;
     sh_nl: in std_logic;
     clock: in std_logic;
     d_out: out std_logic_vector(N-1 downto 0));
end pdt_shift_register;
```

# VHDL description

```
architecture pdt_shift_register_arch of
pdt_shift_register is
  signal reg: std_logic_vector(N-1 downto 0);
  begin
  p0: process(clock,async_clr)
  begin
    if async_clr='1' then
      reg <= conv_std_logic_vector(0,n);
    elsif rising_edge(clk) then
```

# VHDL description

```
if clk_en='1' then
  if sync_clr='1' then
    reg <= conv_std_logic_vector(0,n);
  else
    if ld_nsh='1' then  -- value loading
      reg <= d_in;
    else                      -- shift of the previous value
      if r_nl='1' then  -- right direction
        reg <= r_in&reg(n-1 downto 1);
      else                      -- left direction
        reg <= reg(n-2 downto 0)&l_in;
      end if;     -- r_nl
    end if;                   -- ld_nsh
  end if;                     -- sync_clr
 end if;                      -- clk_en
end if;             -- async_clr
end process p0;
```

# Basic RT-level Functional Blocks

## Combinational

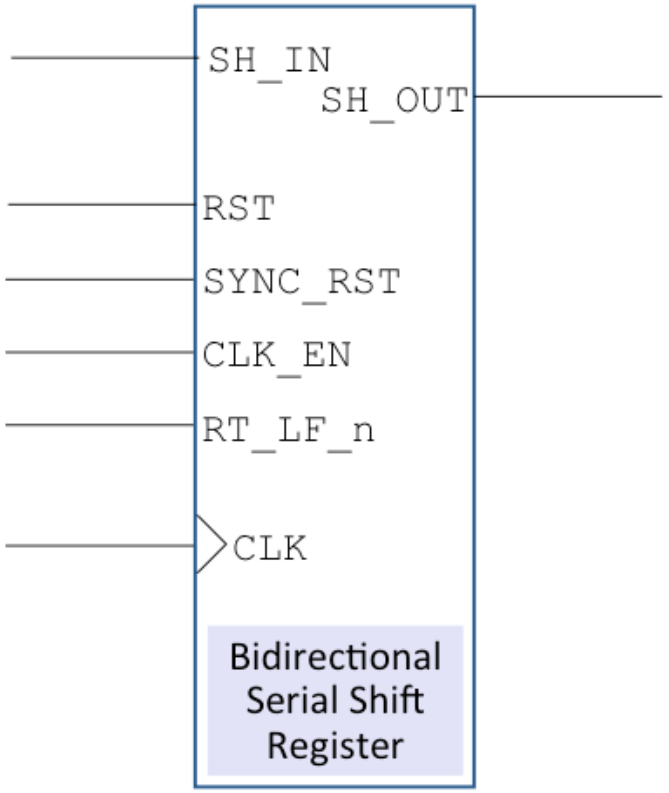
- Adder-Subtracter
- Multiplexer
- Demultiplexer
- Priority Encoder
- Decoder
- Comparator
- ALU

## Sequential

- Registers
- Shift Registers
  - Parallel
  - Serial input
  - Barrel Shifter
- Counters
- Memories

# Serial Input Shift register

- A serial shift register is a shift register without any parallel inputs.



# Basic RT-level Functional Blocks

## Combinational

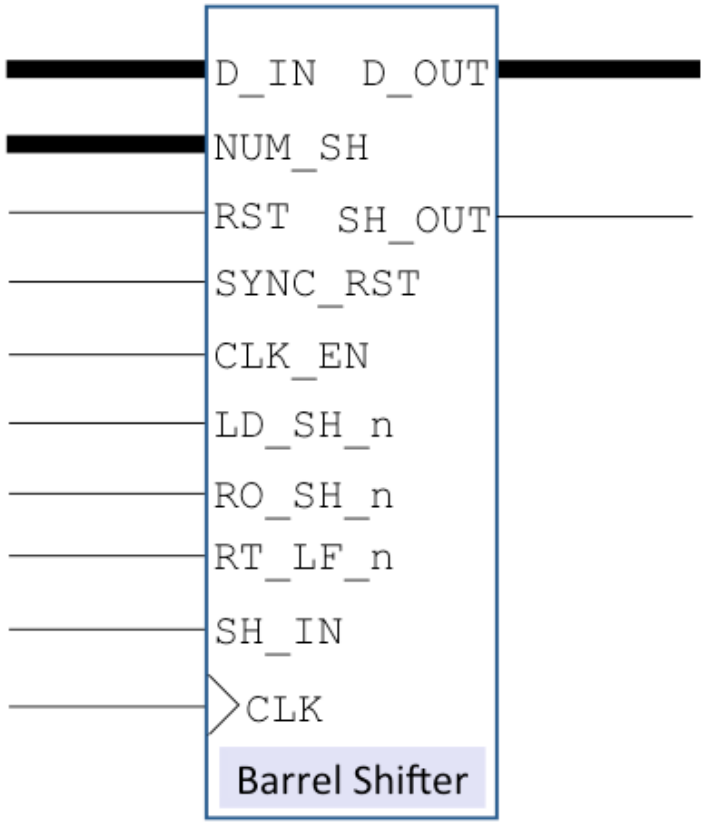
- Adder-Subtracter
- Multiplexer
- Demultiplexer
- Priority Encoder
- Decoder
- Comparator
- ALU

## Sequential

- Registers
- Shift Registers
  - Parallel
  - Serial input
  - Barrel Shifter
- Counters
- Memories

# Barrel shifter

- A Barrel shifter is a shift register that, at any clock cycle, can shift its contents of N positions.



# Basic RT-level Functional Blocks

## Combinational

- Adder-Subtracter
- Multiplexer
- Demultiplexer
- Priority Encoder
- Decoder
- Comparator
- ALU

## Sequential

- Registers
- Shift Registers
  - Parallel
  - Serial input
  - Barrel Shifter
- Counters
- Memories

# Up-down modulo m counters

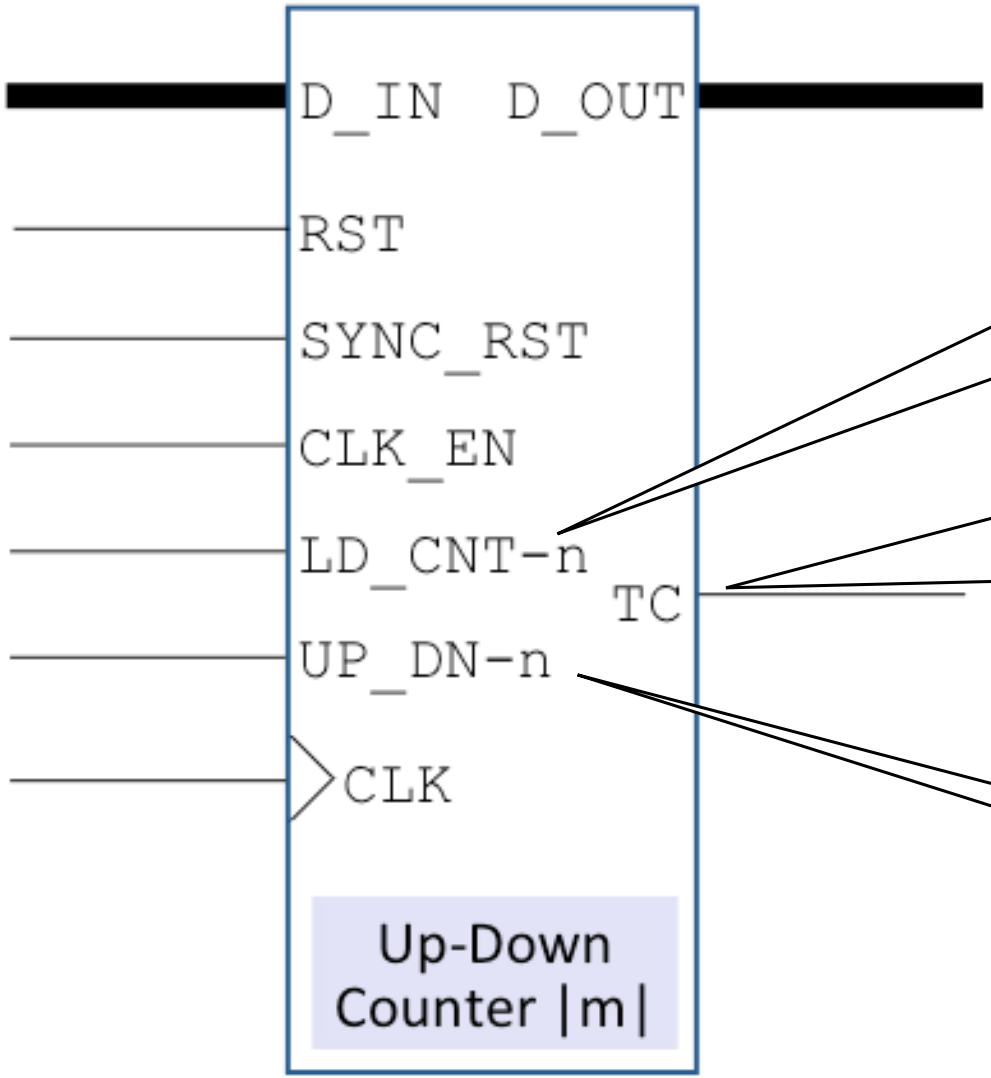
- An up-down modulo m counter is a sequential functional block able to perform the following operations:
  - asynchronous clear
  - synchronous clear
  - n-bit data parallel load
  - holding the stored data
  - incrementing, modulo m, the stored data
  - decrementing, modulo m, the stored data

# The concept of modulo

- Each counter is characterized by its modulo, i.e., the maximum number m of different output configurations it can get.
- It must obviously be:

$$m \leq 2^{\#\,FFs}$$

- A modulo m counter (often referred to as |m|) gets all the values between 0 and m -1.




Operation selection:
it enables *parallel loading* when asserted,
*counting* otherwise

Terminal Count: asserted when the counter is enabled and a counting limit is reached:
m-1 when counting up
0 when counting down

Counting direction

# VHDL description

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity pdt_counter_modm is
  generic (M: positive:=6;  -- counter modulo
           N: positive:=3); -- flip-flops number
  port(
    clk, rst,
    syn_clr, clk_en,
    up_dn_n, ld_cnt_n : in std_logic;
    inbus  : in std_logic_vector(N-1 downto 0);
    outbus : out std_logic_vector(N-1 downto 0);
    tc     : out std_logic );
end pdt_counter_modm;
```

```
architecture pdt_counter_modm_arch of
                          pdt_counter_modm is

signal val: integer range 0 to M-1;

begin

-- flip-flops' number control
  assert (M>2**(N-1)) report "too many flip-flop" severity failure;
  assert (M<=2**N) report "not enough flip flop" severity failure;

-- This process updates the value of signal val
  p0:process(clk,rst)
  begin
    if rst='1' then
      val<=0;
    elsif rising_edge(clk) then
```

```
if clk_en='1' then
  if syn_clr = '1' then           -- sysnchorous reset
    val <= 0;
        else
          if ld_cnt_n ='1' then
            val <= conv_integer(inbus) -- parallel loading
          else                        -- counting
            if up_dn_n = '1' then      -- up counting
              if val = M-1 then
                val <= 0;
              else
                val <= val+1;
              end if;
            else                           -- down counting
              if val = 0 then
                val <= M-1;
              else
                val <= val-1;
              end if;
            end if;   -- up_dn_n
        end if;     -- ld_cnt_n
      end if;       -- syn_clr
    end if;         -- clk_en
  end if;           -- rst
end process p0;
```

```
-- This process updates the value of Terminal Count

p1:process(val,up_dn,clk_en)
begin
  if ( clk_en='1' and up_dn_n = '1' and val=(M-1) ) or
         ( clk_en='1' and up_dn_n = '0' and val=0 )
  then
    tc <= '1';
  else
    tc <= '0';
  end if;
  -- assignment of output value
  outbus <= conv_std_logic_vector(val, N);

end process p1;
end pdt_counter_modm_arch;
```

# Basic RT-level Functional Blocks

## Combinational

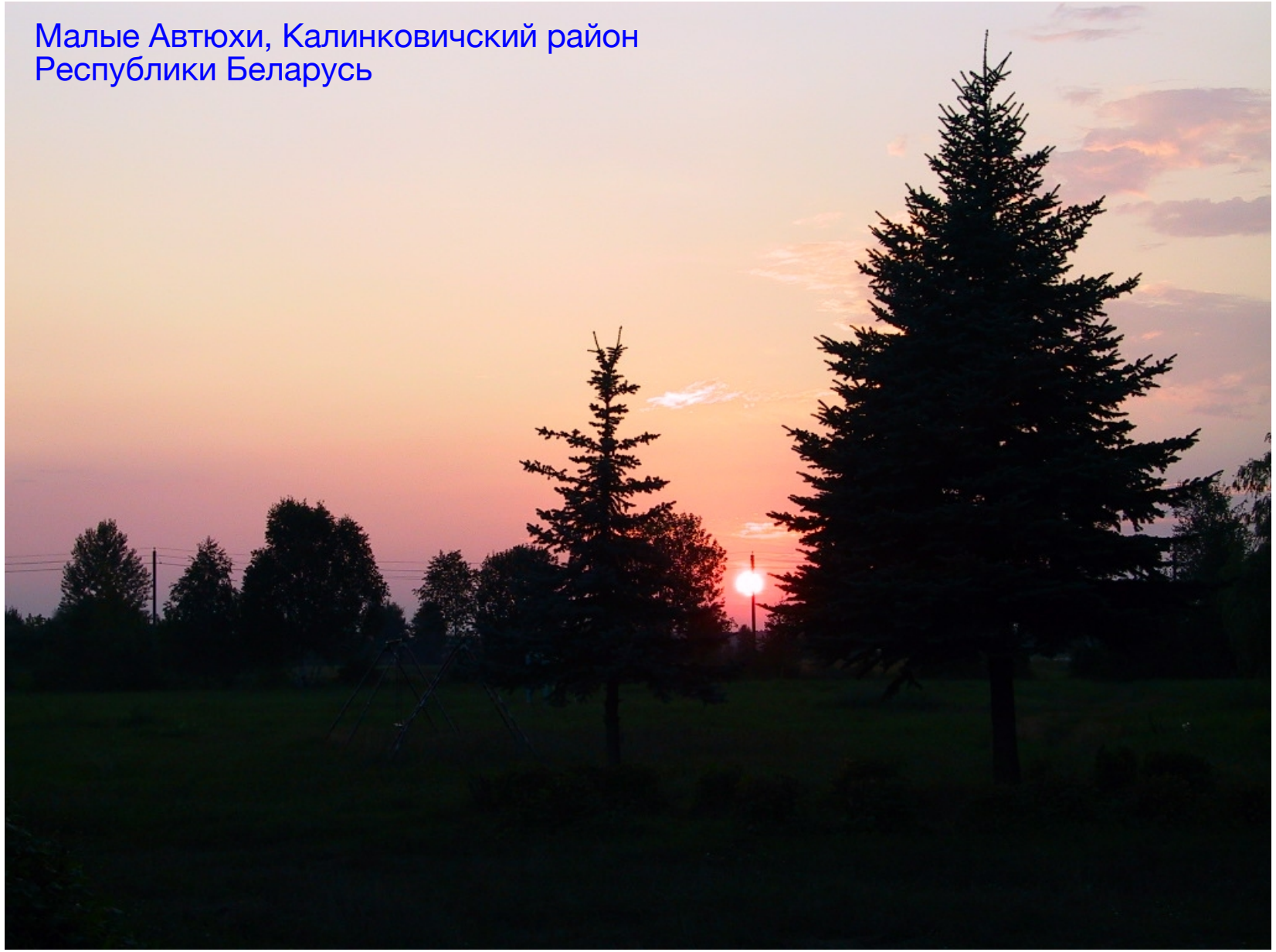
- Adder-Subtracter
- Multiplexer
- Demultiplexer
- Priority Encoder
- Decoder
- Comparator
- ALU

## Sequential

- Registers
- Shift Registers
  - Parallel
  - Serial input
  - Barrel Shifter
- Counters
- Memories

# Remark

- Memories are presented in detail in the Lectures:
  - *HW_C_3 – Memory Devices*
  - *HW_C_4 – Memory Technologies*
- LFSRs are presented in detail in the Lecture:
  - *HW_C_5 – Linear Feedback Shift Resisters - LFRSs*



Малые Автюхи, Калинковичский район
Республики Беларусь


**Paolo PRINETTO**

Director
CINI Cybersecurity
National Laboratory

Paolo.Prinetto@polito.it

Mob. +39 335 227529


CYBER CHALLENGE
CyberChallenge.IT

cini Cybersecurity National Lab

https://cybersecnatlab.it