VHDL

5051158-xxxx

Lecture 2

Digital logic basics - combinatorial logic

VHDL Basics – part 1

V 1.0 17.9.2020
Jarno Tuominen

TURKU AMK
TURKU UNIVERSITY OF
APPLIED SCIENCES

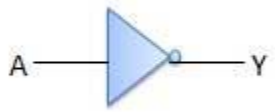
Contents of this lecture

- Digital logic basics (combinatorial logic)
- VHDL basics (for creating combinatorial logic)

Digital logic basics – combinatorial logic

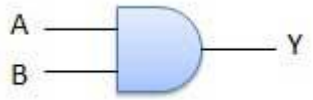
Logic gates

NOT



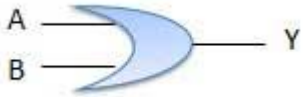
Inputs	Output
A	B
0	1
1	0

AND



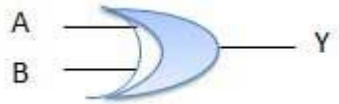
Inputs		Output
A	B	AB
0	0	0
0	1	0
1	0	0
1	1	1

OR



Inputs		Output
A	B	A + B
0	0	0
0	1	1
1	0	1
1	1	1

XOR



Inputs		Output
A	B	A ⊕ B
0	0	0
0	1	1
1	0	1
1	1	0

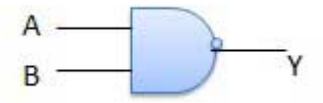
- Basic building blocks for all the digital logic
- The examples are 2-input gates, but there are wider gates as well (like 4-input ANDs and so on)

VHDL: Y **<= not** A**;** Y **<=** A **and** B**;** Y **<=** A **or** B**;** Y **<=** A **xor** B**;**

https://www.tutorialspoint.com/computer_logical_organization/logic_gates.htm

Logic gates (inverted output)

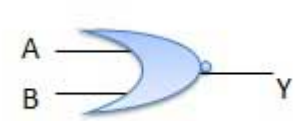
NAND



Inputs		Output
A	B	$\overline{AB}$
0	0	1
0	1	1
1	0	1
1	1	0

VHDL: Y <= A **nand** B;

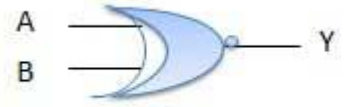
NOR



Inputs		Output
A	B	$\overline{A+B}$
0	0	1
0	1	0
1	0	0
1	1	0

Y <= A **nor** B;

XNOR



Inputs		Output
A	B	A ⊙ B
0	0	1
0	1	0
1	0	0
1	1	1

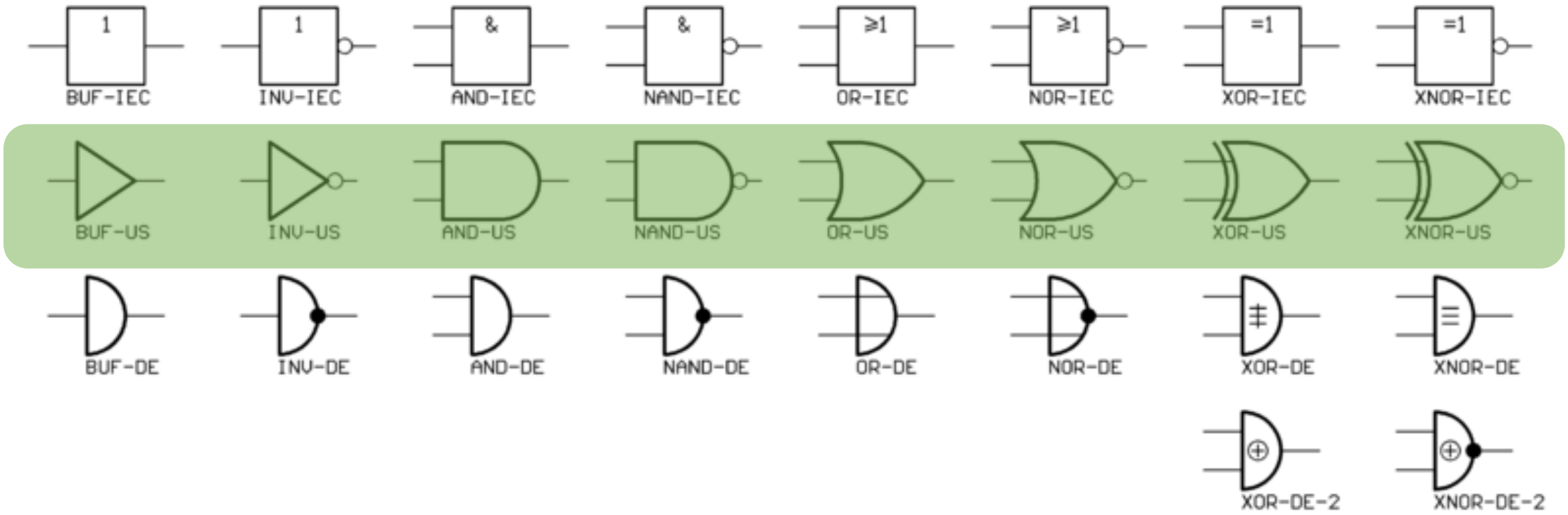
Y <= A **xnor** B;

- NAND is all you need:
  - A NAND B, where A=B is an inverter (NOT)
  - NAND becomes OR (and vice versa) if all the inputs and output are inverted (DeMorgan's theorems)

https://www.tutorialspoint.com/computer_logical_organization/demorgan_theroems.htm

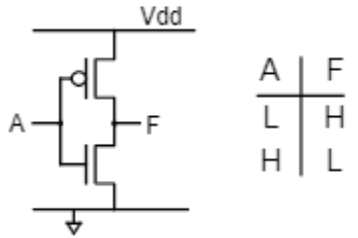
The drawing symbols vary…

- US style clearly the most popular



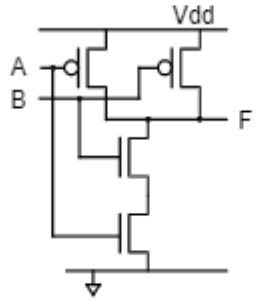
Why NAND/NOR-gates are favored in actual implementations?

- Because they require 2 (CMOS) transistors less than AND/OR
- Less transistors = smaller, cheaper, less power, faster (at least in this case)



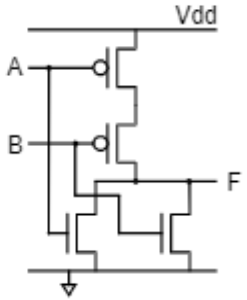
A	F
L	H
H	L

CMOS INVERTER



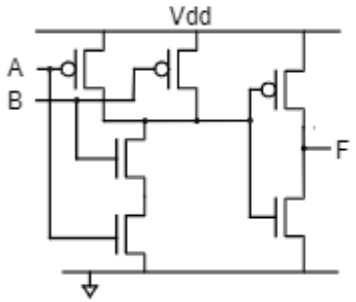
A	B	F
L	L	H
L	H	H
H	L	H
H	H	L

CMOS NAND



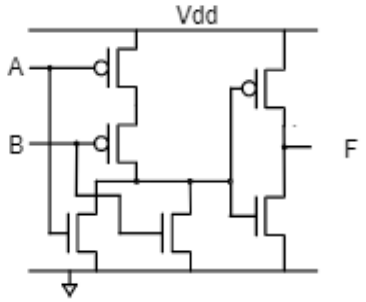
A	B	F
L	L	H
L	H	L
H	L	L
H	H	L

CMOS NOR



A	B	F
L	L	L
L	H	L
H	L	L
H	H	H

CMOS AND



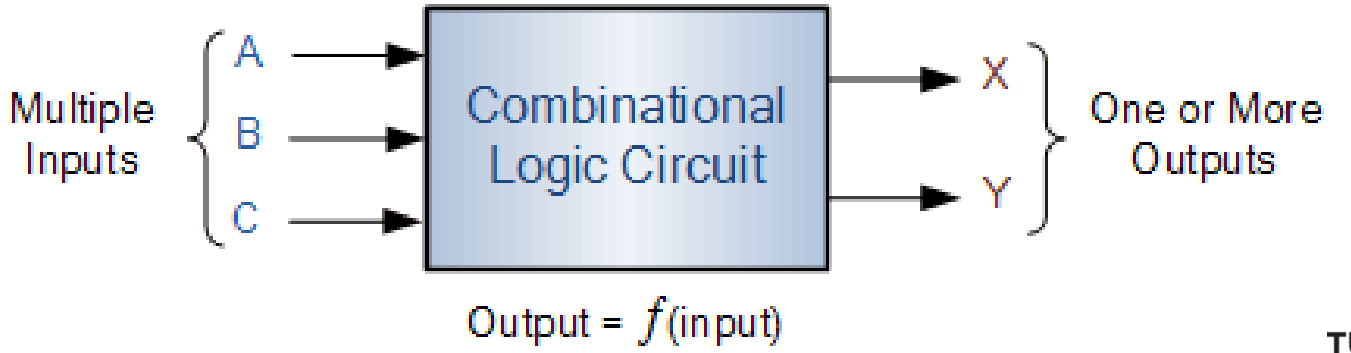
A	B	F
L	L	L
L	H	H
H	L	H
H	H	H

CMOS OR

https://learn.digilentinc.com/Documents/313

What is combinatorial logic?

- Unlike Sequential Logic Circuits whose outputs are dependant on both their present inputs and their previous output state giving them some form of *Memory*. The outputs of **Combinational Logic Circuits** are only determined by the logical function of their current input state, logic “0” or logic “1”, at any given instant in time.
- The result is that combinational logic circuits have no feedback, and any changes to the signals being applied to their inputs will immediately* have an effect at the output. In other words, in a **Combinational Logic Circuit**, the output is dependant at all times on the combination of its inputs. Thus a combinational circuit is *memoryless*.

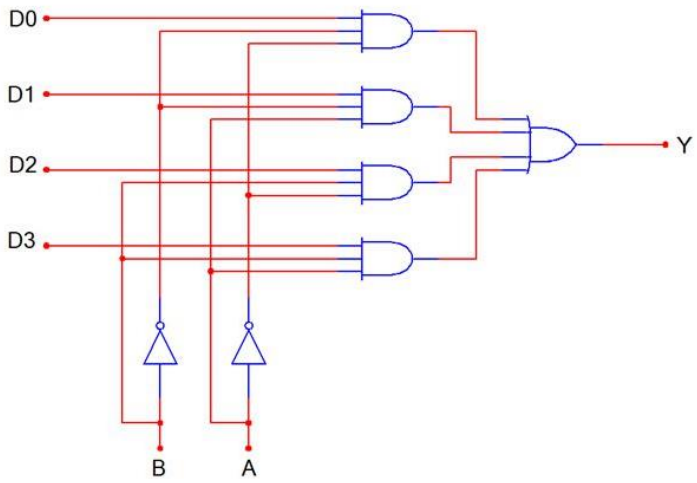


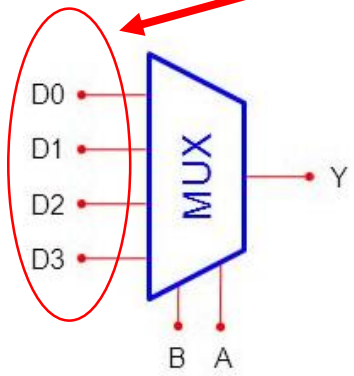
* actually, there is propagation delay always

https://www.electronics-tutorials.ws/combination/comb_1.html

Combinatorial logic circuits – Multiplexer (MUX)

- Selects “one input out of many” to single output
- Example: 4-to-1 mux





B	A	Y
0	0	D0
0	1	D1
1	0	D2
1	1	D3

Note: Inputs D0-D3 are not just static ‘0’s or ‘1’s – they can be any logic signals or signal buses carrying data, clocks, even at different frequencies

```
s <= (B, A); -- just a helper
with s select
  Y <= D0 when "00",
       D1 when "01",
       D2 when "10",
       D3 when others;
```

MUX in VHDL (in architecture)

- Conditional Signal Assignment

```
Y <= A when (s = "00") else
     B when (s = "01") else
     C when (s = "10") else
     D;
```

- Selective Signal Assignment

```
with s select
  Y <= D0 when "00",
       D1 when "01",
       D2 when "10",
       D3 when others;
```

MUX in VHDL (inside processes)

- Using if-elsif-else -statements

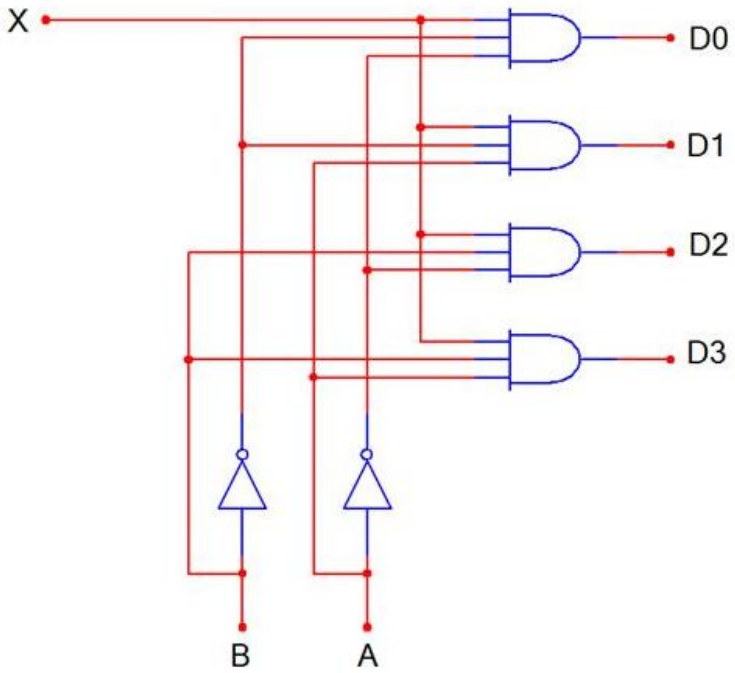
```
process (A,B,C,D,s) is
begin
  if (s = "00") then
      Y <= A;
  elsif (s = "01") then
      Y <= B;
  elsif (s = "10") then
      Y <= C;
  else
      Y <= D;
  end if;
end process;
```

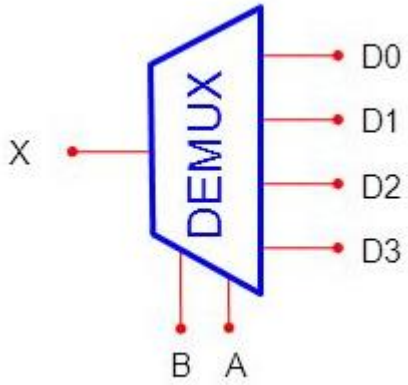
- Using case-statement

```
mux : process (A,B,C,D,s) is
begin
  case s is
    when "00" => Y <= A;
    when "01" => Y <= B;
    when "10" => Y <= C;
    when others => Y <= D;
  end case;
end process mux;
```

Comb. logic circuits – Demultiplexer (DEMUX)

- Maps “one input to one of many outputs”
- Example: 1-to-4 demux





B	A	D0	D1	D2	D3
0	0	X	0	0	0
0	1	0	X	0	0
1	0	0	0	X	0
1	1	0	0	0	X

Assuming X = ‘1’, this becomes 2-to-4 “one-hot” decoder
Of course, ‘X’ can be a data stream or a clock signal, so demux is “routing” the signal.