Fundamentals of DIGITAL LOGIC with VHDL design

- Stephen Brown and Zvonko Vranesic
- McGraw-Hill, 2000

- Slides prepared by P.J. Bakkes (2000) (Edited in June 2003)

Chapter 1
Design Concepts

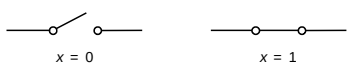
- Read as introduction

Chapter 2
Introduction to Logic Circuits

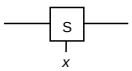
- Digital versus analog circuits
- Binary logic (Afr. Binêre logika)
- Switches
- 0 or 1, ON or OFF, HIGH or LOW

2.1 Variables and Functions

- Reference figure 2.1
- Controlled switches
- Input variable controls a switch
- Logic expression e.g. L(x) = x describes the output as a function of the input variable
- If x = 0, L = 0 and light is off
- If x = 1, L = 1 and light is on

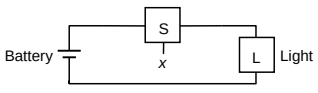


(a) Two states of a switch

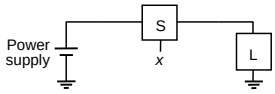


(b) Symbol for a switch

 Figure 2.1 A binary switch



(a) Simple connection to a battery



(b) Using a ground connection as the return path

 Figure 2.2 A light controlled by a switch

Other functions

- Reference figure 2.3 and 2.4
- AND (Afr. EN): L(x1,x2) = x1 • x2
- OR (Afr. OF): L(x1,x2) = x1 + x2
- Other: L(x1,x2,x3) = (x1 + x2) • x3

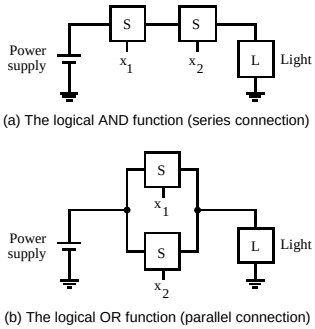


Figure 2.3 Two basic functions

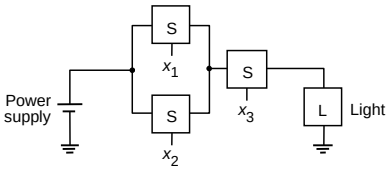


Figure 2.4 A series-parallel connection

2.2 Inversion (Afr. Inversie)

- L(x) = x'
- L(x) = !x
- L(x) = not x
- If x = 0, L = 1
- If x = 1, L = 0

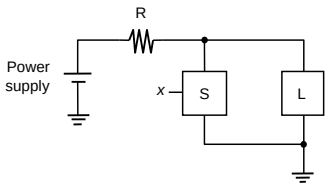


Figure 2.5 An inverting circuit

2.3 Truth tables (Afr. Waarheidstabelle)

- Reference figures 2.6 and 2.7
- Alternative description of a logic function for combinatorial circuits

x_1	x_2	$x_1 \cdot x_2$	$x_1 + x_2$
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1
		AND	OR

Figure 2.6 A truth table for AND and OR

x_1	x_2	x_3	$x_1 \cdot x_2 \cdot x_3$	$x_1 + x_2 + x_3$
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	0	1
1	0	0	0	1
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Figure 2.7 Three-input AND and OR

2.4 Logic gates and networks (Afr. Logiese hekke en netwerke)

- The three basic functions (AND, OR and NOT) can be combined to form logic functions of any complexity.
- They can be represented by symbols (Afr. Simbole) (figure 2.8)
- A combination of these symbols in a drawing is called a circuit diagram or a schematic (Afr. Stroombaandiagram). Figure 2.9.
- Larger circuit uses a network of gates

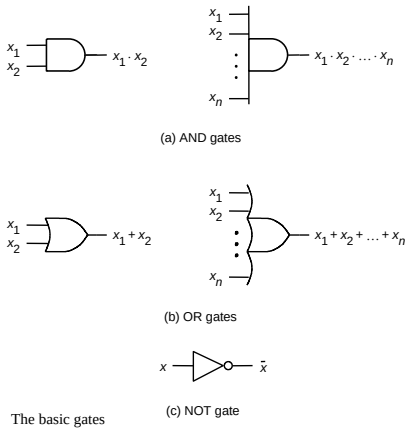


Figure 2.8 The basic gates

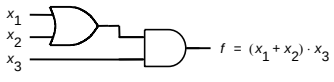
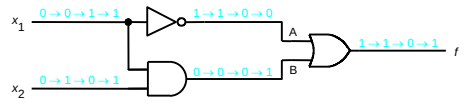


Figure 2.9 An OR-AND function

2.4.1 Analysis of a Logic Network

- Analysis (Afr. analise)
  - Determine the function of an existing circuit
- Synthesis (Afr. sintese)
  - Create a new circuit for an application (main task of an engineer!)
  - Optimization is important
- Reference figure 2.10a



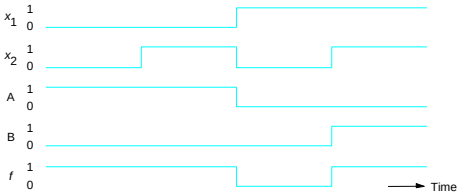
(a) Network that implements $f = \bar{x}_1 + x_1 \cdot x_2$

x_1	x_2	$f(x_1, x_2)$
0	0	1
0	1	1
1	0	0
1	1	1

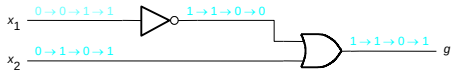
(b) Truth table for f

Timing diagram (Afr. tyddiagram)

- Reference figure 2.10b
- Logic variables can change rapidly with time (up to 1 GHz)



(c) Timing diagram



(d) Network that implements $g = \bar{x}_1 + x_2$

Functional equivalent networks

- Reference figure 2.10d
- Function g is functionally equivalent to f, but timing, cost, etc. can be different.
- Optimization of functions (later).

2.5 Boolean Algebra

- In 1849 George Boole defined an algebraic description of processes involved in thought and reasoning.
- In the late 1930's Shannon applied this to logic circuits.
- Boolean algebra developed into a powerful tool to describe logic circuits.

Axioms of Boolean Algebra

- 0 • 0 = 0
- 1 + 1 = 1
- 1•1 = 1
- 0 + 0 = 0
- 0 • 1 = 1 • 0 = 0
- 1 + 0 = 0 + 1 = 1
- If x = 0, then x' = 1
- If x = 1, then x' = 0

Single variable theorems

- x •0 = 0
- x+1 = 1
- x •1 = x
- x+0 = x
- x •x = x
- x+x = x
- x •x’ = 0
- x+x’ = 1
- x’’ = x

Duality

- Replace all 0’s with 1’s, all 1’s with 0’s, all AND’s with OR’s and all OR’s with AND’s
- More later

Two and three variable properties

- Commutative
  - x•y = y •x
  - x+y = y+x
- Associative
  - x •(y •z) = (x •y) •z
  - x+(y+z) = (x+y)+z
- Distributive
  - x •(y+z) = x •y + x •z
  - x+(y •z) = (x+y) •(x+z)

- Absorption
  - x+x •y = x
  - x •(x+y) = x
- Combining
  - x •y + x •y’ = x
  - (x+y) •(x+y’) = x
- x + x’ •y = x + y
- x •(x’ + y) = x •y

DeMorgan’s Theorem

- (x •y)’ = x’ + y’
- (x+y)’ = x’ •y’

Next page: prove by *perfect induction*

x	y	$x \cdot y$	$\overline{x \cdot y}$	$\overline{x}$	$\overline{y}$	$\overline{x} + \overline{y}$
0	0	0	1	1	1	1
0	1	0	1	1	0	1
1	0	0	1	0	1	1
1	1	1	0	0	0	0
		LHS		RHS		

Applying identities

- See examples 2.1 and 2.2

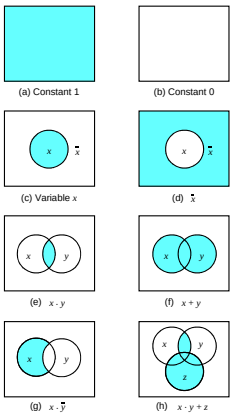


Figure 2.12 The Venn diagram representation

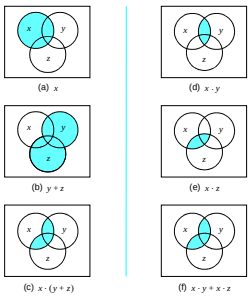


Figure 2.13 Verification of the distributive property

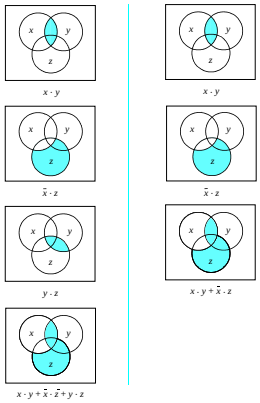


Figure 2.14 Verification of $x \cdot y + \bar{x} \cdot z + y \cdot z = x \cdot y + \bar{x} \cdot z$

2.5.2 Alternative Notation

- AND: x ∧ y
- OR: x ∨ y

2.5.3 Precedence of Operators

- NOT, AND and then OR

2.6 Synthesis using AND, OR and NOT gates

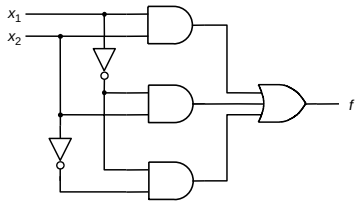
- Synthesis: generate a circuit that implements a function from specifications.
- Simple example:

x_1	x_2	$f(x_1, x_2)$
0	0	1
0	1	1
1	0	0
1	1	1

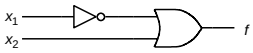
 Figure 2.15 A function to be synthesized

Example

- Reference figure 2.15 and 2.16
- Concept of sum-of-products
- Simplify expression using theorems
- f(x1,x2) = x1 • x2 + x1' • x2' + x1' • x2
- = x1 • x2 + x1' • x2' + x1' • x2 + x1' • x2
- = x1 • x2 + x1' • x2 + x1' • x2' + x1' • x2
- = (x1 + x1') • x2 + x1' •(x2' + x2)
- = x2 + x1'



(a) Canonical sum-of-products



(b) Minimal-cost realization

 Figure 2.16 Two implementations of a function

Row number	x_1	x_2	x_3	Minterm	Maxterm
0	0	0	0	$m_0 = \overline{x}_1\overline{x}_2\overline{x}_3$	$M_0 = x_1 + x_2 + x_3$
1	0	0	1	$m_1 = \overline{x}_1\overline{x}_2 x_3$	$M_1 = x_1 + x_2 + \overline{x}_3$
2	0	1	0	$m_2 = \overline{x}_1 x_2\overline{x}_3$	$M_2 = x_1 + \overline{x}_2 + x_3$
3	0	1	1	$m_3 = \overline{x}_1 x_2 x_3$	$M_3 = x_1 + \overline{x}_2 + \overline{x}_3$
4	1	0	0	$m_4 = x_1\overline{x}_2\overline{x}_3$	$M_4 = \overline{x}_1 + x_2 + x_3$
5	1	0	1	$m_5 = x_1\overline{x}_2 x_3$	$M_5 = \overline{x}_1 + x_2 + \overline{x}_3$
6	1	1	0	$m_6 = x_1 x_2\overline{x}_3$	$M_6 = \overline{x}_1 + \overline{x}_2 + x_3$
7	1	1	1	$m_7 = x_1 x_2 x_3$	$M_7 = \overline{x}_1 + \overline{x}_2 + \overline{x}_3$

 Figure 2.17 Three-variable Minterms and Maxterms

2.6.1 Sum-of-Products and Product-of-Sums

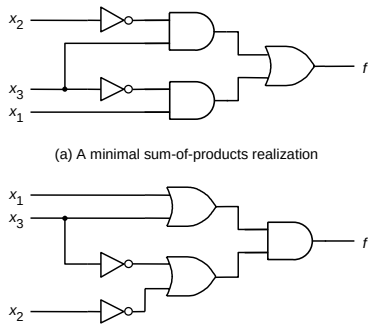
- Introduce more formal terms
- Minterm: for a function of n variables, a product term in which each of the n variables appears *once*, is called a *minterm*.
- Sum-of-Products form: A function f can be represented by an expression that is a logical sum of sum of the minterms.
- See figure 2.17

Row number	x_1	x_2	x_3	$f(x_1, x_2, x_3)$
0	0	0	0	0
1	0	0	1	1
2	0	1	0	0
3	0	1	1	0
4	1	0	0	1
5	1	0	1	1
6	1	1	0	1
7	1	1	1	0

Figure 2.18 A three-variable function

Another example

- See figure 2.18 and 2.19
- f(x1,x2,x3) = x1' • x2' • x3 + x1 • x2' • x3' + x1 • x2' • x3 + x1 • x2 • x2'
- This form is not minimal, but can be reduced to: f = x2' • x3 + x1 • x3'
- Alternative forms:
  - f = ∑ (m1,m4,m5,m6) = ∑ m(1,4,5,6)



(a) A minimal sum-of-products realization

(b) A minimal product-of-sums realization

Figure 2.19 Two realizations of a function

Maxterms

- In stead of considering rows in the truth table for which f = 1, it is also possible to synthesize f considering where f = 0.
- This implies using the duality principle.
- Uses maxterms, which are the complements of minterms.

Product-of-sums form

See figure 2.15

- f' = m2 = x1 • x2'
- From DeMorgan:
- f'' = f = (x1 • x2')' = x1' + x2
- Thus f = m2' = M2 (maxterm)

Another example

- See figure 2.18
- f' = m0 + m2 + m3 + m7
- f = (m0 + m2 + m3 + m7)'
  = m0'•m2' • m3' • m7' (deMorgan)
  = M0 •M2 •M3 •M7
  = (x1+x2+x3) (x1+x2'+x3) (x1+x2'+x3') (x1'+x2'+x3') (product-of-sums)
  = ∏ (M0,M2,M3,M7) = ∏ M(0,2,3,7)

2.7 Design examples

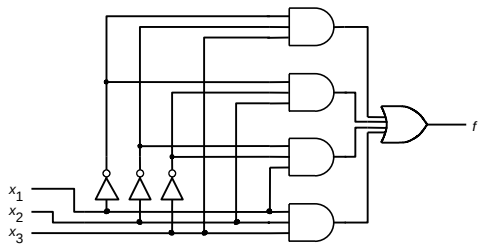
- Design process:
  - Specification of solution to a problem in words
  - Formal specification with truth table
  - Synthesis
  - Implementation
  - Testing
  - Possible iteration

2.7.1 Three-way light control

- Three doors in room each with switch
- One or three on switches must turn light on
- See truth table in figure 2.20
- f = m1 + m2 + m4 + m7
- or f = M0•M3 • M5 • M6
- See figure 2.21. For implementation

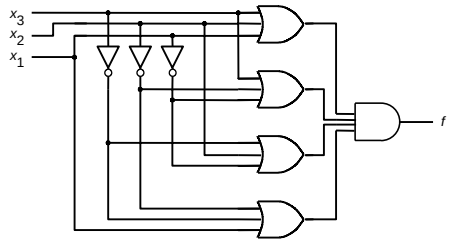
x_1	x_2	x_3	f
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

 Figure 2.20 Truth table for a three-way light controller



(a) Sum-of-products realization

 Figure 2.21 SOP implementation of the three-way light controller



(b) Product-of-sums realization

 Figure 2.21 POS implementation of the three-way light controller

2.7.2 Multiplexer Circuit

- Circuit often used in digital and computer designs
- Switches one of multiple sources of data to a single destination
- See figure 2.22

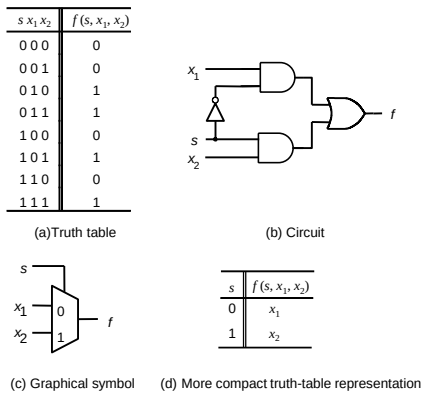


Figure 2.22 Multiplexer

2.8 Introduction to CAD tools

- Lecturers comment: Read only at this stage
- 2.8.1 Design Entry
  - Truth table (figure 2.23)
  - Schematic entry (figure 2.24)
  - Hardware description languages (HDL)
- 2.8.2 Synthesis
- 2.8.3 Functional simulation

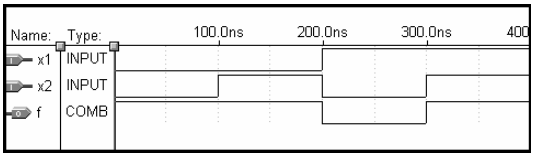


Figure 2.23 Screen capture of the Waveform Editor

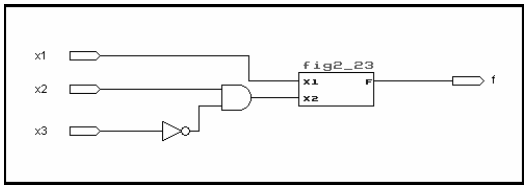


Figure 2.24 Screen capture of the Graphic Editor

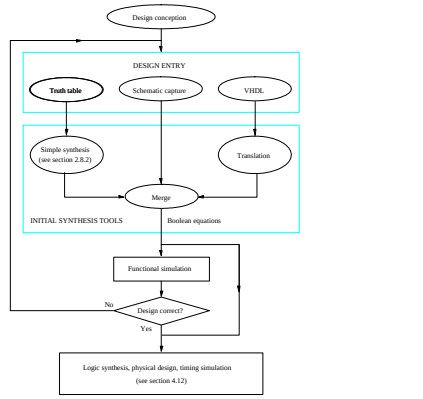
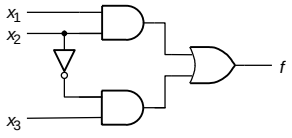


Figure 2.25 The first stages of a CAD system

2.9 Introduction to VHDL

- IEEE standard 1164
- Used for the algorithmic description of digital circuits
- Used for synthesis and simulation
- See figures 2.27 to 2.31




```
ENTITY example1 IS
    PORT ( x1, x2, x3   : IN    BIT ;
           f            : OUT   BIT ) ;
END example1 ;

ARCHITECTURE LogicFunc OF example1 IS
BEGIN
    f <= (x1 AND x2) OR (NOT x2 AND x3) ;
END LogicFunc ;
```

 Figure 2.26 A simple logic function and corresponding VHDL code

```
ENTITY example2 IS
    PORT ( x1, x2, x3, x4   : IN    BIT ;
           f, g             : OUT   BIT ) ;
END example2 ;

ARCHITECTURE LogicFunc OF example2 IS
BEGIN
    f <= (x1 AND x3) OR (NOT x3 AND x2) ;
    g <= (NOT x3 OR x1) AND (NOT x3 OR x4) ;
END LogicFunc ;
```

 Figure 2.30 VHDL code for a four-input function

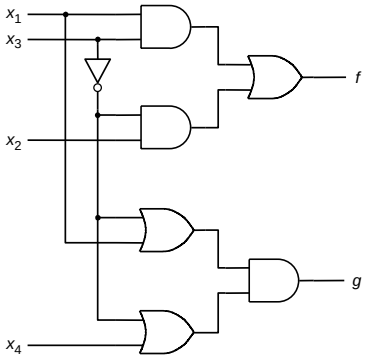


 Figure 2. 31 Logic circuit for four-input function