

Combinational Logic Design

Prof. Dr. Miloš Krstić

- **Boolean values**

Different representations

On/OFF – Vdd/Gnd

True/False

1/0

- **Boolean operators**

NOT

AND

OR

- **Examples**

$A = 1$

$B = C \text{ AND } 0$

$F = \neg(A + B * C)$

$Z = (\neg A + B) * (A + \neg B)$

Truth Tables

- Listing of all possible values of inputs and respective outputs

<u>A</u>	<u>not A</u>
0	1
1	0

<u>A</u>	<u>B</u>	<u>A or B</u>
0	0	0
1	0	1
0	1	1
1	1	1

A XOR B ?

Rules of Boolean Algebra

- **Commutativity**

$$A + B = B + A$$

$$A * B = B * A$$

- **Associativity**

$$A + (B + C) = (A + B) + C$$

$$A * (B * C) = (A * B) * C$$

- **Distributivity**

$$A * (B + C) = A * B + A * C$$

- **Basic Relationships**

$$A * 1 = A$$

$$A + 0 = A$$

$$A * 0 = 0$$

$$A + 1 = 1$$

$$A * A = A$$

$$A + A = A$$

$$A * /A = 0$$

$$A + /A = 1$$

Rules of Boolean Algebra

- **De Morgan's Law**

$$\overline{(A * B)} = \overline{A} + \overline{B}$$

$$\overline{(A + B)} = \overline{A} * \overline{B}$$

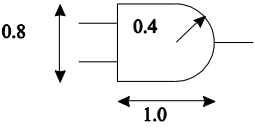
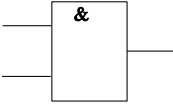
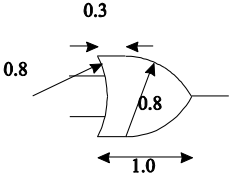
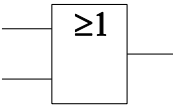
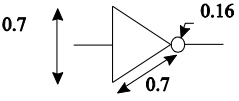
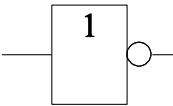
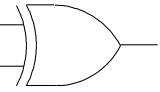
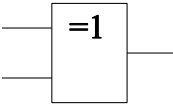
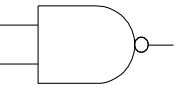
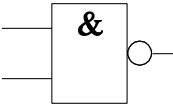
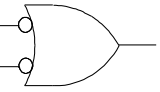
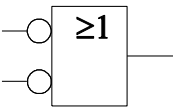
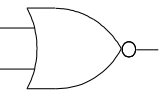
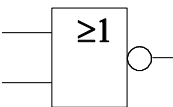
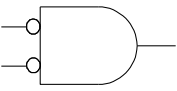
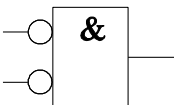
- **Shannon's expansion theorem**

$$F(A, B, C, D, \dots) = (A + F(0, B, C, D, \dots)) * (\overline{A} + F(1, B, C, D, \dots))$$

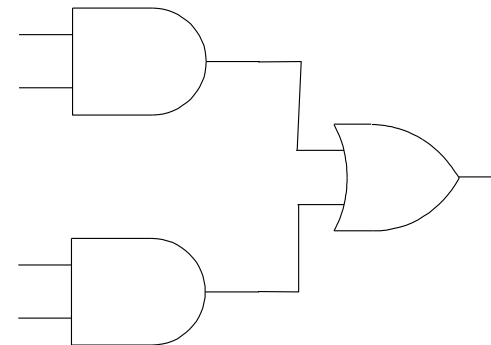
Logic Gates

Logic Symbols

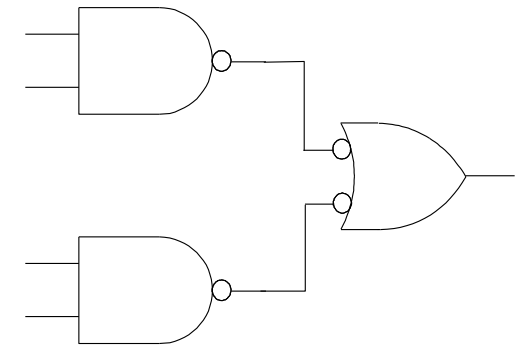
Equivalent Circuit Representation

	MIL-STD-806B	ANSI/IEEE
AND		
OR		
NOT		
XOR		
NAND		
		
NOR		
		

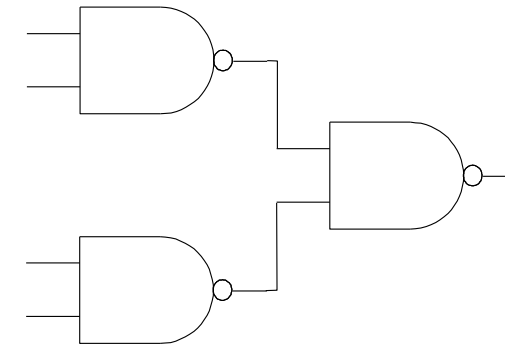
a)



b)



c)



MINTERM and MAXTERM

- **Minterm** – Boolean AND function containing one instance of each variable
- **Maxterm** – Boolean OR function containing one instance of each variable

<u>A</u>	<u>B</u>	<u>C</u>	<u>Z</u>	
0	0	0	1	m0
0	0	1	1	m1
0	1	0	0	M2
0	1	1	0	M3
1	0	0	0	M4
1	0	1	1	m5
1	1	0	0	M6
1	1	1	1	m7

$$Z = m0 + m1 + m5 + m7$$

$$Z = M2 * M3 * M4 * M6$$

Logic Minimization

- Function of a combinational logic circuit can be described by one or more Boolean expressions.

We need optimal implementation of combinational logic!

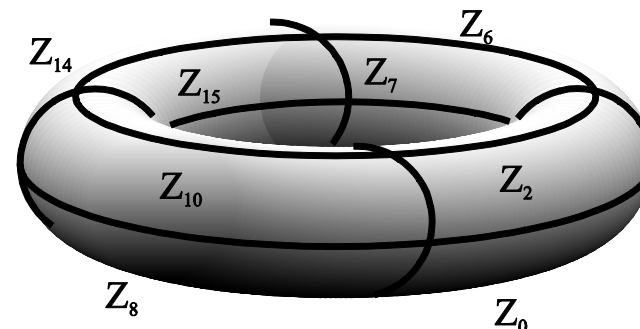
Karnaugh Maps

A	B	Z
0	0	Z_0
0	1	Z_1
1	0	Z_2
1	1	Z_3

		A	0	1
		B	0	1
Z:	0	Z_0	Z_2	
	1	Z_1	Z_3	

		AB	00	01	11	10
Z:	C					
	0		Z_0	Z_2	Z_6	Z_4
	1		Z_1	Z_3	Z_7	Z_5

		AB			
		00	01	11	10
Z:	CD				
	00	Z_0	Z_4	Z_{12}	Z_8
	01	Z_1	Z_5	Z_{13}	Z_9
	11	Z_3	Z_7	Z_{15}	Z_{11}
	10	Z_2	Z_6	Z_{14}	Z_{10}



Karnaugh map

- Grouping patterns**

Circle the largest possible groups

Avoid circles inside circles

		A	0	1
Z:	B			
	0	1	0	
	1	1	0	

		AB			
		00	01	11	10
Z:	C				
	0	0	1	1	0
	1	1	0	0	1

$$Z = B \cdot \bar{C} + \bar{B} \cdot C$$

		AB			
		00	01	11	10
Z:	CD				
	00	1	0	0	1
	01	0	1	1	0
	11	0	1	1	0
	10	1	0	0	1

$$Z = B \cdot D + \bar{B} \cdot \bar{D}$$

Karnaugh map

- Redundant Grouping

		AB			
		00	01	11	10
Z:	CD				
	00	0	0	0	0
	01	0	1	1	0
	11	1	1	0	0
	10	0	0	0	0

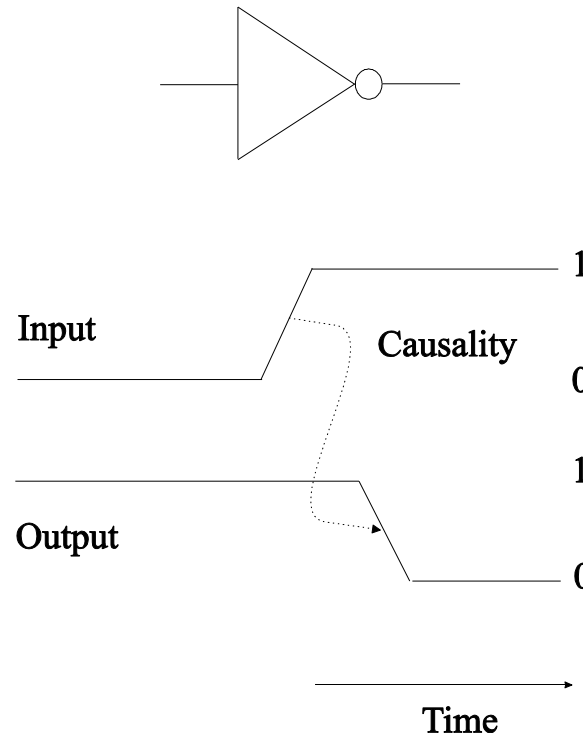
- Don't care

			A	
			B	
				0
				1
Z:	0	0	1	0
	0	1	-	
	1	0	0	
	1	1	1	

Timing

- Timing Diagram**

Boolean gates require some time for propagation
In reality circuits and wires have intrinsic delay



Usually when modeling in VHDL it is not required to handle the timing
In some specific cases is however this recommended
Sole combinational logic, race conditions, hazards

Number Codes

- **Digital representation in set of bits**

- **Integers**

Base 2 – 101_2

hex – $7AF_{16}$

Two's complement

$$-6 = \text{inverting } 0110 + 1 = 1010$$

- **Fixed-point numbers**

$$6.25 = 110.01$$

The point is implicitly stored by knowing the position in advance

- **Floating-point numbers**

S- sign bit, e –exponent, m – mantisa

$$(-1)^s \times 1.m \times 2^e$$

Combinational Code using VHDL

- **Combinational logic is stateless**
Changes in inputs immediately propagate to outputs
In simulator this is however delta cycle delay
- **Entities and architectures**
Basic structures of VHDL
Entity – Symbol (outer view of the block)
Architecture - Implementation

```
entity and2 is  
  port (a, b : in BIT; c: out BIT);  
end entity and2;
```

```
architecture struc of And2 is  
begin  
  c <= a and b;  
end architecture struc;
```

Identifiers, spaces and comments

- **VHDL not case-sensitive, however ... still recommended to follow some rules**

use meaningful names

Don't mix cases, consistent use of cases

Identifiers <15

Don't redefine predefined identifiers (BIT, TIME)

For signals and variables that are active low, this shall be clearly indicated by their name, by suffixing _n

Comments

```
--=====
-- Design units :RS_Decoder (Arch_RS_Decoder) (entity and architecture)
-- File name :RS_Decoder.vhd
--
-- Purpose : Model of Reed Solomon (RS) Decoder for communication systems
-- with FEC (Forward Error Correction)
--
-- Note: Selection of code is performed ...
--
-- Limitations : ....
--
-- Errors : None known
--
-- Library/Package :
-- work.rs_decoder_package
--
-- Author : N.N
-- IHP GmbH, System Design Department
-- e-mail : nn@ihp-microelectronics.com
--
-- Simulator: ModelSim SE-64 10.1d
-----

-- Revision list
-- Version : 1.0
-- Author :N.N.
-- Date : 20.09.2015
-- Changes : New version
-----
```

Handling more complex examples

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

Use K-map to get result !

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;

Why parentheses?

Hierarchy in modules

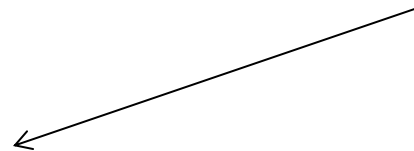
```
entity Or2 is  
  port (x, y : in BIT; z: out BIT);  
end entity Or2;
```

```
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;
```



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;
```

- VHDL code can be hierarchically utilized
- The code which just connects the components uses netlist (structural) style

Netlists

architecture netlist2 of comb_function is

component And2 is

port (x, y : in BIT; z: out BIT);

end component And2;

component Or2 is

port (x, y : in BIT; z: out BIT);

end component Or2;

component Not1 is

port (x : in BIT; z: out BIT);

end component Not1;

signal p, q, r : BIT;

begin

g1: Not1 port map (a, p);

g2: And2 port map (p, b, q);

g3: And2 port map (a, c, r);

g4: Or2 port map (q, r, z);

end architecture netlist;

Signal Assignments

Z <= x and y;

Several ways of modelling delay in VHDL

Intertial delay

Z <= x after 4 ns;

Pulses shorter than 4 ns will be suppressed!

Transport delay

Z <= transport x after 4 ns;

The assignments could be complex

Z <= x and y after 4 ns;

Generics

If we need programmable definition of delay we can use generics

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

```
architecture ex2 of And2 is  
begin  
  z <= x and y after delay;  
end architecture ex2;
```

Constant and open ports

Sometimes we do not want to connect all ports

Example: universal gate: two outputs are “and” and “or” function

invert input defines whether the output should be inverted

We can implement AND, NAND, OR, NOR function

entity universal is

port (x, y, invert : in BIT; a, o : out BIT);

end entity universal;

architecture univ of universal is

begin

a <= (y and (x xor invert)) or (invert and not y);

o <= (not x and (y xor invert)) or (x and not invert);

end architecture univ;

In higher instance:

U0: entity work.universal(univ) port map (x,y,'0', a, open);

Synthesis tool can further optimize circuits if some output is left open

Testbenches

We want to evaluate correctness of our model

Usual way is simulation -> we need stimuli file

We have to define a testbench which reads the stimuli, applies this to DUT and checks the results

```
entity TestAnd2 is  
end entity TestAnd2;
```

```
architecture io of TestAnd2 is
```

```
    signal a,b,c : BIT;
```

```
begin
```

```
    g1: entity WORK.And2(ex2) port map (x=>a, y=>b, z=>c);
```

```
    a<= '0', '1' after 100 NS;
```

```
    b<= '0', '1' after 150 NS;
```

```
end architecture io;
```

Configurations

Alternative way to describe AND

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

architecture ex3 of And2 is
  signal xy : BIT_VECTOR(0 to 1);
begin
  xy <= x&y;
  with xy select
    z <= '1' when "11",
      '0' when others;
end architecture ex3;
```

Testbench with configurations

```
entity TestAnd2 is
end entity TestAnd2;

architecture alternate of TestAnd2 is
  component A2 is
    port (x, y : in BIT; z: out BIT);
  end component A2;
  for all : A2 use entity work.And2(ex2);
  signal a,b,c : BIT;
begin
  g1: A2 port map (x=>a, y=>b, z=>c);
  a<= '0', '1' after 100 NS;
  b<= '0', '1' after 150 NS;
end architecture alternate;
```

```
configuration Tester1 of TestAnd2 is
  for io
    for g1 : And2
      use entity WORK.And2(ex1);
    end for;
  end for;
end configuration Tester1;
```

Configurations

```
entity TestAnd2 is  
end entity TestAnd2;
```

```
architecture remapped of TestAnd2 is  
  component MyAnd2 is  
    generic (dly : DELAY_LENGTH);  
    port (in1, in2 : in BIT; out1: out BIT);  
  end component MyAnd2;  
  signal a,b,c : BIT;  
begin  
  g1: MyAnd2 generic map (6 NS) port map (a, b, c);  
  a<= '0', '1' after 100 NS;  
  b<= '0', '1' after 150 NS;  
end architecture remapped;
```

```
configuration Tester2 of TestAnd2 is  
  for remapped  
    for g1 : MyAnd2  
      use entity WORK.And2(ex2)  
      generic map (delay => dly);  
      port map (x => in1, y => in2, z => out1)  
    end for;  
  end for;  
end configuration Tester2;
```


Conclusions

- **Boolean logic and logic synthesis**
- **Representation**
- **Combinational logic in VHDL**
 - Entity**
 - Architecture**
 - Signal assignment**
 - Generics**
 - Open**
 - Configuration**