

Code EE2C11 en EE2L11

# Manual

Lab course Integrated Circuits  
and Project Design a Chip

October 2017

A. Bakker, A. Frehe, A.J. van Genderen, S. de Graaf, P. Groeneveld  
E.A. Hendriks, N.P. van der Meijs, R. Nouta, C. Verhoeven, J. Liedorp

# Contents

<b>I</b>	<b>General Introduction</b>	<b>9</b>
<b>1</b>	<b>Introduction</b>	<b>9</b>
<b>2</b>	<b>Schedule and Organisation</b>	<b>10</b>
2.1	Schedule . . . . .	10
2.1.1	Quarter 1 . . . . .	10
2.1.2	Quarter 2 . . . . .	10
2.1.3	Quarter 4 . . . . .	10
2.2	Location . . . . .	11
2.3	Assistance . . . . .	11
<b>II</b>	<b>Lab Course Integrated Circuits</b>	<b>13</b>
<b>3</b>	<b>Introduction</b>	<b>13</b>
<b>4</b>	<b>Introduction SPICE Assignment</b>	<b>14</b>
<b>5</b>	<b>Technology Data</b>	<b>14</b>
<b>6</b>	<b>Courselab Instructions</b>	<b>15</b>
<b>7</b>	<b>Information Literacy 2</b>	<b>15</b>
<b>8</b>	<b>Report</b>	<b>15</b>
8.1	General Requirements for the Contents of the Report . . . . .	15
8.2	Requirements for the Structure of the Report . . . . .	16
8.3	Requirements for the Layout of the Report . . . . .	18
8.4	Specific Requirements for the Contents of the Report . . . . .	19
<b>9</b>	<b>Assignment</b>	<b>20</b>

9.1	Parameter Set . . . . .	20
9.2	Unified Model Parameters . . . . .	20
9.3	Optional Assignments . . . . .	22
<b>10</b>	<b>Use of PSPICE</b>	<b>22</b>
10.1	Adding Libraries . . . . .	22
10.2	Exporting from PSPICE . . . . .	22
<b>11</b>	<b>IEEE Citation Style</b>	<b>23</b>
<b>12</b>	<b>IC design software</b>	<b>24</b>
<b>13</b>	<b>IC design tutorial: A hotel switch</b>	<b>27</b>
13.1	Introduction . . . . .	27
13.2	Finite-State Diagram . . . . .	27
13.3	Creating a project . . . . .	28
13.4	VHDL behaviour description . . . . .	29
13.5	Logic synthesis . . . . .	33
13.6	Creating a layout . . . . .	33
13.7	Verification of the layout . . . . .	33
<b>14</b>	<b>Guidelines to write VHDL code</b>	<b>38</b>
14.1	Introduction . . . . .	38
14.2	Behavioural description combinatorial circuit . . . . .	38
14.3	Behavioural description sequential circuit . . . . .	39
14.4	Structural description of a circuit . . . . .	43
14.5	Parameterized descriptions . . . . .	44
<b>15</b>	<b>VHDL restrictions because of design tools</b>	<b>47</b>
<b>16</b>	<b>The FSM introductory assignment</b>	<b>49</b>
16.1	Introduction . . . . .	49
16.2	Creating a Finite-State Diagram . . . . .	49

16.3	Creating the Circuit . . . . .	49
16.4	Possible Assignments . . . . .	51
16.4.1	The Ron Brandsteder Lights . . . . .	51
16.4.2	The Basket . . . . .	52
16.4.3	The Train Security System . . . . .	53
16.4.4	The "Lazy" Lock Door . . . . .	54
16.4.5	The Person Detector . . . . .	55
16.4.6	The Garage Door . . . . .	56
16.4.7	The Soda Dispenser . . . . .	57
16.4.8	The Telephone Circuit . . . . .	58
<b>III</b>	<b>Project Design a Chip</b>	<b>59</b>
<b>17</b>	<b>Introduction</b>	<b>59</b>
<b>18</b>	<b>Project Learning Goals</b>	<b>60</b>
<b>19</b>	<b>Working in a Team (in Dutch)</b>	<b>62</b>
19.1	Inleiding . . . . .	62
19.2	De drie dimensies van het werken in een groep . . . . .	62
19.3	De vergadering . . . . .	63
19.4	Rollen en functies . . . . .	64
19.5	Praktische tips voor de organisatie van de projectgroep . . . . .	67
19.6	De procedure voor probleemoplossing en besluitvorming . . . . .	68
19.7	Evaluatie . . . . .	69
<b>20</b>	<b>The Design Process</b>	<b>71</b>
20.1	Introduction . . . . .	71
20.2	Specification . . . . .	71
20.3	Function Block Diagram . . . . .	72
20.4	Structural System Description . . . . .	72

20.5 Hierarchical Decomposition . . . . .	73
20.6 Abstraction Levels . . . . .	73
<b>21 Guidelines for designing large circuits</b>	<b>75</b>
<b>22 The Group Assignment</b>	<b>78</b>
22.1 Introduction . . . . .	78
22.2 Selection of the assignment . . . . .	78
22.3 System specification . . . . .	78
22.4 General technological boundary conditions . . . . .	79
22.4.1 Available chip area . . . . .	79
22.4.2 Additional boundary conditions . . . . .	79
22.5 System design . . . . .	80
22.6 Implementation of subsystems (the module assignment) . . . . .	80
22.6.1 The assignment . . . . .	80
22.6.2 Report . . . . .	81
22.7 Putting everything together . . . . .	81
22.7.1 Assembling parts with GoWithTheFlow . . . . .	81
22.8 Rapid prototyping . . . . .	82
22.9 Design for testability . . . . .	83
22.9.1 The method for testing after manufacturing . . . . .	83
22.9.2 Considerations during design . . . . .	83
22.9.3 Considerations when creating testbenches . . . . .	84
22.10 Finishing the design . . . . .	84
22.11 Documentation . . . . .	85
<b>23 Ideas for the Group Assignment (in Dutch)</b>	<b>87</b>
23.1 Inleiding . . . . .	87
23.2 De Avant-Garde Klok . . . . .	87
23.2.1 Inleiding . . . . .	87
23.2.2 Specificaties DCF-ontvanger . . . . .	88

23.2.3	Overige specificaties . . . . .	89
23.3	De Infrarood-Besturing . . . . .	91
23.3.1	Inleiding . . . . .	91
23.3.2	Specificaties infrarood-kanaal . . . . .	92
23.4	Een Reactiemeter . . . . .	93
23.4.1	Inleiding . . . . .	93
23.4.2	Werking reactiemeter . . . . .	93
23.4.3	Specificaties reactiemeter . . . . .	93
23.5	De Pong Spelcomputer . . . . .	94
23.5.1	Inleiding . . . . .	94
23.5.2	Het spel . . . . .	95
23.5.3	De spelbesturing . . . . .	95
23.5.4	De aansturing van de monitor . . . . .	95
23.5.5	Het "pong" geluid . . . . .	97
23.5.6	De overige randvoorwaarden . . . . .	97
23.6	Alternatieve opdracht . . . . .	97

## **IV Appendices 99**

### **A Some instructions for using Linux 99**

A.1	Login in for the first time . . . . .	99
A.2	Some linux commands . . . . .	99

### **B ICT Support for Project Work 101**

B.1	Available Services . . . . .	101
B.1.1	Versioning System: GIT or SVN . . . . .	101
B.1.2	Bug Tracker and Wiki: Trac . . . . .	101
B.1.3	Mail group . . . . .	101
B.2	EWI Projects Server . . . . .	102
B.2.1	Creating and Managing Projects . . . . .	102

B.2.2	Repository Address . . . . .	104
<b>C</b>	<b>De Sea-of-Gates Chip</b>	<b>105</b>
C.1	Globale beschrijving . . . . .	105
C.2	De kern van de chip, vanuit circuit-standpunt . . . . .	106
C.3	De kern van de chip, vanuit een layout-standpunt . . . . .	107
C.4	Eisen en afspraken met betrekking tot de metallisatie . . . . .	108
C.5	Meetgegevens van de Sea-of-Gates chip . . . . .	110
C.6	Structuur in de rand van de chip . . . . .	111
<b>D</b>	<b>De SoG Cellenbibliotheek (oplib)</b>	<b>113</b>
D.1	Inleiding . . . . .	113
D.2	De cellen . . . . .	114
D.2.1	iv110 . . . . .	114
D.2.2	no210 . . . . .	115
D.2.3	no310 . . . . .	116
D.2.4	na210 . . . . .	117
D.2.5	na310 . . . . .	118
D.2.6	ex210 . . . . .	119
D.2.7	buf40 . . . . .	120
D.2.8	tbuf10 . . . . .	121
D.2.9	tin10 . . . . .	122
D.2.10	mu111 . . . . .	123
D.2.11	mu210 . . . . .	124
D.2.12	de211 . . . . .	126
D.2.13	dfn10 . . . . .	128
D.2.14	dfr11 . . . . .	130
D.2.15	dfa11 . . . . .	132
D.2.16	osc10 . . . . .	134
D.2.17	ln3x3 . . . . .	135

D.2.18	lp3x3 . . . . .	136
D.2.19	mir_nin, mir_nout . . . . .	137
D.2.20	mir_pin, mir_pout . . . . .	139
D.2.21	bond_leer . . . . .	141
D.2.22	bond_bar . . . . .	142
<b>E</b>	<b>Analoge Deelschakelingen</b>	<b>143</b>
E.1	Inleiding . . . . .	143
E.2	Analoge signaalbewerkingen in de groepsopdracht . . . . .	144
E.3	Bouwstenen voor analoge circuits op de SoG wafer . . . . .	146
E.4	Voorbeelden van implementaties . . . . .	147
E.4.1	Toetsenbord-uitlezing . . . . .	147
E.4.2	DA-converter . . . . .	148
E.4.3	Sinus-generatie met een DAC . . . . .	150
E.4.4	Eindversterker . . . . .	150
E.4.5	Power on reset . . . . .	151
<b>F</b>	<b>Testing SOG chips with the Logic Analyzer</b>	<b>153</b>
F.1	Introduction . . . . .	153
F.2	Overview . . . . .	154
F.3	Obtaining the Reference Files . . . . .	155
F.4	Preparing the Input Data for the Logic Analyzer . . . . .	155
F.5	Setting up the Logic Analyzer . . . . .	155
F.6	Mounting and connecting the SOG chip . . . . .	155
F.7	Using the Logic Analyzer . . . . .	156
F.8	Comparing the Analyzer Output with the Reference Output . . . . .	156
F.9	Trouble Shooting . . . . .	156
F.10	Further Reading . . . . .	156
	<b>References</b>	<b>157</b>



## **Part I**

# **General Introduction**

## **1 Introduction**

This manual is to be used both for the lab courses associated with the course "Integrated Circuits" as well as for the project "Design a Chip".

The lab courses of "Integrated Circuits" take place in the first quarter of the second year of the BSc. programme Electrical Engineering. The project "Design a Chip" takes place during the second quarter of that year. These parts are closely related to each other: various activities within the lab are a preparation for the project.

The manual is divided into four parts: (1) a general introductory part, (2) a part focusing on the lab courses of Integrated Circuits (3) a part that is about the project Design a Chip, and (4) appendices that provide information that is of importance for both the lab courses as well as the project, such as a detailed description of the Sea-of-Gates chip and a description of the available library cells.

Note that part of this manual is still in Dutch, parts of it are in English, as a preparation for translating the complete manual into English.

## 2 Schedule and Organisation

### 2.1 Schedule

For the lab courses of "Integrated Circuits", 4 sessions are scheduled in the first quarter. For the project "Design a Chip", 16 sessions are scheduled in the second quarter. In the fourth quarter, there will be a session where the chips that were designed during the project, will be tested.

Besides the scheduled sessions, it is expected that students also work on the assignments during their own time. For the project, for example, the scheduled time is  $16 \times 4 \text{ hours} = 64 \text{ hours}$ . The total study load of the project is 5 EC, which corresponds to  $5 \times 28 \text{ hours} = 140 \text{ hours}$ . Therefore, it is expected that students work on the project in their own time during  $140 - 64 = 76 \text{ hours}$ .

Below the planning of the lab courses and the project is roughly described. A more detailed description will appear on Blackboard.

#### 2.1.1 Quarter 1

During the first quarter, within the context of the lab course of Integrated Circuits, first an assignment is done where simulations are done for a CMOS circuit. A report has to be written about this assignment. During a second assignment, the student is introduced to the Sea-of-Gates design process. For the first two assignments, students will work in pairs. Next, at the end of the first quarter, a group meeting will take place where the students will discuss the chip that they will design during the second quarter. During this meeting, it is expected that a plan is made for the design during the second quarter.

#### 2.1.2 Quarter 2

During the second quarter, a Sea-of-Gates chip will be designed by a group of 7-10 students. The quarter begins by first setting good specifications for the group design. Next, the design is subdivided into 3-5 modules of approximately equal complexity. The different modules are then designed in groups of 2-3 students and a report is written about this. Then, the total system is implemented by connecting the different modules, finalizing the modules, and by creating the final layout. At the end of the second quarter, the design has to be handed in, together with the final report. The assessment takes place during a presentation and defense of the project by the group.

#### 2.1.3 Quarter 4

In the third and at the beginning of the fourth quarter, the chip that was designed during the project, will be manufactured at the Else Kooi Lab, provided that verification results show that there is a good chance that the chip will be working correctly. Next, at the end of the fourth quarter, the chip is tested.

## **2.2 Location**

The project will take place in the Tellegen hall of the EWI building (Building 36 Mekelweg 4). PCs are available where students can log in with their NetID. During the first assignment of the first quarter, software will be used that is running under Windows, during the rest of the first quarter and during the second quarter, software will be used that is running under Linux. See Appendix A for a short introduction to Linux.

## **2.3 Assistance**

Student assistants will be available for technical questions and guidance. Each project group of 7-10 students will also be supervised by a tutor, where each tutor supervises two groups.



## Part II

# Lab Course Integrated Circuits

### 3 Introduction

During the course labs that are part of Integrated Circuits (Geïntegreerde Schakelingen) 4 half days are available for the following 3 activities:

- A SPICE simulation assignment where parameters for a simple MOS transistor are derived. This assignment is described in sections 4 till 11. A report has to be written about this assignment, which is evaluated by a tutor.
- A step-by-step tutorial to get acquainted with the Sea-of-Gates design flow, followed by the design of a simple Finite-State Machine, using this design flow. This is described in Section 12 till 16. The result should be marked by one of the assistants.
- The last activity is a group meeting where initial plans are discussed for the design that will be made during the EPO-3 project in the second quarter. See Sections 22 and 23 for how this project will be done and for ideas on group designs.

## 4 Introduction SPICE Assignment

The following sections describe a SPICE simulation assignment for the course labs with the course EE2C11 Integrated Circuits (Geïntegreerde Schakelingen). The lab both contributes to the learning goals of the course, and it prepares for the EPO-3 project EE2L11 Design a Chip at TU Delft.

The following are the three main learning objectives:

- To increase the insight in the characteristics and behaviour of MOS transistors.
- To practice in designing and carrying out experiments (in this case, simulations) in order to measure a certain parameter, show a certain effect or to test a hypothesis.
- To improve reporting skills, in particular with respect to reporting of technical-scientific research activities and results.

In this lab, you will carry out SPICE simulations, according to certain moderately detailed instructions. You should also interpret your results and report about this interpretation according to a specific format.

This document is outlined as follows. Section 5 describes the technology data for the lab. Section 6 describes the 'how' of the lab. Section 7 introduces a BB module on information literacy that you should work through. It is about finding and using scientific information, i.e. citing of references. Section 8 introduces the reporting requirements for this lab. This includes a description of the (minimum) required contents. Section 9 describes the actual simulation assignment. Finally, Appendix 10 presents some tips for the using SPICE, in particular how to work with model parameter files and how to construct reporting-quality figures.

## 5 Technology Data

The simulation assignments require you to use the transistor parameters that go with the technology of the EPO-3 Sea of Gates image. These parameters can be found on BB. The following conditions apply:

Nominal $V_{DD}$ :	5V
Minimum channel length:	1.6 $\mu\text{m}$
Maximum channel length:	6.4 $\mu\text{m}$
Minimum channel width:	2.4 $\mu\text{m}$
Maximum channel width	10000 $\mu\text{m}$

The SPICE transistor parameters are according to the so-called *level 2 model*. See paragraph 3.3.4 of the Rabaey book. This model is also known as the *Grove-Frothman model*, so named because of the developers of this model.

## 6 Courselab Instructions

Execute in teams of 2 persons the assignment as presented in Section 9. The following conditions apply:

- You can do the assignments in teams of 2, possibly with a team of 3 in case of uneven number of group members. This has to be accepted by the TA's.
- You have to write and submit for assessment a report describing your work and results. Both (or all three) team members are equally responsible for all parts of the report, and can be questioned about it. There will be a review of the report with an instructor. The reporting instructions can be found in Section 8. The report has to be accepted by your tutor to get a pass for the assignment.
- You have to submit some key parameters that you derive in your assignment separately from your report, via Google forms. These results will be evaluated for consistency. If deemed questionable or not consistent, you may be asked for explanation or correction. The results will also be analysed statistically, showing potential effects of variability in CMOS.

## 7 Information Literacy 2

This lab includes and uses an online instruction module called Information Literacy 2 (Informatievaardigheden 2). This is a sequel module of the IV 1 module in the second semester of the first year. The focus of this second module is how to find and cite scientific information. Please see BB for the module and corresponding instructions for how to work it through.

## 8 Report

You must write a report on the assignment. The type of report that is being asked is a technical-scientific research report. It has to comply to a number of requirements, some of these relate to the structure and form of the report, others relate to contents. The most important rule here is the following: use *common sense* and use other documents as an example.

Please remember that good communication skills, both oral and written, are an important asset of engineers and scientists. This is the time to practice and further develop those skills.

### 8.1 General Requirements for the Contents of the Report

- You have to write the report as a standalone report, not as a document that only makes sense when simultaneously reading (or having read) this course lab manual.
- The contents of the report should contribute to the message that you want to convey. No more and no less. This means that you should not add contents that doesn't contribute to the message.

In turn that means that you need to think and rethink your proposed message very carefully while planning your report.

- Generally speaking, a scientific report or lab report should transfer knowledge to the reader, and enable the reader (at least in theory) to *repeat your experiments* in order to verify (or falsify) your results.
- The requirement above provides useful criteria about the level of detail to include in your report. It means, if only, that you need to document your assumptions and boundary conditions as well as your methods and tools. A such, tools include include physical tools (such as specific measurement equipment) and software tools. It for example also means that you need to include non-trivial settings your software tools. In this specific case, you are asked to carefully document the SPICE model that you have used (see Section 8.4).

In practice, the contents also strongly depends on limits to the number of pages allowed. Thus, the main challenge becomes to present the maximum amount of useful information in a readable and pleasant form.

- Information that is important but not critical to understand the main message of the report or would interrupt the flow of information too much, usually is best placed in an appendix.
- In scientific reports, interpretation of results is extremely important. This is true for all kinds of results, including e.g. measurements and simulations as well as new theoretical results. Just presenting results without discussion and placement in context is not useful, and won't be accepted in this course lab.
- Numerical results might require a discussion of the error margin.
- In science, correctly citing other work is of prime importance. The source/origin of any concept, idea, thought etc. should always be clear, and that requires all information that can not be considered common knowledge to be cited. Errors in this aspect are heavily charged, and can lead to disqualification. Play it safe. Cite all material that you use from other sources. If this is a picture/graph or similar, you can add a citation in the caption of the figure (and in the text where your first refer to that figure).

Building on the work of others is not a form of weakness, on the contrary. Science and technology can only make progress by using the work of those who came before you as a starting point. This calls for complete traceability of ideas, which in itself is an important reason for citing correctly. Another important reason is one of ownership of ideas: people should receive credit when others use their *intellectual property*.

Furthermore, please realize that your work becomes more credible and stronger if you use strong, high-quality and authoritative references.

The BB module 'Information Literacy 2' offers techniques and methods for finding and citing other work - you should use this. Also see Section 8.3 and 11) for *how* to cite (i.e. preferably in IEEE style).

## 8.2 Requirements for the Structure of the Report

- The report should have a logical structure, with an introduction, a problem description, a body (main part) and a conclusion. In some cases, the conclusion section is preceded by a discussion,



in other cases, the discussion is included in the conclusion section, see below. The report should also have a so-called *abstract*. The conclusion is followed by the references and, if applicable, the appendix or appendices.

- The abstract is a brief summary at the beginning of the report, as a kind of teaser. It should help the readers to decide if reading the report is useful or not, and actually encourage the target audience to indeed read the report.
- The introduction describes the problem to be solved, without going too deeply into this. The introduction also provides the motivation and context of your work: what is important, what bigger problem are resolved or addressed, etc... If applicable, the introduction also describes related work (with references). This means that it describes what others have done to the same or similar topics, what is going well, and which open problems remain to be solved or issues to be improved. The last part of the introduction traditionally is a description of the structure of the rest of the report.
- In a report like this, the problem statement can probably be incorporated completely in the introduction (in other situations, the introduction will only contain a summary of the problem with a full description in a separate section or chapter). In some cases, it works better to paraphrase the assignment in your own words, in other cases a literal copy of the assignment should be included. Use judgment. When the assignment is not fully specified and/or ambiguous, please indicate how you have interpreted the assignment and how you have restricted/defined it further.
- The body of the report can contain different parts. For example, it can contain a part with background knowledge and — theory. This part then has the goal of summarizing existing theory in such a way that the rest of the report can be read and understood more easily. One or more subsequent parts can contain your own work, how many parts these are really depends on the situation. A more or less common structure is background knowledge — method (explanation of experiment setup) — results, but this is only a rule of thumb. If these parts are short, they can be combined in one section or one chapter.
- The discussion section reflects upon the results achieved. Results w/o discussion or reflection seldom have the impact they could otherwise have. The discussion explains which goals have been achieved, which haven't, etc. Also, the discussion can compare the results to the expected outcomes. If there are discrepancies, what could be possible reasons for this? What are other strong points of the work, what are the weak ones? What would you do differently on another occasion? What are possible sources of measurement errors or interpretation errors, etc. What possibilities do you see to further improve your work? Also discuss, if applicable, possible implications of your work. If you made a discovery, what could be the impact on future developments?
- The conclusion is very important. Usually, it is relatively short and briefly summarizes the main results and reflection upon these results. When the conclusion section needs to be long, it is better to add a separate discussion section (see previous item). The conclusion is the place for any recommendations for subsequent work/research. (If this would become too long, it can also become a separate section.) The conclusion is also the place for some brief final remarks.

Be aware that some readers only read (or skim!) the abstract and the conclusion before deciding to read (no) more. This can also be a guideline for you to decide what to include in the conclusion.

- More comprehensive reports also include a table of contents, preface, etc. That is probably not the case for these reports.

### 8.3 Requirements for the Layout of the Report

- The report should look good and 'groomed', but not particularly fancy. Rather, sober and business-like. Spelling and grammar mistakes should be avoided. Failure in this respect means a smaller *signal to noise ratio* of your message, hampering communication quality and grade :-).
- If you overlooked item one above: The report should look good and 'groomed', but not fancy. Spelling and grammar mistakes should be avoided.
- Provide a clear title page that includes names and student numbers. Report title pages usually also contain information on the context (in this case: course lab for EE2C11 at TU Delft), date, and if necessary version information.
- Graphs and other images should preferably be 'vector images' rather than 'bitmaps'. When bitmaps can not be avoided, make the resolution and image quality settings high enough so that so-called 'JPEG artifacts' (also 'compression artefacts') are not seen.
- Graphs and images should have clearly readable titles. The text and numbers on the axes must be in a font that matches the size of the graphics, which should match the size of the body text. The axis should be properly labeled, and legends provided as necessary. Captions should be provided, and should be descriptive. Table captions go above the tables, figure captions go below the figures.
- The bibliography must be prepared in accordance with the usual rules, preferably in the so-called IEEE style. This style is the most common style in the field of Electrical Engineering. See Section 11.
- Simple and small formulas etc. can be given in line, and larger and/or more important formulas should be centered on a line by itself. If numbered, the number appears within round brackets at the right margin. By way of example, for real numbers  $a$ ,  $b$  and  $c$ , the roots of the quadratic equation  $ax^2 + bx + c = 0$  are given by

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \quad (1)$$

You can refer to this equation by Eq. (1). However, not all equations need to be numbered. Please remember that all symbols need to be defined.

- You should use a common font, font size and page layout (margins, line spacing, etc.). It may be in one or in two column format. In LaTeX or Word.

## 8.4 Specific Requirements for the Contents of the Report

Apart from the reporting on the assignment (see Section 9), in which you a.o. present the methods applied and results achieved and place them properly into context, the report should include material as specified below. Here, you should use that what you have learned from the IV 2 module.

### Validity of the SPICE model

The report should specifically address the validity of the transistor model that you will use with PSPICE. This model is the so-called *Level 2* model, also known as the *Grove-Frohmman model*.

In general, simulation results can only be accurate and applicable if based on a model that is accurate enough. Now, it turns out that the transistors that need to be simulated in this lab show *velocity saturation* and therefore the SPICE model that you will use should include this effect. This is not automatic, since it turns out that a simpler transistor model such as *Level 1* doesn't include velocity saturation, hence wouldn't be suitable. In the report, you should address this issue by finding a suitable publication of good quality that describes/discusses the *Level 2* model, and adding a discussion in your report in which you use the authority of that publication to positively address the suitability of the *Level 2* model for your purposes.

### Handling of Variability

The report should specifically address how variability is dealt with in modern IC design.

In practice, transistor parameter will vary from device to device, from chip to chip and from wafer to wafer. This is caused by both systematic and random fluctuations in the manufacturing process. In the lab, all groups will work with a different set of transistor parameters. Compared to the baseline SPICE parameters of the SoG technology, the following parameters are assumed to be variable.

parameter	$3\sigma$ variation
$V_{T0}$	$\pm 15\%$
$\mu_0$	$\pm 15\%$
$x_d$	$\pm 15\%$

Here,  $3\sigma$  variation assumes a Gaussian/normal distribution, with the given percentages as 3 times the  $\sigma$  value. That means that 99.7% of all parameters are expected to be in the indicated range from the nominal/average value.

Dealing with such parameter variations is a real-world problem in state of the art chip design. Depending on the particular values of parameters across a chip, different chips can show great differences in performance (speed, power dissipation, etc.). Read Section 3.4 of the Rabaey book. For this assignment, you are required to add an explanation of about 300 - 500 words on how variability is managed in chip design in practice, with the aim of optimizing the fraction of chips that meet performance without the design becoming uneconomical e.g. because of maintaining large but unnecessary safety margins. Support your explanation with at least one good-quality scientific reference not older than 10 years.

## 9 Assignment

### 9.1 Parameter Set

As explained above, all groups of students work with a different parameter set, simulating CMOS device variability as if you were all measuring a different device. The three parameters that we assume to be subjected to variability are  $V_{T0}$ ,  $\mu$  and  $x_d$ . You can find the parameters of your group by executing a Matlab script with a study number as an argument. Please use the lowest study number of both (or three) group members for non-ambiguity. The Matlab script is called `parameters.m` and can be found on the BB page of EE2C11, in the 'contents' folder. The script contains instructions for how to use it.

To simulate a transistor subject to variations, you have to change the model parameters in the `ModelLibEPO3.lib` file. (The base file can also be found on BB). Please note that in the `.lib` file,  $V_{T0}$  is identified as `vt0`,  $\mu$  as `uo` and  $x_d$  as `ld`. See Appendix 10 for how to work with the model files.

### 9.2 Unified Model Parameters

Determine for an NMOS transistor from the EPO-3 Sea of Gates technology, subject to the parameter variations as explained above, the following parameters of the Unified transistor model from the Rabaey book:

$$V_{T0} \quad k \quad \lambda \quad V_{DSAT}$$

If you have those parameters, compare in your report the results of the Unified Model with SPICE simulations, similar to what has been done in Figure 3.25 from Rabaey. To this end, you have to make a graph, or multiple graphs, with the two sets of curves for both the SPICE simulation and an evaluation of the Unified Model using your parameters. One approach is to create a Matlab script for the Unified Model, and use it to plot the model results as well as the exported waveform data from PSPICE. You have to discuss and explain the differences.

A set of suitable Matlab scripts are provided in BB, along with the other resources. A zipfile called 'mos\_matlab\_models.zip' contains a reasonably extensive set of matlab scripts. They are mostly sufficiently self-documenting. A good script to use for the purposes of the lab would be `mos.m`, but you have to provide your own scaffolding code to do the comparison and produce the required graph(s). Other scripts do provide some scaffolding code. The models are from [1].

The recommended approach for finding the parameters of the unified model is given below.

#### Determination of $V_{T0}$ and $k$

$V_{T0}$  and  $k$  can be derived from a square root of  $I_D$  vs  $V_{GS}$  plot (the square root of  $I_D$  on the vertical axis and  $V_{GS}$  on the horizontal axis), in the saturation region. Explain (in the report) that  $V_T$  is given by the intersection with the x-axis. Also give and explain a formula for computing  $k$  from the slope of  $\sqrt{I_D}$  vs  $V_{GS}$  and estimate  $k$  in this way. If you are interested: also try an approach in which you plot  $i_D$  (not the square root) against  $V_{GS}$ .

For this, you should experiment with different values of  $V_{DS}$  in the range of about 50 mV to 5 V, and notice that you will find different values of the threshold voltage — this is consistent with the so-called DIBL effect (see the Rabaey book, Section 3.3.3). You will also find different values for the slope, which is because the effect of channel length modulation ( $1 + \lambda V_{DS}$ ) is mixing in. Compare and explain the  $V_T$  variations so found in your report. For fitting the Unified Model, you should use the value of the threshold voltage for the case in which the DIBL effect is not manifest as the value for  $V_{T0}$ .

### Alternative Approach for $k$

Alternatively,  $k$  can be derived from the slope of  $I_D$  vs  $V_{DS}$  at  $V_{DS} = 0$ . Derive a corresponding formula and use this to estimate  $k$ .

You will probably see that this value of  $k$  is significantly smaller than the value of  $k$  derived in the previous approach. This is not an error of either method. In fact, mobility (and thus also  $k$ ) depends on the vertical field strength (which in itself depends on  $V_{GS}$ ) because of an effect called *surface scattering*. This effect is modelled in the Level 2 SPICE model, but not in the Unified Model. For the Unified Model, it is better to use the latter (smaller) value of  $k$ . In your report, you have to compare the different methods to obtain  $k$ , and the values, and explain why you pick one of these values for fitting the Unified Model.

### Determination of $\lambda$

$\lambda$  can be derived from the slope of the  $I_D$  vs  $V_{DS}$  plot at fixed  $V_{GS}$ , in the saturation region. If you linearly extrapolate the saturation current to the left, it intersects the horizontal axis at a (negative) voltage  $V_E$ <sup>1</sup>.

For typical MOS devices in practice, this intersection is each time at about the same voltage. In the Unified Model, with the drain current being given as  $I_D = I'_D(1 + \lambda V_{DS})$ , this intersection is always at the same voltage. Prove this fact, and give the formula for  $\lambda$  as a function of the intersection voltage  $V_E$ . Subsequently, perform the extrapolation for different values of  $V_{GS}$  and take some average value for the estimate of  $V_E$  and derive  $\lambda$ .

### Determination of $V_{DSAT}$

$V_{DSAT}$  can not easily be derived mathematically, yet a visual estimation from a set of  $I_D$  vs  $V_{DS}$  curves with  $V_{GS}$  as parameter, and optimizing the fit of the corresponding curves of the Level-2 model and the Unified model, will do. (You will actually see that  $V_{DSAT}$  depends on  $V_{GS}$ , take a suitable average value.)

<sup>1</sup> We use the symbol  $V_E$  since this intersection with the horizontal axis is similar to what happens with bipolar transistors because of the so called Early effect (named after the discoverer of that effect, James M. Early) This effect causes extrapolated collector currents for different  $V_{BE}$  to intersect in a common point (the *Early voltage*) on the negative  $V_{CE}$  axis.

### 9.3 Optional Assignments

#### Determination of Effective Channel Length

See the description of effective channel length on page 92 of the Rabaey book, and find the paper by Chern [2]. Use the method in that paper to find  $x_d$ . You can compare  $x_d$  with the corresponding parameter in the SPICE model that you have used. If you have  $x_d$ , you can further calculate  $k'$ .

#### Determination of Gate Capacitance

Study Example 3.9 of the Rabaey book and use it to extract the gate capacitance. If you have the gate capacitance, use it with the other parameters that you have extracted to further compute/estimate  $t_{ox}$ ,  $\mu_n$  and  $\xi_c$ . Although  $\mu$  is technically an input of your work, it is interesting to see how close you can match that value.

## 10 Use of PSPICE

### 10.1 Adding Libraries

To use the correct spice models for the simulations, the EPO3 Model Library has to be made available for PSpice. This works as follows.

1. Download the `ModelLibEPO3.zip` from the BB page of EE2C11 and unpack this file in your home directory. You should see `ModelLibEPO3.slb`, `.lib` and `.olb`.
2. Start PSpice Schematics.  
*Start → All Programs → Engineering → PSpice Student → Schematics*
3. Add the EPO3 Symbol Library (.slb).  
*Options → Editor Configuration → Library Settings → Browse → Add*
4. Add the EPO3 Model Library (.lib).  
*Analysis → Library and Include Files → Browse → Add Library*

**NOTE:** *ModelLibEPO3.olb is not needed to add specifically, but is necessary for simulation anyway.*

Use the `NENH` and `PENH` for the NMOS and PMOS transistors respectively. Parameters can be adapted upon double-clicking on a component.

### 10.2 Exporting from PSPICE

An easy way to export figures (waveforms, schematics), is to print to pdf. The pdf can be used easily in other programs (e.g. Latex).

---

Data points can be exported into Matlab or Excel, for further processing. Exporting from PSpice works by selecting the name of the required curve and using → Copy (or Ctrl+C). Then paste in the program.

## 11 IEEE Citation Style

The IEEE is the largest international professional association in the field of Electrical Engineering. As such, it is also a publisher of many scientific journals and other periodicals, conference proceedings and standards, and very influential at this. The so-called *IEEE Style* is in many cases the preferred style guide for many types of documents in the field. Formatting of citations and bibliographies is an important element of style, and as such the IEEE citation style [3]) is preferred for the course lab (and for many other reports and documents during your education, usually inclusive of your BSc thesis, for that matter).

You are not required to follow the exact detailed rules as outlined in the official IEEE style guides (such as [3], also cited above), but it *is* recommended to follow the most distinctive aspects of this style:

- Citations are numbered, and included in the text between square brackets (not superscripts).
- The bibliography is a list of references at the end of the document, ordered in order of appearance in the text, and numbered with the same square brackets.
- Author names are included, as the first element(s) of the bibliography item, usually *initials, lastname*. See for example reference [2].

See the citation style and references in this manual, and generally in all works published by the IEEE, as an example. The information that has to be included in the bibliography items must make the citation precise and should aid retrieval/searching for the document. In general, the information to be included is the same as in other styles such as APA.

## 12 IC design software

In this chapter the various computer programs for designing and verifying an integrated circuit will be discussed. It is not too complicated to deal with these computer tools. You will soon become familiar with it. However, the problem of IC design is that we have to deal with many components that all should be laid out on the chip correctly, without any errors. The computer will help you with that, but programs are not always smart enough and they sometimes also contain errors. As a result, the computer programs may sometimes produce not so optimal or even incorrect results. Therefore, it is important that results are inspected critically and that extensive verification/simulation is performed.

The computer programmes that you will be going to use originate from various sources. Partly they are commercial tools, like VHDL-simulation program ModelSim that you have already been using in the first year, and the logic synthesis tool from Synopsys. The design system OCEAN/NELIS, to create IC layouts, was developed at our own university by master and PhD students. In this chapter we will first see which steps are necessary to implement a circuit in Sea-of-Gates technology. These steps are indicated in Figure 1.

We can distinguish the following steps:

- First the VHDL-files have to be created.  
For a certain part of the design (an "entity") first a "behaviour" description or a "structural" description has to be created. How this is done is further described in Chapter 14. Additionally, a VHDL-file for a testbench has to be created. With the testbench, the entity can be tested for correct functioning using simulation. The testbench will consist of the entity to be tested and signals that are applied to the inputs of the entity. Through the outputs of the entity the behavior can be verified.
- So the next step is that the created description are verified for correctness by means of a VHDL-simulator. Depending on the results of the simulation the next step can be done, or, in case the simulation results show that the design is not working properly, the VHDL-descriptions have to be modified. The updated descriptions are simulated again, and these steps are repeated until simulation shows correct circuit behavior.
- The next step is logic synthesis for the behavior descriptions. For this, besides the VHDL-descriptions, also a library with basic cells (logic gates, flip-flops etc.) is needed, from which the circuit to be designed can be constructed.

During synthesis the following steps can be distinguished:

- The VHDL-descriptions are converted into logic functions.
  - These functions are being optimized.
  - The optimized functions are implemented using the cells from the library.
  - From the synthesized circuits, a structural VHDL-code is generated, thus consisting of only library-cells and their connections.  
At the same time a SLS-file is generated that describes the circuit also in another format. This description will be used to create the layout.
- The synthesized circuit next has to be simulated. This can be done using the same testbench that was also used for simulation of the original VHDL-code. Now, however, by means of a



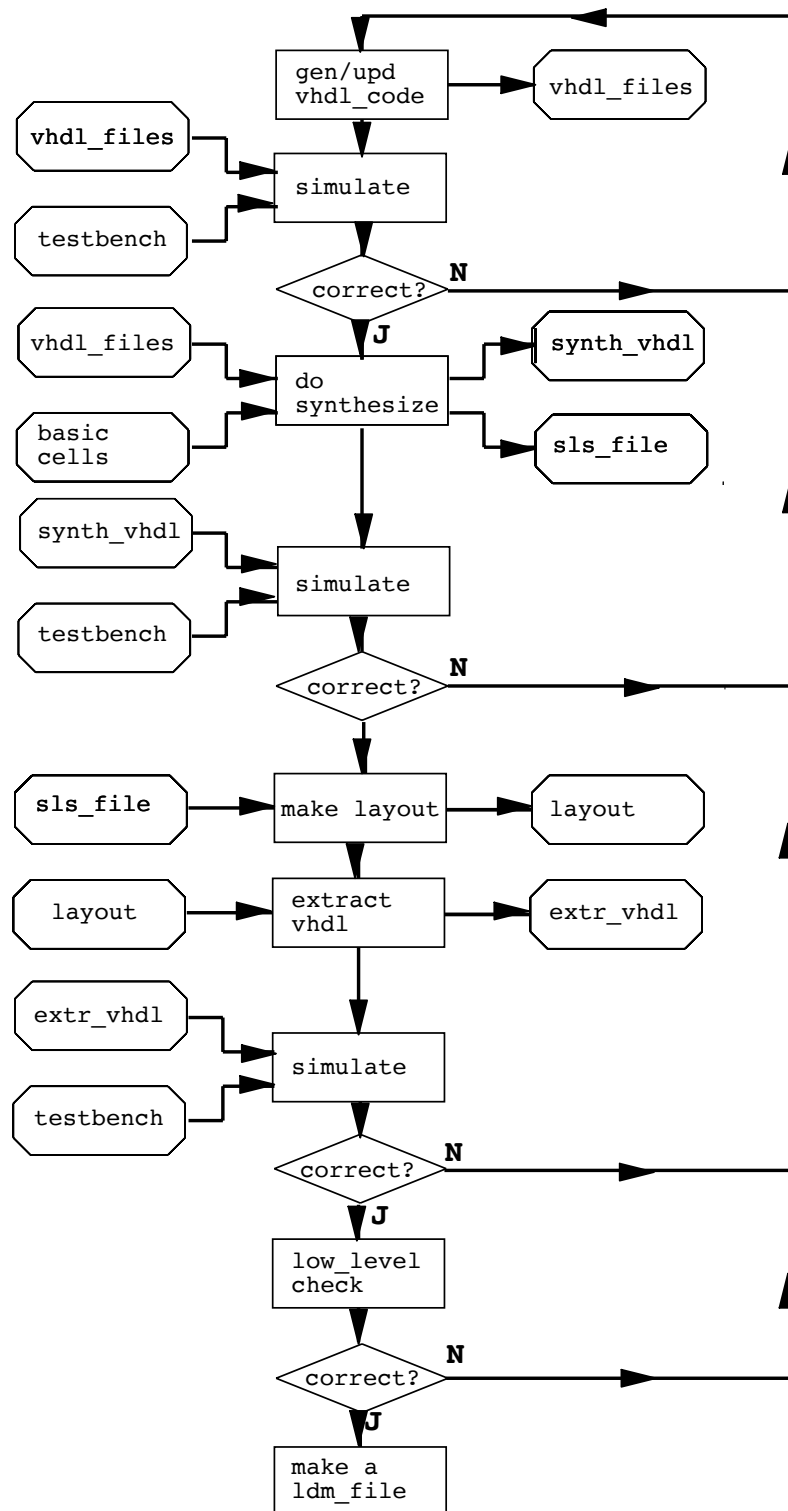


Figure 1: The design-flow for IC design

configuration statement it has to be specified that the synthesized version of the circuit should be used.

When simulation reveals that the synthesized circuit also has a correct behavior, the next step can be executed. Otherwise, the original VHDL-description has to be modified, such that the synthesized circuit also shows correct behavior.

- Next a layout is generated from the SLS-descriptions. The cells present in the SLS-description are first automatically placed on the layout area using a placement-tool (for this the tool *madonna* or the *row placer* can be used). If necessary, the placement can be adjusted by hand. After that, the connections between the cells can be generated using the routing-tool *trout*.
- To check the result, VHDL-code can be extracted from the layout, which will consist of all basic cells that are part of the designed circuit, including their connections. This circuit-extraction takes place using the *space*.
- Also this description should be simulated using the testbench, in which case now, using a configuration statement, it is specified that the extracted version of the circuit is used. When this simulation is successful, the next step can be executed. If the simulation is not successful, then a closer inspection of the results should show what is wrong and what should be improved in the design.
- As a pre-last step of the design process, another simulation can be executed that simulates the design at the transistor level. This step consists of the following sub-steps:
  - The transistor-circuit is extracted from the layout and the unused transistors from the sea-of-gates image are first removed.
  - From the initial simulation results for the VHDL-description, a command-file and a reference-signal-file are created. The command-file contains the input signals that are applied to the circuit-under-test, and the reference-file contains the results of the (reference) simulation.
  - The transistor-level simulation is now executed using the command-file.
  - Next, the result of the simulation, a result-file, is compared to the reference-file obtained from the VHDL-simulation. If both results match, the design is ready and the last step can be executed. If the results do not match, a closer look has to be taken at the differences to find out what exactly is wrong and how this can be solved.
- The last step is that a file is created that describes the layout of the design (a ldm-file).

## 13 IC design tutorial: A hotel switch

### 13.1 Introduction

In this section an example will be given about how to design a simple circuit. First, from a given specification, a Finite-State Diagram is created. Next, based on this, a VHDL behaviour description is derived, which is then simulated and synthesized. Finally the layout is created. Before going through this tutorial, it is recommended to read Appendix A.

Assignment: Design a circuit that controls a lamp using 3 push buttons (s1, s2, s3). When 1 or more buttons is pushed, the lamp will switch to on or off, depending on the current state. An overrule switch (ov) can be used to prevent the lamp from switching to another state. Figure 2 gives an overview of the circuit. An external OR gate is used to generate the OR function s of input signals s1, s2 and s3. The circuit to be designed is in the black-box with question mark.

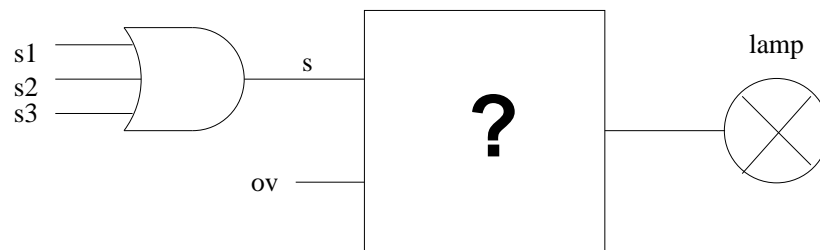


Figure 2: Design of a hotel switch

### 13.2 Finite-State Diagram

Figure 3 shows the Finite-State Diagram that can be created for the circuit.

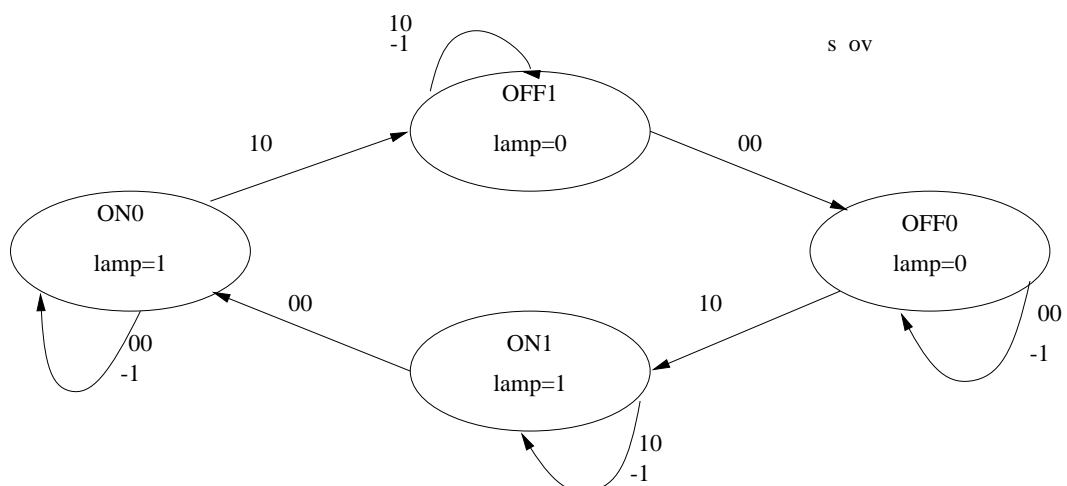


Figure 3: Finite-State Diagram for the de hotel switch

### 13.3 Creating a project

First be sure that once the command `op_init` has been executed in a terminal window, to set the search paths to the software tools. (see Appendix A).

Now, start the program *GoWithTheFlow* by clicking on the corresponding icon at the DeskTop. This will show the interface of the tool as shown in Figure 4.

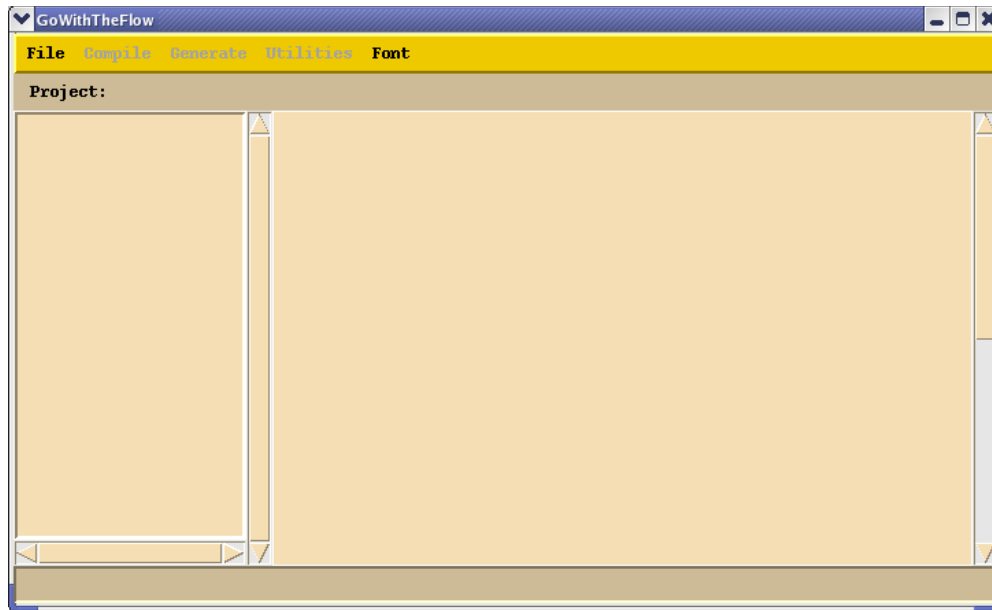


Figure 4: The program *GoWithTheFlow* after start up

Next, create a project directory using the command `File → New project`. A window as in Figure 5 will be appear. Specify in the field behind Selection the name of a non-existing directory and click on OK.

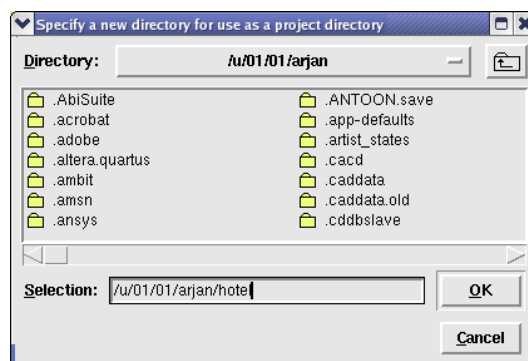


Figure 5: The creation of a project directory

## 13.4 VHDL behaviour description

First enter the entity description for the hotel circuit by clicking on File → New entity and filling out the widget that appears as in Figure 6. The tab key can be used to go through the next column. Press the enter key at the end of each line.

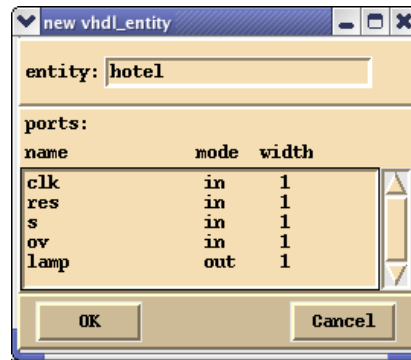


Figure 6: Specifying a new entity

After clicking on OK, the corresponding VHDL description will appear in a new window, see Figure 7. Click on Compile. After confirming that the description will be saved as a file, the description will

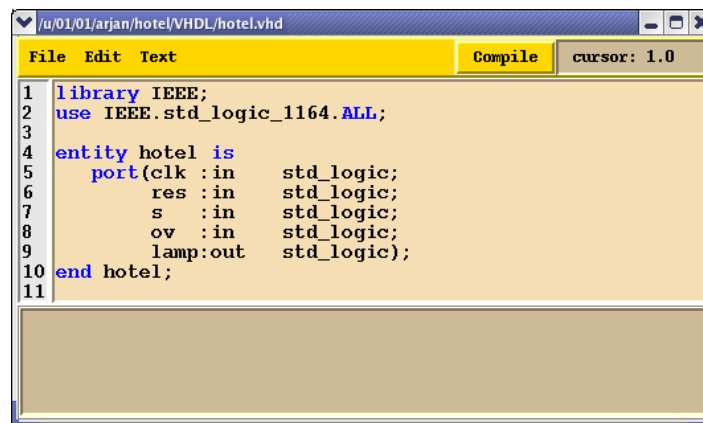


Figure 7: De VHDL code voor de hotel entity

be stored in the database of *GoWithTheFlow* and will be compiled.

Next, click on the rectangle "hotel", select Add Architecture and enter the following behaviour description. Notice that (similar to what is described in Chapter 14) the first process statement (lbl1) describes that state registers, and the second process statement (lbl2) describes how the next state and output signals are derived from the current state and the input signals.

```

library IEEE;
use IEEE.std_logic_1164.ALL;

```

```

architecture behaviour of hotel is

```

```
type lamp_state is (OFF0, OFF1, ON0, ON1);
signal state, new_state: lamp_state;
begin
  lbl1: process (clk)
  begin
    if (clk'event and clk = '1') then
      if res = '1' then
        state <= OFF0;
      else
        state <= new_state;
      end if;
    end if;
  end process;
  lbl2: process(state, s, ov)
  begin
    case state is
      when OFF0 =>
        lamp <= '0';
        if (s = '1') and (ov = '0') then
          new_state <= ON1;
        else
          new_state <= OFF0;
        end if;
      when ON1 =>
        lamp <= '1';
        if (s = '0') and (ov = '0') then
          new_state <= ON0;
        else
          new_state <= ON1;
        end if;
      when ON0 =>
        lamp <= '1';
        if (s = '1') and (ov = '0') then
          new_state <= OFF1;
        else
          new_state <= ON0;
        end if;
      when OFF1 =>
        lamp <= '0';
        if (s = '0') and (ov = '0') then
          new_state <= OFF0;
        else
          new_state <= OFF1;
        end if;
    end case;
  end process;
end behaviour;
```

Click on Compile. The program will first ask to save the description to a file, on which you should answer yes. Then, when there are no errors in the description, a rectangle with the text "behaviour" will appear below the rectangle "hotel". This means that the description has been stored in the database of *GoWithTheFlow*. In case there are compilation errors, these errors will be shown and they will first have to be solved before it is possible to continue.

Next place the cursor on the "behaviour" rectangle, click with the left mouse button and select Add Configuration. A window will appear that proposes a new for the VHDL configuration file. Click on OK. The VHDL description for the configuration file will be shown. Next save this file and compile it.

In a similar way as described above, create an entity , a behaviour and a configuration description for a testbench for the hotel circuit. Call the entity "hotel\_tb" (it is not necessary to add ports to this testbench entity) and use the following behaviour description:

```
library IEEE;
use IEEE.std_logic_1164.ALL;

architecture behaviour of hotel_tb is
    component hotel
        port (clk : in std_logic;
              res : in std_logic;
              s   : in std_logic;
              ov  : in std_logic;
              lamp : out std_logic);
    end component;
    signal clk: std_logic;
    signal res: std_logic;
    signal s: std_logic;
    signal ov: std_logic;
    signal lamp: std_logic;
begin
    lbl1: hotel port map (clk, res, s, ov, lamp);
    clk <= '0' after 0 ns,
           '1' after 100 ns when clk /= '1' else '0' after 100 ns;
    res <= '1' after 0 ns,
           '0' after 200 ns;
    s <= '0' after 0 ns,
         '1' after 600 ns,
         '0' after 1000 ns,
         '1' after 1400 ns,
         '0' after 1800 ns,
         '1' after 2200 ns,
         '0' after 2600 ns,
         '1' after 3000 ns,
         '0' after 3400 ns,
         '1' after 3800 ns;
    ov <= '0' after 0 ns,
```

```

    '1' after 1800 ns,
    '0' after 2600 ns;
end behaviour;

```

The main window of *GoWithTheFlow* should now present itself as shown in Figure 8.

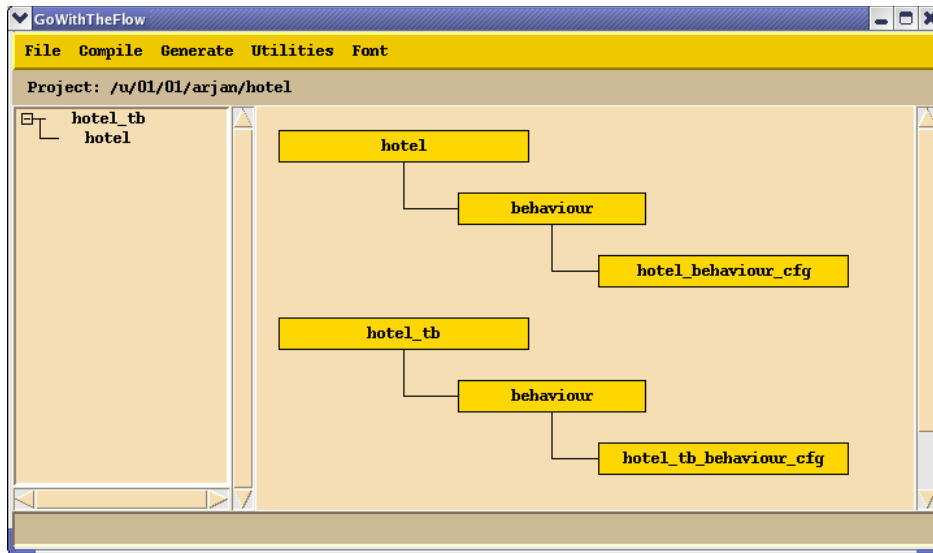


Figure 8: *GoWithTheFlow* after adding behaviour and testbench

Start a VHDL simulation by clicking on `hotel_tb_behaviour_cfg` with the left mouse button and select simulate. After simulating over for example 4000 ns ModelSim will show a result in a wave window as shown in Figure 9. To be able to perform a switch-level simulation (a simulation at transistor level)

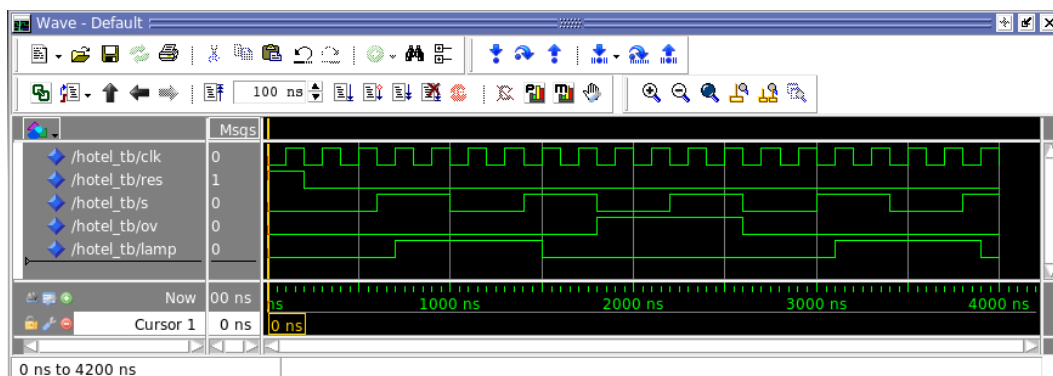


Figure 9: Result of VHDL simulation with ModelSim

later on for the layout of the circuit, we will now first create a .lst file that contains a description of the simulation input and output signals. Therefore, select in the Tools menu of the wave window the command `Make.list.file`. Next, select the hotel behaviour description and click on OK. And finally, save the file as `hotel.lst`.



### 13.5 Logic synthesis

Next we will do a synthesis step for the behavioural description of the hotel entity. In addition, the behavioural description is transformed into a network description consisting of cells from the Sea-of-Gates cell library. To do this, click the left mouse button on the hotel's configuration rectangle and select *synthesize*. The synthesizer window will now appear. Click *makeSyntScript* and then *Synthesize*. Optionally, clicking on *Show circuit* can be used to see a schematic of the synthesized circuit. Then click *Compile* to compile the VHDL description and *Parse\_sls* to place a SLS circuit description in the backend (OCEAN/NELISIS) library. There will now be rectangles for the synthesized VHDL description and the sls circuit description be visible in *GoWithTheFlow*.

The synthesized VHDL description can now also be simulated with the previously created test bench. First, add a configuration to the synthesized VHDL description. Then click on the testbench behaviour description and add a new (second) configuration. Give the new configuration a name that refers to the synthesis, for example, *hotel\_tb\_behaviour\_syn\_cfg*. It will now be asked which version of hotel should be chosen for the new configuration of *hotel\_tb*. Select the synthesized version and click OK. For the new configuration, a simulation can now be started.

### 13.6 Creating a layout

A layout can be made from the synthesized circuit description. Click on the rectangle circuit and select *Place & route*. The program *seadali* will now be launched. Go to the automatic tools menu, and then execute the steps *place* (using *madonna*) and *route* (using *trout*). Afterwards, a layout as shown in Figure 10 will be displayed. Then write the layout using the database menu (return to main menu via -return-) and exit the layout editor.

An alternative to *Place & route* by means of *seadali* consists of first placing the components with the tool *row placer*. To do so, select the *Run row placer* option on the rectangle circuit. In general, this placement tool will give a better (more compact) placement than as is obtained by means of *seadali*. However, this placer can only be used to place components from the cell library and can not be used for circuits that contain other designed cells. Also here, the routing should be done with *trout* in *seadali*.

### 13.7 Verification of the layout

The layout can now be checked in 2 ways. The first way is to extract a VHDL description from the layout and simulate it with ModelSim. To do this, use the left mouse button on the layout rectangle and select *Extract vhd*. In the new window, click *Get*, then *Write*, and then *Compile*. Then add a configuration as previously done. Now, for the testbench behaviour description of hotel, make another configuration where the extracted version of hotel is selected. In this way the extracted VHDL description can be simulated. The main window of *GoWithTheFlow* will now display an image as in Figure 11.

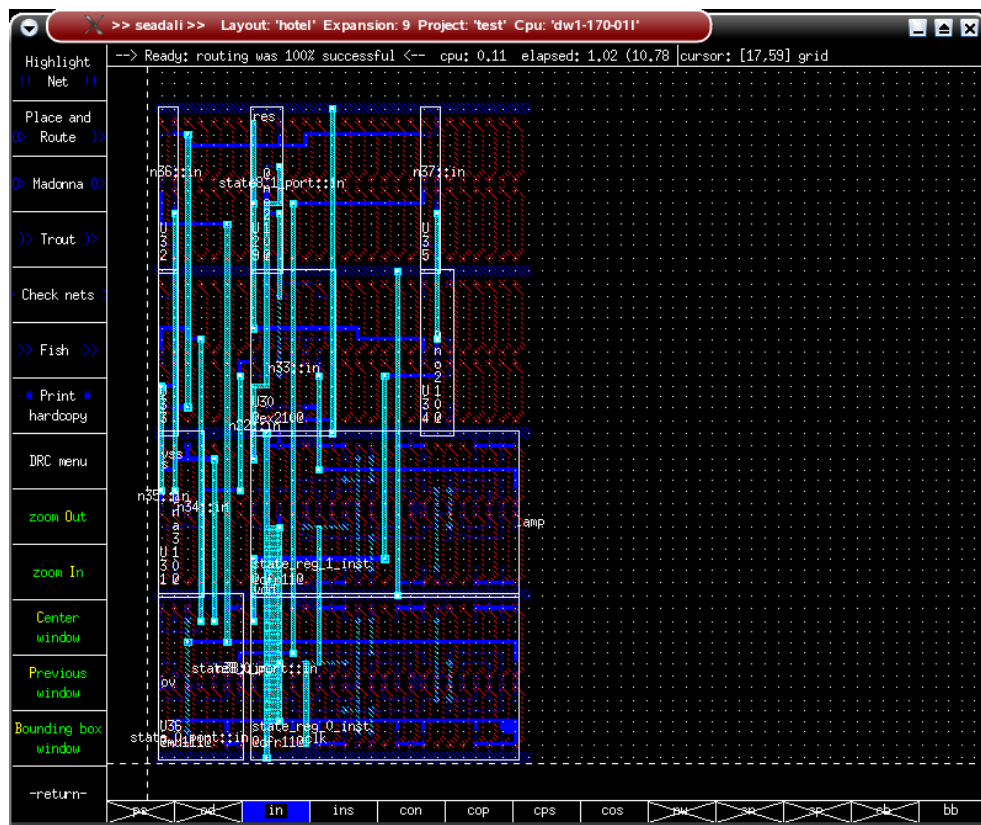


Figure 10: A layout for the hotel circuit

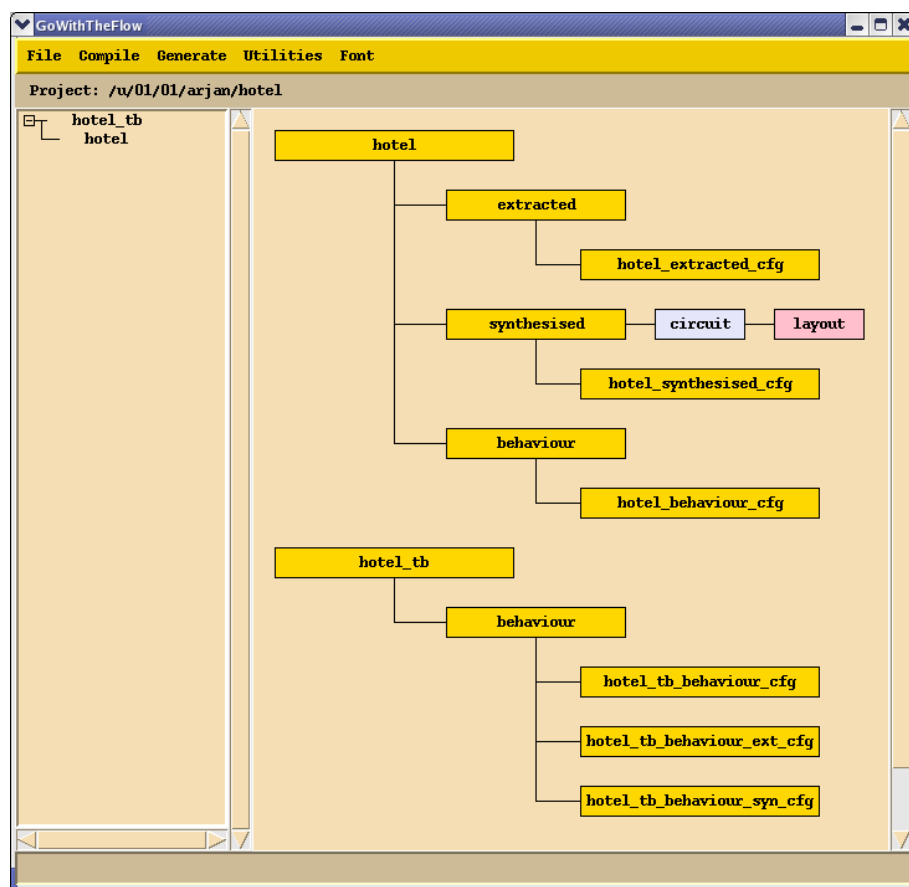


Figure 11: The main-window of the program *GoWithTheFlow* after going through all the steps of the hotel example

A more accurate way of verification consists of simulating a transistor description that is extracted from the layout. To do this, you first need to create a suitable command file for the switch-level simulator. Click Generate → Command file, and then click File → Generate from, and select the previously created file hotel.lst. Save the generated command file with File → Save as hotel.cmd. To simulate, click on the layout rectangle and select Simulate. After clicking Doit, an extraction will take place first and then a switch-level simulation. Afterwards, click ShowResult to see a result as in Figure 12. After zooming in, also the delays times for the output signal lamp can be observed.

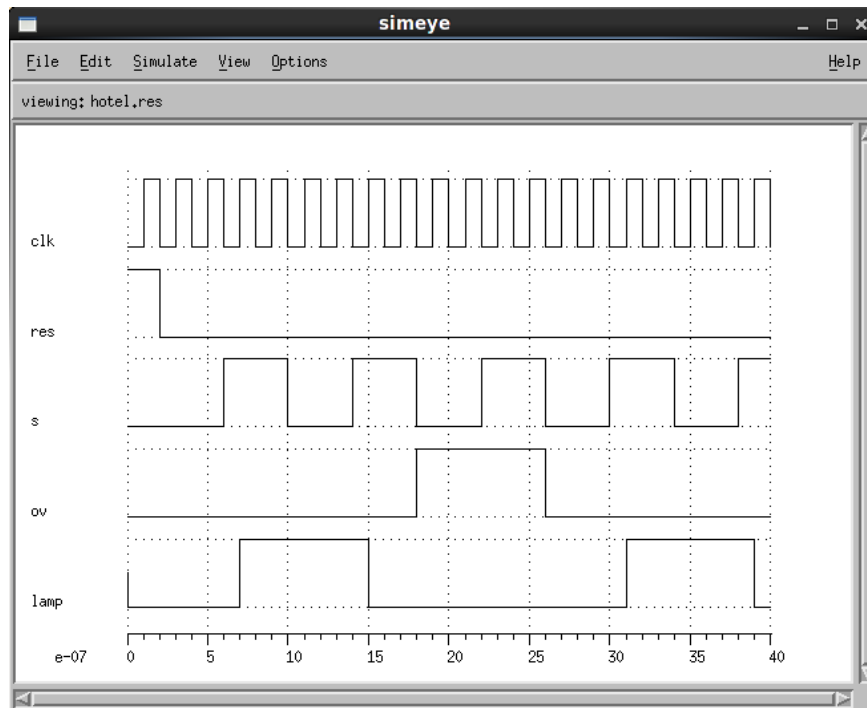


Figure 12: Switch-level simulation result

Although in this case it is clear that the results of the simulation on the transistor level match the results of the original VHDL simulation, we can also compare both simulation results by using the program *GoWithTheFlow*. To do so, we need a simulation reference file (.ref) file from the previously created .lst file. Therefore, in menu Utilities of *GoWithTheFlow*, click on Generate → Reference file. In the new window that appears click File → Generate from and select hotel.lst. Then click File → Write to save file hotel.ref and close the window. Then, click Utilities → Compare to have the compare window appear. Use File → Read ref to read hotel.ref and File → Compare Res to read hotel.res and compare them with hotel.ref. The output signals will be compared at times just before the next rising clock flank (see Figure 13). If everything is correct, output signals may only be different during the initialization period at the beginning of the simulation, when the reset is high.

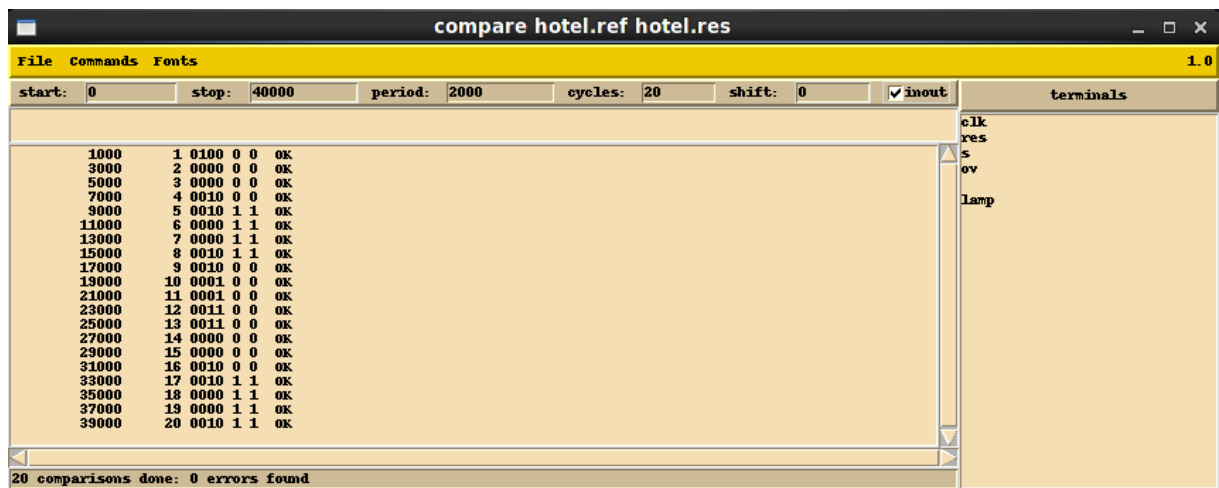


Figure 13: The comparison of simulation results

## 14 Guidelines to write VHDL code

### 14.1 Introduction

This chapter provides guidelines for writing "good" VHDL code. Following these guidelines will ensure:

- Clear code that is easy to debug.
- A great chance of successful logical synthesis (translation to hardware).
- A good match between simulation results and behaviour of the synthesized hardware.

First of all, it is important that the design is properly divided into hierarchical blocks, such that a block (entity) can be described by not too much VHDL code. When the VHDL description for an entity becomes too extensive, it is advisable to divide the entity into sub-entities, of course as much as possible on the basis of the functionality of the various components.

Also remember that VHDL code is not written on a way like for example a C program is written. Always realise that any VHDL entity describes a piece of hardware that corresponds to a combinatorial circuit, a sequential circuit (for example a FSM controller, a counter or a register), or a structural circuit (i.e., a circuit in which different other sub-circuits are connected together). Note that if one can not say how the VHDL code can be realized in hardware, the synthesis software will usually also not be to do that, or it may generate something that shows sometimes unexpected behavior.

Considering this, we will distinguish the following types of VHDL code and discuss them separately.

- Behavioural description combinatorial circuit
- Behavioural description sequential circuit
- Structural description of a circuit

Avoid as much as possible to bring these types together in one entity. The entities at the lowest level in the hierarchy will be of the type "behavioural", while the entities higher in the hierarchy will be of the type "structural".

Finally, we will also describe in this chapter how to make a "parameterized" VHDL description.

### 14.2 Behavioural description combinatorial circuit

A behavioral description of a combinatorial circuit must define the values of the outputs for all input combinations that may occur. If this is not done completely, the synthesis software will assume that previous values must be remembered in some cases and memory elements like latches will be generated for the circuit, This of course is not the intention and will give unwanted circuit behaviour.

With a combinatorial circuit, the architecture body will contain statements like process statements and data flow statements. The following is an example of a combinatorial circuit with a process statement.

Important when using a process statement is that in the sensitivity list all input signals for that process must occur.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity combinatorial is
    port (a, b, c : in std_logic;
          y       : out std_logic
    );
end combinatorial;

architecture behaviour of combinatorial is
begin
    -- a, b and c are input signals, y is output signal
    lbl1: process (a, b, c)
    -- All input signals must be in the sensitivity list
    begin
        -- Compute output value(s) based on input values
        -- Important: for every input combination, each
        -- output should receive a value !
        if ((a = '1' or b = '1') and c = '0') then
            y <= '1';
        else
            y <= '0';
        end if;
    end process;
end behaviour;
```

The following describes the same combinatorial circuit, but now in a more compact way using a dataflow statement:

```
architecture behaviour of combinatorial is
begin
    y <= (a or b) and (not c);
end behaviour;
```

### 14.3 Behavioural description sequential circuit

We assume that every sequential circuit that we design can be described by a finite state machine as shown in Figure 14. Therefore, in describing a sequential circuit we make a distinction between a register section that remembers the state ("statereg") and a combinatorial part that calculates the new state and the values of the outputs, based on the input signals and the current state ("combinatorial"). The combinatorial circuit "combinatorial" can be described in a similar way as outlined in the previous section. So set the output values for all input combinations!

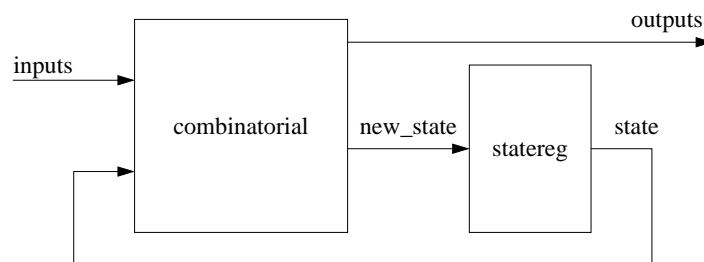


Figure 14: Schematic for a finite-state machine.

In the description of the part that remembers the state it is a good design practice to implement the memory elements by positive-edge triggered D-flipflops. Then we still have an option to choose flipflops with a synchronous reset or flipflops with an asynchronous reset. It is also possible to use flipflops without a reset, but this is not recommended since then it will not be possible to start operating from a fixed state.

When we choose a synchronous reset, when "states" displays the current state, "new\_state" the following state, "RESET\_STATE" the reset state, "clk" the clock signal and "res" the reset signal, then we can describe the section for storing the state with

```

statereg: process (clk)
begin
    if (clk'event and clk = '1') then
        if res = '1' then
            state <= RESET_STATE;
        else
            state <= new_state;
        end if;
    end if;
end process;

```

An example of a complete description of a sequential circuit then is:

```

library IEEE;
use IEEE.std_logic_1164.all;

entity fsm is
    port (clk      : in std_logic;
          reset    : in std_logic;
          input1   : in std_logic;
          output1  : out std_logic
    );
end fsm;

architecture behaviour of fsm is
    type state_type is (STATE1, STATE2, STATE3);

```



```

    signal state, new_state: state_type;
begin
    statereg: process (clk)
    begin
        -- Because we use positive-edge triggered memory elements
        -- (flipflops) with synchronous reset, we assign a new
        -- state when clock goes to '1'
        if (clk'event and clk = '1') then
            if res = '1' then
                state <= STATE1;      -- assign reset state
            else
                state <= new_state;
            end if;
        end if;
    end process;

    combinatorial: process (state, input1)
    begin
        -- Assign output values based on state only (Moore machine)
        -- and compute new_state based on current state and input
        -- signals. Important: for all possible states, every output
        -- should receive a value and new_state should be defined.
        case state is
            when STATE1 =>
                output1 <= '0';
                if (input1 = '0') then
                    new_state <= STATE2;
                else
                    new_state <= STATE1;
                end if;
            when STATE2 =>
                output1 <= '1';
                new_state <= STATE3;
            when STATE3 =>
                output1 <= '1';
                new_state <= STATE1;
        end case;
    end process;
end behaviour;

```

When we would choose an asynchronous reset then the first process would look like this:

```

statereg: process (clk, res)
begin
    -- State register using positive-edge triggered memory elements
    -- and an asynchronous reset (note that res now is in the sensitivity
    -- list since register must do something when res changes).

```

```

    if res = '1' then
        -- assign reset state
        state <= STATE1;
    elsif (clk'event and clk = '1') then
        state <= new_state;
    end if;
end process;

```

It is important to note that for the above description, during synthesis, STATE1 should be encoded with only 0 bits, because in our cell library only flipflops are available that can be reset asynchronously to the value 0. Because in the above example, STATE1 is the first value in the type enumeration, this will probably go well automatically. In other cases, it may be necessary to force the state encoding to encode the reset state with only 0 bits. This can be specified in the VHDL code by extending the state type in the following way with 2 attribute statements:

```

type state_type is (STATE1, STATE2, STATE3);
attribute enum_encoding : string;
attribute enum_encoding of state_type : type is "00 01 10"

```

such that STATE1, STATE2 en STATE3 are encoded by respectively 00, 01 en 10.

Another example of a sequential circuit is a 4 bit counter as given below. Note that in this case the state is formed by the 4 bits of the unsigned vector count and that the outputs are formed by the 4 bits of the std\_logic\_vector count\_out, derived directly from this.

```

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity counter is
    port (clk      : in std_logic;
          reset    : in std_logic;
          enable   : in std_logic;
          count_out : out std_logic_vector (3 downto 0)
    );
end counter;

architecture behavioural of counter is
    signal count, new_count : unsigned (3 downto 0);
begin
    statereg: process (clk)
    begin
        if (rising_edge (clk)) then
            if (reset = '1') then
                count <= (others => '0');
            else

```

```

        count <= new_count;
    end if;
end if;
end process;

combinatorial: process (count, enable)
begin
    if (enable = '1') then
        new_count <= count + 1;
    else
        new_count <= count;
    end if;
end process;

count_out <= std_logic_vector (count);

end architecture behavioural;

```

## 14.4 Structural description of a circuit

A structural description can be seen as a physical network/circuit of components and connections between them. Depending on which components are included, the structural description will be describing a combinatorial or a sequential circuit. The components are representations for entities and architectures of other parts from the design. The components may also be cells from the cell library that is available.

The following is an example of a structural description. In addition to the textual entry of the VHDL code, a structural description can also be created graphically using a schematic editor.

```

library IEEE;
use IEEE.std_logic_1164.all;

architecture circuit of network is
    -- First the declaration of the components used
    component comp1
        port (x: in std_logic; y: out std_logic);
    end component;
    component comp2
        port (p, q: in std_logic; r: out std_logic);
    end component;
    -- second the declaration of the nets (internal nodes)
    signal net1, net2, net3 : std_logic;
begin
    -- connections between nets and pins
    output1 <= net2;
    -- instances of components and their connections

```

```

    comp1_1: comp1 port map (x=>net1, y=>net3);
    comp1_2: comp1 port map (x=>net2, y=>net1);
    comp2_1: comp2 port map (p=>net3, q=>input1, r=>net2);
end circuit;

```

### 14.5 Parameterized descriptions

Using a design parameter that may be set at different values, can best be done by using a package. This package should then be included in all VHDL files where the parameter is used. In this way, there is only one place where the value of the parameter is set (in the package) and the value is referenced in other files. A package file "parameter\_def.vhd" to specify a parameter N can for example look like this:

```

package parameter_def is
constant N : INTEGER;
end parameter_def;

package body parameter_def is
constant N : INTEGER := 8;
end parameter_def;

```

To use this parameter N then in an entity or architecture description, the package should be called at the top of the file as follows:

```

use work.parameter_def.all;

```

In this way we can, for example, modify the previously given 4-bit counter into an N-bit counter in the following way:

```

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use work.parameter_def.all;

entity counter is
    port (clk      : in std_logic;
          reset    : in std_logic;
          enable   : in std_logic;
          count_out : out std_logic_vector (N-1 downto 0)
    );
end counter;

architecture behavioural of counter is
    signal count, new_count : unsigned (N-1 downto 0);

```

```
begin
  statereg: process (clk)
  begin
    if (rising_edge (clk)) then
      if (reset = '1') then
        count <= (others => '0');
      else
        count <= new_count;
      end if;
    end if;
  end process;

  combinatorial: process (count, enable)
  begin
    if (enable = '1') then
      new_count <= count + 1;
    else
      new_count <= count;
    end if;
  end process;

  count_out <= std_logic_vector (count);

end architecture behavioural;
```

In principle, it is also possible to parameterize a design with generic statements. However, then the synthesis software will create new names for entities, which leads to inconsistencies in the database of the project.



## 15 VHDL restrictions because of design tools

In this section the restrictions are discussed that have to be taken into account when using the design software. Adhering to these restrictions will prevent problems that sometimes only manifest at a later stage. **Therefore, it is in your own interest that you take these restrictions into account!**

- The "ports" of entities of VHDL descriptions that need to be synthesized may *only* be of type *std\_logic* or *std\_logic\_vector*. This is to prevent that all kinds of conversions have to be made when trying to use the same testbench for different descriptions. Also the conversion from VHDL to sics may fail otherwise.
- For the names of signals, ports and entities use *lowercase letters*. Otherwise, with the various data conversions that are taking place during the design, errors may occur.
- The names that are used for entities and ports must be distinguishable in the first 14 letters.
- Entity, architecture and configuration descriptions should preferably each be stored in a separate file.
- For a layout cell, the name should be equal to the name of the corresponding circuit cell. This is to prevent naming problems when extracting the VHDL code from a layout.
- Signals of type "inout" may only be used for "analogue" signals.
- A port of an entity may not be unconnected in its architecture description; This also applies for all members of the port when the port is a *std\_logic\_vector*.

The logic synthesis software for generating the circuits also sets restrictions on the VHDL descriptions. In addition to the general instructions in chapter 14 for making "synthesizable" VHDL descriptions, the following should be taken into account:

- Initializations of signals in their declaration are ignored.  
Thus, for example, *signal a: std\_logic := '0';* is read as *signal a: std\_logic;*
- After-clauses in statements are ignored.  
Thus, for example, *a <= '1' after 2 ns;* is read as *a <= '1';*
- Process statements must be created with a process list in which, in a register description (without asynchronous reset), only the clock signal is present, or, in a description of a combinatorial circuit, *all* signals are present that are read during the execution of the process.
- For case and select statements, an output value should be defined for *all* possible input values.
- Also, in every branch of case and if statements, *all* output signals should get a value.
- As asynchronous reset value, only the value 0 (encoded) can be used. See also Section ??.
- Do not use generic statements to parameterize designs, but use constants that are defined in a package. See also Section 14.5.

To be able to view the time delays of the gates in the circuits, the accuracy of the sls simulation time is default set to 100 ps. The maximum smulation time during switch-level simulation, without having an 'overflow' problem, then is 100 ms. *Take this also into account when simulating VHDL code, and adjust the clock frequency if necessary*



## 16 The FSM introductory assignment

### 16.1 Introduction

The FSM introductory assignment will further familiarize you with Sea-of-Gates design environment. With this assignment, a simple control (Finite State Machine) must be designed. Just as with the IC design tutorial in chapter 13 all steps of the design flow - from Finite State Diagram to layout - have to be completed.

First of all, the assignment must be carefully studied and the specifications, if incomplete, further determined. Then a Finite-State Diagram must be created. That is, it has to be decided which different states the circuit design can have and how, under the influence of input signals, transitions take place, see [4]. From the state diagram, a VHDL-description can be derived and tested, see [5]. Then this description can be synthesized. The synthesized circuit can be entered in the OCEAN/NELIS system and a layout can be generated. After a circuit extraction of the layout also the circuit implemented on the fishbone image can be tested for its functional operation, both at VHDL gate-level level and the transistor level.

### 16.2 Creating a Finite-State Diagram

Study your assignment, see Section 16.4.

Create a Finite-State Diagram according to the Moore model. That is, the output signals may not directly depend on the input signals, but only on the state itself. First, check what the input and output signals are and what conditions can occur. Give all the states and signals a functional name. Determine what the values of the output signals are for each state. Determine the state transitions as a function of the input signals. Make sure all possible combinations are covered.

### 16.3 Creating the Circuit

The next section will discuss what steps should be taken to eventually become a (working) circuit. The entire design can be done by means of the *GoWithTheFlow* interface. The following subsequent steps must now be taken into account:

- Create a VHDL behaviour description for the Finite-State Machine based on the Finite-State Diagram. See also chapter ???. Also make a testbench for the FSM.
- Compile the created VHDL files
- Simulate the testbench and see if the circuit behaves as expected. If necessary, edit the VHDL description. Also create the list file(s) for a later comparison with simulation results for the circuit extracted from the layout.
- Synthesize the VHDL description of the circuit. This synthesis provides two descriptions:
  - A new VHDL description, consisting of components from the OCEAN/NELIS library and their interconnections.

This can be compiled and then tested for correct operation by means of simulation. In case of incorrect operation, the VHDL behavior description may need to be modified.

- A description of the circuit in the SLS format.

This description can be used as a circuit input for the OCEAN/NELIS system.

- Generate through *madonna* (placer) and *trout* (router) of *seadali* the layout of the circuit.
- Extract the VHDL code from the created layout and simulate it to see if the layout works well.
- Generate reference and command files to test the circuit on transistor level.
- Run a simulation on the circuit with the generated command file.
- Compare the outcome of this simulation with the generated reference file. Apart from differences during the period when the reset is high, no other differences may occur.

## 16.4 Possible Assignments

### 16.4.1 The Ron Brandsteder Lights

To make T.V. shows more apparent it is necessary to regularly turn on and turn off lights on and around the stage, thus rendering the content of the show which needs less attention. The lights that should thought of here are in a circle. The number of lamps are a multiple of three. Each third consecutive lamp is connected to the same switch (see figure 15). This creates three groups lamps. Each group is switched on or off by a separate switch.

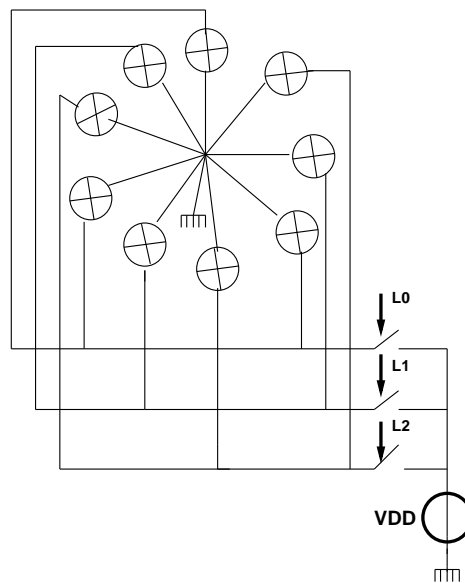


Figure 15: Connections for Ron Brandsteder Lights

You are instructed to design a controller for these switches. The control must have the following possibilities:

- All lights are off.
- The three groups are turned on clockwise after each other.
- The three groups are turned on counter-clockwise after each other.
- All lights remain in the same position.

The input of the controller consists of two control signals  $s_0$  and  $s_1$ . The output consists of three output signals  $L_0$ ,  $L_1$  and  $L_2$ , each one operating a switches.

### 16.4.2 The Basket

Joop van den Beginne has made a new T.V. program for the new T.V. season. During the program, the participants must throw a ball into a basketball net. When this succeeds, a horn goes off. Besides the horn signal, the basket board also has lamp that is off when the game starts that starts blinking when one hit has been made, and that stays on when a second hit has been made. The quiz master can bring the circuit in the initial state by using a push button.

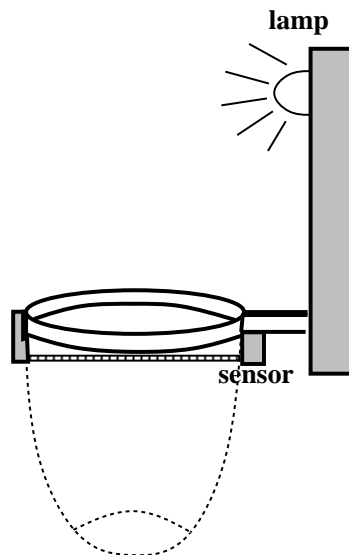


Figure 16: Situation for the basket

You should design the controller for the lamp and the horn on the basket board, taking into account the following conditions:

- On the basket ring there is a light sensor that gives a logical '1' when a ball is going through the ring. This signal is synchronised with the clock and lasts exactly one clock cycle.
- The offered clock has a frequency suitable for a lamp to blink (5 Hz).
- The horn should be audible for at least two clock pulses.

### 16.4.3 The Train Security System

A new station is being built on the Madurodam-Lutjebroek route. To ensure a good flow of intercity trains, an extra track will be added to the platforms. so that the main track remains free for passing trains.

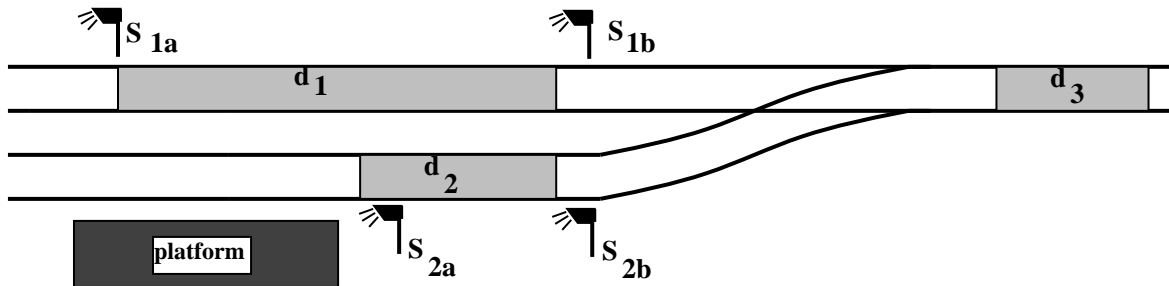


Figure 17: Situation train tracks

To prevent accidents, a signal system has been installed. Four signals have been placed ( $S_{1a}$ ,  $S_{1b}$ ,  $S_{2a}$ ,  $S_{2b}$ ) that are controlled via logical signals ( $S_1$  en  $S_2$ ). When a logical signal has the value '1' the corresponding signals  $S_{na}$  en  $S_{nb}$  are respectively yellow and red. In other cases, both signals are green. To control the signals, there are also 3 sensors: ( $d_1$ ,  $d_2$ ,  $d_3$ ) They give a logical '1' when the train is in the corresponding track.

You are given the assignment to design a circuit that ensures that the signals are controlled in such a way that no accidents can happen. The design should also be such that when at the same time both a train on the main and side track arrive, the train on the main track takes precedence.

Further it holds that:

- The braking path for a train is smaller than the distance between  $S_{na}$  and  $S_{nb}$ .
- It is not possible that a train activates 2 sensors at the same time.
- Trains go only from left to right on a track.

#### 16.4.4 The "Lazy" Lock Door

The assignment is to design the control for two doors that act as an anti wind door, just like the back door of the electrical building. The configuration of the 2 doors A and B has been drawn in Figure 18.

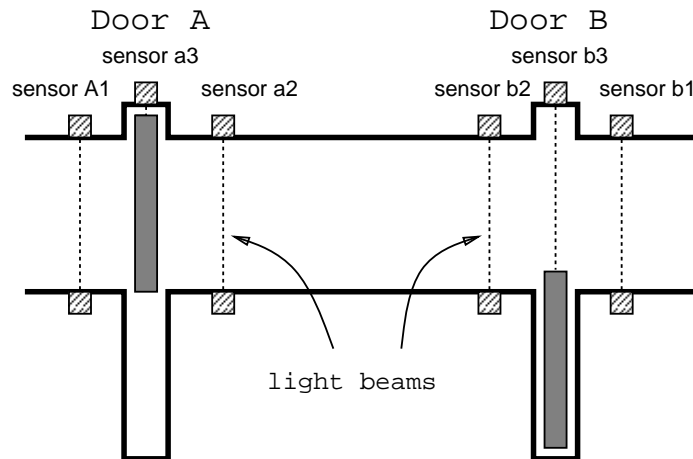


Figure 18: Situation lock door

At each door there are 3 light beam sensors mounted to detect persons A sensor delivers a logic '1' signal when the light beam becomes interrupted. The logic for the transmission must comply with the following rules:

- In every situation, there at least one door closed.
- The door can not close as sensor a3 (or b3) gives a signal. This is to prevent nasty accidents.
- Individuals must be able to go through the door from left to right and from right to left.
- To reduce wear on the door mechanism, the circuit is "lazy", that is, it does not return to the state with 2 doors closed.
- When at the same time at door A and at door B one person arrives, the person at door A receives priority.
- In case of reset or after switch on the power, both doors should be closed

### 16.4.5 The Person Detector

At many events it is desirable to know afterwards how many people have been present. In order to measure this, a fairly narrow (60 cm) corridor has been made through which only one person can pass at the same time. In this corridor two light locks (d1, d2) are present shortly after each other, each consisting of a combination of two infrared transmitters and receivers (a, b). If both the a and b transmitter-receiver combination are interrupted, a high signal is issued by the sensor in question. Signal d1 is thus high if both d1a and d1b are interrupted.

The direction of travel is from left to right. The output is a signal that causes a counter to be incremented.

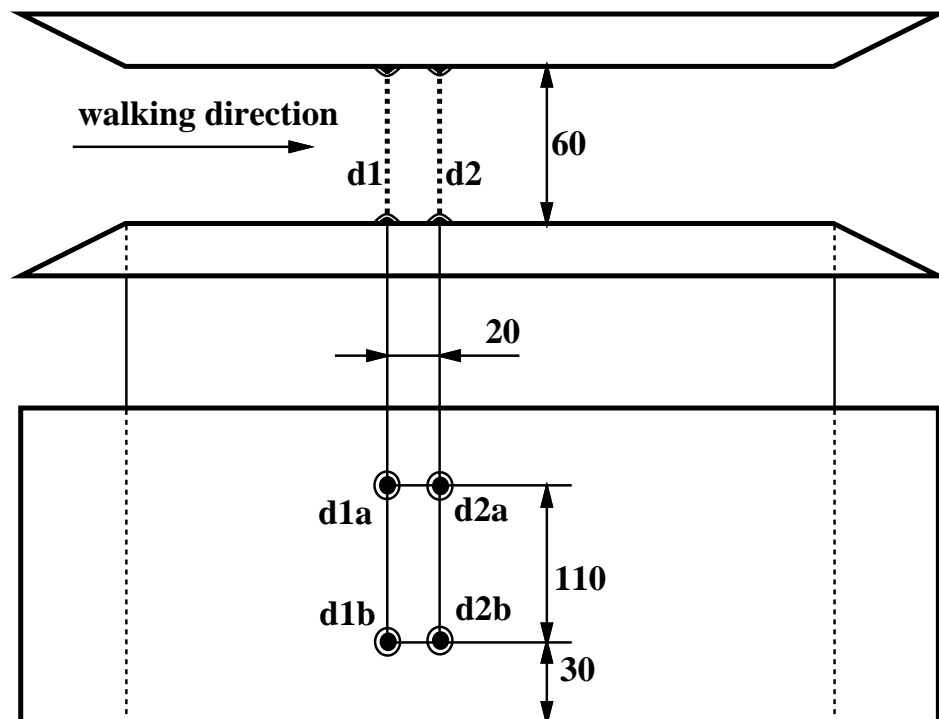


Figure 19: Situation person detector

You are given the assignment to make a control for the person detector that meets the above requirements.

Other conditions are:

- A person who walks in the opposite directions should not be detected.
- It is not possible to connect the count signal to the clock input of the counter.

Optional: Create a circuit that also detects people coming from the right and then delivers an count-down signal.

### 16.4.6 The Garage Door

In order not to get out of his car every time to open and close the garage door, Mr. B. Modaal wants to install a garage door that automatically opens his garage door when he arrives and closes when he leaves.

In addition to the door, an infrared (IR) sensor (d2) is present that delivers a "high" signal when it detects the signal emitted by a transmitter in the car. There is also a metal detector loop (d3) in the garage floor with which it can be detected whether there is a car in the garage. Finally, at the door next to the house, there is another push button (d1) that allows the garage door to be manually operated.

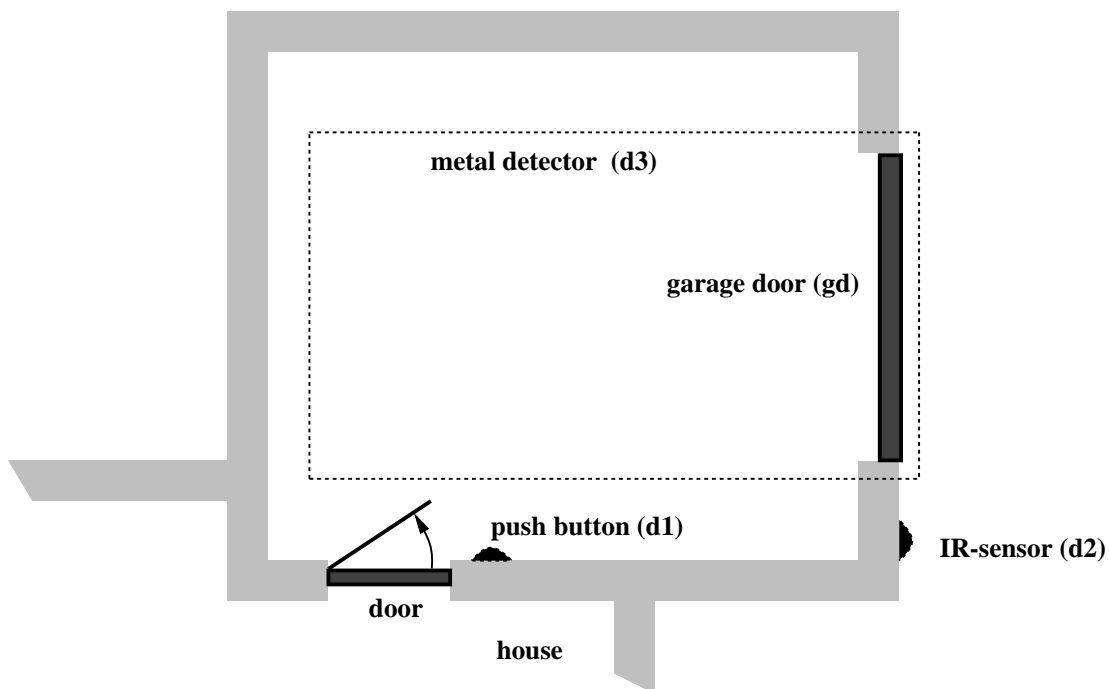


Figure 20: Situation garage

You are instructed to design a controller that allows the garage door to be operated with the sensors present.

Other terms and conditions:

- When the push button d1 is pressed, the garage door should always open or close the garage door.
- If the IR sensor d2 is activated, the garage door must open or close only if the garage is empty (d3 is low).
- When signal gd to control the garage door is high, the garage door will open or stay open, when it is low, the garage door will close or stay closed.



### 16.4.7 The Soda Dispenser

Design a control for the handling of the money deposited in a soft drinks dispenser.

The cash dispenser can handle coins of 20 cents and euros, but can not return change. The price of a soda can is 1,20 euro. A sketch of the transport system is given in figure 21.

The coins are sorted on their size after being thrown in. The coins of 20 cents go in the first channel and are detected by sensor S1. The euros end up in the second channel and are detected by sensor S2. After detection, the sensors give a high signal for at least one (typically more than one) clock pulse.

If 1 coin of 20 cents and 1 euro has been deposited (or vice verse), there must be one pulse on a relay R2 to release a can of soda. This relay signal needs to be high exactly 1 clock cycle. Pushing the reset button causes that the money deposited so far, will be returned. This is achieved by a high signal on a relay R1. Also, relay R1 must be activated if 2 coins of 20 cents or 2 euros have been thrown in. That is, in case of an incorrect payment, all the money deposited so far is returned.

In Figure 21 the relays R1 and R2, that are respectively used to returning the money and releasing the can, are not shown.

**Option:** Consider an extension of the system, which also allows the deposit of 6 coins of 20 cents.

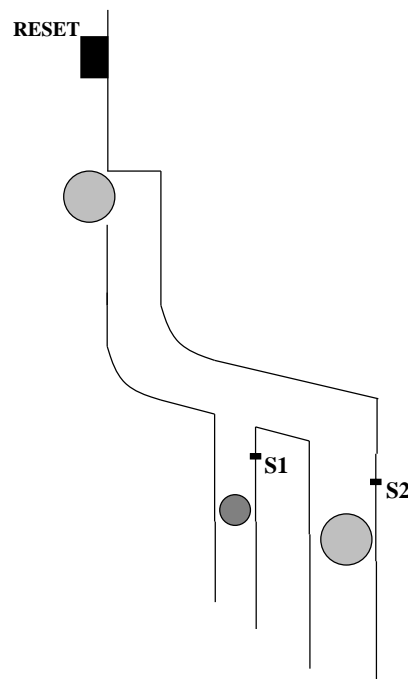


Figure 21: Overview coin system soda dispenser

### 16.4.8 The Telephone Circuit

A student who is not so far studying has the following problem: He, together with a roommate, shares the telephone connection. So far, the phones were connected in parallel, but they find it annoying that they are talking to each other when they both pick up the phone at the same time. When this happens, the caller does not understand anything then.

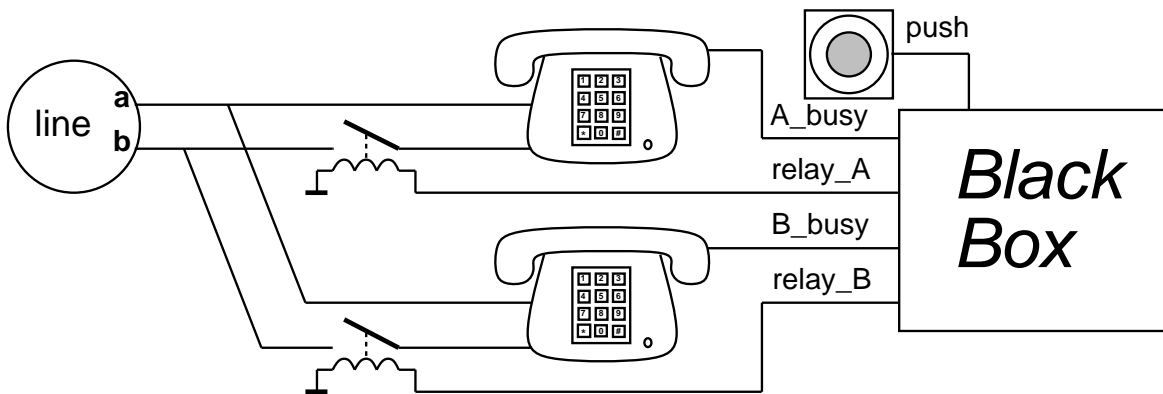


Figure 22: Situation telephone circuit

The students have thought that they can solve the problem with the above circuit. The phones are now both via a relay connected to the telephone line. In addition, device A has a push button that can be used to forward the connection. You are requested to fill in the 'black box' with a circuit that must meet the following requirements:

- When called, both phones must ring.
- When one of both persons picks up the phone, the other phone becomes disconnected from the line.
- When both persons pick up the phone at the same time, phone A is given priority.
- When the pushbutton is pressed, both devices are connected to the telephone line. In this way, the students can connect the caller to the other phone.

The input of the circuit consists of three signals. The signals A\_busy and B\_busy indicate a logical '1' as the horn of the phone in question is not on the hook. The signal "push" gives a logic '1' as the pushbutton is pressed. The outputs relay\_A and relay\_B ensure that the devices are connected to the telephone line. In addition, there is a clock signal that is not shown. The circuit does not have a separate reset signal.

## **Part III**

# **Project Design a Chip**

## **17 Introduction**

During this project the student learns to design in a group. It is a follow-up to the projects from the first year of the Bachelor of Electrical Engineering, but more emphasis is placed on structured hierarchical design and on working together in a group. The students are divided into groups of 7-10 people and design a chip in a Sea-of-Gates technology. The chip will be made in the second semester at the Else Cooi Lab, after which the students can test the chip at the end of the second semester.

During this project, a lot of knowledge is used from the first-year courses of digital systems A and B. Furthermore, knowledge and skills are also used from the first year courses of linear circuits, semiconductors and amplifier circuits, and the first year projects. Finally, knowledge is also obtained from the course integrated circuits, that is given in the first quarter of the second year. Especially the practical part of this course, which is described earlier in this manual d (Part II), is an important preparation for this project.

The following chapters in Part II are important for this project. Chapter 13 contains a tutorial for using the IC design environment on the basis of an example where a simple control (hotel switch) is designed. Chapter 14 provides guidelines for writing VHDL code. In chapter 15 some important restrictions related to the use of the software, are discussed.

In addition, this part of the manual contains the following information. In chapter 18 the objectives of the project are formulated. Chapter 19 describes a number of aspects of working in a team. Chapter 20 gives an introduction to the design process like this during This project will be used. In chapter 21 are guidelines for designing large digital IC circuits given. Chapter 22 gives important information about how the group assignment further needs to be elaborated. Chapter 23 finally gives ideas for possible group assignments.

## 18 Project Learning Goals

This project contributes to the fact that the student will experience and understand a design process within a group. The emphasis is on working towards one jointly agreed outcome, while at the same time each individual provides his/her own original contribution. In the project, both design methods, electrical engineering skills, as well as social aspects play a role. The learning goals are listed below. They will partly have a "learning of" character, partly a "getting familiar with" character.

- *Create a design using a global product specification, including boundary conditions.*  
It is important to specify in advance as clearly as possible to which the product to be designed must comply. Furthermore, during every project that the engineer will do later on, boundary conditions will play a role. These may include boundary conditions related to the specification, as well as boundary conditions regarding available time and money. In the project, the student will have to deal with both aspects. He/she must properly specify the (sub) assignment and realize it within a tight schedule.
- *Design in a systematic and hierarchical way.*  
The student learns to design in a structured way. This becomes evident during hierarchical design, whereby the design is recursively divided into subsystems.
- *Use analysis as a tool for synthesis during the design process.*  
During the project, the student will have to think design-oriented. It will not be his/her task to analyze an existing system, but to design a new system or possibly adjust an existing system. With this, analysis of the design, on different levels in the hierarchy, is an important tool for synthesis. In the project this means that the student will have to analyze the design at different levels by means of simulation.
- *Be able to analyse a design at different levels of abstraction.*  
In the project there are simulation programs available for different levels of abstraction: logical level, switch level and transistor level. Furthermore, a FPGA board is available for prototyping. The student will have to understand which abstraction levels for simulation/verification are useful at certain stage during the design.
- *Use different types of models during design.*  
The student must understand the importance of modeling: that not every detail can be taken into account at every abstraction level. A model must be as simple as possible, but also not so simple that the effects you want to describe are not visible.
- *Take into account testability.*  
During the design, the student must take into account that his designed product must be tested later on to check for proper operation and that, if the product does not work properly, it should be possible to obtain information what exactly is not working properly. In other words, the student must learn to handle test procedures and must be able to generate and execute a test plan. In the project, the students are eventually confronted with the result of their own design after the chip has been manufactured.
- *Perform the above design process within a group.*  
In his/her later professional career, the engineer will usually work on a project in a team. This

means that the engineer must be able to collaborate and communicate with others. This does not only require technical knowledge, but also social skills. That means skills to deal with people in a group to perform a task together with these people. Important items include: listening to each other, expressing own ideas and thoughts, stimulating the other members and not depressing, accepting leadership, giving guidance, etc.

- *Make a design as a part of a group design.*

Projects will often be too big to be handled by one or two persons. The engineer must be able to divide such a bigger project into subtasks and to specify the requirements for the subtasks. Then, he/she must be able to finish a subtask within the agreed requirements. The "Design a Chip" project assignment is also of a size that the students have to subdivide the assignment into smaller subtasks, and to make clear and tight requirements for each subtasks. When working on a subtask, the student will be confronted with the consequences when he/she does not comply to what has been agreed upon.

- *Use modern computer tools during the design.*

The modern engineer has a wide range of computer aids (design tools) at his/her disposal to support the design process. The student will be get acquainted with different tools for the design process.

- *Make documentation about the design.*

The students will use documentation in the project on, among other things, design tools and library cells. But they will also provide documentation themselves, about their own design. This latter documentation will be used for example when testing the designed circuits.

## 19 Working in a Team (in Dutch)

### 19.1 Inleiding

Een van de belangrijkste kenmerken van het project is, dat er wordt gewerkt in groepen, d.w.z. dat de leden van de groep samen een bepaalde, gemeenschappelijke einddoelstelling moeten realiseren.

Werken in groepen betekent, dat de groepsleden niet alleen vak-inhoudelijk bezig zijn met het project, maar ook dat zij

- samen moeten kunnen werken (werken met anderen, teamwork) en
- samen moeten kunnen communiceren (o.a. discussiëren en vergaderen).

Beide zaken zijn aanmerkelijk minder gemakkelijk dan men op het eerste gezicht zou veronderstellen.

In een team kunnen werken en kunnen communiceren zijn vormen van sociale vaardigheid. Het zijn extra dimensies van projektonderwijs. Bij individuele studie is iedere student immers afzonderlijk bezig met de leerstof.

Een van de voordelen van projektonderwijs is, dat de student gedwongen wordt, naast het opdoen van zuiver technische kennis (het leren van het "vak"), zich ook bezig te houden met sociale vaardigheden.

Ook in zijn latere beroepspraktijk zal de ingenieur immers bijna nooit alleen (als enkeling) technisch handelen; ook dan zal hij met anderen moeten samenwerken en dus ook met anderen moeten communiceren.

Om te voorkomen dat de groepsleden geheel onvoorbereid aan hun taak zouden moeten beginnen, worden in dit hoofdstuk enkele facetten van de genoemde vaardigheden behandeld. Volledigheid is onmogelijk. Maar bestudering van het hier gegeven minimum aan theoretische achtergrond en toepassing van de daaruit af te leiden praktijk-adviezen is een zeer zinnige zaak.

### 19.2 De drie dimensies van het werken in een groep

In de inleiding werden "samen kunnen werken" en "samen kunnen praten" vormen van sociale vaardigheid genoemd. Wat is sociale vaardigheid? Dat is: bekwaamheid in sociaal handelen, anders gezegd: bekwaamheid in het omgaan met mensen in een groep ten behoeve van een met en door de leden van die groep te verrichten taak. Als een groep op weg is naar een doel, dan kunnen aan dit "op weg zijn" drie aspecten worden onderscheiden:

1. Tijdens de groepsbijeenkomst wisselen de groepsleden, om tot hun gemeenschappelijk doel te komen, verzamelde kennis en informatie uit; ze selecteren de informatie en brengen er een logische samenhang in aan; ze bespreken vaktechnische zaken en resultaten van literatuurstudie en onderzoek; ze rekenen en konstrueren, etc. Kortom: de groepsleden zijn inhoudelijk bezig met de uitvoering van een taak, met de verwezenlijking van de doelstelling. Men noemt dit de inhoudsdimensie van het groepsgedrag.

2. Aan het bespreken van en de discussie over de inhoudelijke elementen dient - terwille van de begrijpelijkheid, overzichtelijkheid en doelmatigheid - duidelijk vorm te worden gegeven. Er moet volgens bepaalde "spelregels" worden gepraat. Voor de discussie over een probleem moet tijdens de bijeenkomst een verantwoorde werkwijze worden uitgestippeld; de besprekingen dienen gestructureerd te verlopen; men moet methodisch naar oplossingen zoeken, zich beradend op criteria en randvoorwaarden waaraan de aangedragen oplossingen moeten voldoen; besluiten moeten weloverwogen worden genomen. Kortom: de groepsleden moeten tijdens hun besprekingen van de inhoudelijke elementen van het project een bepaalde aanpak volgen: ze moeten een procedurele vorm geven aan de inhoud. Men noemt dit de procedure-dimensie van het groepsgedrag.
3. Tijdens de groepsbijeenkomst ontstaat er een bepaalde sfeer: individuen blijven in een groep weliswaar individuen, maar gaan tegelijk ook gedeeltelijk op in het groepsgebeuren. De groepsleden reageren op elkaar, waardoor zij een ander gedragspatroon gaan vertonen dan het louter individuele. Groepsleden beïnvloeden elkaar bewust en onbewust; ze beïnvloeden elkaar positief en negatief, waardoor een klimaat van enthousiasme en coöperativiteit, maar anderzijds ook van vrijblijvendheid of soms zelfs van verveling, irritatie of gedeprimeerdheid kan ontstaan: er ontstaan waarde-oordelen over de personen in de groep en hun bijdragen; er groeit gemotiveerdheid, een wil tot samenwerking en inschikkelijkheid, of anderzijds een neiging tot wedijver en konflikt; er kan sprake zijn van subgroepvorming, met goede of slechte gevolgen voor het groepsgeheel. Kortom: tussen groepsleden groeien onderlinge relaties en gevoelens, vormen van interactie en groepsdynamiek. Men noemt dit de procesdimensie van het groepsgedrag.

Inhoud, procedure en proces zijn dus drie aspecten van het werken in groepen. Deze drie dimensies beïnvloeden elkaar voortdurend. Ze zijn wel te onderscheiden maar niet te scheiden.

Als over inhoudelijke elementen alleen maar chaotisch (ongestructureerd) of in een verziekt klimaat (onwerkbaar sfeer) gediscussieerd wordt, komt men niet ver; althans heel wat minder ver dan men zou kunnen komen, en ook met heel wat minder plezier in het werk. Inzicht in en beheersen van de genoemde drie dimensies zijn onontbeerlijk voor het bereiken van een goed eindresultaat in een goede werksfeer.

### 19.3 De vergadering

De projectvergadering is het moment waarop de groep daadwerkelijk samen aan het werk is. Hier laten mensen zien wat hun vorderingen zijn en hier wordt besloten hoe er verder wordt gegaan. Voor het begin van de vergadering is er een agenda opgesteld die door de voorzitter aan de groep bekend wordt gemaakt (schrijf deze agenda op, of lees de agenda door). Nu weet ieder wat hem/haar te wachten staat gedurende de vergadering. De voorzitter (die de agenda van tevoren opstelt) moet er voor zorgen dat deze een goede opbouw heeft:

- Belangrijke punten eerst: deze mogen nooit door tijdgebrek niet goed aan de orde komen.
- Zorg dat er samenhang is tussen de te behandelen punten (logische opbouw: Als je het een weet, kun je verder met het volgende).
- De agendapunten moeten zo worden geformuleerd dat deelnemers weten waar het precies over gaat.

Een agenda heeft gedeeltelijk een vaste opbouw:

- vaste punten
  - opening en mededelingen
  - wijzigingen in de agenda (uitsluitend als er zeer dringende redenen daarvoor zijn)
  - notulen/verslag + besluitenlijst van de vorige vergadering
  - binnengekomen en verzonden stukken
- variabele punten
  - de eigenlijke agendapunten (in weloverwogen volgorde en goed geformuleerd, zie boven)
- vaste punten
  - vaststelling datum, plaats, tijd en concept-agenda van de volgende vergadering
  - rondvraag
  - sluiting

## 19.4 Rollen en functies

**Rol van de groepsleden** In een vergadering vervult ieder lid van de groep een bepaalde functie. Wanneer je dus bijvoorbeeld geen voorzitter of notulist bent, ben je toch mede verantwoordelijk voor het goede verloop van de groepsbijeenkomst. Dat betekent in ieder geval een goede voorbereiding van wat er gaat gebeuren (presentatie van het werk dat je gedaan hebt of je gedachten bepalen over een bepaald onderwerp). De volgende personen hebben een bijzondere functie in de vergadering:

**De voorzitter** De voorzitter is verantwoordelijk voor een efficiënt verloop van de vergadering. Dat betekent dat er voor hem/haar relatief veel tijd in de voorbereiding gestoken moet worden. Aan de hand van de notulen van de vorige vergadering en m.b.v. het tijd-werkschema kan een agenda opgesteld worden. Van tevoren moet de voorzitter in gedachten bepalen wat het uiteindelijke resultaat moet zijn van ieder te behandelen punt: Een besluit, op de hoogte brengen van andere groepsleden of bijvoorbeeld met elkaar van gedachten wisselen om een mening te vormen over een bepaald onderwerp. Wanneer je je dat niet van tevoren realiseert weet gedurende de vergadering misschien helemaal niemand meer waar de groep nu precies mee bezig is.

Het is zinvol om van tevoren te schatten hoeveel tijd een bepaald onderwerp in beslag gaat nemen. Hierdoor voorkom je dat je een te lange agenda maakt voor de beschikbare tijd waardoor er misschien bepaalde beslissingen te overhaast worden genomen. Belangrijke beslispunten dienen vooraf op schrift geformuleerd te worden. Tijdens de vergadering is de voorzitter formeel de leider. Dat betekent dat hij/zij bepaalt wie er aan het woord is en in mag grijpen wanneer er zaken ter sprake komen die volgens hem/haar niets zinnigs bijdragen aan het doel van het agendapunt. De hele groep moet echter meewerken en stil zijn wanneer de voorzitter aan iemand anders het woord heeft gegeven. Dat lijkt kinderachtig maar komt het ordelijk (snel en duidelijk) vergaderen ten goede. Het ligt voor de hand dat er een goede samenwerking moet zijn tussen de groep en de voorzitter. Wanneer de voorzitter zich te autoritair opstelt zullen de mensen zich of gepasseerd voelen of ongeïnteresseerd raken door



de vervelende sfeer die kan ontstaan. Wanneer de voorzitter echter "te vriendelijk" is, zal de vaart snel uit de vergadering zijn. Er zit geen structuur meer in, omdat niemand die er bewust inbrengt. De voorzitter moet dus wel duidelijk aanwezig zijn. Het ideaal lijkt ook hier weer in het midden te liggen. De zgn. democratische stijl van leidinggeven zorgt ervoor dat de voorzitter voortdurend oplet wat er in de groep leeft en van daaruit ook handelt. De democratisch leider ziet de groep als iets waar voorzichtig mee moet worden omgesprongen maar waar wel duidelijk leiding aan moet worden gegeven. Respekt voor elkaar (goed luisteren!) is een van de belangrijkste aspecten voor het slagen van de groepssamenwerking. De voorzitter moet de aandacht van de groepsleden vragen bij het te behandelen onderwerp. Dat kan hij/zij bijvoorbeeld doen door regelmatig samen te vatten wat er besproken is. Het maken van aantekeningen kan daarbij een grote hulp zijn. Soms is het van belang om zaken op het bord of op papier te zetten. Hierdoor wordt de aandacht van de groep "centraal" gehouden. In feite verloopt het oplossen van een probleem, met bijbehorende besluitvorming, in een vergadering net als in het hele project (zie paragraaf 19.6):

- Fase 1 Beeldvorming: Doelstellingen, randvoorwaarden, uitgangspunten, probleemstellingen, etc. Hierin bepaal je: "Waar hebben we het precies over en wat moet ons resultaat zijn".
- Fase 2 Het genereren van oplossingen: Dit is een wat brainstormachtig gebeuren waarbij alle (voor de groep) denkbare oplossingen op tafel moeten komen. Schrijf alles op!
- Fase 3 Concentreer fase: Onder leiding van de voorzitter worden de criteria geformuleerd waarmee de oplossingen beoordeeld gaan worden. In grote lijnen vallen er nu al de nodige oplossingen af. Al pratend over de oplossingen zullen er meer criteria komen en tenslotte komt men bij de besluitvorming.
- Fase 4 Besluitvorming: De voorzitter moet hier zorgvuldig te werk gaan. Spreek duidelijk af waarover precies besloten gaat worden en zorg dat niemand "vergeten" wordt. Iedereen moet de kans hebben zijn/haar voorstel te verdedigen om te voorkomen dat men er later op terug wil komen.
- Fase 5 Besluit-uitvoering: Hierin moet afgesproken worden wie wat gaat doen. Probeer het werk eerlijk te verdelen. Vriendjes gaan graag samen in een subgroep waardoor er wel eens mensen uit de boot vallen en met een minder boeiend onderwerp worden opgescheept. De voorzitter moet in de volgende vergadering de uitvoering van het besluit controleren.

**De notulist** Van iedere vergadering moet een verslag gemaakt worden. Dat is van belang om onduidelijkheid over de genomen besluiten te voorkomen en controle uit te kunnen oefenen. Zorg er dus ook voor dat de notulen bewaard worden en voor iedereen te vinden zijn (b.v. ruim voor de volgende vergadering in het postvak). De volgende zaken moeten in de notulen vermeld staan: Of men kiest voor uitgebreide notulen of voor een beknopt verslag, altijd zullen de volgende zaken vermeld moeten worden:

- datum en tijdsduur van de vergadering
- aanwezig/afwezig (namen van de aanwezigen opsommen - functie van voorzitter en verslaglegger apart bij de desbetreffende namen vermelden -, bij de afwezigen vermelden of niet met of zonder bericht van verhindering is).

- weergave van de behandeling van alle agendapunten (volgorde van de agenda aanhouden). altijd vermelden:
  - wat is er besloten?
  - werkafspraken (wie doet wat voor wanneer?)
- besluitenlijst (deze is tweeledig):
  - opsomming van alle genomen besluiten en gemaakte afspraken.
  - opsomming van besluiten en werkafspraken uit vroegere vergaderingen, die om welke reden dan ook nog niet zijn afgewerkt.

NB.: In de notulen/het verslag dient objectief te worden vermeld, wat er gezegd/gebeurd is. Geen eigen meningen of kommentaar van de verslaglegger. Ook de eigen deelneming aan de discussie behoort door de verslaglegger objectief te worden weergegeven.

Wanneer de groep gekozen heeft voor een roulerend voorzitterschap (een andere voorzitter per vergadering), kan het nuttig zijn om de notulist van periode "n" de voorzitter voor periode "n+1" te laten zijn. Deze weet dan precies wat er gaande is.

**De taken van de deelnemers** Het projectgroepswork, met daarin als belangrijke componenten de discussie en vergadering, is een gezamenlijk proces van alle groepsleden, van voorzitter en deelnemers.

Te vaak denken de deelnemers dat het wel en wee van een bijeenkomst uitsluitend in handen ligt van de voorzitter. Maar zelfs de beste voorzitter moet falen als de deelnemers niet meewerken.

De taak van de voorzitter is veelomvattend en niet gemakkelijk; van de deelnemers mag men vragen, dat zij de voorzitter de uitvoering van zijn taak mogelijk maken. Dat is hun voornaamste taak; alle andere vloeien eigenlijk daaruit voort. Indirekt zijn ze op de vorige bladzijden al ter sprake geweest. We zetten de voornaamste, voor zover zij de groepsvergadering betreffen, nu nog even op een rij.

- Verwacht mag worden, dat alle deelnemers met functies (verslaglegging, secretariaat, archivering) loyaal met de voorzitter meewerken en in goed overleg steeds tijdig hun plichten vervullen.
- Alle deelnemers behoren zich voor te bereiden op de bijeenkomst. Dat begint al met "kleinigheden" als op tijd aanwezig zijn. In feite is het opzettelijk of nonchalant te laat komen een belediging jegens de andere deelnemers en de voorzitter, die wel de moeite hebben genomen om op tijd te komen; "het vergeten" van een vergadering getuigt evenmin van een juiste instelling, nog afgezien van het feit dat zo'n vergadering daardoor kan inboeten aan doelmatigheid. Er ontstaat vaak extra werk in een later stadium. De echte voorbereiding omvat natuurlijk ook het zich bezinnen op de agendapunten: het tevoren bestuderen van documentatie-materiaal; het prepareren van bijdragen, niet alleen qua inhoud, maar ook qua presentatie en formulering, het verzamelen van (eventueel visueel) bewijsmateriaal bij argumenten, etc.
- Alle deelnemers die in een vorige vergadering taken opgedragen gekregen hebben, dienen deze voor de volgende nauwgezet te volbrengen.

- Gedurende de vergadering zelf mag niet alleen van de voorzitter, maar ook van de deelnemers verwacht worden dat zij aantekeningen maken; daardoor kunnen veel overbodige herhalingen en onnodige afdwalingen voorkomen worden.
- Deelnemers moeten zich tijdens de vergadering onthouden van gedrag dat een bijeenkomstodeloos lang doet duren, het goede klimaat verstoort en irritatie wekt bij de andere aanwezigen. Zij moeten daarom o.a.:
  - goed naar elkaar luisteren
  - elkaar laten uitpraten en niet steeds in de rede vallen
  - meewerken aan het binnen de vastgestelde tijdgrenzen blijven (niet het woord nemen als er niets is aan te vullen op wat anderen al gezegd hebben; mensen die omslachtig vorige sprekers herhalen en er vervolgens aan toevoegen dat ze het er helemaal mee eens zijn, zijn beruchte tijdvreter).
  - bij onverhoopt uitlopen van de vergadering niet er vandoor gaan of met tekenen van ongeduld en bedekte verwensingen "de tijd uitzitten" (dit geldt ook voor de rondvraag).
  - loyaal meewerken (bij democratisch genomen besluiten) aan de besluitvoering, ook al was men aanvankelijk tegen.
- Ieder groepslid heeft tot taak, gedurende de vergadering de voorzitter te steunen door het vervullen van informele leiders-taken, voorzover dat in zijn vermogen ligt en het gewenst is.

## 19.5 Praktische tips voor de organisatie van de projectgroep

- Duidelijkheid is een eerste vereiste voor een goede organisatie.
- Met heldere afspraken kan een belangrijk deel van de duidelijkheid bereikt worden.
- Wanneer dan iedereen zich aan deze afspraken houdt, is reeds veel gewonnen.
- Voortgangskontrolle kan dit bevorderen.
- In het Projektonderwijs hangt veel af van de vergaderingen van de projektgroep.
- Een goede voorbereiding op deze vergaderingen is van veel belang voor het slagen ervan.
- De voorzitter moet deze vergadering primair voorbereiden:
  - agenda samenstellen
  - problemen formuleren
  - stukken tevoren rondzenden.
- Er moeten harde afspraken gemaakt worden: wie doet wanneer wat.
- In de notulen dienen deze afspraken te worden vastgelegd. Uitgebreide notulen zijn meestal niet nodig, zelfs ongewenst want dan worden ze niet meer gelezen. Leg geen hele discussies vast! Hooguit de konklusies.

- Loop bij de start van de volgende vergadering deze afspraken na bij wijze van voortgangskontrolé: heeft iedereen zijn taak gedaan? Kan iedereen voldoende vooruit? Zullen de gegevens op tijd aanwezig zijn?
- Probeér taken zo vroeg mogelijk te verdelen zodat men zich daar vast op kan voorbereiden.
- Hanteer, zowel bij het maken van afspraken als bij de voortgangskontrolé, het tijdschema als uitgangspunt.

## 19.6 De procedure voor probleemoplossing en besluitvorming

Voor een correcte probleemoplossing en besluitvorming geldt de volgende procedure in vijf fasen:

1. In de eerste fase, die van de beeldvorming of probleemstelling zorgt de voorzitter ervoor, dat allen een helder beeld krijgen van wat het probleem precies inhoudt. Dit is nl. lang niet altijd voor alle deelnemers onmiddellijk duidelijk. Hierbij kunnen de volgende vragen de revue passeren:
  - hoe en in welke context is het probleem ontstaan? (oorzaken)
  - voor wie is het een probleem?
  - is het een eenmalig of structureel probleem?
  - is alle informatie beschikbaar? Welke gegevens ontbreken nog?
  - is het probleem in eerste instantie correct geformuleerd? Wat ontbreekt?

De beantwoording van deze vragen brengt vaak nieuwe facetten aan het licht of vereenvoudigt het aanvankelijk verwarde beeld. Aan het einde van de beeldvormingsfase blijkt vaak, als resultaat van de discussie, herformulering van het probleem noodzakelijk; in ieder geval moeten alle deelnemers nu een volledig en identiek beeld hebben van het probleem: onvolledige beeldvorming kan zich wreken tijdens de volgende fasen. Het kan nuttig zijn, het probleem in zijn definitieve formulering op het bord te schrijven.

2. In de tweede fase (creatieve fase of brainstorming) leidt de voorzitter het zoeken van de groep naar mogelijke oplossingen. Hij benadrukt dat de leden zich niet hoeven te beperken tot pasklare oplossingen; iedere suggestie is welkom. Wellicht kunnen ze in een latere fase gecombineerd worden tot een werkelijke oplossing. Creatief en associatief denken is tijdens de brainstorming zeer welkom. De brainstorming kan frappante resultaten opleveren op voorwaarde dat de voorzitter niet toestaat, dat de deelnemers onmiddellijk commentaar geven op voorstellen van andere deelnemers. Dit is noodzakelijk. Een brainstorming is gedoemd te mislukken als deelnemers zich geremd voelen in hun creatief denkproces door mogelijk schampere reacties. De kritische afweging komt pas in fase 3; de creatieve fase is bedoeld voor het vrij genereren van ideeën, en het elkaar enthousiasmeren in een alles-is-mogelijk- niets-is-gek-klimaat. De voorzitter noteert voor allen duidelijk zichtbaar op het bord alle suggesties en ideeën die geopperd worden. Als niemand meer iets aan de inventarisatie heeft toe te voegen, gaat hij over naar de derde fase.
3. In de derde fase (concentratie fase) begint de voorzitter met het samen met de groep formuleren van de criteria waaraan oplossingen moeten voldoen. Dit zou ook aan het einde van de eerste

fase kunnen gebeuren. De kans is dan echter groot dat de groepsleden door hun vroegtijdige kennis van deze criteria zich tijdens de creatieve fase geremd voelen en niet durven komen met "gewaagde" ideeën. Vervolgens worden de suggesties aan de criteria getoetst en waar mogelijk deeloplossingen met elkaar gekombineerd. Eerst laat men nu die voorstellen vallen, die het minst aan de criteria voldoen. Al discussiërend, combinerend en schrappend komt men tot een steeds strengere selectie, tot er waarschijnlijk enkele haalbare oplossingen overblijven. Is geen enkele oplossing bruikbaar, dan zal men de criteria ruimer moeten stellen en het laatste deel van de procedure moeten herhalen. De voorzitter ziet erop toe dat ook tijdens deze fase alle groepsleden in de gelegenheid zijn hun visie te geven of hun bijdrage te leveren. Dat betekent niet dat iedereen iets moet zeggen, maar wel dat iedereen iets moet kunnen zeggen. Ook ziet hij toe op logische redeneerwijze en argumentatie: voor- en tegenstanders van bepaalde oplossingen willen nog wel eens met louter emotionele middelen hun standpunt verdedigen of het gelijk aan hun zijde krijgen.

4. In fase vier (besluitvorming) wordt de definitieve oplossing gekozen (het besluit genomen). Zonodig herhaalt de voorzitter de overgebleven oplossingen, schrijft ze op het bord met daarnaast de criteria en konsekventies. Alle pro's en kontra's van de oplossingen worden tegen elkaar afgewogen (eventueel ook op het bord schrijven, ze zijn dan beter met elkaar te vergelijken). Tenslotte valt het besluit. Het prettigst is het, als het besluit unaniem genomen wordt; dan is iedereen voor dezelfde oplossing en zal zich ook geheel voor de uitvoering van het besluit inzetten. Is unanimité niet haalbaar, dan is consensus het meest aangewezen principe. Dat betekent dat niemand zich tegen de keuze van een bepaalde oplossing verzet. Zijn er wel tegenstanders, dan wordt het besluit genomen bij meerderheid van stemmen.
5. Is het besluit eenmaal genomen, dan dient de groep zich nog te bezinnen op de uitvoering ervan (vijfde fase: besluit-uitvoering). Er dienen afspraken gemaakt en regelingen getroffen te worden (wie doet wat voor wanneer?). De voorzitter ziet erop toe dat deze afspraken en regelingen in de notulen en het verslag worden opgenomen.

De loyaliteit eist dat groepsleden die aanvankelijk tegen het genomen besluit waren, zich achter het besluit stellen (mits het uiteraard op correcte wijze genomen is) en meewerken aan de besluit-uitvoering. Wellicht is het voor de voorzitter mogelijk, bij de taakverdeling rekening te houden met de aanvankelijke standpuntbepaling van die groepsleden.

## 19.7 Evaluatie

De evaluatie beoogt beschrijving en waardering van de wijze waarop gewerkt wordt en, daaruit volgende, - voorzover nodig en mogelijk - verbetering van de activiteiten en het gedrag.

Evalueren is eigenlijk een vorm van feedback geven. We zouden kunnen zeggen dat bij de evaluatie feedback wordt gegeven over alle aspecten van het groepswork: alle activiteiten worden gewaardeerd, en wel aan de hand van criteria (normen), die impliciet of expliciet worden ontleend aan de doelstellingen van de groep.

Voorwaarde voor een geslaagde evaluatie is, dat er een open sfeer heerst, waarin ieder groepslid alles ter sprake moet kunnen brengen wat hem op het hart ligt, zonder dat hem dat wordt kwalijk genomen, dus zonder dat andere groepsleden zich persoonlijk gegriefd voelen. Men moet zijn opmerkingen

echter wel kunnen beargumenteren, d.w.z. kunnen zeggen wat men heeft waargenomen (de voor allen zichtbare, objectieve feiten) en hoe men het waargenomene heeft geïnterpreteerd. Waarneming en interpretatie moeten dus steeds duidelijk onderscheiden worden. Tijdens de bespreking blijkt dan wel of de interpretatie van de een overeenkomt met wat de ander bedoelde

## 20 The Design Process

### 20.1 Introduction

From the moment someone gets an idea until the moment the product becomes available in a shop, many design steps need to be done. This design trajectory is very similar for many products. Such a generally valid trajectory is depicted in Figure 23.

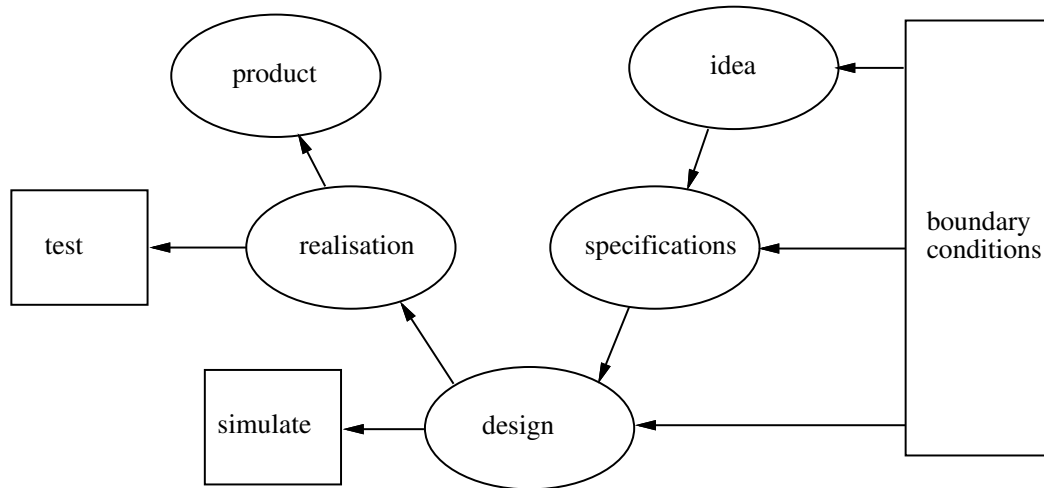


Figure 23: General design trajectory

The same applies to the design of an Application Specific Integrated Circuit (ASIC). After the creation of an idea, specifications must be drawn up that the circuit must meet, including boundary conditions that apply. Then the circuit is designed. First at logic level and circuit level, and finally the layout level. From the layout, the chip is fabricated. Next the chip is tested and, if all is good, the chip is taken in mass production. With this last step we came to the final product. In this project we will go through the entire range from specification up to the test. However, the emphasis will be on designing an IC, thus obtaining a circuit and layout from the specifications and the boundary conditions.

### 20.2 Specification

If there is an idea for a product and one wishes to proceed with the design of the product, then the first step in the design process is to put things in line to which the product must comply, either the preparation of a program of requirements (PoR). These requirements can be subdivided as follows:

#### Functional requirements

These are the requirements related to the product's function: What should the product do? What are the main features? What are the secondary functions?

#### Boundary conditions

These are more the demands that are made due to all kinds of limitations that have to be taken into account. For example, restrictions that result from the environment in which the product is

used (it must not exceed 1x1x1 cm, it must be able to work on a battery) and limitations due to the design and production process (the design must be ready within 5 months, the product may not cost more than 2 euros).

For example, let's take the Avant-Garde clock design (see section 23), which uses a DCF radio signal to adjust itself to the correct time. The Avant-Garde clock should display the hours and minutes on a digital display, the seconds on an analog display, and play the Big Ben melody every 15 minutes (See also the group assignment in section 23.2). For this design we could compile the following program of requirements.

### **PoR - Avant-Garde clock**

#### **Functional requirements**

Main function: The product must show the right time in hours and minutes on a digital display, in seconds on an analog display, and it should use as a time reference the 77.5 kHz DCF radio signal.

Side function: The Big Ben melody has to be played every 15 minutes.

#### **Boundary conditions**

- The electronic control of the product must be realized on an area of  $0.4 \text{ cm}^2$  of a CMOS Sea-of-Gates chip  $1.6 \mu\text{m}$  technology.
- The above-mentioned circuit must work at a clock frequency of 6.144 MHz.
- The design must be ready in 4 months.

## **20.3 Function Block Diagram**

Based on the PoR we can start the design and first we will look at the various sub-functions we need to realize. We can then come to a function block diagram as shown in Figure 24. In the function block diagram, each block represents an action and each arrow represents a start, intermediate or end result.

The use of a function block diagram as an initial step during the design, has the advantage that the system is described at a high level and that any alternatives for the implementation at the structural level, as described in the following section, can be better evaluated.

## **20.4 Structural System Description**

Based on the function block diagram, we can now choose how to implement the different sub-functions in a system. In this project, the design is intended to be realized on a Sea-of-Gates chip, and most of the blocks from the function block scheme will therefore be mapped onto a Sea-of-Gates chip. We then come up with a structural description of the design as shown in Figure 25. In Figure 25, for the Avant-Garde clock, all parts except the receiver, the digitizer and peripherals such as loudspeakers and displays, are mapped onto the Sea-of-Gates chip. Additionally, an extra subsystem will be added that is not directly found in the function block diagram but that is required for the implementation of several parts, namely a clock generator for different clock frequencies.



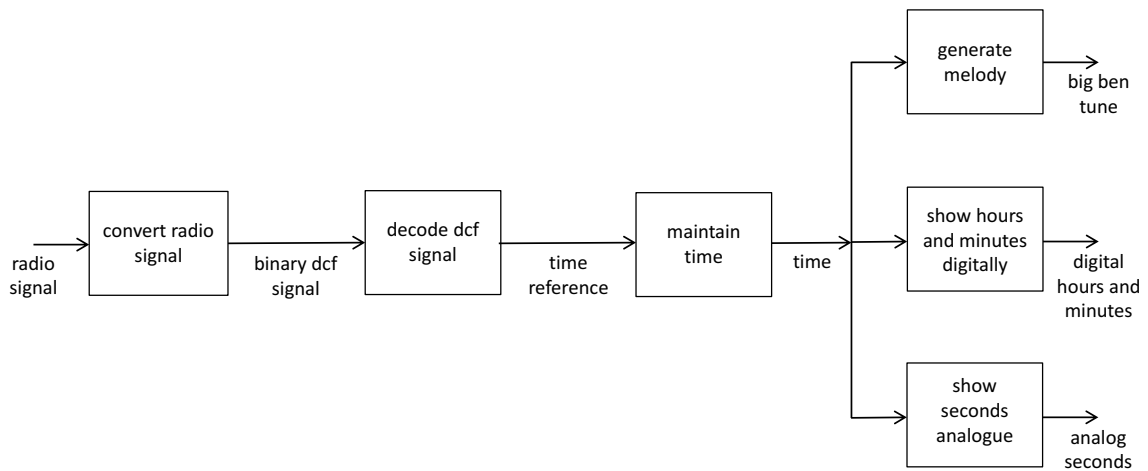


Figure 24: Function block diagram for the Avant-Garde clock

## 20.5 Hierarchical Decomposition

Usually, after the first subdivision, the subsystems are still too big to design each of them directly. A next step is then that subsystems are further subdivided into smaller subsystems. In figure 26 this is done for the subsystem "clock" from Figure 25.

This can be done recursively until the subsystems are small enough to be designed immediately, or until the subsystems corresponding to subsystems that have previously been designed (for example, a flipflop that has been designed and that is part of the Sea-of-Gates cell library). When we look at the VHDL description for the system, then each subsystem from the design will correspond to an entity in the VHDL description.

## 20.6 Abstraction Levels

Different abstraction levels may be used for each subsystem during the design of the system. For example, in IC design we may describe the subsystem at the logical level, the circuit level and/or the layout level. The abstraction level used depends on the design phase in which one is working and on the type of verification that needs to be done at that moment. For each abstraction level, there are several tools that can be used for design and verify at that level. For logical level verification, for example, a VHDL simulator such as ModelSim is used, for circuit-level verification a circuit simulator such as PSpice, and for Layout Level Verification a layout to circuit extractor in combination with a switch-level simulator.

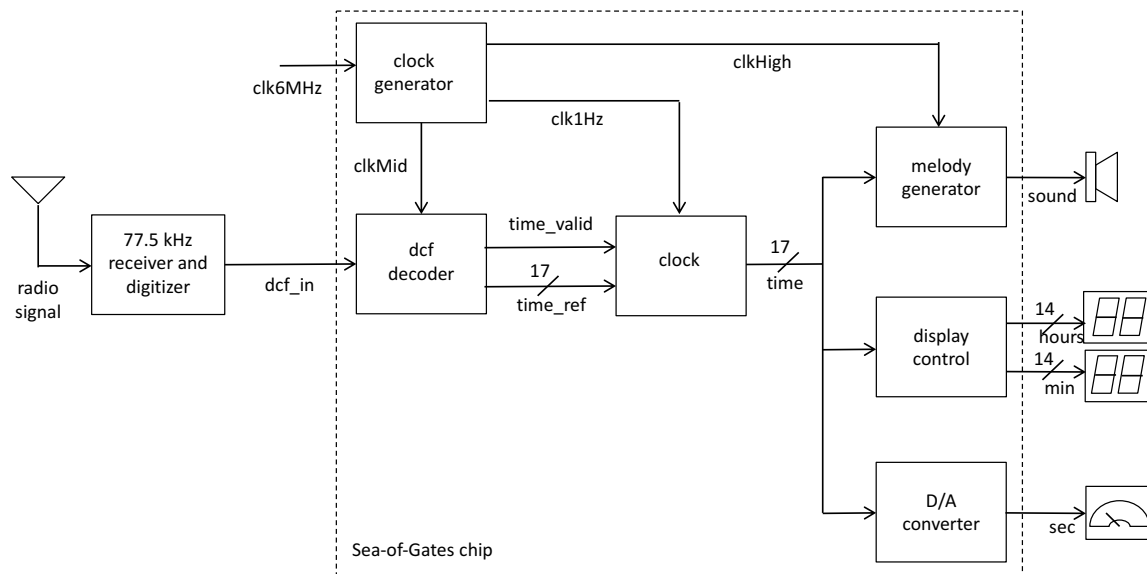


Figure 25: A structural description of the design for the Avant-Garde clock.

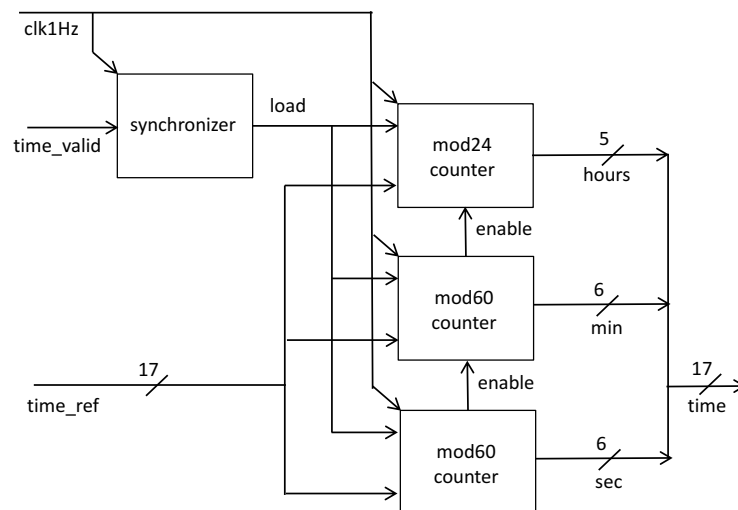


Figure 26: The subsystem clock subdivided into smaller subsystems

## 21 Guidelines for designing large circuits

When designing an IC, we have computers, software and programmable hardware (for rapid prototyping) at our disposal. These tools are essential because of the very large and still growing complexity of ICs. However, they are useless if they are not used with intelligence. In addition, the real design work remains the job of the engineer.

In order to streamline the design process, it is necessary to make some arrangements regarding to the design strategy and use of the tools. Therefore, in this chapter, some (design) rules will be given.

**Design hierarchically** Hierarchy is absolutely necessary to maintain a good overview of a complicated circuit. As a rule of thumb one can say that for a good overview no more than 5 to 10 sub-blocks should be defined at a hierarchical level. The software is also based on a hierarchical design. Tools like the placer and router have the greatest difficulty with large "flat" designed circuits.

An exception to this rule is the circuits made with "logical synthesis". For this, the argument of the "Overview" is not so important. The limitation is then with the software. As a rule of thumb, you must adhere to no more than 100 to 200 instances per hierarchical level.

**First, perform a proper verification of the entire circuit by means of simulation and rapid prototyping, before you start creating the layout** There are numerous analysis steps in the design process. One of the most important checks you will often do is simulation of the circuit. Another way is rapid prototyping, involving special hardware (an FPGA) that is programmed to behave according to the VHDL description of the design. Performs these types of verification before you begin to create the layout. Both methods complement each other and allow to notice and fix errors early in the design stage. The benefits are shorter iteration loops when errors are detected. Although thorough verification often seems time consuming, it turns out in practice that this time investment is well-rewarded.

**Keep in mind that you are describing hardware when creating VHDL descriptions** When making a VHDL description, always keep in mind that the description represents a digital circuit. Although it is beneficial to create high level VHDL descriptions (for example,  $c \leq a + b$  for the addition of 2 integer numbers) and translate this through the logic synthesizer to a network with logical ports and flip-flops, always keep in mind that the VHDL description will eventually be translated to a combinatorial and/or sequential circuit according to the Moore model. If the VHDL description made is not clear enough with respect to the behavior of the circuit, then the logic synthesizer will not be able to translate the VHDL behavioral description into a structural description, or the generated circuit will display unwanted/unpredictable behavior. More information on this can be found in Chapter 14.

**Design FSMs according to the Moore-model** Finite State Machines can be designed according to the Moore model or the Mealy model. The difference between these two design models is that with the Moore model the outputs are only dependent on the current state of the circuit, while with the Mealy model the output signals may also be dependent directly on the input signals. In practice, this means that "Moore FSMs" are less susceptible to interference and easier to design correctly. A disadvantage is a sometimes larger amount of states and consequently a larger impact on the silicon

surface. However, this disadvantage does not outweigh the benefits (certainly when you do not have much experience in chip design yet).

**Design synchronous circuits** Synchronous design means that all signals in a circuit after a time will be stopped and then resume at a certain time. This is necessary to keep an overview of the mix of signals and their different delay times. Activity is resumed only after it is sure that all signals have come to a rest. The (central) signal that indicates when the circuit is resumed, we call the clock. Another advantage of these synchronous or "clocked" systems is that always occurring spikes in the circuit, do not cause any problem. When the flip-flops read their new input value, this value will always be stable, provided that the clock frequency is chosen low enough.

**Only use edge-triggered flip-flops for data storage** Memory elements can read their new value during a clock edge or during the entire period that the clock signal is "high" (a pulse). The fault sensitivity of so-called edge-triggered flip-flops is much lower than that of the pulsed triggered variant. For this reason, only the edge-triggered flip-flops should be used during the project. The `oplib` (library) only contains flip-flops of this type.

**Be careful with clock signal** The clock signal is at the heart of the circuit. It is very important that at all flip-flops the clock flank arrives simultaneously, because otherwise things may get mixed up. In practice, however, this will not happen exactly. This phenomenon, also known as "clock-skew" can cause that your circuit does not work, while logically it is correct. Also simulators often do not notice clock skew. It is therefore important that you consider this phenomenon in the design process and that there is as little clock skew as possible. In practice, there are a couple of rules:

- Do not use logic in the clock signal (this may also give rise to spikes in your clock signals which is even a worse problem).
- Buffer your clock signal sufficiently and use the same number of buffers in each "branch" of your clock tree in order to have similar delay times at the ends of the tree. This means that between each flip-flop in your circuit and the original clock signal there is the same number of buffers.
- Attach a more or less equal load to each buffer.

**Do not use dynamic circuits** Dynamic circuits use the available capacity to temporarily store information. The best known example of a dynamic circuit is the dynamic RAM (Random Access Memory) or shortly DRAM, that is used in most computers. The advantage of dynamic above ordinary "static" circuits is that they occupy less silicon surface, which is important in large repeating structures (for example memories). Disadvantages are the requirement of a multi-phase clock and poor testability at low frequencies. Because of these disadvantages, we conclude during the project dynamic memory or logic cells should not be used.

**Only use positive logic** Signals can be defined positively and negatively. Examples of positive definitions are "light on" or "person detected". A widely used example of a negative definition is "non-reset" or *reset*. The latter is often done in terms of saving: it sometimes saves an inverter. The disadvantage of negative logic is that it is not intuitive. In your mind, an inversion has to be applied. For the project we conclude that we should preferably use positive logic.

**Use a buffer when an output has load that is too large** When an output is loaded, the "edge" of the outgoing signal becomes less steep. Because of this, your circuit will be slower, and also more power will be dissipated. In a clock line, you have the additional problem of an increasing clock skew. In order to keep these effects within limits, it's important that you use buffers when appropriate. To help you to decide when a buffer is needed, the library cells indicate the values of the input capacity and the fanout. If the load exceeds the fanout value, it is wise to place a buffer (e.g. buf40).

## 22 The Group Assignment

### 22.1 Introduction

During the group assignment, you will be making a complex design with the whole group. All aspects of working in a group and working on a large design will be covered. Background information for this has already been provided earlier in this manual. For project planning, the reader is referred to [6]. Version management is discussed in Appendix B.

In general, the approach for the group assignment can be divided into the following tasks:

- Selection of the assignment
- System specification
- System design
- Implementation of subsystems (modules)
- Putting everything together
- Finishing the design
- Documentation

These tasks are of course not all strictly separated. For example, creating the documentation can best be done during all previous steps.

### 22.2 Selection of the assignment

For the selection of an assignment, the reader is referred to Section 23.

### 22.3 System specification

During system specification it must be clearly defined which requirements the design must meet. This corresponds to the creation of a program of requirements such as described in Section 20.2. Below, the different tasks are summarized:

- Create a short description in words what the circuit should do.
- Determine boundary conditions from the assignment description.
- Add additional, external boundary conditions.

For general technological boundary conditions that apply to this project, see Section 22.4.

Let your system specification be evaluated by an assistants and by the tutor.

## 22.4 General technological boundary conditions

### 22.4.1 Available chip area

The total area on a Sea-of-Gates chip is subdivided into 4 bond\_bars (see Appendix D.2.22). The area available for each group consists of 2 bond\_bars below each other. This corresponds to a chip area of approximately  $0.4 \text{ cm}^2$ , or 40,000 transistor pairs. An impression of the size of a bond\_bar can be obtained by inspecting the layout of a bond\_bar instance in *seadali*.

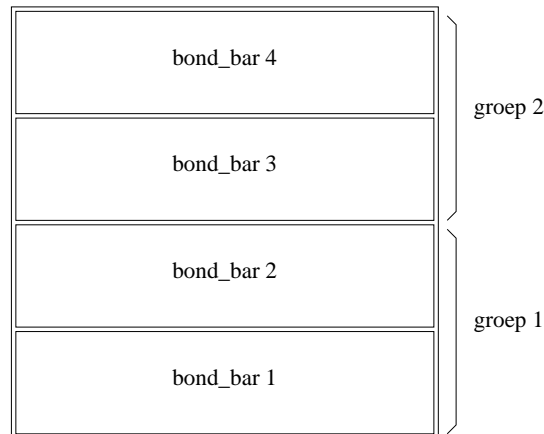


Figure 27: The chip area has been subdivided into 4 bond\_bars

Note that by far not all transistors can effectively be used for the design. The amount of components that can be used for a design will depend on a variety of factors, e.g. which components are used, whether long connections are required etc. As a rule thumb, one can assume that for a design that fits on the area of 2 bond\_bars **no more than 200-250 flip-flops can be used**.

A group can choose to put on the surface of 2 bond\_bars one large design, or it can choose to place a separate design on each bond\_bar (for example, on the first bond\_bar the complete circuit and on the second bond\_bar only parts of the circuit intended for testing). However, keep in mind that with each design only the connections (pins) of only one bond\_bar can be used. Per bond\_bar, 32 I/O connections are available. The power pins are not included here, they are connected separately. So when choosing to use the whole area of both bond\_bars for one large design, only the connections of one bond\_bar can be used.

When the option is chosen to create 2 separate designs on the available chip area (with 2 different bond\_bars connected), one design may use a part of the area of the other design.

### 22.4.2 Additional boundary conditions

- Only FSMs (Finite State Machine) of type Moore may be used. More information can be found in [4].
- When the circuit starts up, all FSMs should go to an initial known state by means of a reset signal.

- To generate a clock signal, a crystal of 6,144 MHz can be used.
- The aim is to minimize the number of components outside of the chip. However, the chip dissipation is also limited. This gives a compromise for the maximum current that electronics may dissipate for the control of the LEDs, etc.
- The supply voltage for the IC is 5 Volt.
- The IC is fabricated in a semi-custom CMOS process, for which specifications are given in appendix C.

## 22.5 System design

During system design, a first translation of the system specification into a global design takes place. These steps are described in Section 20.3 and Section 20.4. This is summarized in the following way:

- Subdivision of the system into separate subsystems.
- Description in words what the different subsystems have to do.
- Description of the interfacing (communication between the different subsystems).
- Global elaboration of the subsystems. Determining specifications and boundary conditions for the subsystems. Also the testability has to be considered.
- Mutual tuning of the subsystems (naming, timing, signals, frequencies). Determine which signals will be inputs or outputs (also for testing purposes). Make sure the distribution is such that the subsystems can be tested separately.
- Distribution of subsystems across the group, make a time plan. Make arrangements for mutual consultation.

Let the system design be evaluated by an assistants and by the tutor.

## 22.6 Implementation of subsystems (the module assignment)

### 22.6.1 The assignment

This step in the design process corresponds to what has been discussed in Section 20.5 and Section 20.6.

During the implementation of the subsystems (modules) it is the idea that each subsystem is designed by a group of 2-3 students and each group writes a report about it that is evaluated by the tutor.

Before you begin the implementation, let the tutor know which subsystem you are going to design and what the specs are.

In practice, it will occur that later, when the subsystems are connected to each other, the specifications of the subsystems must be adjusted. It is therefore not necessary to optimize the implementation of



each subsystem and to create the layout. However, synthesise the subsystem to the Sea-of-Gates cell library to get an impression of the size of the subsystem (number of logic ports and flipflops) and perform a simulation of the synthesized circuit to see if also the synthesized i VHDL description works correctly.

It is important that clear specifications are made and that, for the moment, the design is implemented according to these specifications. The assignment must be completed within approximately 3 weeks.

### 22.6.2 Report

The design of the module requires a short and concise report containing:

- Title of design and names of designers,
- The agreed specification for the module,
- A brief elaboration of the design with diagrams and state diagrams (whatever is applicable).
- Simulation results of the synthesized circuit, which show that the circuit meets the specifications.
- References to any used literature.

This report should be sent to the assistant and to the tutor together with the VHDL structural and behavior files that are created (so no synthesized VHDL files).

## 22.7 Putting everything together

After all subsystems (modules) have been designed, the subsystems are merged into one system and further elaboration of the subsystems takes place. Of course, it is important to test (simulate) regularly the system obtained so far. Testing the entire system can already be done when for the design only a mixed structural/behavioral VHDL description has been created. It is not necessary to wait until the layout is implemented. More about testable designs can be found in Section 22.9. Also Rapid prototyping (see Section 22.8) can be used to test the entire system at an early stage. Sometimes it will turn out that the initial specifications can not be met. Then alternatives will have to be devised or, in some cases, the system specification needs to be adjusted.

### 22.7.1 Assembling parts with GoWithTheFlow

Probably, parts of the design are created in different projects. It is possible to combine the different components with *GoWithTheFlow* with one or more of the following:

- By importing data from an entity in one project into another project.  
In the target project, use the command File → Import project cell, and select first the project from which parts are to be imported. Next select the relevant entities. Please note that all sub entities of an entity must first be imported before an entity can be imported.

- Import an entity's layout, including the layout of all sub-entities, into another project.  
First, in the source project, export a layout from the project by clicking on the appropriate layout icon and next using the Export tree command. A .tar file with the layout data will be created. Then import the layout into the target project as a result. The command File → Import layout tree.
- By importing all design data from a project into another project.  
First, export the data of the source project with the command File → Export project archive. This will create a .pa.tar file with the project data. Then, open the target project and import the data into this project with the command File → Import project archive. This command will transfer the following data: VHDL descriptions, circuit and layout, synthesis data, schematics, components, .lst, .ref and .cmd files. Note that data in the target project will be overwritten with new data when this data has the same (entity) name.

## 22.8 Rapid prototyping

In addition to the IC design software, also programmable hardware is available in the form of an Altera FPGA board, which allows the design to be verified during "run-time". This form of verification makes it possible to detect mistakes in the design that are much harder to find with simulation alone. While with simulation the user must specify all input signals and interpret all output signals at "the logic level", with rapid prototyping the hardware peripherals can be used to handle input and output at a "high level" It is advisable to verify the design with rapid prototyping after the simulation of the design has been successful, and before the layout of the design is created.

The following steps will be taken:

- After a first verification using the VHDL simulator the VHDL description of (a part of) the design can be synthesized for the FPGA present on the Altera board. This is done using the program *quartus*. Only behavioral and structural / circuit VHDL descriptions should be used for this. No synthesized VHDL descriptions for the Sea-of-Gates cell library. The program *GoWithTheFlow* has a command Utilities → Create Quartus DE Project to create a *quartus* project for a DE1 or DE2 Altera board.
- The (bitstream) file that is obtained after synthesis in *quartus* is used to configure the FPGA, again with *quartus*,
- On the Altera board there are several peripherals (switches, LEDs, LCD display, etc.) which can be used in conjunction with the FPGA. Other, extra, peripherals (loudspeaker, monitor, etc.) can also be connected using the connectors present on the board.
- The programmed board is "run-time" tested in various ways.
- If the design does not meet the requirements the original VHDL description is modified and the verification process goes back to the first step again.

To debug the design while it is running on the FPGA. it is possible to use the SignalTap II Embedded Logic Analyzer. A tutorial for this tool can be found on Brightspace.

Note that a working prototype on the FPGA board does not automatically mean that the VHDL description is finished. The "hardware" of the Sea-of-Gates chip will generally behave differently compared to the hardware of the FPGA board, and proper operation on the board does not imply a proper operation of the chip. A simulation of the for Sea-of-Gates cells synthesized VHDL description or (even better) a switch-level simulation for the circuit extracted from the layout of the chip will give more clarity on this.

## 22.9 Design for testability

After the circuit has been manufactured, it will be tested by the group later in the study year. Testing the manufactured circuit has a dual purpose:

- Check that the circuit meets the specifications.
- If the circuit does not meet the specifications: Try as accurately as possible to find out what the reason is for the fact that the circuit does not work properly, so that during a possible redesign the problem can be fixed.

It is very important that at an early stage during the design it is taken into account that the circuit will be tested later on. This because in the final phase of the design it is often difficult to modify the circuit for this. Therefore, the following section discusses how the testing will take place and what requirements are for the circuit (and the simulations) so that adequate testing is possible later on.

### 22.9.1 The method for testing after manufacturing

Testing of the circuit after manufacture is performed in the following 2 ways. In both cases, the test are performed on 2 a 3 pcs of assembled circuits in a DIL40 package.

- The circuit is connected to a logic analyzer. Via the logic analyzer, logic signals are applied to the inputs of the circuit and the logic signals at the outputs are observed. This is similar to simulating the circuit during the design, but now with the real hardware instead of the (VHDL) simulation model. To assess whether the circuit works, test vectors of a earlier VHDL simulation are used and it is checked if the outputs of the actual circuit give the same results as during simulation (assuming that the earlier VHDL simulation gave good results). For more information on this form of testing, see appendix ?? of the project manual.
- External components (switches, oscillator crystal for clock, displays, speakers etc) are connected to the circuits and the circuit is being tested in real time.

### 22.9.2 Considerations during design

Because of the testability of the design and the possible localization of problems, attention should be paid to the following points during the design:

- Make sure the outputs of the circuit, with a given set of input signals and after the reset, always go through a known, fixed set of values. When the logic analyzer is used, only then it is possible to make a comparison with the reference set of output values as previously obtained via simulation. Moreover, when the circuit can not be initialized to a known state, it may be possible that the circuit does not start working properly.
- When a 6.144 Mhz clock signal is used and one wants to watch the behavior of the circuit after 1 minute (real-time) then it means that there would be  $60 \times 6144000$  (about 360 million) clock cycles. That's way too much for using the switch-level simulator, and also a ModelSim simulation will probably take a lot of time. To solve this problem, solutions can be found in the following directions: (1) give the circuit a special testing mode in which it goes through interesting states while using a much slower clock signal. (2) give the circuit an option to start from a particular state after which the circuit reaches the states that are interesting for testing, much faster.
- When a circuit is not working properly, it is often not possible to see only from the outputs where the problem is. Therefore, try to connect as many as possible "interesting" internal signals to the output pins of the IC if there are still pins available. The number of input/output pins available is 32, excluding supply pins. To be able to effectively connect more inputs and outputs, one may think of demultiplexing and multiplexing some pins. Note that this will also require some additional pins for controlling the demultiplexers and multiplexers.

### 22.9.3 Considerations when creating testbenches

For the entire circuit, at least one test bench must be made which allows the circuit to be tested with the switch-level simulator and later, possibly, also with the logic analyzer. This test bench must meet the following requirements:

- Due to the finite resolution of the switch-level simulator, the simulation time should not exceed 100 ms. At a clock frequency of 6.144 MHz, that means the simulation can not be longer than about 614,000 clock periods.
- An input signal must remain input signal throughout the simulation. I.e. It is not allowed to set a logical value for some time to a pin and then "release" the signal during the rest of the simulation.
- For a specific set of input signals, after the reset always the same set of outputs should occur. E.g. after the reset, the outputs can not be dependent on an accidental random start state.

When a test bench will also be used for use with the logic analyzer. the testbench should not contain - due to memory limitations of the logic analyzer - more than 65,000 clock cycles.

## 22.10 Finishing the design

At the end of the assignment you need to make the layout suitable for placement on the Sea-of-Gates chip and you need to submit several files.

One of the things to be considered is that on the total circuit the library cell `osc10` (Appendix D.2.16) must be added in order to control the clock signal. The clock input signal for your design must be connected to terminal `f` of the `osc10` cell. The terminals `xi` and `xo` serve as connecting pins for the crystal to be brought outside (i.e., they become ports of the top-level entity; respectively with type `in` and type `inout`) and the terminal `e` (enable) must be attached to `VDD`. The layout for the total design must now be made such that it fits on 1 or 2 `bond_bar`s (depending on what has been decided on how to use the available chip area, see Section ??). It is often useful for this last placement to do the placement by hand, where 1 or 2 `bond_bar`s are temporarily placed in the background as reference for the available area. However, do not forget to remove this `bond_bar` (s) afterwards

A `bond_bar` should be connected to the layout obtained so far using the command `Add bond_bar` from the Utility menu of *GoWithTheFlow*. Here it is has to be specified which design terminals (ports of the top-level entity) connect to which IC pins. The `bond_bar` has 32 terminals for this purpose. `Vdd` and `gnd` will be connected separately. This will create a new circuit cell consisting of 2 components: your design and the `bond_bar`. Using the command `Place & route` this circuit must be converted into a layout. Place the your design manually over the `bond_bar`, such as indicated in Figure 28, and route them together with *trout*, without using the border terminals option. Note that the terminals of the `bond_bar` are located on the lower half of the area, at the left and at the right side.

To increase the chance on a working design it is further **important that this final layout (including bond\_bar) is always verified by means of a switch-level simulation**. This switch-level simulation can for example be based on the test bench that will later be used with the logic analyzer.

As a last step, create for the final layout, including `bond_bar`, an `ldm` file with the command `Make ldm`.

The files to be submitted to the project staff are:

- The LDM file for the total layout, created with the command `Make ldm`
- The corresponding `.buf` file that contains information about how the I/O pins should be connected. This file will automatically be generated when executing `Utilities → Add bond_bar`.)
- At least one `.ref` file with which the circuit can be tested. A `.ref` file is created from a `.lst` file using the `Generate → Reference file` command. Multiple `.ref` files can be submitted for different tests, e.g. to test different parts of the circuit.

## 22.11 Documentation

Important when writing documentation is that the documentation should make clear

- What the design does
- How the design should be used
- Which parts the design is composed of
- How the different parts have been implemented

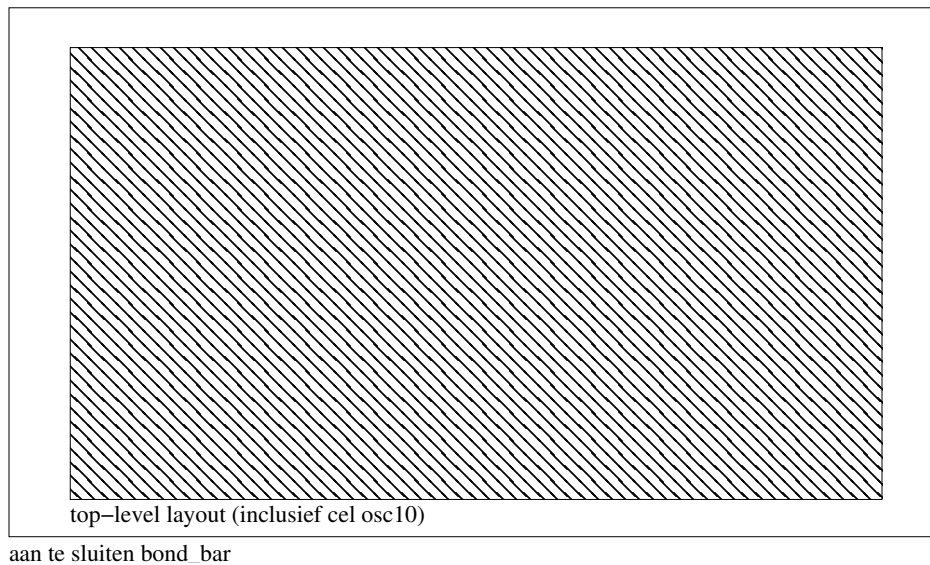


Figure 28: Placement of the top-level layout on top of the layout of the bond\_bar

- That the design has been sufficiently verified
- How the design can be tested after fabrication
- What the important conclusions/experiences are from the project.

Assume that the reader of the report has the same theoretical and practical knowledge of the field as yourself, but that he/she knows nothing yet about your specific design.

Structural and behavioral VHDL code can be included in the appendix of the report. VHDL code for important components may be included in the main text, but it is often better to use state diagrams and schematic descriptions. An overview of how the circuit is divided into sub-circuits should in any case be part of the report. VHDL code generated by the synthesizer does not belong in the report. However, relevant simulation results should be included. One or more layouts can optionally be part of the report.

## 23 Ideas for the Group Assignment (in Dutch)

### 23.1 Inleiding

In dit hoofdstuk worden beschrijvingen gegeven van de mogelijke hoofdopdrachten:

- de Avant-Garde Klok,
- de Infrarood-Besturing,
- de Reactiemeter,
- de Pong Spelcomputer,
- Alternatieve opdracht.

Eén van deze opdrachten zult u met uw groep gaan ontwerpen.

Naast de beschrijvingen van de opdrachten vind u ook bij elke opdracht een aantal specificaties waaraan de schakelingen moeten voldoen. Deze specificaties zijn niet volledig, maar bedoeld als eerste aanzet. Zelf zult u nauwkeuriger moeten specificeren.

Voor analoge deelschakelingen die mogelijk voorkomen in het ontwerp wordt men verwezen naar appendix E.

### 23.2 De Avant-Garde Klok

#### 23.2.1 Inleiding

Klokken zijn behalve technische hoogstandjes ook altijd mode-objecten geweest. Iedere periode in de geschiedenis kent zijn eigen karakteristieke uitvoering van een tijdmetr. Het ontwerpen van zoiets “gewoons” als een klok kan daarom nog heel bijzonder zijn. De informatie-overdracht van klok naar mens kan op talloze manieren verzorgd worden. Dit loopt van de ouderwetse wijzerplaat met wijzers tot aan de elektroden op de grote hersenen met een bionisch RS232-interface. Puur technisch gezien wordt van klokken een steeds grotere nauwkeurigheid verlangd. Dit alles in beschouwing genomen blijft het nog steeds een uitdaging om een klok te ontwerpen die behalve als functioneel technisch object ook zijn plaats in de wereld verdient door zijn uitvoeringsvorm. De directeur van TECHNO-GADGET had op een druilerige zondagmiddag het idee gekregen om een klok met een ultieme nauwkeurigheid te maken. Hij dacht ineens aan een zender die in Duitsland staat, die met atoomnauwkeurigheid de tijd gencodeerd uitzendt, de zogenaamde DCF-zender. Na verdere ontwikkeling van zijn ideeën, heeft hij contact opgenomen met de beroemde Italiaanse designer Verstamo, om een klok te ontwerpen die eeuwigheidswaarde heeft.

Er zijn talloze manieren om de tijd zichtbaar te maken. De ontwerper heeft gekozen moderne middelen te gebruiken, om de dynamiek van de techniek op de bezitter van de klok over te dragen. Hiervoor gebruikt hij een LED-display voor de aanduiding van de uren, zeven LEDs voor de aanduiding van de weekdag en twee draaispoelmeters voor de aanwijzing van de minuten en seconden. Om de gebruikers te helpen de klok zo te plaatsen dat deze het DCF-sigaal kan ontvangen, is ook op een

strategische plaats een LED aangebracht, die aangeeft of het DCF-sigitaal wordt ontvangen. Zo mogelijk zal de klok ieder kwartier het wijsje spelen dat op dat moment ook door de Big Ben in Londen wordt gespeeld. Speciaal voor mensen die ooit in Londen geweest zijn maakt dit de klok natuurlijk extra aantrekkelijk.

Helaas heeft Verstamo geen verstand van elektronica, dus de chip die dit alles moet besturen zal door de echte deskundigen ontworpen moeten worden. Hier is natuurlijk haast bij, want zoals alle leken denkt ook Verstamo dat elektronisch alles zo maar kan. Hij heeft zijn klok dus al verkocht aan een grote fabrikant die reeds een grote reclamecampagne heeft opgezet. Verstamo kan dan ook een claim van ongeveer 100 miljoen euro aan geïnvesteerd reclamegeld verwachten indien het ontwerp niet op tijd komt. Zijn adviseurs hebben hem al gewaarschuwd dat simulaties ter verificatie van de chip veel tijd kunnen gaan kosten. Wordt bijvoorbeeld een schakeling gesimuleerd met een klokfrequentie van 6 MHz waaruit een puls per week moet komen, dan moet hij beseffen dat de simulator niet veel meer dan 6 miljoen gebeurtenissen per seconde rekentijd kan simuleren en dat de simulatie dus ook makkelijk een week zal kunnen gaan duren. Verder zitten er in een week bijzonder veel seconden, zodat data-opslag voor de simulator ook wel eens een probleem zou kunnen gaan worden. Ook later, wanneer de chip uit de fabriek komt zal het ondoenlijk zijn de chip een week lang in een tester te stoppen om te kijken of de weekaanduiding werkt. De chip zal niet op een veel hogere klokfrequentie kunnen werken, dus hier zal weinig winst te behalen zijn.

*Het is daarom zaak ruime aandacht te besteden aan simuleerbaarheid en testbaarheid en niet aan een grote hoeveelheid (ontestbare) elektronische grapjes.* Het aanbrengen van verschillende testmogelijkheden zoals het “opdelen” van delerketens die los testbaar zijn binnen een redelijke tijd is van zeer groot belang.

### 23.2.2 Specificaties DCF-ontvanger

- Als invoer dient een antenne-sigitaal, dat de DCF-code bevat en door de al bestaande ontvanger wordt omgezet naar een sigitaal dat direct als invoer kan dienen voor de schakeling, die voor de verwerking van dit sigitaal zorgt. Deze signalen hebben logische niveaus die door deze schakeling verwerkt kunnen worden, maar nog niet gesynchroniseerd en gedecodeerd zijn.
- Als uitgang van de DCF-klok dienen enige displays die de tijd en ook een luidspreker die de tonen van de Big Ben melodie hoorbaar maakt. Bij het genereren van deze tonen kunnen verschillende luxe-klassen worden onderscheiden. De tonen kunnen een verschillende tijdsduur hebben, maar ook een volume dat in de tijd afneemt. Dit geeft het effect of een echt klokkenspel wordt bespeeld. Deze opties brengen wel een ingewikkelder en groter ontwerp met zich mee!
- De nauwkeurigheid van de klok wordt geleverd door de beroemde DCF-tijdzender. Deze zender zendt op 77.5 kHz een sigitaal uit in de vorm van een serie bits. Elke minuut wordt er een reeks van 59 bits uitgezonden die de complete tijd weergeeft (tijd, dag, maand, jaar en controle-bits). De afwijking is ongeveer 1 seconde per 300.000 jaar, dus dat is net voldoende. Een bijzonder simpele ontvanger (die niet ontworpen hoeft te worden) levert aan de uitgang een digitaal sigitaal (pulsen van 100 ms en 200 ms), waaruit de tijd bepaald kan worden.
- De ontvangst van het DCF-sigitaal zal doorgaans goed zijn. Als de zender echter niet ontvangen wordt, of als er storingen worden gedetecteerd, dan zal de klok moeten besluiten het voorlopig maar met het eigen kristal te doen. Zodra goede ontvangst weer mogelijk is, zal de

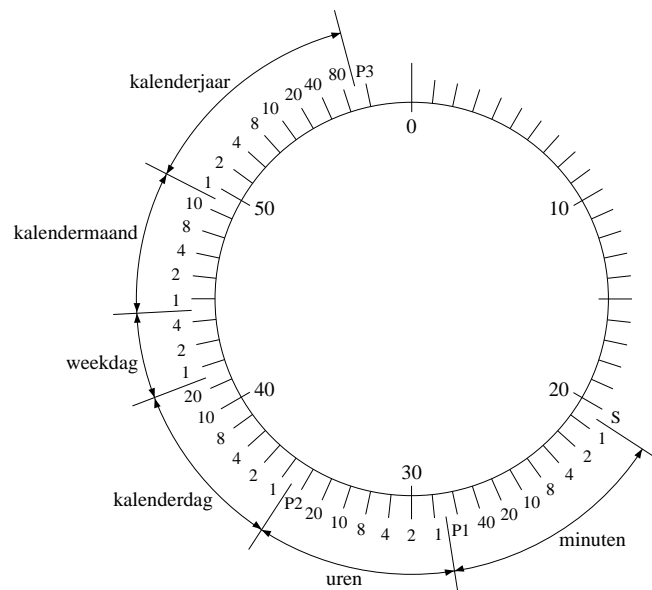


klok zich weer precies gelijk zetten. Wanneer een DCF-sigitaal wordt ontvangen, moet de klok dit aangeven door het laten branden van een LED.

- De firma TECHNOGADGET verwacht dat de eerste serie exclusieve klokken nog voor grote bedragen verkocht kunnen worden. Wanneer de klokken bekend worden, wil iedereen zo'n klok kopen. Om iedereen in staat te stellen zo'n klok te kopen, moeten ook goedkopere versies gemaakt kunnen worden. Door verschillende mogelijkheden niet te gebruiken, kan een goedkope versie worden aangeboden. Door een modulier ontwerp te maken, is het eenvoudig om bepaalde onderdelen weg te laten. Dit heeft als voordeel dat de verschillende circuitdelen los van elkaar getest kunnen worden, omdat ze ook los van elkaar kunnen werken. Dit is ook prettig voor de fabrikant, omdat deze bij uitval tijdens de fabricage, wanneer bijvoorbeeld het speelwerk defect is, de klokken kan gebruiken in een wat goedkopere versie van de DCF-klok zonder geluid.

### 23.2.3 Overige specificaties

**DCF-sigitaal** De codering van het DCF-sigitaal ziet er als volgt uit:

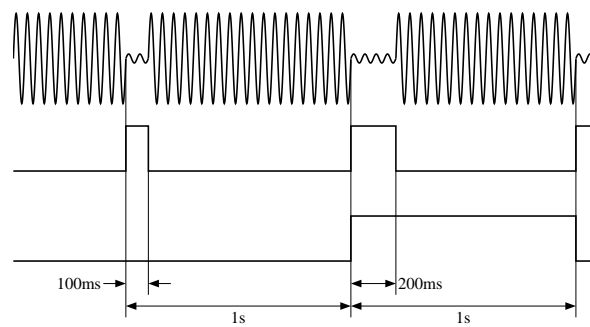


Figuur 29: Codering van het DCF-sigitaal

- Alle waarden worden gegeven in BCD-code.
- Er zijn drie pariteitsbits (controle-bits: P1, P2, P3) aanwezig. Deze kunnen genegeerd worden.
- Bij de weekdag is maandag als "001" gecodeerd. De overige coderingen zijn triviaal.

Het sigitaal zelf ziet er ongeveer zo uit als bovenaan aangegeven is in figuur 30

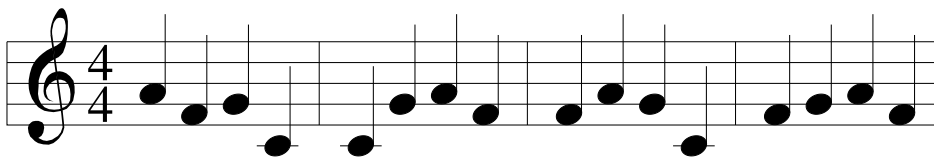
Een (overigens vrij eenvoudige) ontvanger, die u niet zelf hoeft te maken, maakt er een sigitaal van dat lijkt op het middelste sigitaal. Dit sigitaal wordt aangeboden aan de chip.



Figuur 30: Vorm van het DCF-signaal

De DCF-zender genereert om de seconde een puls met een breedte van 100 ms (voor een '0') of 200 ms (voor een '1'). Op de 60-ste seconde wordt geen puls uitgezonden. Het onderste signaal geeft aan hoe u dit zou kunnen decoderen.

**Big Ben melodie** de Big Ben melodie ziet er als volgt uit:



Figuur 31: Big Ben melodie

De toonhoogten die gebruikt worden zijn:

C	1046.5 Hz
F	1396.9 Hz
G	1567.9 Hz
A	1760.0 Hz

**Draaispoelmeter** De draaispoelmeters zijn lineair en vragen voor een volledige uitslag een stroom van 2 mA. De impedantie is kleiner dan 100  $\Omega$ .

**Uitgangen** De uitgangen van de DCF-klok kunnen bij een te grote belasting gebufferd worden. Om een goede aansturing van de buffers mogelijk te maken, moet de uitgangsstroom van de (analoge) signalen (extern) regelbaar zijn.

## Toetsen

impedantie:                      uit:                      > 10 M $\Omega$   
    aan:                      < 100  $\Omega$

stabiliseringstijd: < 50 ms

### Luidspreker

impedantie: 50  $\Omega$  ( $\pm$  10 %)  
resonantie-frequentie: 440 Hz  
bandbreedte: 350-5000 Hz  
vermogen: < 0.3 W  
rendement: 100 % (aanname)

**Funcities** Er zijn een aantal duidelijk te onderscheiden functies in de klok waardoor de firma TECHNOGADGET in staat is deelontwerpen uit te besteden aan verschillende teams:

- De DCF-decoder, die in staat is de dcf-pulsen te decoderen en om te zetten in logische niveaus.
- De schakeling die besluit of er een geldige datareeks is binnengekomen uit de ontvanger.
- Een “autonome klok” die parallel de DCF-data kan laden als deze geldig blijkt te zijn, maar die zelfstandig kan lopen in andere gevallen.
- Een display sectie, die de LEDs en de draaispoelmeters kan aansturen.

## 23.3 De Infrarood-Besturing

### 23.3.1 Inleiding

Een instituut beschikt over een ionenbron, die ingesteld wordt m.b.v. een vijftal parameters, die alle kunnen worden ingesteld door een getal tussen de 0 en 99. Deze ionenbron is geplaatst in een afgeschermd ruimte omdat hij op een hoge spanning moet zijn aangesloten. Probleem is nu, dat steeds wanneer voor een proef met deze bron een bepaalde parameter moet worden gewijzigd, de gehele opstelling moet worden uitgeschakeld om de betreffende parameter te kunnen instellen. Gelukkig voor dit instituut liep er een slimme TUD-student stage, die het idee opperde om een infrarood kanaal te maken, waarmee de waarden voor de parameters naar de bron-opstelling zouden kunnen worden overgebracht, zonder de opstelling uit te schakelen. Hij merkte op dat de devices voor het instellen van de parameters reeds een busaansluiting hadden, waarop, in binaire code de waarde van de parameter kon worden geplaatst, waarna deze waarde werd overgenomen wanneer de enable lijn van het betreffende device gedurende tenminste 100  $\mu$ s ‘hoog’ werd. Bovendien wist hij dat er van die handige telefoon- toetsenbordjes bestaan, die heel goed als invoermedium van gegevens kunnen dienen.

Helaas, toen dit allemaal bedacht en besproken was met de directie van het instituut (waar veel tijd in ging zitten), was zijn stagetijd om. Maar gelukkig kan een nieuwe op-goep uitkomst brengen door dit infrarood kanaal te realiseren, door een zend- en een ontvang-chip voor dit probleem te maken met de onderstaande specificaties.

### 23.3.2 Specificaties infrarood-kanaal

- Het infrarood-kanaal zal bestaan uit twee chips: een zender-chip, die een pulsreeks uitzend om de parameters in te stellen, en een ontvanger-chip die de uitgezonden pulsreeks decodeert en de parameters de juiste waarden geeft.
- Als invoermedium voor het kanaal zal gebruik worden gemaakt van een telefoon toetsenbord, waarbij de volgende afspraken gelden:
  - De nummers op het toetsenbord worden gebruikt om de waarden van de parameters op te geven.
  - De \*-toets wordt gebruikt om het device waarvan de parameter moet worden veranderd aan te geven. Steeds wanneer op deze toets wordt gedrukt zal het volgende device worden aangewezen.
  - De #-toets wordt gebruikt om de opgegeven waarde van de parameter in het aangegeven device te laden.

Bovendien dient er rekening mee te worden gehouden, dat het toetsenbord een ontdendertijd heeft van 40 ms.

- De zender moet tevens een uitgang hebben, waarop een 7-segment display kan worden aangesloten, waarop de laatst ingedrukte toets is zichtbaar gemaakt. Hierbij moet de \*-toets worden getoond als een 'A' (advance) en de #-toets als een 'U' (update).
- De ontvanger moet een 7-bits brede uitgang hebben, waarop de 7-bits binaire code voor de parameter-waarde wordt geplaatst en een uitgang voor het enable bit van ieder device, waarmee de parameter-waarde in dit device kan worden geplaatst.
- De laatst verstuurde waarden van elk device, dit zijn dus de waarden waarop de bron staat ingesteld moeten worden zichtbaar gemaakt op een LCD. Voorts moeten hierop ook de nieuw in te stellen waarde worden getoond, alsmede een indicatie voor het geselecteerde device waarop deze waarde zal worden geplaatst.  
Hiertoe is een LCD aanwezig (met documentatie), waarop 2 regels van elk 16 karakters kunnen worden weergegeven.
- Daar de zender-chip en ontvanger-chip alleen met elkaar communiceren via het infrarood-kanaal en er verder geen verbinding tussen de twee chips mogelijk is, wordt er dus ook voor het zend- en ontvang-gedeelte gebruik gemaakt van een afzonderlijke klokpuls.  
Er moet dus rekening mee worden gehouden, dat deze twee klokpulsen geen fase-koppeling hebben. Ook moet er rekening worden gehouden met een eventueel verschil in frequentie van beide pulsen van maximaal 5%.
- De pulsduur van de enable-signalen van de devices moet minimaal 100  $\mu$ s bedragen, en gedurende minimaal 50  $\mu$ s voor deze puls tot 50  $\mu$ s na deze puls moet de waarde op de parameter-uitgangen stabiel zijn.

## 23.4 Een Reactiemeter

### 23.4.1 Inleiding

Tijdens een feestelijke bijeenkomst raakten een drietal studenten in gesprek over hun uiterst snelle reactietijd op een gebeurtenis (zelfs na een aantal glazen bier). Natuurlijk vond elk van de drie dat hij toch verreweg de kortste reactietijd had. De discussie liep hoog op, tot één van de studenten op het idee kwam om een schakeling te maken, die voor eens en altijd kon uitmaken wie van hen werkelijk het snelst was. Ze bedachten hierna zo'n schakeling die de hieronder beschreven werking zou moeten hebben. Helaas is het tot een realisatie van de schakeling nooit meer gekomen. Maar hier kan een op-groep wellicht uitkomst brengen.

### 23.4.2 Werking reactiemeter

De reactiemeter moet twee cijfers weergeven: een in te stellen referentiecijfer en een steeds veranderend reactiecijfer. Voorts moet voor elke deelnemer een drukknop aanwezig zijn en een tijdweergave van hun reactietijd.

Wanneer het referentiecijfer en het reactiecijfer gelijk zijn moeten de deelnemers zo snel mogelijk hun drukknop indrukken. Wie het eerst heeft gedrukt behoudt zijn tot dan toe gebruikte reactietijd. Bij de twee andere deelnemers moet het verschil tussen hun reactietijd en de reactietijd van de snelste deelnemer bij hun totale reactietijd worden opgeteld. Wie zijn toegestane totale reactietijd (bijv. 1 seconde) heeft gebruikt valt af. Degene die overblijft is de winnaar.

Om te voorkomen, dat 'slimme' deelnemers constant hun drukknop indrukken of ingerukt houden moet ook een straf tijd aan de reactietijd van een deelnemer worden toegevoegd, wanneer hij zijn drukknop indrukt terwijl de cijfers niet gelijk zijn.

### 23.4.3 Specificaties reactiemeter

In verband met het bovenstaande moet de reactiemeter voldoen aan de volgende specificaties:

- Er moet een in te stellen referentiecijfer worden weergegeven.
- Er moet een reactiecijfer worden gemaakt en getoond, dat om de seconde (pseudo-)random verandert.
- Er moet een drietal drukknoppen aanwezig zijn waarop moet worden gedrukt wanneer beide cijfers gelijk zijn.
- Er moet een drietal tijdweergaven zijn, waarop de geaccumuleerde reactietijden van de drie deelnemers zichtbaar zijn.
- Voor elke deelnemer is een indicatie aanwezig, die aangeeft wanneer een deelnemer zijn totale reactietijd heeft overschreden.
- Er moet een reset knop aanwezig zijn, waarmee de schakeling kan worden gereset.
- De reactietijden moeten worden gemeten met een nauwkeurigheid van 0.01 seconde.

- Voor de klok van de schakeling mag gebruik worden gemaakt van een externe pulsgenerator.
- Voor de totale reactietijd die mag worden gebruikt moet een geschikte waarde in de orde van grootte van 1 seconde worden gekozen.
- De 'straf' voor het foutief indrukken van de drukknop mag door de ontwerper(s) van de schakeling worden bepaald.

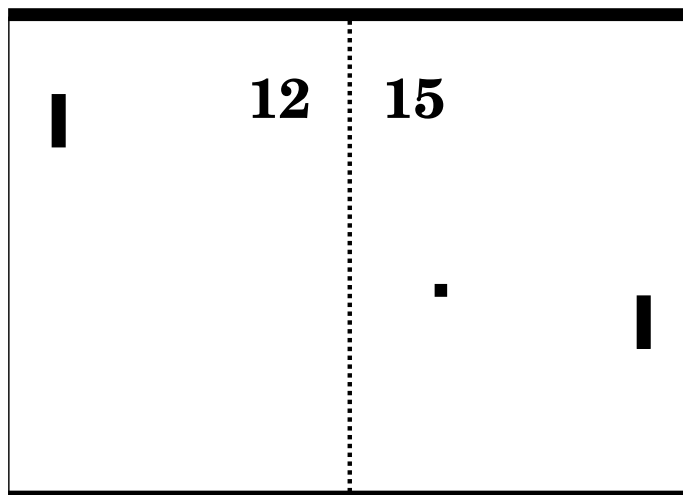
Voor de weergaven van de genoemde grootheden kan worden gekozen uit twee opties:

- Voor het reactiecijfer en het referentiecijfer wordt gebruik gemaakt van een 7-segments display, de tijdsoverschrijding wordt aangegeven met een led en de waarden van de reactietijden, worden via (op de chip te ontwerpen) digitaal-analoog omzeters door draaispoelometers aangegeven.
- Alle grootheden worden getoond op een lcd-schermje met een afmeting van 2 regels met elk 16 karakters.

## 23.5 De Pong Spelcomputer

### 23.5.1 Inleiding

De opdracht is om op een Sea-Of-Gates chip een spelcomputer te maken waarvan de uitgangen kunnen worden aangesloten op de VGA-ingang van een monitor en waarmee een spel gespeeld kan worden dat populair was in de begintijd van de spelcomputers: "pong tennis". Zie de afbeelding hieronder. Voor meer achtergrondinformatie over "pong", zie ook <http://www.pong-story.com/intro.htm>.



Figuur 32: Het beeld van de Pong spelcomputer

### 23.5.2 Het spel

Het idee van het spel is dat twee spelers hun racket omhoog en omlaag bewegen om een balletje te laten terugkaatsen dat zich schuin over het speelveld beweegt. Het balletje voert volkomen elastische botsingen uit tegen de boven en onderwand. Ook tegen de rackets worden volkomen elastische botsingen uitgevoerd, echter met het verschil dat wanneer de bal het racket raakt op de onderste helft de bal altijd terugkaatst naar beneden en wanneer de bal het racket raakt op de bovenste helft de bal altijd terugkaatst naar boven. Elke keer wanneer het balletje botst tegen een wand of racket is het geluid "pong" te horen. Wanneer een speler de bal mist en de bal achter hem passeert krijgt de tegenstander 1 punt erbij. Wanneer één van de spelers 15 punten heeft, heeft deze speler het spel gewonnen.

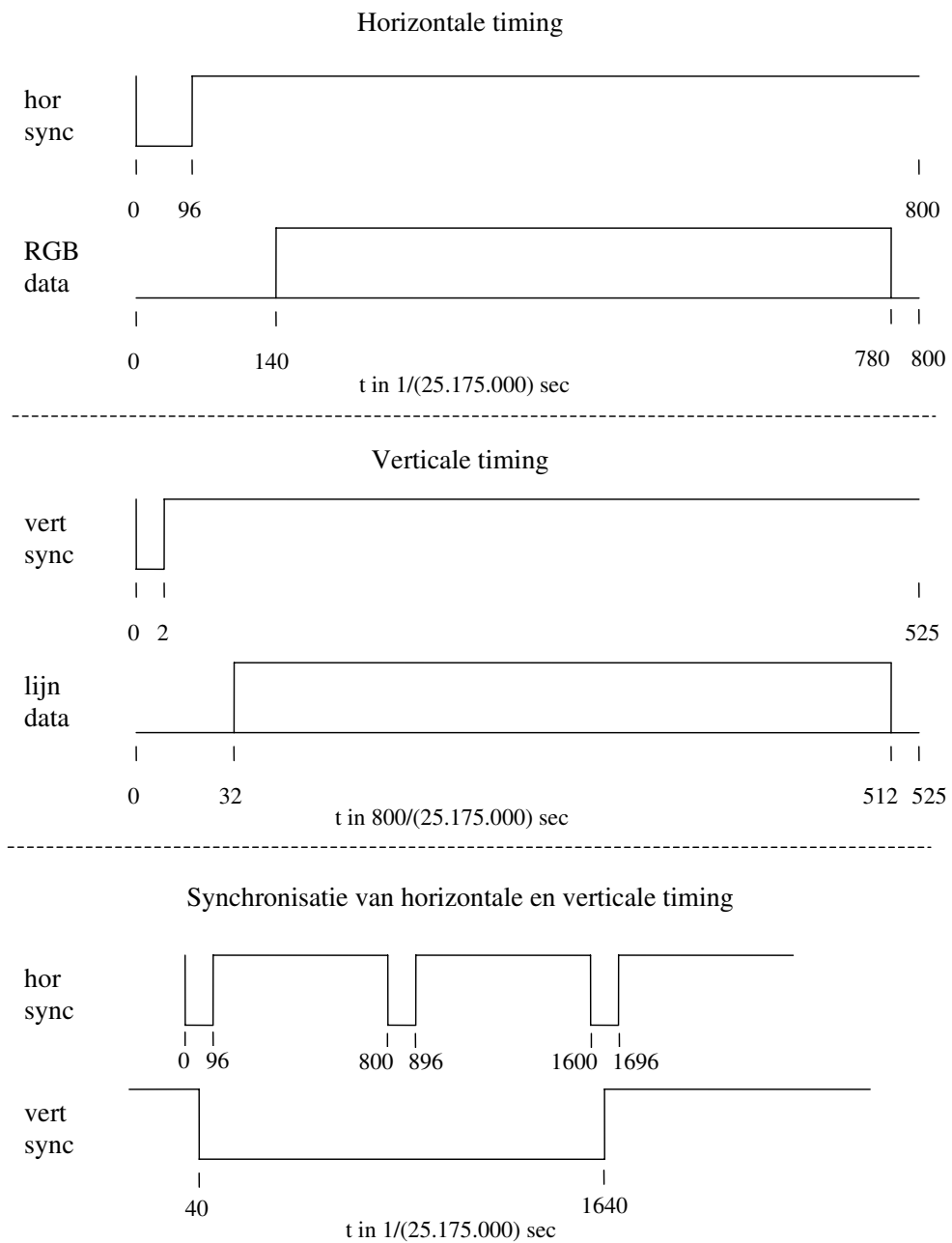
### 23.5.3 De spelbesturing

Elke speler heeft twee drukknoppen: één om het racket omhoog te laten gaan en één om het racket omlaag te laten gaan. Verder zijn er:

- een reset knop om de stand op 0 - 0 te zetten en het spel te starten.
- een schakelaar waarmee 2 verschillende balsnelheden kunnen worden ingesteld:
  - horizontaal: ongeveer 500 pixels/sec, verticaal: ongeveer 250 pixels/sec.
  - horizontaal: ongeveer 1000 pixels/sec, verticaal: ongeveer 500 pixels/sec.
- een schakelaar waarmee 2 verschillende afmetingen van de rackets kunnen worden ingesteld:
  - 64 pixels verticaal
  - 32 pixels verticaal

### 23.5.4 De aansturing van de monitor

De chip dient een VGA-sigitaal te genereren voor een beeld van 640x480 pixels. Een korte schematische beschrijving van deze standaard is te zien in figuur 33. Voor verdere informatie, zie bijvoorbeeld [http://www.epanorama.net/documents/pc/vga\\_timing.html](http://www.epanorama.net/documents/pc/vga_timing.html). Merk op dat 5 uitgangssignalen nodig zijn voor een VGA-sigitaal: De waarde voor het rode pixel, de waarde voor het groene pixel, de waarde voor het blauwe pixel, een horizontaal sync-sigitaal en een verticaal sync-sigitaal. De laatste twee signalen zijn digitale signalen van 0 of 5 Volt en corresponderen dus met de op de Sea-Of-Gates chip gebruikte waarden voor een logische 0 en 1. De RGB pixelwaarden kunnen in principe analoge waarden hebben tussen 0 en 0.7 Volt. Voor ons spel zullen we de RGB pixels alleen helemaal aan of helemaal uitzetten en zullen er 3 externe level-shift schakelingen beschikbaar zijn waarmee de 5 Volt uitgangswaarde van de Sea-Of-Gates chip kan worden omgezet naar 0.7 Volt. Hoewel volgens de definitie van de 640x480 pixel mode VGA-standaard een klok van 25.175 MHz vereist is (om elke klokpuls van 39.7 nsec. een R, G en B pixelwaarde over sturen), blijkt deze frequentiewaarde niet zo kritisch te zijn en zullen de meeste VGA-monitors nog werken wanneer er een afwijking van 5 % in de timing optreedt. Merk verder op dat wanneer de resolutie van de objecten in het spel niet te groot wordt gekozen en elke reeks van  $n$  ( $n = 2, 4$  of  $8$ ) achtereenvolgende pixels dezelfde waarde hebben er verder ook volstaan kan worden met een  $n$  keer zo lage klokfrequentie.



Figuur 33: VGA-timing



### 23.5.5 Het "pong" geluid

Via een kleine luidspreker van 50 Ohm die direct op de chip wordt aangesloten dient het "pong" geluid gegenereerd te worden (een 1kHz blokgolf signaal).

### 23.5.6 De overige randvoorwaarden

Voor dit ontwerp mag een chip-oppervlak van 2 bond\_bars worden gebruikt.

## 23.6 Alternatieve opdracht

Naast bovengenoemde opdrachten kan een groep ook zelf een opdracht bedenken en zelf de specificaties opstellen. Enige suggesties voor een alternatieve opdracht zijn:

- Een ander soort spelcomputer: bijvoorbeeld Snake, Tetris, Packman, etc.
- Een thermostaat.

Zo'n soort opdracht dient (1) aan dezelfde algemene randvoorwaarden te voldoen als de andere opdrachten (zie sectie 22.4), en (2) eenzelfde complexiteit te hebben (d.w.z. een chip-oppervlak van 1 à 2 bond\_bars te omvatten). Verder dient de opdracht eerst door de begeleiding te worden goedgekeurd voordat met de verdere uitwerking wordt begonnen.



## Part IV

# Appendices

## A Some instructions for using Linux

### A.1 Login in for the first time

To use the project software, you must be log in under linux, using your NetID username and password. If you have problems with login in, please contact the EWI service desk at the ground floor near the main entrance.

After login in for the first time, you must do the following once:  
Open a terminal window and type in the following command:

```
source /data/public/common/software/opprog/bin/op_init
```

This ensures that your file .bashrc is modified, so that every time you log in, your search paths are set to the location of the tools of the project. All programs used during the project will then be available. This procedure should be executed only once.

### A.2 Some linux commands

Some common Linux commands that can be given in a terminal window are:

**ls** Show the contents of a current directory. The option -l gives details about each file and subdirectory.

**mkdir** Create a new directory.

`mkdir <dir_name>` create a directory <dir\_name> below the current directory.

**cd** Change the current directory.

`cd` go to your home directory  
`cd ..` go to the parent directory  
`cd <dir_name>` go to the specified directory

**cp** Copy a file.

`cp <file_from> <file_to>` make a copy of <file\_from> under the name <file\_to>  
`cp <file_from> <dir_to>` make a copy of <file\_from> in directory <dir\_to>  
with the name<file\_from>

**rm** Remove a file.

`rm <file_name>` remove the file <file\_name> from the file system.  
`rm -rf <dir_name>` remove the directory <dir\_name> and all files and subdirectories in it.  
uit het file\_systeem

**man** Give an explanation to a command.

man rm                      give explanation to the command rm

For file and directory names, pattern matching characters like the character `*` can be used. For example, the following command copies all files ending on `.vhd` to a directory `proj/VHDL`

```
cp *.vhd proj/VHDL
```

## **B ICT Support for Project Work**

When working in a project group, you have to be able to work together on documents and source code, keep track of bugs and problems, contact your group members, etc. There are several online options you can use to share documents, work together on source code, or create mailing lists, such as Dropbox or GitHub. All these options store data on publicly available servers, which may be undesirable from a security or privacy perspective. The EWI Projects Server offers support for these tasks by allowing you to create a versioning system repository, complete with bug tracking, wiki, and group email – all stored within servers of the TU Delft.

### **B.1 Available Services**

The projects server offers several services, each of which is briefly described in the following paragraphs.

#### **B.1.1 Versioning System: GIT or SVN**

You can create a GIT or SVN repository. Such a repository can be used to keep track of your text-based files, such as  $\text{\LaTeX}$  files and source code (including Matlab m-files). When working with multiple people on the same files, the system allows you to merge in changes. It also allows you to safely experiment with changes to the files. You can revert unwanted changes and only change the files in the online repository when you're sure the modified files are correct.

The repository contains the entire history of your project, so it can also serve as a project archive.

GIT and SVN offer similar, but not identical functionality. Both systems are supported on the lab computers, under both Windows and Linux. For Windows, a graphical user interface is available.

Note that a versioning system is less suited for use with binary type files, such as Word documents. When a text file is modified, only the changes are stored in the repository. This keeps the repository small and allows you to merge in changes. Modified binary files are replaced. For these types of files you're recommended to use Blackboard.

#### **B.1.2 Bug Tracker and Wiki: Trac**

The Trac bug tracker can be used to document bugs. This way, all project members know about existing bugs and any project member can choose to work on one of those bugs. Trac also provides a wiki. The wiki provides you with editable wiki-style web pages that can be used to document the project, or the structure of the project repository or anything else.

#### **B.1.3 Mail group**

The projects server provides you with a group-email address. This will enable you to easily send an email to all project members. The email address is derived from the project name.

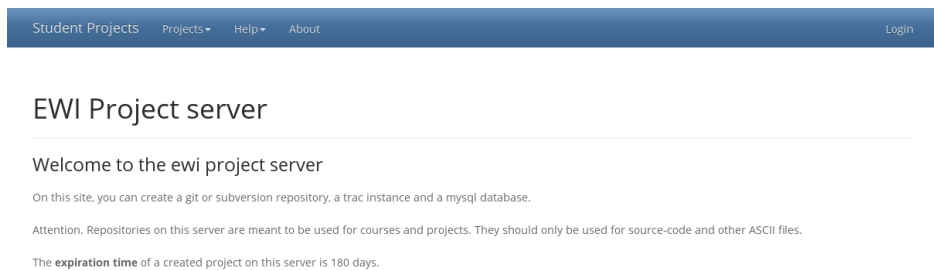


Figure 34: Screenshot of the EWI Projects Server main webpage

## B.2 EWI Projects Server

You can find the EWI Projects Server at the following webaddress:

`https://projects.ewi.tudelft.nl/`

A screenshot of the projects server webpage is shown in Figure 34. Any EWI student can create a repository on the EWI Projects Server. Login with your NetID and password using the button on the right in order to be able to create a project. Using the **Projects** drop-down menu, you can create and manage projects. The **Help** menu contains basic information on using GIT and SVN.

### B.2.1 Creating and Managing Projects

Via the menu **Projects**→**Create project** you can create a new project. This will show you the webpage shown in Figure 35. When you create a new project, the following components will be created:

- Either a GIT repository or an SVN repository
- A Trac bug-tracker (including wiki)
- A project group email address
- A MySQL database<sup>2</sup>

During the creation of a project, you can choose between a GIT or an SVN repository and furthermore, choose between several different access specifications for the repository, Trac, and Wiki. You can also specify whether or not the repository name should be visible for anyone in the project list.

As the creator of the project, you're automatically added to the list of users, you can add additional users by adding their NetID in the field **Members netid**.

---

<sup>2</sup>The MySQL database is probably of little use for the EPO project.

Student Projects Projects ▾ Help ▾ About

## Create a new project

**Project name**

**Description**

**Repository type**

☐ Enable anonymous access for repository

**Trac wiki access**

**Trac ticket access**

☒ Show in public project list

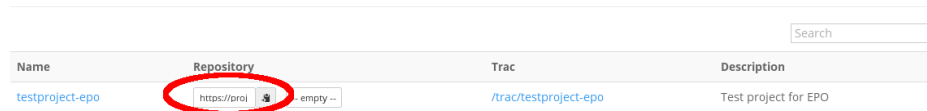
**Members netid**

Add netid of all members of this project

Create Project

Figure 35: Screenshot of the Create a new project webpage

## Projects you created




Name	Repository	Trac	Description
<a href="#">testproject-epo</a>	<a href="https://prol">https://prol</a>  - empty --	<a href="#">/trac/testproject-epo</a>	Test project for EPO

Figure 36: Screenshot of the My Projects webpage. The highlighted area contains the repository link.

### B.2.2 Repository Address

The creator of the repository will receive an email with the project data. Every project member can also find the repository link in the overview on the Projects website under **Projects→My projects**, as shown in Figure 36.



## C De Sea-of-Gates Chip

### C.1 Globale beschrijving

De chip waar de projectontwerpen op gemaakt worden is van het "semi-custom" type. Dit houdt in dat de grootte van de chip vast staat en dat ook de grootte, ligging en aantal van de te gebruiken transistoren al bekend is. In feite is de chip al voor een groot deel vervaardigd, alleen de metallisatie die nodig is voor de gewenste schakeling, moet nog worden aangebracht. De gekozen structuur op de reeds gefabriceerde wafers (plakken silicium met een doorsnede van 4 inch) is die van de moderne gate-array: de Sea-of-Gates structuur. Deze structuur leent zich primair voor digitale schakelingen, maar analoge zijn ook mogelijk.

Fabricageproces:

Een  $1.6\ \mu\text{m}$  nwell-CMOS proces (het Philips C3DM proces).

Grootte van de chip:

- 10 x 10 mm, (46 exemplaren op een 4 inch plak)
- 144 aansluitpinnen waarvan de functie programmeerbaar is door middel van metallisatiepatronen,
- 200.000 transistoren (de helft is pmos, de andere helft is nmos),
- afmeting nmos transistor:  $1.6 \times 23.2\ \mu\text{m}$ ,
- afmeting pmos transistor:  $1.6 \times 29.6\ \mu\text{m}$ .

De metallisatie van de wafers, het uitzagen en afmonteren van de chips gebeurt in het Else Kooi Lab. Voor de metallisatie zijn 4 maskerstappen nodig: 2 metaal- en 2 kontaktgaten maskers.

De grootte van de chip is zodanig gekozen dat meerdere typen ontwerpen gemaakt kunnen worden. Ook is het niet bij voorbaat onmogelijk om een redelijk grote schakeling te realiseren. De belangrijkste voorwaarde is dat het Else Kooi Lab in staat moet zijn de metallisatie aan te brengen. Hieronder volgt een lijst met de maskers die het Else Kooi Lab voor de metallisatie moet maken (en dus door de ontwerper gespecificeerd) en hun functie. Voor de volledigheid is een opsomming toegevoegd van de maskers die nodig waren om de sea-of-gates chip te maken.

Else Kooi Lab maskers:

- co (con, cop, cps) : Definieert posities van kontaktgaten in het oxide tussen de onderste metaallaag(in) en od en ps gebieden.
- in : Definieert de onderste (eerste) metaallaag.
- cos : Definieert de positie van kontaktgaten in het oxide tussen eerste en tweede(bovenste) metaallaag(ins).

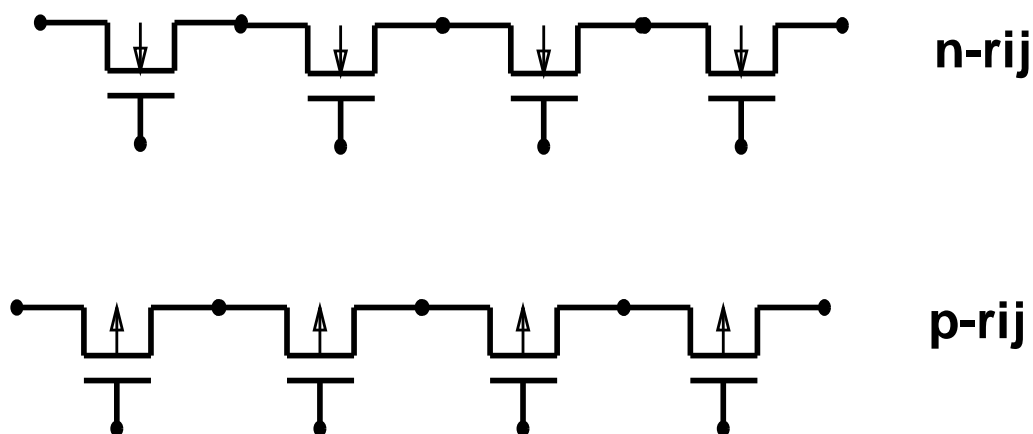
- ins : Definieert de bovenste (tweede) metaallaag.

Overige maskers:

- nw : nwell, definieert de gebieden waarin p-transistoren kunnen worden gedefinieerd.
- od : Definieert de actieve gebieden: plaats van source, gate en drain van beide typen transistoren.
- sn : Definieert die delen van de actieve gebieden die van het n-type zijn.
- sp : Definieert die delen van de actieve gebieden die van het p-type zijn.
- ps : Definieert gates van transistoren.

## C.2 De kern van de chip, vanuit circuit-standpunt

De chip heeft een sea-of-gates structuur en bestaat eenvoudig uit een groot aantal rijen transistoren zoals hieronder getekend. Er zijn evenveel rijen n- als p-type transistoren (zie figuur 37)



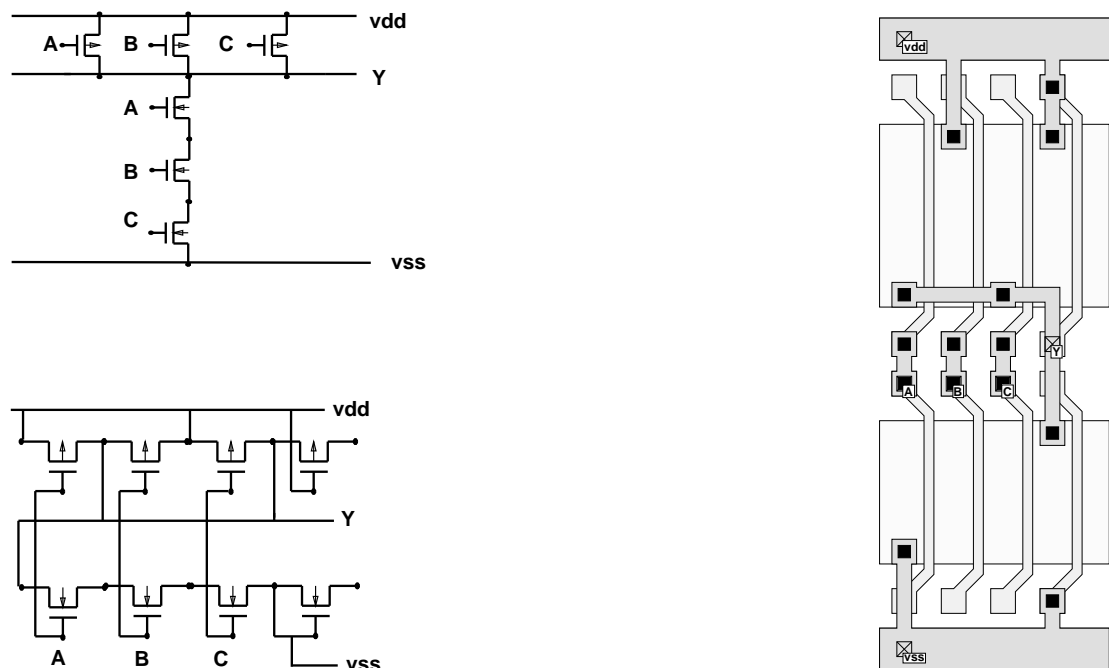
Figuur 37: Schematische voorstelling van een pmos- en een nmos-rij op de sea-of-gates chip

Elke rij op de chip telt ongeveer 1000 transistoren. Elk knooppunt in elke rij is toegankelijk vanuit de eerste metaallaag. De tweede metaallaag kan gebruikt worden indien kruisingen tussen metaalsporen nodig zijn. Naast een rij n-transistoren ligt altijd een rij p-transistoren.

De rijenstructuur lijkt onhandig maar is dit in de praktijk niet. In een schakeling zijn namelijk altijd rijen transistoren te ontdekken.

Als voorbeeld een 3-input nand schakeling (zie figuur 38).

In de structuur is duidelijk dat alle transistoren keurig op een rij gelegd kunnen worden en met een eenvoudig verbindingspatroon (in metaal) verbonden kunnen worden tot een 3-input nand. De in de structuur meest rechts getekende transistoren dienen alleen als scheiding van de schakeling met de rest van de transistoren. De herhaling in de y-richting van n- en p-rijen transistoren is als volgt: . . . . nppnnppnnppn . . . (zie ook figuur 39).



Figuur 38: Schema van de 3-input nand, de afbeelding hiervan op een rijenstructuur en de layout op het sea-of-gates image

Dit heeft o.a. als reden dat de p-transistoren op deze manier meer gegroepeerd zijn voor grotere nwell gebieden.

De basisrepetitie van 4 rijen ( $n_{ppn}$ ) komt op de chip 44 keer voor.

In de tabellen 1 en 2 staan enkele elektrische eigenschappen van het cmos proces.

masker	weerstand ( $\Omega/\square$ )
$n^+$ OD	55
$p^+$ OD	75
poly-Si	25-60
in	0.045
ins	0.03

Tabel 1: Nominale interconnectie weerstand cmos proces

### C.3 De kern van de chip, vanuit een layout-standpunt

In figuur 39 is een stukje layout gegeven van de basisstructuur. Duidelijk zijn de rijen transistoren zichtbaar. De p-transistor is breder dan de n-transistor. Het verschil in breedte is gelijk aan de ruimte die nodig is voor een extra metaalspoor en contactgat op de actieve gebieden van de p-transistoren. Tussen de twee rijen p-transistoren is plaats voor het voedingsspoor met daaronder een od gebied

	Oppervlakte capaciteit (aF/ $\mu\text{m}^2$ )	Rand capaciteit (aF/ $\mu\text{m}$ )	opmerking
$n^+$ OD naar substraat	190	310	junctie capaciteit
$p^+$ OD naar substraat	450	570	junctie capaciteit
poly naar substraat	49	54	
metaal1 naar substraat	25	45	
metaal 1 naar OD	49	54	
metaal1 naar poly	49	55	
metaal2 naar substraat	13	49	
metaal2 naar OD	15	53	
metaal2 naar poly	22	62	
metaal1 naar metaal2	43	81	

Tabel 2: Nominale interconnectie capaciteiten cmos proces

om de nwell te kunnen verbinden met de hoogste spanning in de schakeling (de voedingsspanning). Onder het aardspoor ligt een OD gebied om het p-substraat aan aarde te kunnen leggen.

Ook valt nog op dat alle afmetingen rechthoekig zijn, behalve de gate-aansluitingen waarin 45-graden lijnen zijn gebruikt. Dit is gedaan uit efficiency overwegingen.

Wat de layout betreft:

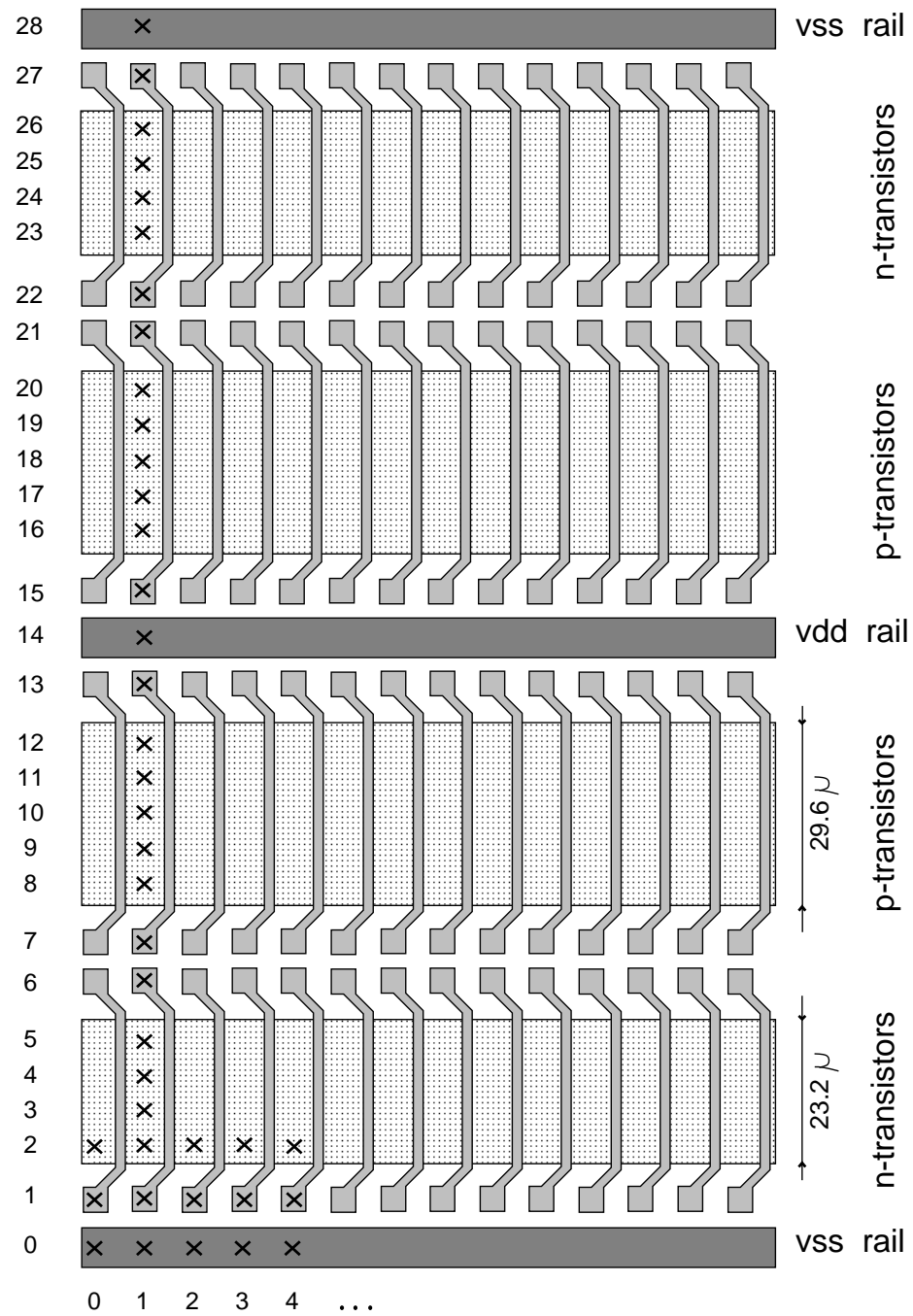
De steek in de x-richting is  $8.0 \mu\text{m}$ .

De steek in de y-richting is  $197.6 \mu\text{m}$ .

De layout-eenheid is 1 lambda ( $1 \text{ lambda} = 0.2 \mu\text{m}$ ).

#### C.4 Eisen en afspraken met betrekking tot de metallisatie

- Horizontaal lopend over het actieve gebied van een rij n-transistoren is plaats voor maximaal 4 metaalsporen.  
Horizontaal lopend over het actieve gebied van een rij p-transistoren is plaats voor maximaal 5 metaalsporen.
- Source- en drain gebieden en ook gate-aansluitingen) kunnen alleen maar verbonden worden met metaal1 (in). Verbinding van source, drain en gate met metaal2 (ins) kunnen alleen maar lopen via metaal1.
- Kontakten van metaal1 met source, drain of gate wordt gemaakt middels kontaktgaten. Deze kontaktgaten mogen alleen gedefinieerd worden op de grid-punten (zie figuur 39). Kontakt van metaal1 (in) en pmos source en draingebied wordt gemaakt met cop, kontakt van metaal1 (in) en nmos source en draingebied wordt gemaakt met con en kontakt van metaal1 (in) met de gates (poly-silicium) wordt gemaakt met cps.  
Overigens mogen deze drie kontaktlagen (con, cop en cps) bij het maken van de layout door elkaar worden gebruikt omdat bij het maken van het masker voor deze gaten ook geen onderscheid wordt gemaakt.



Figuur 39: Layout van het fishbone sea-of-gates image.

- Kontakten tussen metaal1 (in) en metaal2 (ins) worden gemaakt met COS.
- Een con- of cop- of cps-kontakt en een COS kontakt mogen niet boven elkaar worden gedefinieerd.
- Op de plaats van een gate kontaktaansluiting mag geen COS kontakt worden gedefinieerd.
- Metaal1 (in) loopt overwegend horizontaal, metaal2 (ins) loopt overwegend verticaal.

## C.5 Meetgegevens van de Sea-of-Gates chip

**Karakteristieken van de nmos en pmos transistoren** In figuur 41 is de gemeten drainstroom  $I_d$  uitgezet tegen de drain-sourcespanning  $V_{ds}$  met de gate-sourcespanning  $V_{gs}$  als parameter.

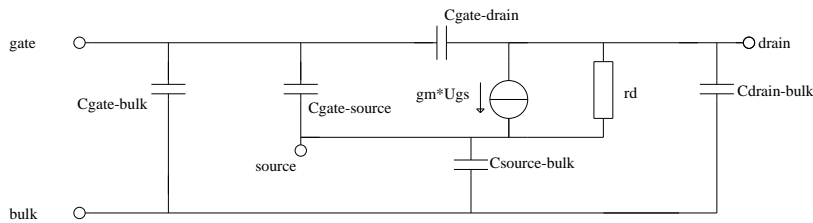
In figuur 42 is de gemeten drainstroom  $I_d$  uitgezet tegen de gatespanning  $V_{gs}$  met de drain-sourcespanning  $V_{ds}$  als parameter.

De maximale stroom voor een enkele nmos transistor ( $V_{gs} = V_{ds} = 5$  V) is ca. 5.5 mA. Voor een enkele pmos transistor is de maximale stroom ( $V_{gs} = V_{ds} = -5$  V) gelijk aan ca. 2.7 mA.

**Kleinsignaalmodel van de mos transistor** In figuur 40 is een eenvoudig kleinsignaalmodel gegeven van de mos transistor. De waarde van de verschillende capaciteiten zijn nog niet gemeten. Voor de drainstroom in verzadiging geldt ( $V_{ds} > V_{gs} > V_t$ , zie ook [7]):

$$I_d = \frac{\beta}{2} (V_{gs} - V_t)^2 (1 + \lambda V_{ds})$$

De gemeten waarde van de verschillende parameters zijn te vinden in tabel 3.



Figuur 40: Een eenvoudig kleinsignaalmodel van de mos transistor

Parameter	nmos	pmos
$V_t$	0.7 V	-1.2 V
$\beta$	0.65 mA/V <sup>2</sup>	0.13 mA/V <sup>2</sup>
$\lambda$	0.027 V <sup>-1</sup>	0.096 V <sup>-1</sup>
$r_d$	7.5 kΩ	10 kΩ

Tabel 3: Enkele gemeten parameters van de nmos en pmos transistoren

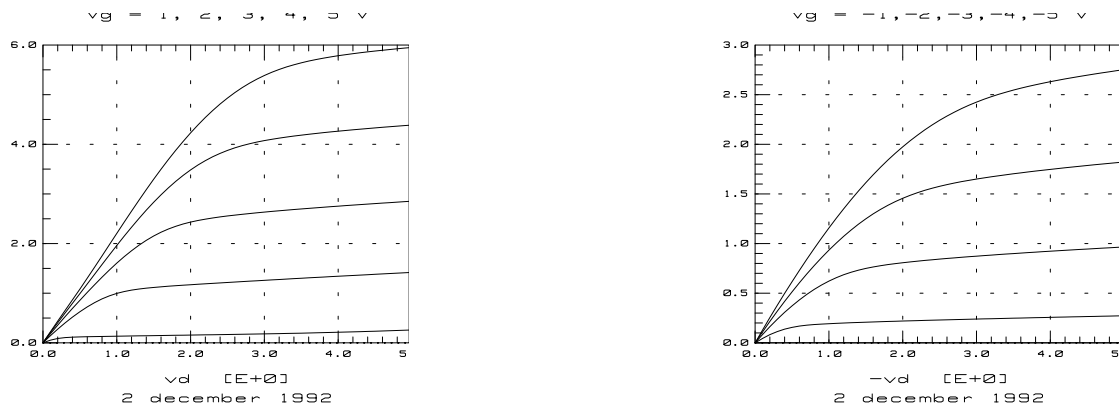
**Weerstandswaarden** In tabel 4 zijn de gemeten waarden te vinden van een aantal weerstanden en vergeleken met de berekende waarden uit gegevens van het cmos-proces.

Grootheid	Berekende waarde	Gemeten waarde
Contactgateweerstand		
<b>in-ins (cos)</b>	max. $0.2 \Omega$	$0.13 \Omega$
<b>in-ps (cps)</b>	max. $50 \Omega$	$22 \Omega$
Metaalweerstand		
metaal1 ( <b>in</b> )	$45 \text{ m}\Omega/\square$	$56 \text{ m}\Omega/\square$
metaal2 ( <b>ins</b> )	$30 \text{ m}\Omega/\square$	$26 \text{ m}\Omega/\square$
Polysiliciumgateweerstand		
nmos gate	$690 \Omega$	$700 \Omega$
pmos gate	$790 \Omega$	$950 \Omega$

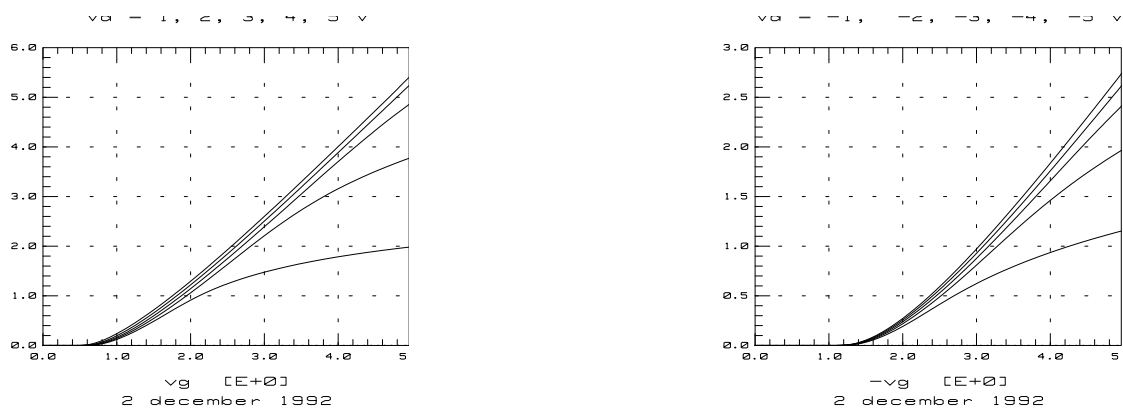
Tabel 4: Enkele weerstandwaarden van de sea-of-gates chip

## C.6 Structuur in de rand van de chip

Er zijn in de rand 144 aansluitposities, verdeeld in 8 groepen van elk 18. Elke groep van 18 aansluitingen heeft een voedingspositie en een aardpositie. De andere 16 zijn programmeerbaar d.m.v. metallisatiepatronen tot een aantal bufferfuncties, b.v.: digitaal in, digitaal uit, analoog in, analoog uit. Alle 8 aardposities in de rand zijn met elkaar verbonden (via de chiprand). Voor de programmering van de rand van een chip wordt verwezen naar een afzonderlijke handleiding. Voor de liefhebbers liggen er in de ruimten tussen de buffers in de rand nog polyweerstand. In elke tussenruimte liggen er 8 van elk  $2.5 \text{ k}\Omega$  (= 100 vierkanten). Tenslotte ligt in de rand linksonder een grote poly-rechthoek die nuttig kan zijn voor een capaciteit. Deze weerstanden en capaciteiten zijn echter niet te gebruiken in het project omdat ze vanwege de kwarten-indeling van de chip niet zijn aan te sluiten.



Figuur 41: De gemeten drainstroom van een nmos en pmos transistor uitgezet tegen de drain-sourcespanning met de gate-sourcespanning als parameter



Figuur 42: De gemeten drainstroom van een nmos en pmos transistor uitgezet tegen de gate-sourcespanning met de drain-sourcespanning als parameter



## D De SoG Cellenbibliotheek (oplib)

### D.1 Inleiding

In dit hoofdstuk zal een korte beschrijving gegeven worden van de bibliotheekcellen die beschikbaar zijn voor het project. De beschrijving bestaat voor elke cel uit:

- Functie
- Terminal aansluitingen
- IEC-symbool
- Waarheidstabel
- Timing parameters
- Equivalent chip oppervlak
- Fanout

Er zijn vier soorten timing parameters:

$T_{PLH}$  en  $T_{PHL}$  De vertragingstijd van de cel bij eenheidsbelasting (0.12 pF)

$\Delta T_{PLH}$  en  $\Delta T_{PHL}$  De vertragings-coëfficiënt bij capacatieve belasting

$C_{in}$  De ingangscapaciteit

$T_{su}$  en  $T_{hold}$  Setup- en hold-tijden van de flipflops

De vertragings-tijden en -coëfficiënten worden afzonderlijk gedefinieerd voor een opgaande (LH) en neergaande (HL) flank van het uitgangssignaal. De totale vertragingstijd wordt berekend met de formule:

$$T_{P_{tot}} = T_P + \Delta T_P (C_{load} - C_{unit})$$

waarbij  $C_{load}$  de belastingscapaciteit voorstelt en  $C_{unit}$  de eenheidsbelasting.

Deze belastingscapaciteit is de som van de ingangscapaciteit van de aangestuurde cellen plus de capaciteit van de verbindingsdraden.

De eenheid van (equivalent) chip oppervlakte is gedefinieerd als het kleinst mogelijke stukje van het fishbone image en bestaat uit één nmos en één pmos transistor. De grootte hiervan is ongeveer 800  $\mu\text{m}^2$ .

De fanout is de belasting waarboven de totale vertragingstijd *afneemt* als een buffer wordt toegevoegd.

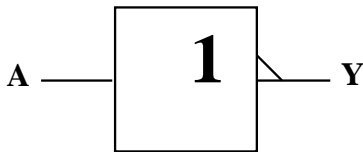
D.2 De cellen

D.2.1 iv110

Functie: Inverter

Terminals: (A, Y, vss, vdd)

IEC-symbol:



Waarheidstabel:

A	Y
L	H
H	L

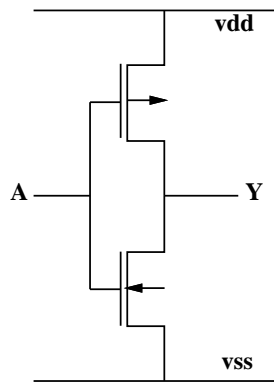
Timing parameters:

Parameter	From	To	Min	Typ	Max	Unit
$T_{PLH}$	A	Y	0.13	0.19	0.35	ns
$T_{PHL}$	A	Y	0.15	0.23	0.40	ns
$\Delta T_{PLH}$	A	Y	-	-	1.1	ns/pF
$\Delta T_{PHL}$	A	Y	-	-	0.8	ns/pF
$C_{in}$	A	vss	-	0.12	0.18	pF

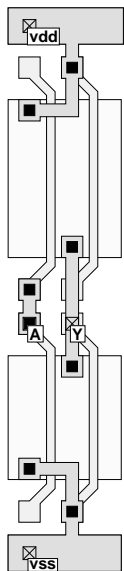
Equivalent chip oppervlak: 2

Fanout: 3.0 pF

Circuit:



Layout:

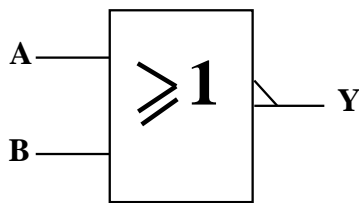


D.2.2 no210

Functie: 2-input nor

Terminals: (A, B, Y, vss, vdd)

IEC-symbol:



Waarheidstabel:

A	B	Y
L	L	H
-	H	L
H	-	L

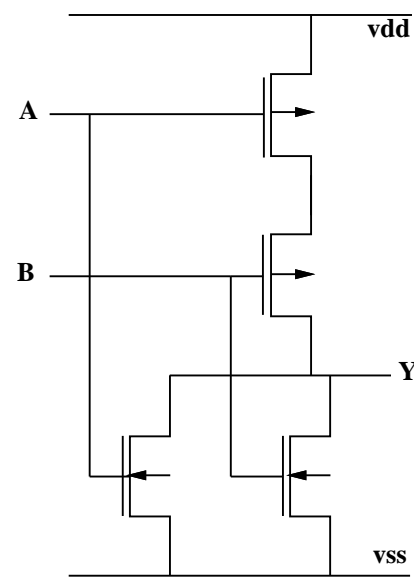
Timing parameters:

Parameter	From	To	Min	Typ	Max	Unit
$T_{PLH}$	A	Y	0.23	0.34	0.59	ns
$T_{PHL}$	A	Y	0.10	0.25	0.38	ns
$T_{PLH}$	B	Y	0.21	0.32	0.59	ns
$T_{PHL}$	B	Y	0.10	0.22	0.40	ns
$\Delta T_{PLH}$	any	Y	-	-	1.8	ns/pF
$\Delta T_{PHL}$	any	Y	-	-	0.8	ns/pF
$C_{in}$	any	vss	-	0.12	0.18	pF

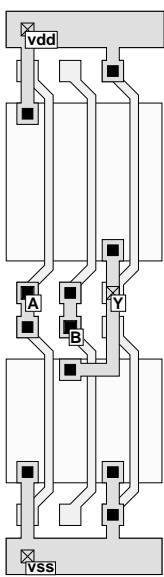
Equivalent chip oppervlak: 3

Fanout: 2.4 pF

Circuit:



Layout:

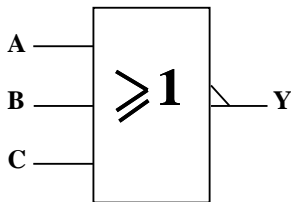


D.2.3 no310

Functie: 3-input nor

Terminals: (A, B, C, Y, vss, vdd)

IEC-symbol:



Waarheidstabel:

A	B	C	Y
L	L	L	H
-	-	H	L
-	H	-	L
H	-	-	L

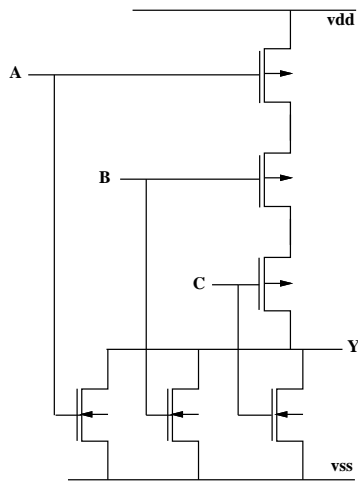
Timing parameters:

Parameter	From	To	Min	Typ	Max	Unit
$T_{PLH}$	A	Y	0.36	0.54	1.1	ns
$T_{PHL}$	A	Y	0.08	0.27	0.47	ns
$T_{PLH}$	B	Y	0.36	0.53	1.1	ns
$T_{PHL}$	B	Y	0.08	0.27	0.47	ns
$T_{PLH}$	C	Y	0.30	0.46	1.1	ns
$T_{PHL}$	C	Y	0.08	0.24	0.41	ns
$\Delta T_{PLH}$	any	Y	-	-	2.4	ns/pF
$\Delta T_{PHL}$	any	Y	-	-	0.9	ns/pF
$C_{in}$	any	vss	-	0.12	0.18	pF

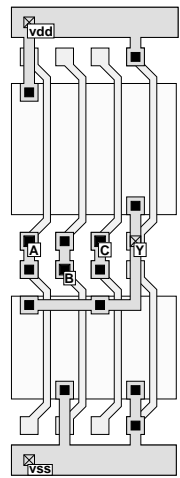
Equivalent chip oppervlak: 4

Fanout: 1.9 pF

Circuit:



Layout:

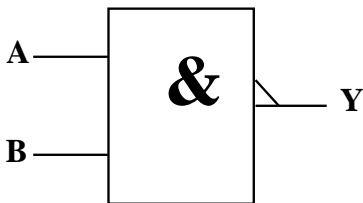


D.2.4 na210

Functie: 2-input nand

Terminals: (A, B, Y, vss, vdd)

IEC-symbol:



Waarheidstabel:

A	B	Y
-	L	H
L	-	H
H	H	L

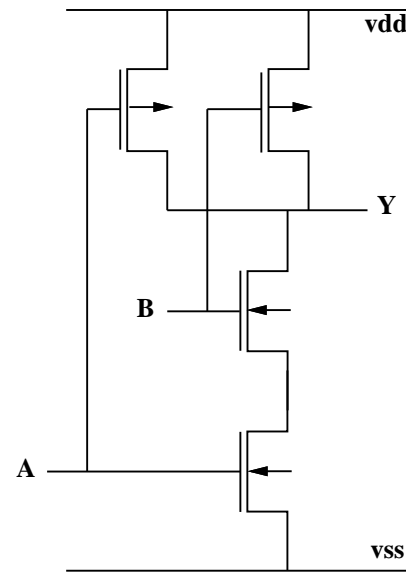
Timing parameters:

Parameter	From	To	Min	Typ	Max	Unit
$T_{PLH}$	A	Y	0.08	0.33	0.50	ns
$T_{PHL}$	A	Y	0.19	0.29	0.52	ns
$T_{PLH}$	B	Y	0.08	0.20	0.36	ns
$T_{PHL}$	B	Y	0.20	0.30	0.52	ns
$\Delta T_{PLH}$	any	Y	-	-	1.0	ns/pF
$\Delta T_{PHL}$	any	Y	-	-	1.1	ns/pF
$C_{in}$	any	vss	-	0.12	0.18	pF

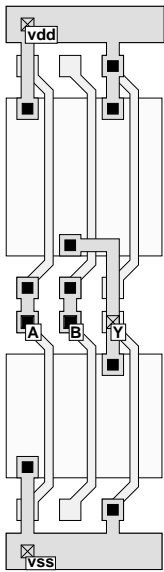
Equivalent chip oppervlak: 3

Fanout: 2.9 pF

Circuit:



Layout:

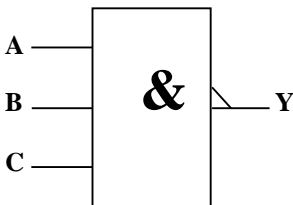


D.2.5 na310

Functie: 3-input nand

Terminals: (A, B, C, Y, vss, vdd)

IEC-symbol:



Waarheidstabel:

A	B	C	Y
-	-	L	H
-	L	-	H
L	-	-	H
H	H	H	L

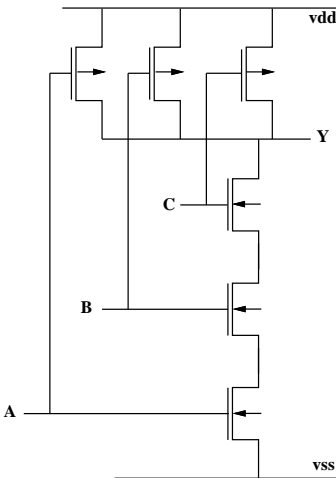
Timing parameters:

Parameter	From	To	Min	Typ	Max	Unit
$T_{PLH}$	A	Y	0.06	0.37	0.61	ns
$T_{PHL}$	A	Y	0.26	0.39	0.77	ns
$T_{PLH}$	B	Y	0.06	0.35	0.58	ns
$T_{PHL}$	B	Y	0.27	0.40	0.77	ns
$T_{PLH}$	C	Y	0.06	0.21	0.37	ns
$T_{PHL}$	C	Y	0.26	0.39	0.77	ns
$\Delta T_{PLH}$	any	Y	-	-	0.9	ns/pF
$\Delta T_{PHL}$	any	Y	-	-	1.4	ns/pF
$C_{in}$	any	vss	-	0.12	0.18	pF

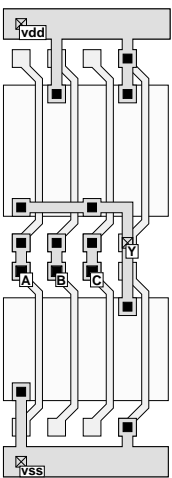
Equivalent chip oppervlak: 4

Fanout: 2.5 pF

Circuit:



Layout:

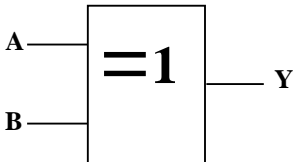


D.2.6 ex210

Functie: Exclusive or

Terminals: (A, B, Y, vss, vdd)

IEC-symbol:



Waarheidstabel:

A	B	Y
L	L	L
L	H	H
H	L	H
H	H	L

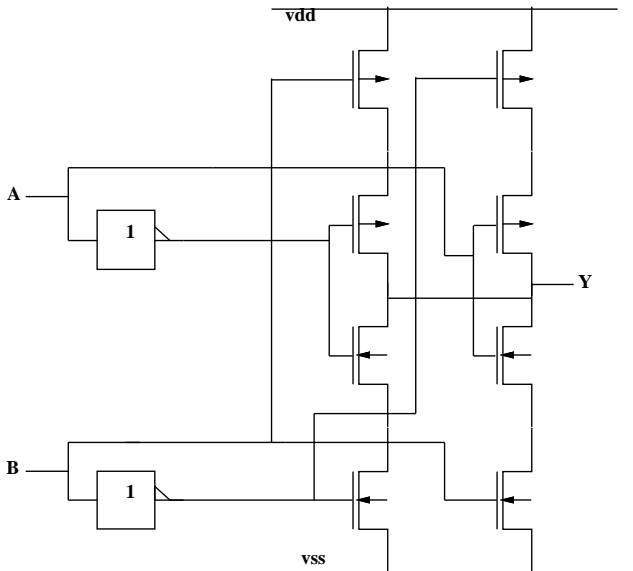
Timing parameters:

Parameter	From	To	Min	Typ	Max	Unit
$T_{PLH}$	any	Y	0.22	0.69	1.6	ns
$T_{PHL}$	any	Y	0.24	0.41	0.68	ns
$\Delta T_{PLH}$	any	Y	-	-	1.4	ns/pF
$\Delta T_{PHL}$	any	Y	-	-	1.1	ns/pF
$C_{in}$	any	vss	-	0.24	0.36	pF

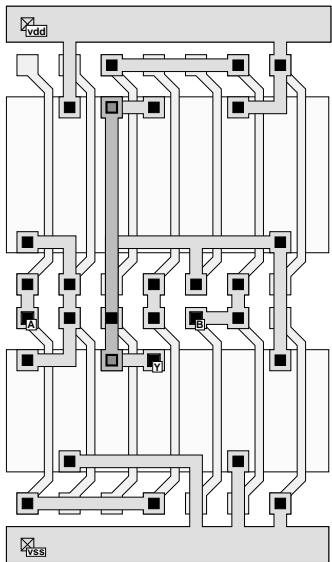
Equivalent chip oppervlak: 7

Fanout: 2.4 pF

Circuit:



Layout:

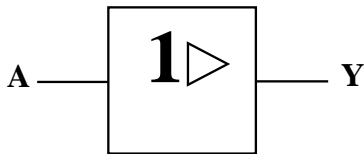


**D.2.7 buf40**

Functie: Buffer (4x-drive)

Terminals: (A, Y, vss, vdd)

IEC-symbol:



Waarheidstabel:

A	Y
L	L
H	H

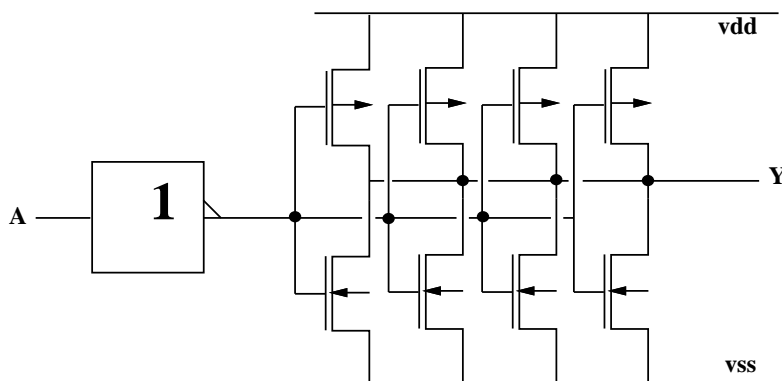
Timing parameters:

Parameter	From	To	Min	Typ	Max	Unit
$T_{PLH}$	A	Y	0.46	0.69	1.0	ns
$T_{PHL}$	A	Y	0.56	0.84	1.3	ns
$\Delta T_{PLH}$	A	Y	-	-	0.3	ns/pF
$\Delta T_{PHL}$	A	Y	-	-	0.4	ns/pF
$C_{in}$	A	vss	-	0.12	0.18	pF

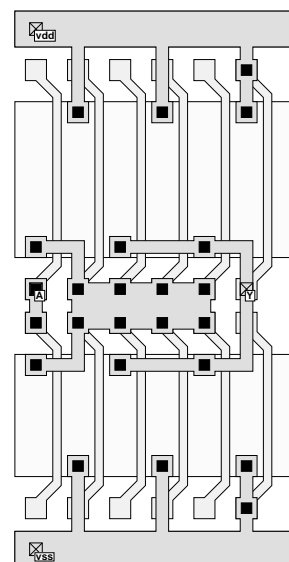
Equivalent chip oppervlak: 6

Fanout: 12.0 pF

Circuit:



Layout:



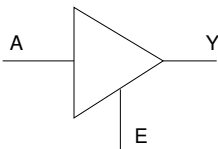


D.2.8 tbuf10

Functie: Tri-state buffer

Terminals: (A, E, Y, vss, vdd)

IEC-symbol:



Waarheidstabel:

A	E	Y
-	L	Z
L	H	L
H	H	H

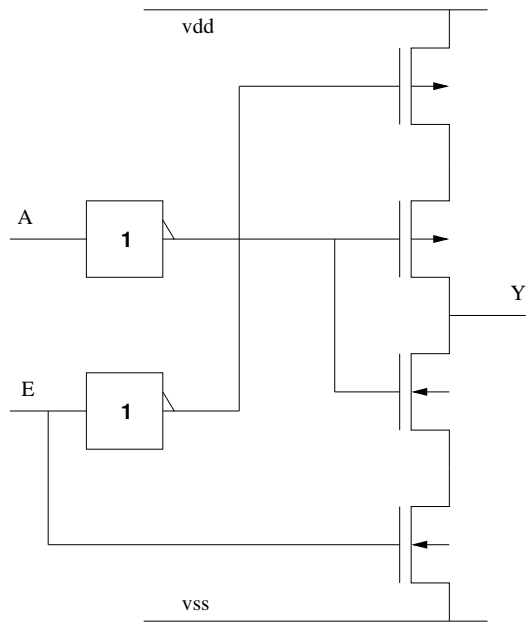
Timing parameters:

Parameter	From	To	Min	Typ	Max	Unit
$T_{PLH}$	A	Y	0.36	0.55	0.99	ns
$T_{PHL}$	A	Y	0.33	0.49	0.87	ns
$\Delta T_{PLH}$	A	Y	-	-	1.8	ns/pF
$\Delta T_{PHL}$	A	Y	-	-	1.1	ns/pF
$C_{in}$	A	vss	-	0.12	0.18	pF

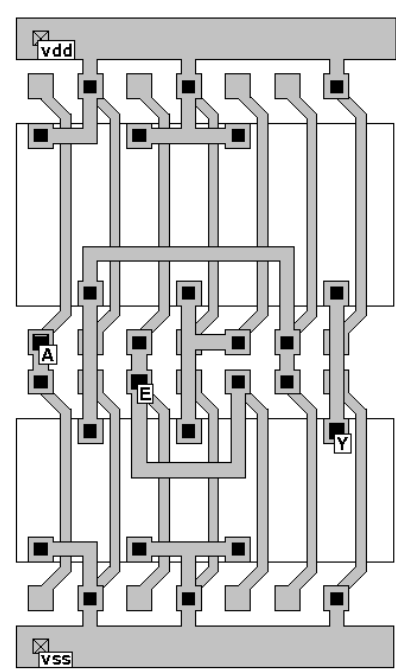
Equivalent chip oppervlak: 7

Fanout: 2.5 pF

Circuit:



Layout:

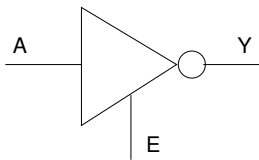


**D.2.9 tinv10**

Functie: Inverting tri-state buffer

Terminals: (A, E, Y, vss, vdd)

IEC-symbol:



Waarheidstabel:

A	E	Y
-	L	Z
L	H	H
H	H	L

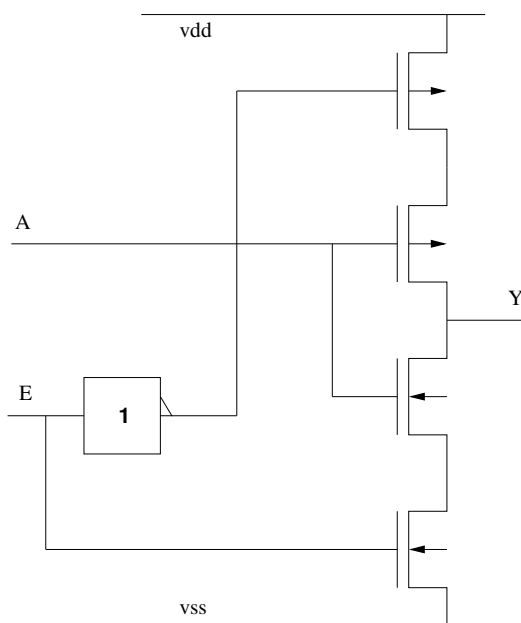
Timing parameters:

Parameter	From	To	Min	Typ	Max	Unit
$T_{PLH}$	A	Y	0.21	0.32	0.59	ns
$T_{PHL}$	A	Y	0.20	0.30	0.52	ns
$\Delta T_{PLH}$	A	Y	-	-	1.8	ns/pF
$\Delta T_{PHL}$	A	Y	-	-	1.1	ns/pF
$C_{in}$	A	vss	-	0.12	0.18	pF

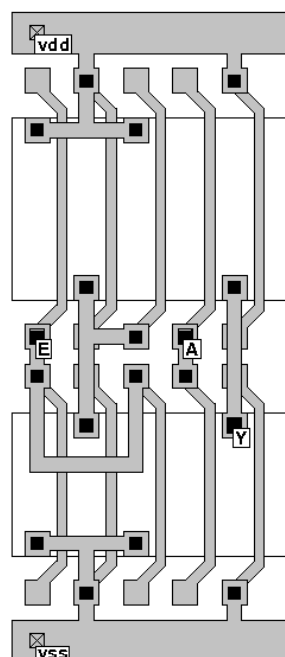
Equivalent chip oppervlak: 5

Fanout: 2.5 pF

Circuit:



Layout:

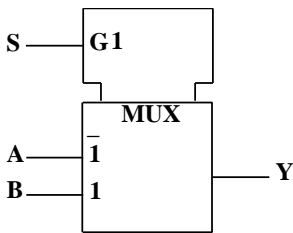


D.2.10 mu111

Functie: 2-line-to-1-line data selector/multiplexer

Terminals: (A, B, S, Y, vss, vdd)

IEC-symbol:



Waarheidstabel:

S	A	B	Y
L	L	-	L
L	H	-	H
H	-	L	L
H	-	H	H

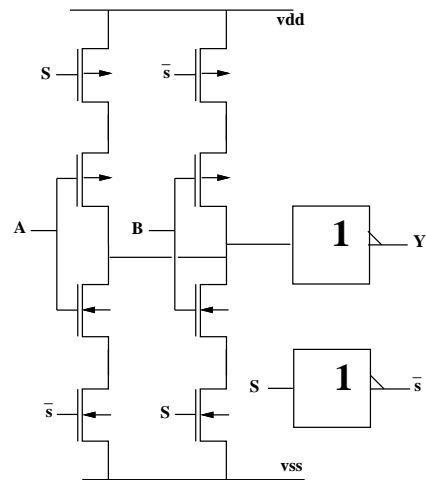
Timing parameters:

Parameter	From	To	Min	Typ	Max	Unit
$T_{PLH}$	A, B	Y	0.31	0.50	0.81	ns
$T_{PHL}$	A, B	Y	0.37	0.60	0.98	ns
$T_{PLH}$	S	Y	0.37	0.60	0.95	ns
$T_{PHL}$	S	Y	0.37	0.65	1.1	ns
$\Delta T_{PLH}$	any	Y	-	-	1.2	ns/pF
$\Delta T_{PHL}$	any	Y	-	-	1.0	ns/pF
$C_{in}$	A, B	vss	-	0.12	0.18	pF
$C_{in}$	S	vss	-	0.24	0.36	pF

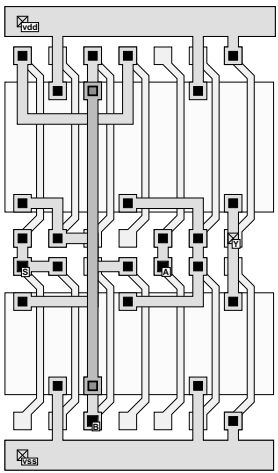
Equivalent chip oppervlak: 7

Fanout: 3.0 pF

Circuit:



Layout:

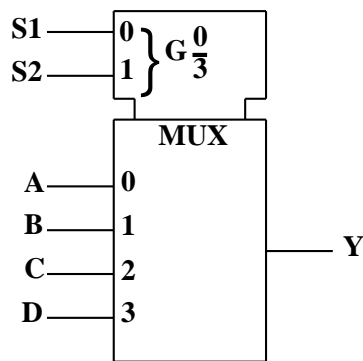


**D.2.11 mu210**

Functie: 4-line-to-1-line data selector/multiplexer

Terminals: (S1, S2, A, B, C, D, Y, vss, vdd)

IEC-symbol:



Waarheidstabel:

S2	S1	A	B	C	D	Y
L	L	L	-	-	-	L
L	L	H	-	-	-	H
L	H	-	L	-	-	L
L	H	-	H	-	-	H
H	L	-	-	L	-	L
H	L	-	-	H	-	H
H	H	-	-	-	L	L
H	H	-	-	-	H	H

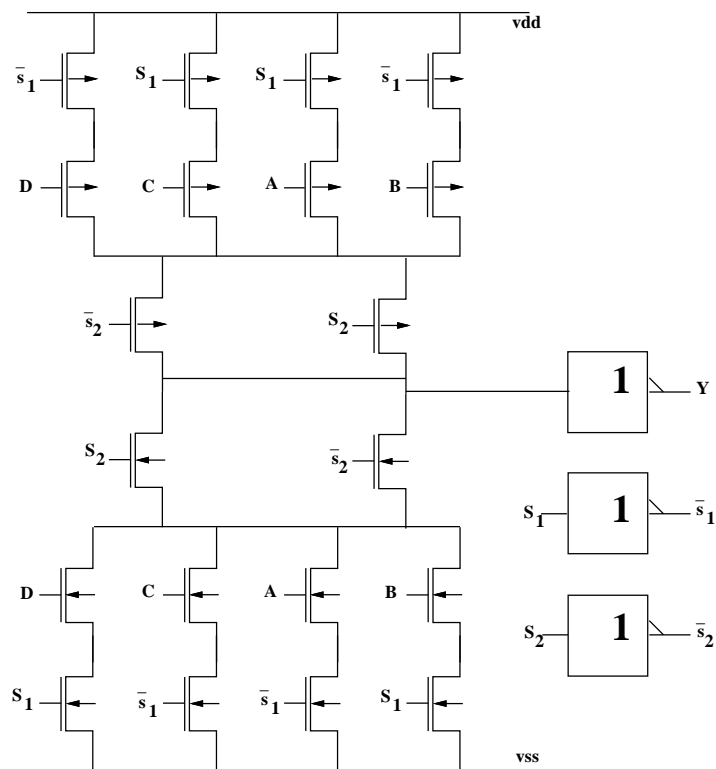
Timing parameters:

Parameter	From	To	Min	Typ	Max	Unit
$T_{PLH}$	A, B, C, D, S1	Y	0.88	1.3	2.0	ns
$T_{PHL}$	A, B, C, D, S1	Y	1.4	2.1	3.2	ns
$T_{PLH}$	S2	Y	0.77	1.2	1.7	ns
$T_{PHL}$	S2	Y	0.57	0.86	1.3	ns
$\Delta T_{PLH}$	any	Y	-	-	1.3	ns/pF
$\Delta T_{PHL}$	any	Y	-	-	1.4	ns/pF
$C_{in}$	A, B, C, D	vss	-	0.12	0.18	pF
$C_{in}$	S1	vss	-	0.36	0.54	pF
$C_{in}$	S2	vss	-	0.24	0.36	pF

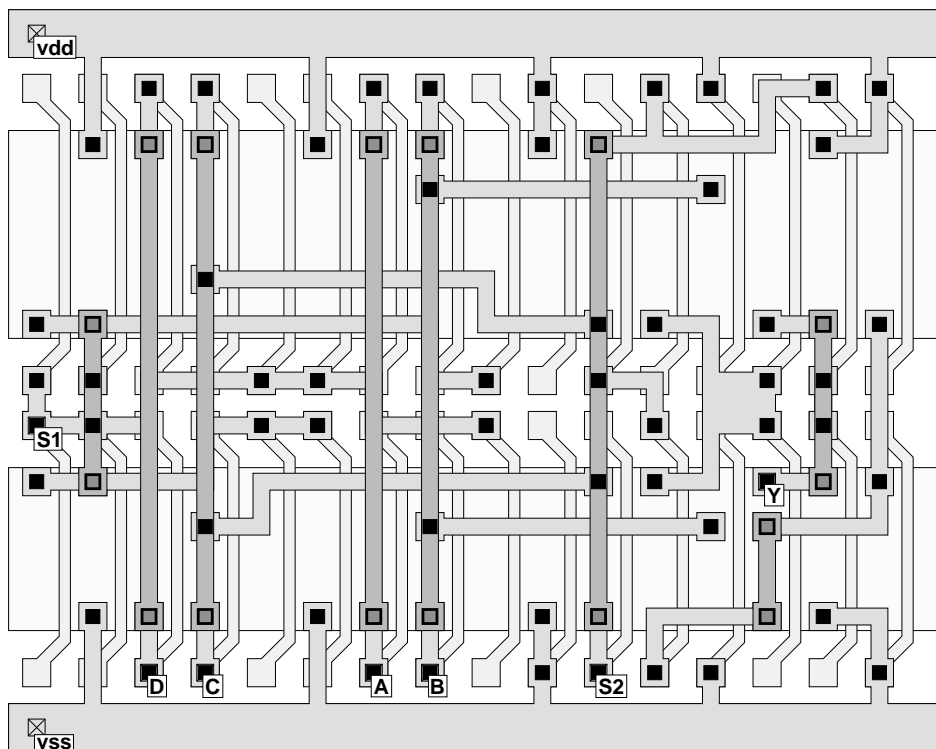
Equivalent chip oppervlak: 16

Fanout: 3.0 pF

Circuit:



Layout:

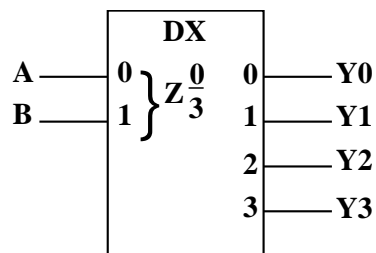


**D.2.12 de211**

Functie: 2-to-4 decoder/demultiplexer

Terminals: (A, B, Y0, Y1, Y2, Y3, vss, vdd)

IEC-symbol:



Waarheidstabel:

B	A	Y0	Y1	Y2	Y3
L	L	H	L	L	L
L	H	L	H	L	L
H	L	L	L	H	L
H	H	L	L	L	H

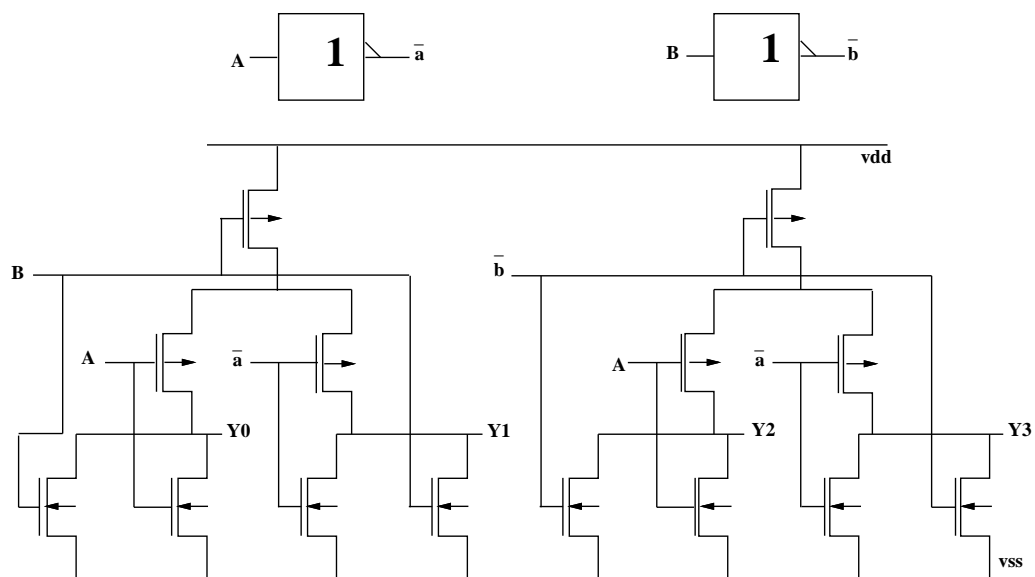
Timing parameters:

Parameter	From	To	Min	Typ	Max	Unit
$T_{PLH}$	A	Y0,Y2	0.23	0.34	0.51	ns
$T_{PHL}$	A	Y0,Y2	0.17	0.25	0.38	ns
$T_{PLH}$	A	Y1,Y3	0.41	0.62	0.93	ns
$T_{PHL}$	A	Y1,Y3	0.36	0.54	0.81	ns
$T_{PLH}$	B	Y0,Y1	0.20	0.30	0.45	ns
$T_{PHL}$	B	Y0,Y1	0.15	0.22	0.33	ns
$T_{PLH}$	B	Y2,Y3	0.38	0.57	0.87	ns
$T_{PHL}$	B	Y2,Y3	0.34	0.51	0.76	ns
$\Delta T_{PLH}$	any	any	-	-	1.8	ns/pF
$\Delta T_{PHL}$	any	any	-	-	0.8	ns/pF
$C_{in}$	A	vss	-	0.36	0.54	pF
$C_{in}$	B	vss	-	0.30	0.45	pF

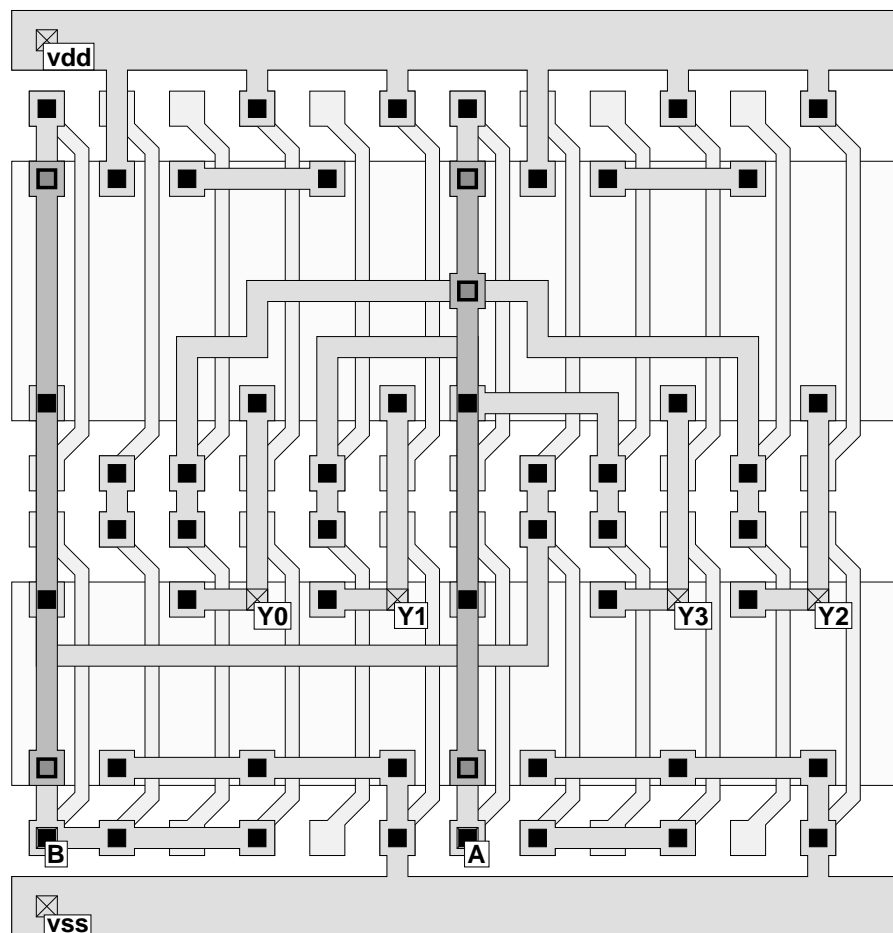
Equivalent chip oppervlak: 12

Fanout: 2.4 pF

Circuit:



Layout:

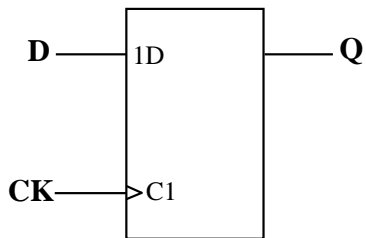


**D.2.13 dfn10**

Functie: D-type positive edge-triggered flipflop

Terminals: (D, CK, Q, vss, vdd)

IEC-symbol:



Waarheidstabel:

D	CK	Q
L	↑	L
H	↑	H

Timing parameters:

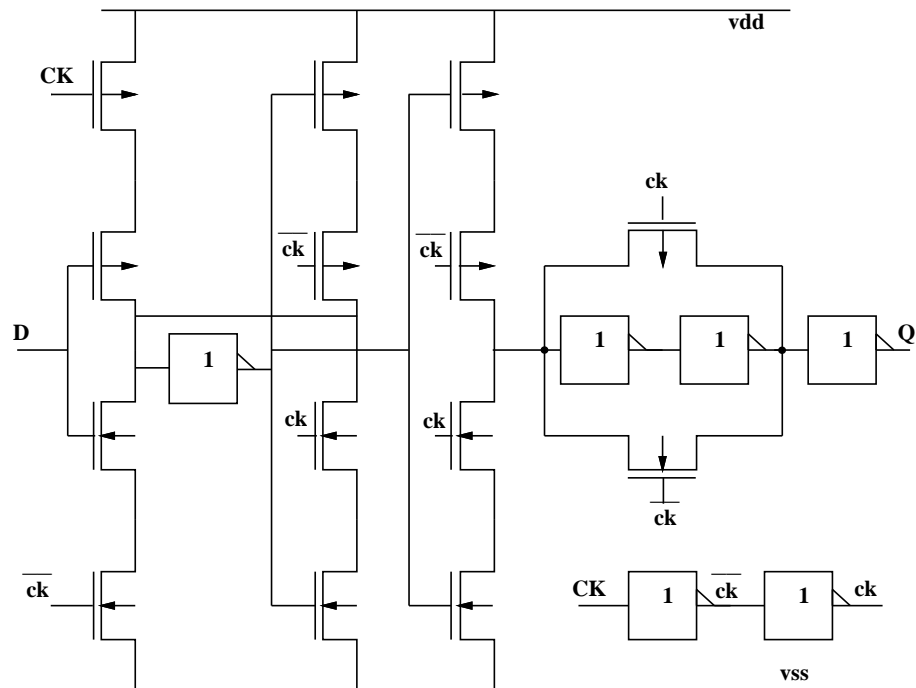
Parameter	From	To	Min	Typ	Max	Unit
$T_{PLH}$	CK	Q	1.4	2.1	3.2	ns
$T_{PHL}$	CK	Q	1.3	1.9	2.9	ns
$\Delta T_{PLH}$	CK	Q	-	-	1.1	ns/pF
$\Delta T_{PHL}$	CK	Q	-	-	0.8	ns/pF
$T_{su}$	D	CK	-	-	1.4	ns
$T_{hold}$	D	CK	-	-	0.9	ns
$C_{in}$	D	vss	-	0.12	0.18	pF
$C_{in}$	CK	vss	-	0.18	0.27	pF

Equivalent chip oppervlak: 17

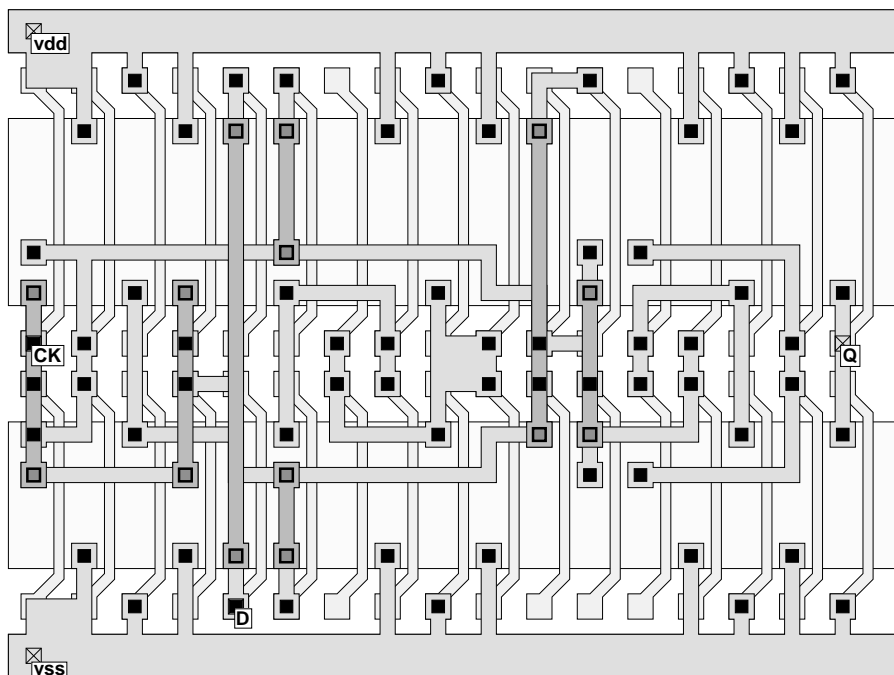
Fanout: 3.0 pF



Circuit:



Layout:

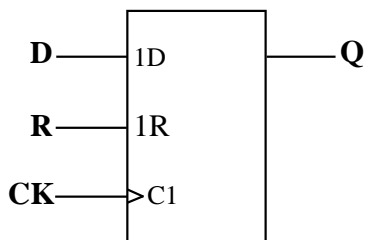


**D.2.14 dfr11**

Functie: D-type positive edge-triggered flipflop with synchronous reset

Terminals: (D, R, CK, Q, vss, vdd)

IEC-symbol:



Waarheidstabel:

R	D	CK	Q
L	L	↑	L
L	H	↑	H
H	-	↑	L

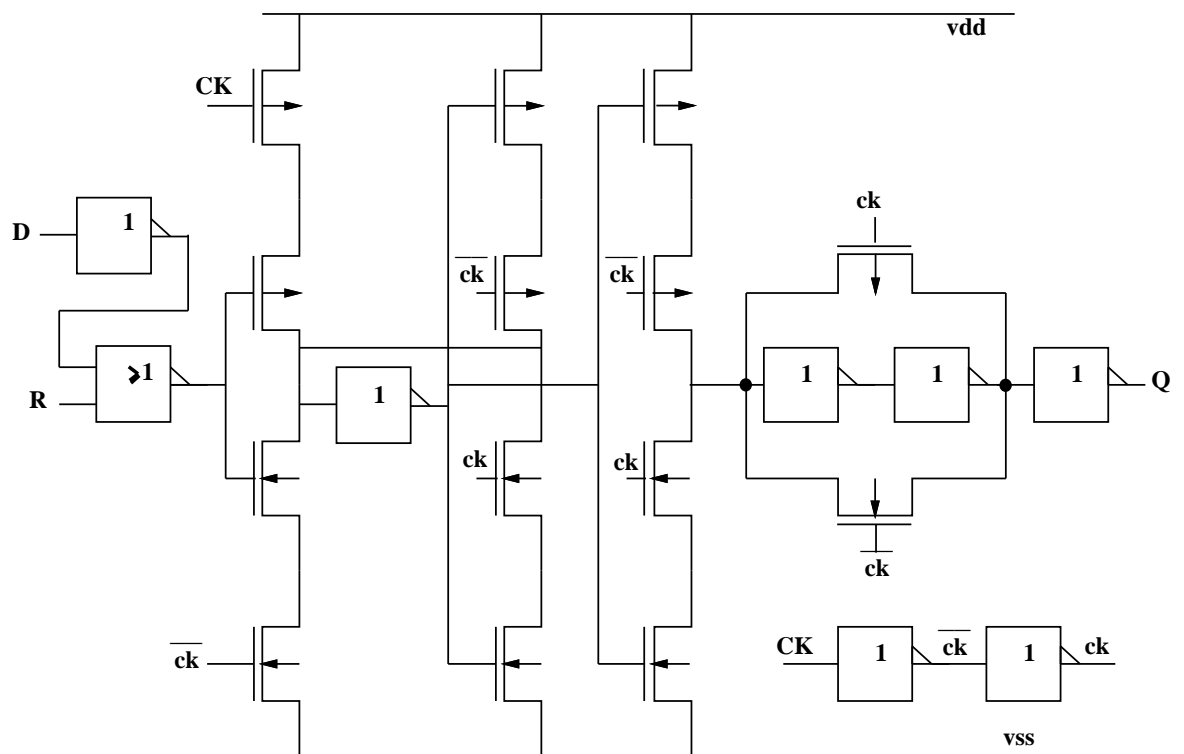
Timing parameters:

Parameter	From	To	Min	Typ	Max	Unit
$T_{PLH}$	CK	Q	1.4	2.1	3.2	ns
$T_{PHL}$	CK	Q	1.3	1.9	2.9	ns
$\Delta T_{PLH}$	CK	Q	-	-	1.1	ns/pF
$\Delta T_{PHL}$	CK	Q	-	-	0.8	ns/pF
$T_{su}$	D,R	CK	-	-	1.4	ns
$T_{hold}$	D,R	CK	-	-	0.9	ns
$C_{in}$	D,R	vss	-	0.12	0.18	pF
$C_{in}$	CK	vss	-	0.18	0.27	pF

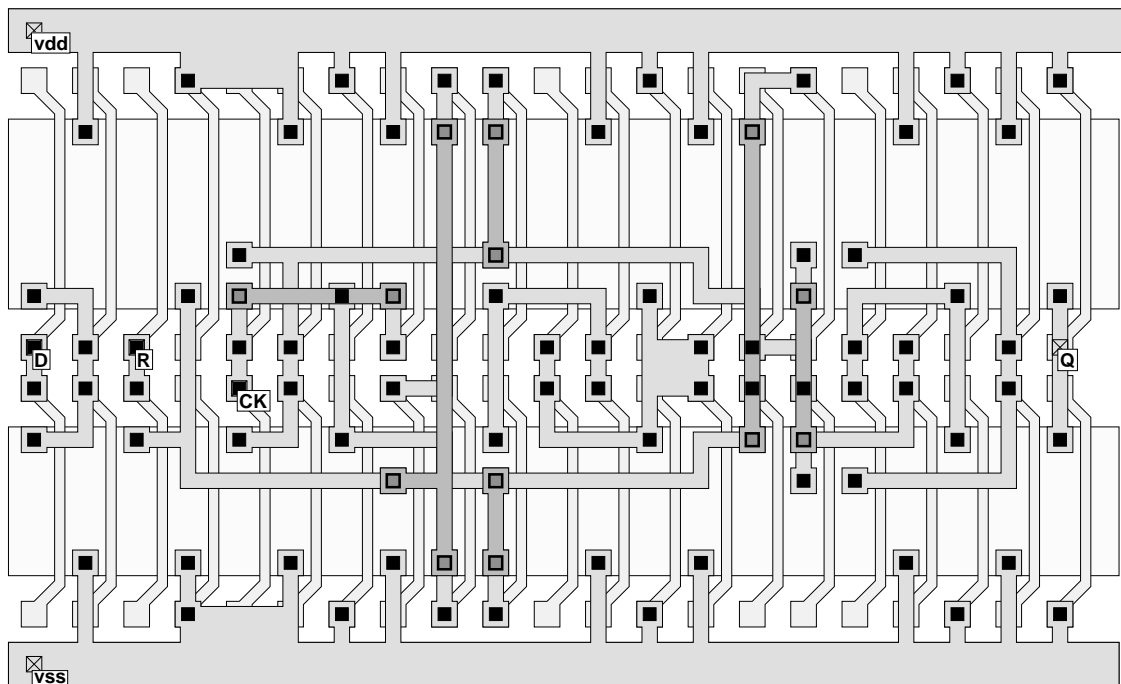
Equivalent chip oppervlak: 21

Fanout: 3.0 pF

Circuit:



Layout:

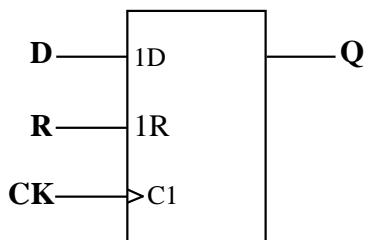


**D.2.15 dfa11**

Functie: D-type positive edge-triggered flipflop with asynchronous reset

Terminals: (D, R, CK, Q, vss, vdd)

IEC-symbol:



Waarheidstabel:

R	D	CK	Q
L	L	↑	L
L	H	↑	H
H	-	-	L

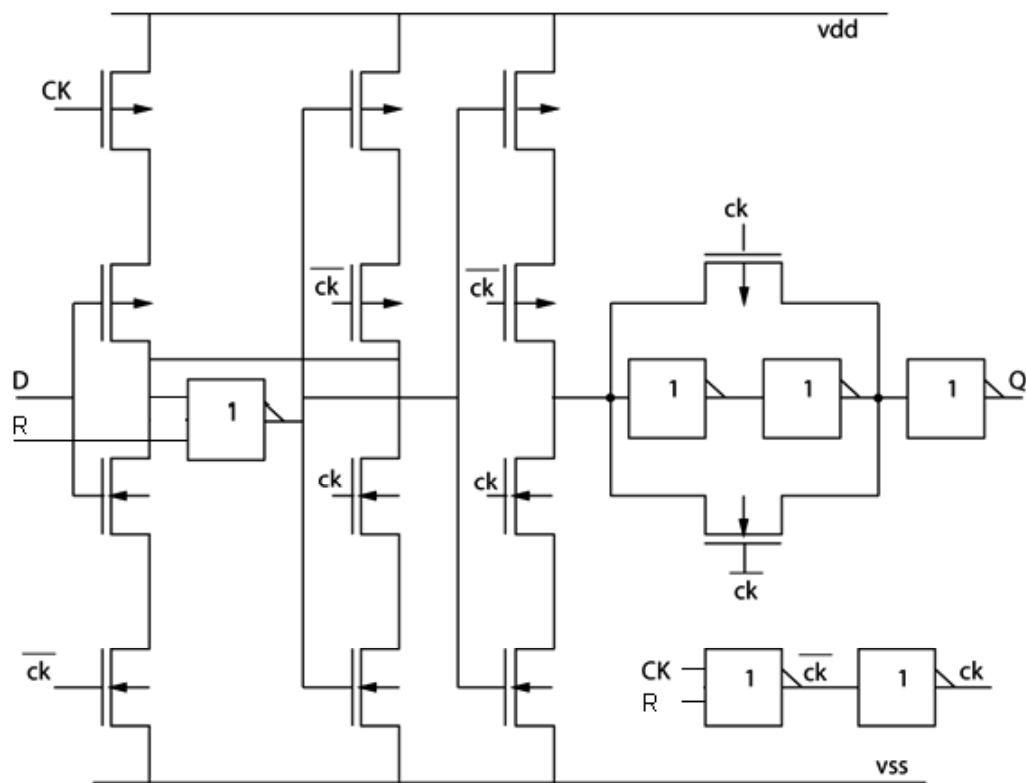
Timing parameters:

Parameter	From	To	Min	Typ	Max	Unit
$T_{PLH}$	CK	Q	1.4	2.1	3.2	ns
$T_{PHL}$	CK	Q	1.3	1.9	2.9	ns
$\Delta T_{PLH}$	CK	Q	-	-	1.1	ns/pF
$\Delta T_{PHL}$	CK	Q	-	-	0.8	ns/pF
$T_{su}$	D,R	CK	-	-	1.4	ns
$T_{hold}$	D,R	CK	-	-	0.9	ns
$C_{in}$	D,R	vss	-	0.12	0.18	pF
$C_{in}$	CK	vss	-	0.18	0.27	pF

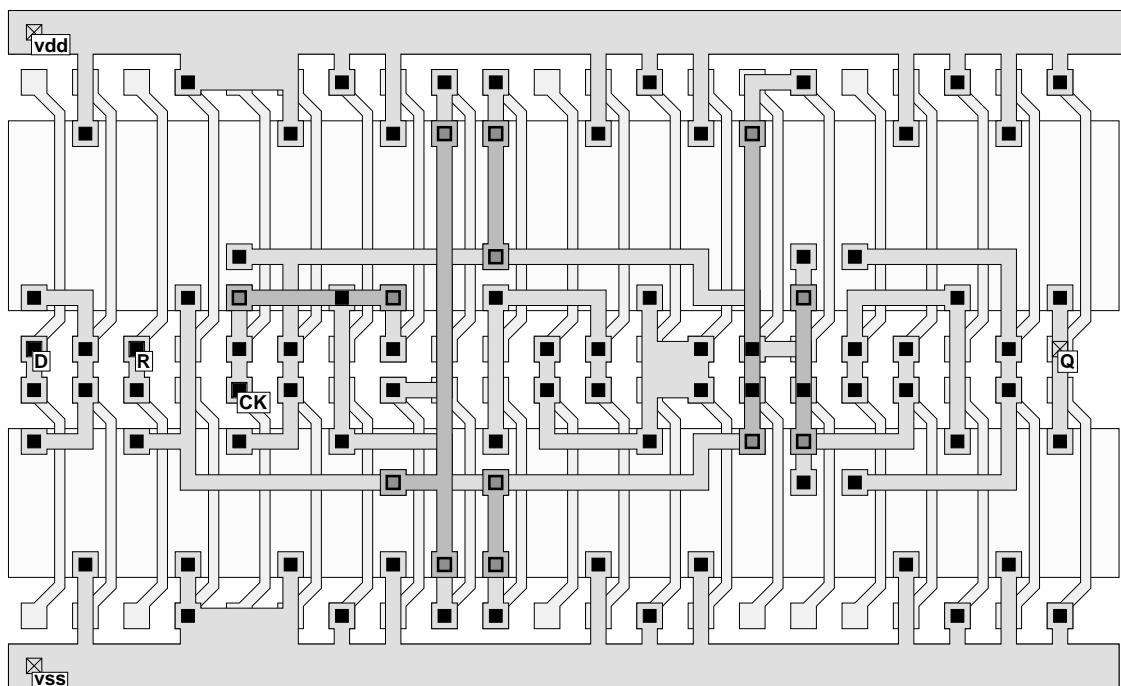
Equivalent chip oppervlak: 19

Fanout: 3.0 pF

Circuit:



Layout:

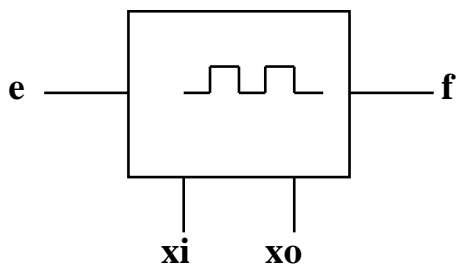


**D.2.16 osc10**

Functie: Crystal oscillator with enable

Terminals: (e, f, xi, xo, vss, vdd)

IEC-symbol:



Frequentie bereik: 1 MHz t/m 20 MHz

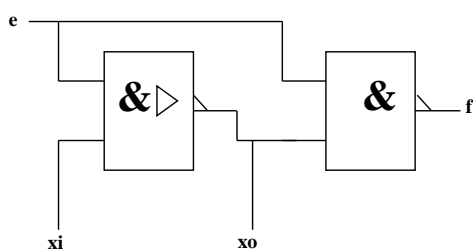
Equivalent chip oppervlak: 16

Beschrijving:

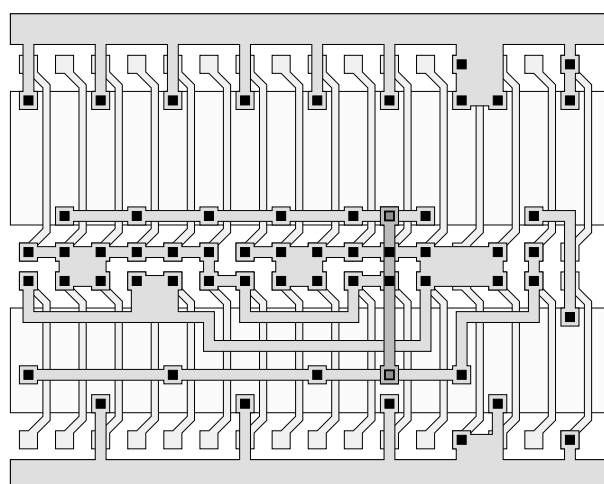
xi en xo zijn de aansluitingen voor het kristal. xi moet verbonden worden met een port van het type input, xo dient verbonden te worden met een port van het type inout.

Als e (enable) hoog is, is de oscillator actief. Als e laag is, is de uitgang f hoog.

Circuit:



Layout:



**D.2.17 ln3x3**

Functie: NMOS compoundtransistor, basiscel voor analoge schakelingen

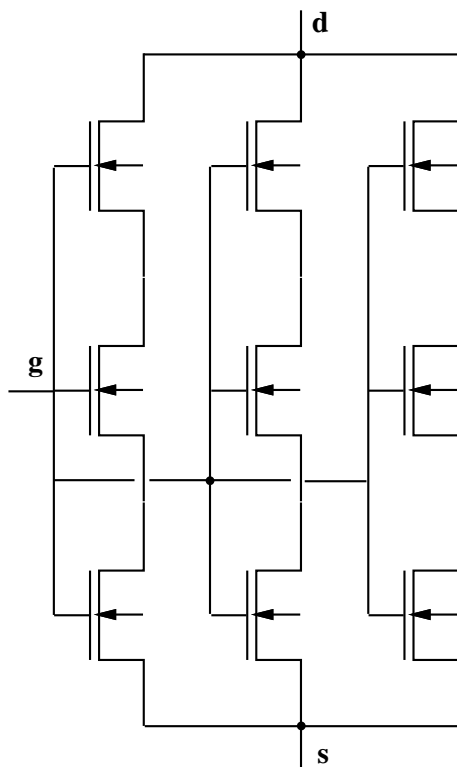
Terminals: (g, d, s, vss, vdd)

Equivalent chip oppervlak: 10

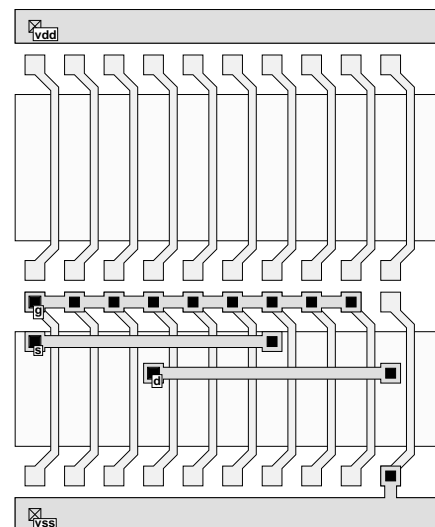
Beschrijving:

De NMOS compoundtransistor ln3x3 bestaat uit 9 NMOS basistransistoren die in een matrix van 3x3 geschakeld zijn.

Circuit:



Layout:



### D.2.18 lp3x3

Functie: PMOS compoundtransistor, basiscel voor analoge schakelingen

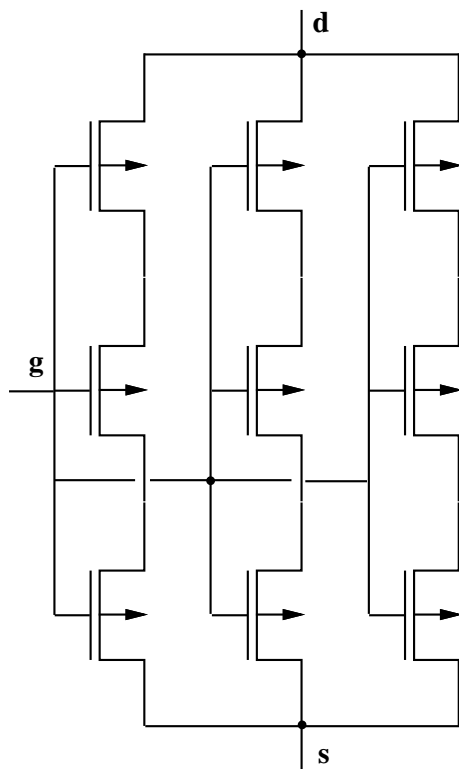
Terminals: (g, d, s, vss, vdd)

Equivalent chip oppervlak: 10

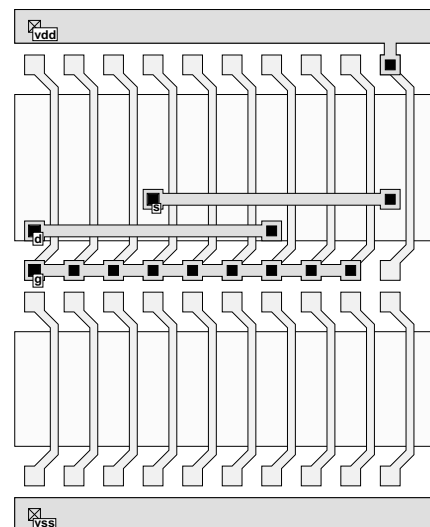
Beschrijving:

De PMOS compoundtransistor lp3x3 bestaat uit 9 PMOS basistransistoren die in een matrix van 3x3 geschakeld zijn.

Circuit:



Layout:





**D.2.19 mir\_nin, mir\_nout**

Functie: NMOS spiegel, basiscel voor stroomspiegel

Terminals mir\_nin: (i, g, vss, vdd)

Terminals mir\_nout: (i, g, o, vss, vdd)

Equivalent chip oppervlak: 19

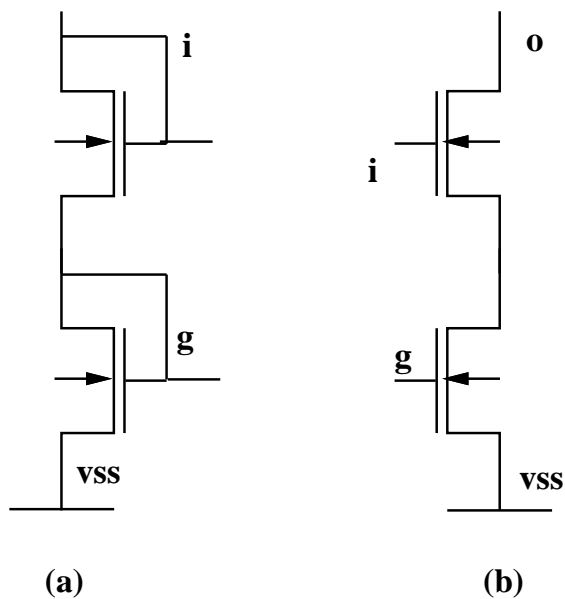
Beschrijving:

De NMOS spiegel is een gecascadeerde stroomspiegel. Hij bestaat uit een ingang mir\_nin en een uitgang mir\_nout. Beide kunnen in een spiegel meerdere keren gerepeteerd worden om schaling te verkrijgen. De gebruikte transistoren zijn zgn. compound transistoren.

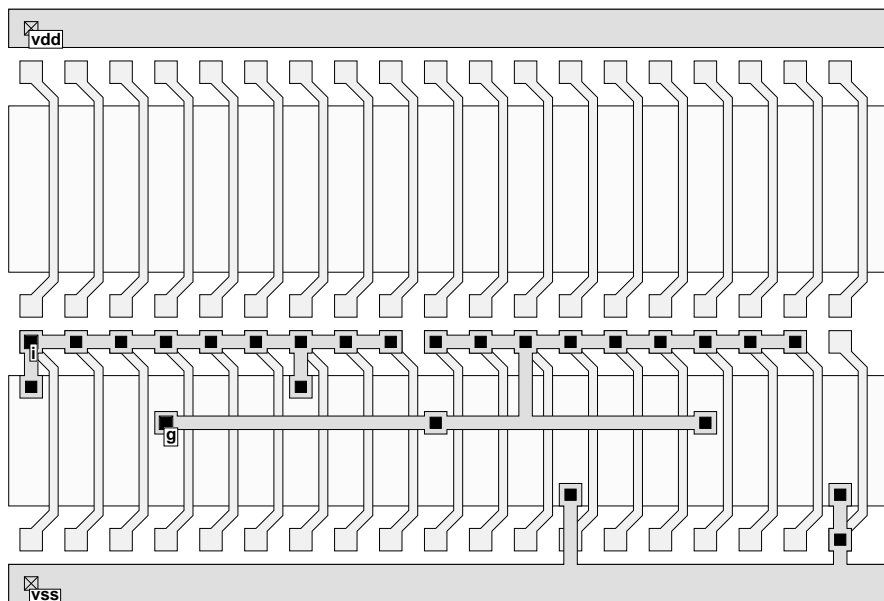
Circuit:

(a) mir\_nin

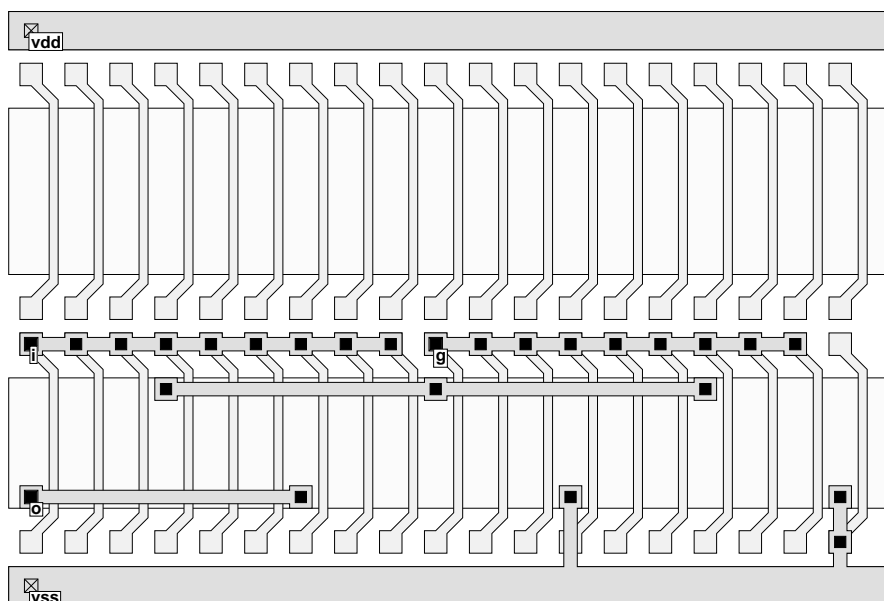
(b) mir\_nout



Layout mir\_nin:



Layout mir\_nout:



**D.2.20 mir\_pin, mir\_pout**

Functie: PMOS spiegel, basiscel voor stroomspiegel

Terminals mir\_pin: (i, g, vss, vdd)

Terminals mir\_pout: (i, g, o, vss, vdd)

Equivalent chip oppervlak: 19

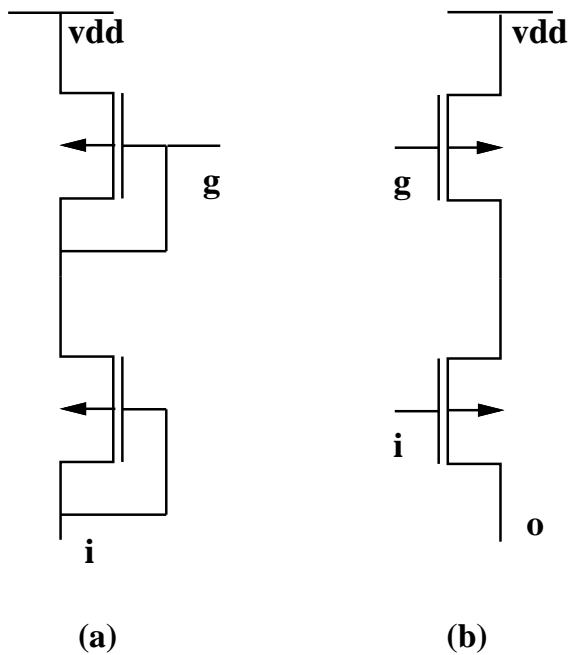
Beschrijving:

De PMOS spiegel is een gecascadeerde stroomspiegel. Hij bestaat uit een ingang mir\_pin en een uitgang mir\_pout. Beide kunnen in een spiegel meerdere keren gerepeteerd worden om schaling te verkrijgen. De gebruikte transistoren zijn zgn. compound transistoren.

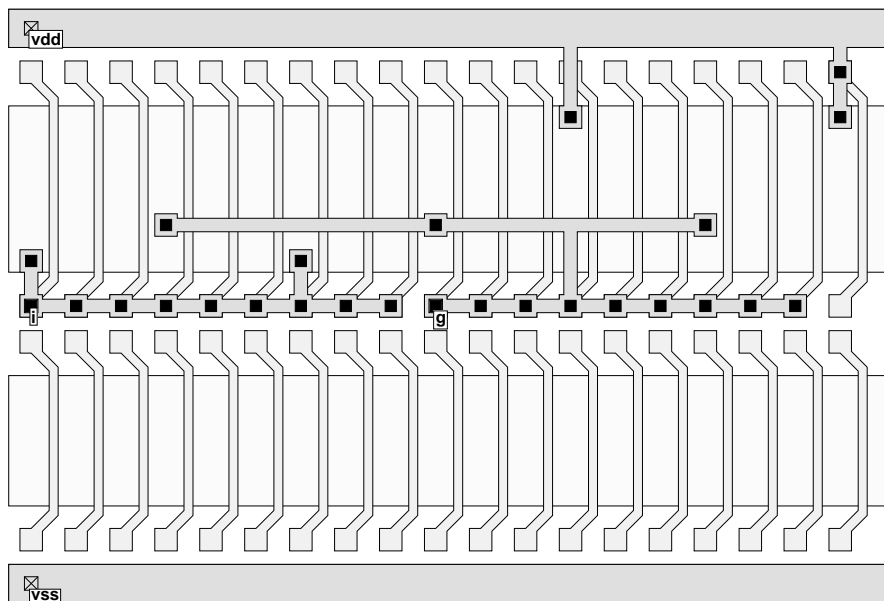
Circuit:

(a) mir\_pin

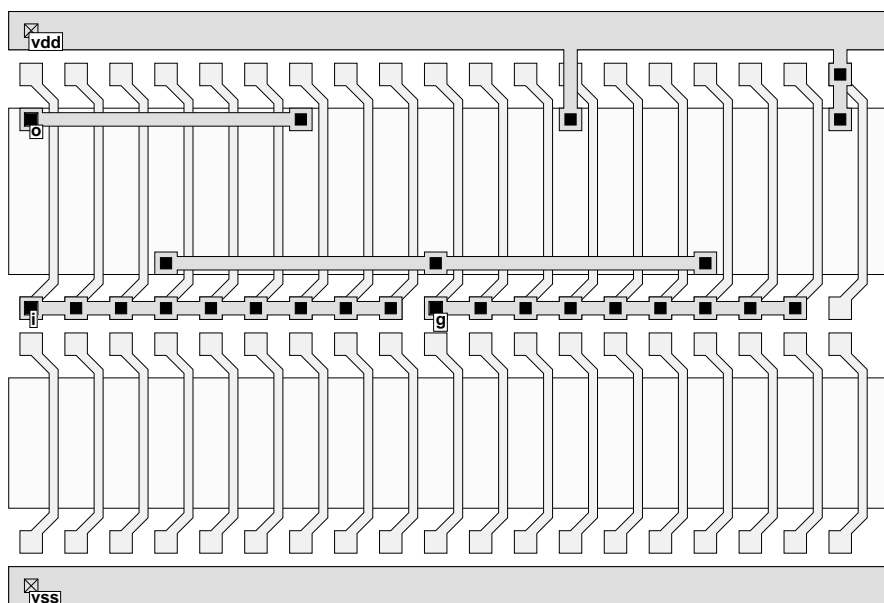
(b) mir\_pout



Layout mir\_pin:



Layout mir\_pout:



**D.2.21 bond\_leer**

Functie: Basiscel voor kleine SoG-schakelingen

Terminals: (vss, bf1, bf2, bf3, vdd, bf4, bf5, bf6)

Equivalent chip oppervlak: 800

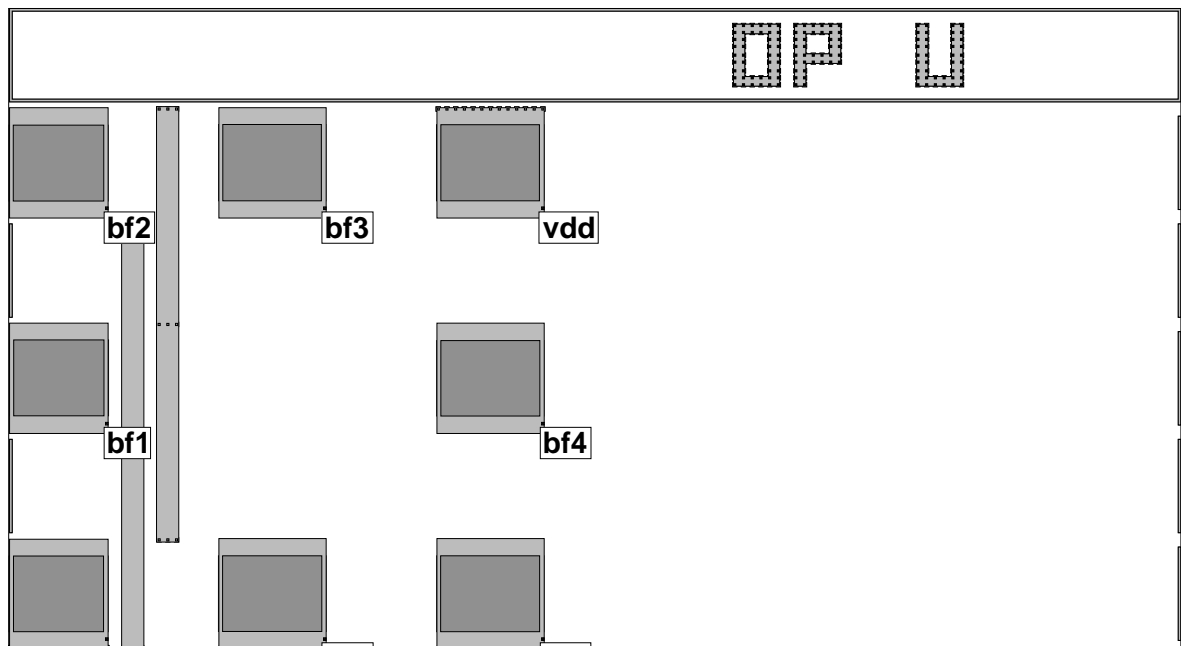
Maximum grootte schakeling: 350

Aansluitingen:

- Aantal: 8, waarvan 6 vrij te kiezen, 2 voeding (vss, vdd)
- Soort beveiliging: protectiedioden, geen buffers

Meetmogelijkheden: 8-pins probe-kaart

Layout:



**D.2.22 bond\_bar**

Functie: Basiscel voor kwartindeling SoG-chip

Terminals: (t1, t2, t3, t4, t5, t6, t7, t8, t9, t10, t11, t12, t13, t14, t15, t16, t17, t18, t19, t20, t21, t22, t23, t24, t25, t26, t27, t28, t29, t30, t31, t32)

Equivalent chip oppervlak: 25000

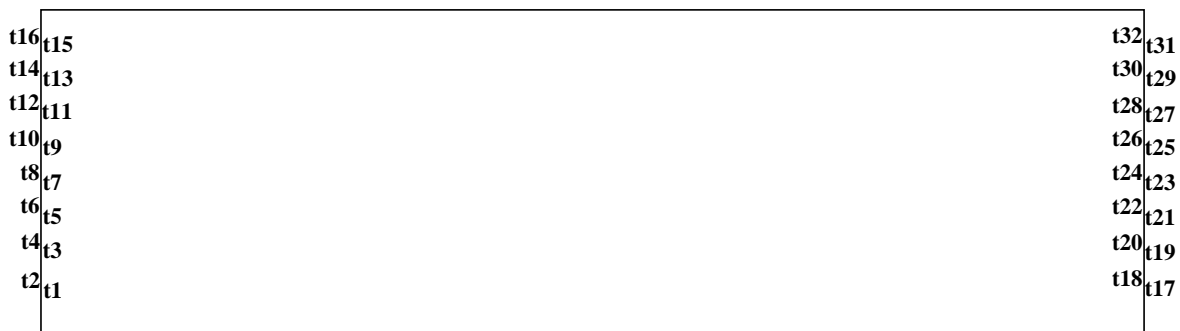
Maximum grootte schakeling: 20000

Aansluitingen:

- Aantal: 36, waarvan 32 vrij te kiezen, 2 voeding (chip), 2 voeding (buffers)
- Soort beveiliging: protectiedioden, buffers (in, out, direct)

Meetmogelijkheden: 36-pins probe-kaart, afmontage DIL-40

Layout:



## E Analoge Deelschakelingen

### E.1 Inleiding

Een kenmerk van een digitaal signaal is het feit dat slechts twee signaalniveaus van betekenis zijn voor de informatie-inhoud van dat signaal. Schakelingen die digitale signalen verwerken worden digitale schakelingen genoemd. Soms is het zinnig om voor de codering van informatie in een signaal meer dan twee niveaus te gebruiken. Het maximale aantal signaalniveaus dat nuttig gebruikt kan worden, hangt af van het maximaal toegelaten signaalniveau en het kleinst onderscheidbare verschil tussen twee signaalniveaus. Een veel gebruikte maat hiervoor is het dynamische bereik van een signaal. Schakelingen die dit soort signalen verwerken heten analoge schakelingen.

Elektronische schakelingen zullen bij het verwerken van signalen altijd het signaal in zekere mate aantasten. Het is de taak van de ontwerper om de schakelingen zo te ontwerpen dat de schade tot een minimum beperkt blijft. Bij digitale schakelingen zijn er slechts twee signaalniveaus van belang. Alle tussenliggende signaalniveaus worden slechts “gepasseerd” tijdens de overgang van het ene bekende naar het andere bekende niveau. Hoe de digitale schakeling van het ene niveau omschakelt naar het andere is niet aan erg gecompliceerde eisen gebonden. In het algemeen is het feit dat een digitale schakeling binnen een bepaalde tijd schakelt een voldoende garantie voor een goede werking. Dit heeft tot gevolg dat de modellering van de schakeling op dit gebied erg simpel kan zijn. Een model hoeft tijdens het omschakelen niet nauwkeurig te beschrijven op welk tijdstip een bepaald tussenliggend signaalniveau gepasseerd wordt, daar deze signaalniveaus toch geen informatie bevatten.

Bij analoge signalen en circuits is dit anders. Daar worden alle mogelijke signaalniveaus gebruikt om de informatie in het signaal te coderen. De modellering van analoge circuits zal daarom gecompliceerder zijn dan die van digitale circuits. Dit heeft tot gevolg dat op plaatsen in het ontwerptraject waar de modellering ter sprake komt het ontwerptraject voor analoge schakelingen zal afwijken van dat voor digitale schakelingen. Het betekent echter ook dat voor een zeer groot gedeelte van het ontwerptraject er geen onderscheid gemaakt hoeft te worden tussen analoge en digitale schakelingen. Het is eigenlijk ook niet verstandig om schakelingen het label analoog of digitaal te geven. Het gaat om het karakter van het signaal dat verwerkt wordt of de schakeling die het verwerkt analoog of digitaal genoemd moet worden. Ontwerphulpmiddelen zoals layout editors, design-rule checkers, routers, die geen weet hebben van het karakter van de signalen die verwerkt worden, kunnen met evenveel gemak op analoge als op digitale schakelingen worden toegepast. Alleen hulpmiddelen die zwaar leunen op de modellering van de schakelingen zullen voor analoge schakelingen duidelijk anders zijn dan voor digitale. De circuit simulator is daarvan wel het meest sprekende voorbeeld. Digitale schakelingen zijn meestal groot van omvang. Om zeer grote schakelingen nog efficiënt te kunnen simuleren is het belangrijk dat de modellering van de componenten waaruit de schakeling is opgebouwd zo simpel mogelijk is. Hierbij kan met veel succes gebruik gemaakt worden van het feit dat de te verwerken signalen digitaal zijn en dat er dus maar twee signaalniveaus onderscheiden hoeven te worden. Simulators voor digitale schakelingen zullen dus gekenmerkt zijn door de toepassing van simpele modellen op grote hoeveelheden componenten. Door de simpele modellen die gebruikt worden, zijn ze echter meestal niet geschikt voor het nauwkeurig simuleren van analoge schakelingen. Voor analoge schakelingen zal daarom gebruikt gemaakt worden van een simulator die werkt met gecompliceerde, nauwkeurige modellen. Dit beperkt dan weer de grootte van de schakeling die nog efficiënt gesimuleerd kan worden. Gelukkig zijn analoge schakelingen meestal veel kleiner dan digitale, zodat de simulator toch efficiënt gebruikt kan worden.

## E.2 Analoge signaalbewerkingen in de groepsopdracht

**Inleiding** Tijdens het ontwerp van de groepsopdracht zullen er in een aantal gevallen analoge signalen bewerkt of gegenereerd moeten worden. Deze signalen worden vaak opgedrongen door de "buitenwereld", omdat in de natuur bijna alle signalen meer dan twee niveaus kunnen aannemen. In deze paragraaf zullen we een aantal analoge signaalbewerkingen op een rijtje zetten die u kunt tegenkomen bij het maken van uw ontwerp.

**Uitlezen van een toetsenbord** Een toetsenbord bestaat uit een aantal schakelaars die open of gesloten kunnen zijn. De impedantie van een schakelaar die gesloten is, is lager dan van een schakelaar die open is. In het geval van het folie-toetsenbord dat voor het practicum beschikbaar is, is de impedantie in gesloten toestand ongeveer  $100\ \Omega$  en in open toestand enkele  $M\Omega$ 's. De impedantie van een schakelaar zal moeten worden omgezet in een digitaal signaal. Hier vindt dus een eenvoudige één-bit AD-conversie plaats.

**Generatie van sinusvormige signalen** Omdat een digitaal signaal slechts twee niveaus kan aannemen, zal zo'n signaal er altijd blokvormig uitzien. De "beste" benadering van een sinus met een digitaal signaal is een blokgolf met een duty-cycle van 50%. Als we een dergelijke blokgolf met een amplitude van  $\frac{\pi}{4}E_0$  opdelen in zijn verschillende frequentie-componenten, krijgen we:

$$\frac{\pi}{4}E_0\mathbf{B}(\omega t) = E_0\sin(\omega t) + \frac{E_0}{3}\sin(3\omega t) + \frac{E_0}{5}\sin(5\omega t) + \frac{E_0}{7}\sin(7\omega t) + \dots \quad (2)$$

De harmonische distorsie van dit signaal is de som van het vermogen van de hogere harmonischen gedeeld door de grondharmonische. Dus:

$$\frac{\left(\frac{E_0}{3}\right)^2 + \left(\frac{E_0}{5}\right)^2 + \left(\frac{E_0}{7}\right)^2 + \dots}{E_0^2} \quad (3)$$

en dit is gelijk aan:

$$\left(\frac{1}{3}\right)^2 + \left(\frac{1}{5}\right)^2 + \left(\frac{1}{7}\right)^2 + \dots = \sum_{n=1}^{\infty} \frac{1}{(2n+1)^2} \quad (4)$$

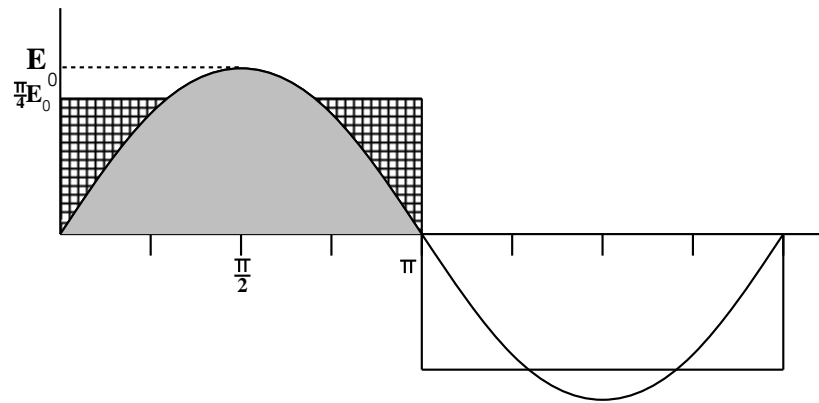
Dit kunnen we ook schrijven als:

$$\sum_{n=1}^{\infty} \frac{1}{n^2} - \sum_{n=1}^{\infty} \frac{1}{2n^2} - 1 = \frac{\pi^2}{6} - \frac{\pi^2}{24} - 1 = \frac{\pi^2}{8} - 1 \approx 0.23 \quad (5)$$

Op een iets meer intuïtieve manier kunnen we dit ook zien aan de hand van figuur 43:

De harmonische distorsie is, zoals al eerder gezegd, de som van het vermogen van de hogere harmonischen gedeeld door de grondharmonische. Het vermogen van de grondharmonische is in de





Figuur 43: Figuur ter verduidelijking van bewijs harmonische distorsie blokgolf

figuur de oppervlakte onder de sinus. De som van de vermogens van de hogere harmonischen is de oppervlakte onder de blokgolf minus de oppervlakte onder de sinus. De harmonische distorsie is dus:

$$\frac{\text{opp. blok} - \text{opp. sinus}}{\text{opp. sinus}} = \frac{\pi \cdot \frac{\pi}{4} E_0 - \int_0^\pi E_0 \sin x \, dx}{\int_0^\pi E_0 \sin x \, dx} = \frac{\frac{\pi^2}{4} - 2}{2} = \frac{\pi^2}{8} - 1 \quad (6)$$

Als we dus een signaal willen genereren met een lagere harmonische distorsie dan 0.23 of -6.4 dB, kan dat niet met een digitale schakeling.

Er zijn verschillende mogelijkheden om een sinusvormig signaal te genereren, b.v.:

- Met behulp van oscillatoren die elektronisch afstembaar zijn (Voltage Controlled Oscillator = VCO). Het digitale deel zal dan via een DA-converter (digitaal-analoog omvormer) de oscillator op de juiste frequentie afstemmen.
- Met behulp van een teller en een DA-converter. De teller zal een DA-converter zodanig aansturen dat aan de uitgang van de DA-converter een sinusvormig signaal verschijnt van de juiste frequentie.
- Door het digitale circuit een blokvormig (digitaal) signaal van de juiste frequentie te laten leveren wat daarna gefilterd moet worden om hogere harmonischen in het signaal te verwijderen.
- Met behulp van puls-breedte modulatie. Een hoogfrequent draaggolf wordt puls-breedte gemoduleerd met een gekwantificeerde sinus. De draaggolf wordt vervolgens uitgefilterd.

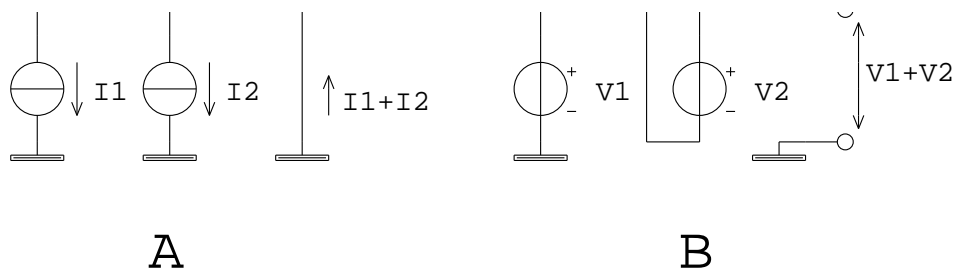
**Versterken** Versterken is de meest gebruikte analoge signaalbewerking. Aan de ingang van een ontwerp is meestal versterking nodig omdat het inkomende signaal vaak te zwak is om direct gedigitaliseerd te worden. Aan de uitgang omdat het digitale signaal vaak te weinig vermogen heeft om direct een actuator<sup>3</sup> aan te sturen.

<sup>3</sup> Voorbeelden van actuatoren zijn: een luidspreker, robot-arm, beeldbuis, enz.

**Optellen** Het optellen van twee signalen kan zowel digitaal als analoog gebeuren. In dit hoofdstuk zullen we ons beperken tot de analoge methode. In dit laatste geval worden de twee op te tellen signalen eerst afzonderlijk gegenereerd en daarna opgeteld. De twee gegenereerde signalen zullen beschikbaar zijn als spanning of als stroom. Stromen kunnen vrij simpel worden opgeteld door ze simpelweg in hetzelfde knooppunt te laten vloeien (parallel schakeling). Door de uitgangen van twee stroombronnen die ieder een sinusvormige stroom produceren met elkaar te verbinden is de optelling gerealiseerd.

Om spanningen te kunnen optellen moeten de spanningsbronnen in serie geschakeld worden. Dit betekent dat tenminste één van de twee bronnen zwevend zal moeten zijn. Het maken van zwevende spanningsbronnen is moeilijk, daarom worden van de spanningen meestal eerst stromen gemaakt (bijvoorbeeld d.m.v. een weerstand) die daarna worden opgeteld.

In figuur 44 is het optellen van spanningen en stromen te zien. Figuur 44A toont het optellen van stromen, figuur 44B het optellen van spanningen.



Figuur 44: Het optellen van spanningen en stromen

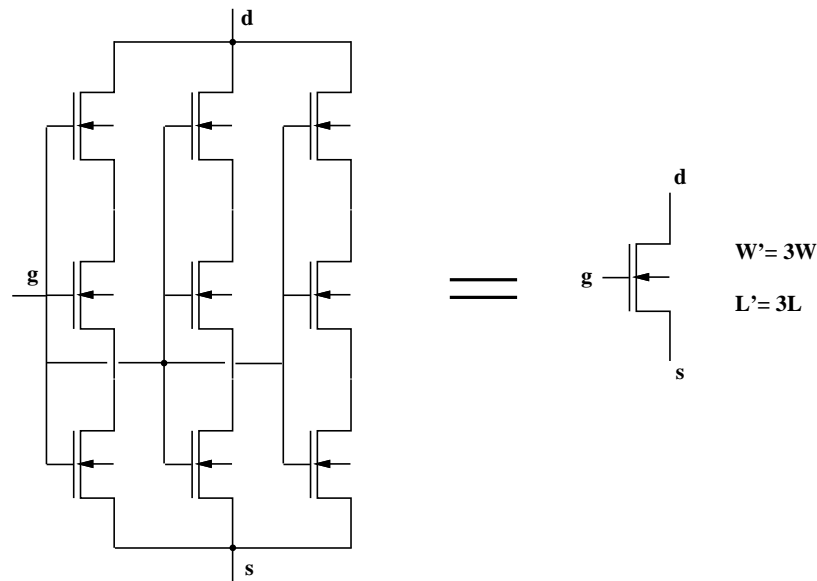
### E.3 Bouwstenen voor analoge circuits op de SoG wafer

Er zijn voor de implementatie drie verschillende soorten componenten beschikbaar:

1. MOS-transistoren
2. Weerstanden
3. Condensatoren

**MOS-transistoren.** Op de sea-of-gates chip is van de PMOS en de NMOS transistor maar één type beschikbaar. Dit zijn transistoren met een zeer kort kanaal. Om toch iets aan schaling van  $W/L$  verhoudingen te kunnen doen moeten er transistoren parallel en in serie geschakeld worden. Door twee transistoren parallel te schakelen neemt de breedte ( $W$ ) van de zo ontstane compound-transistor met een factor twee toe. Door er twee in serie te schakelen (gates aan elkaar) ontstaat een twee keer zo lange.

Eén van de nadelige effecten van een kort kanaal is de lage uitgangsimpedantie die de transistor daardoor heeft. Dit is o.a. bijzonder vervelend in stroomspiegels, waarvan veel gebruik gemaakt wordt. Daarom is er een compound-transistor van  $3 \times 3$  transistoren gemaakt die bij de ontwerpen zal worden gebruikt, zie figuur 45. In de bibliotheek staan deze compound-transistoren onder de naam  $1n3 \times 3$  (nmos) en  $1p3 \times 3$  (pmos).



Figuur 45: De compound-transistor.

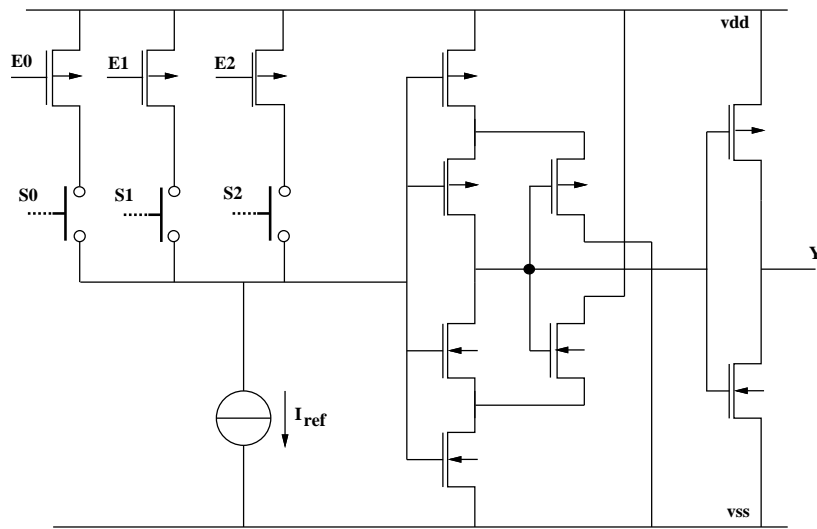
**Weerstand** Het is mogelijk de gate van de mos-transistoren te gebruiken als weerstand. Elke gate heeft twee aansluitingen waartussen zich een stukje poly-silicium bevindt dat de gate vormt. De weerstand tussen de twee aansluitingen wordt bepaald door de weerstand van dit stukje poly-silicium en ligt rond de  $700 \Omega$ . Door deze weerstandjes in serie of parallel te schakelen kunnen weerstanden van verschillende waarden gemaakt worden. De source en de drain van de betreffende transistoren worden dan niet gebruikt. De “gate-weerstandjes” matchen goed zodat op deze manier nauwkeurig geschaalde weerstanden kunnen worden gemaakt.

**Capaciteiten** Er zijn verschillende manieren om op de Sea-of-Gates chip capaciteiten te maken. Als er een “zwevende” (die niet met een aansluiting aan de  $v_{ss}$  verbonden is) capaciteit gemaakt moet worden, kan gebruikt gemaakt worden van de capaciteit tussen de eerste en tweede metaallaag. Capaciteiten die met één poot aan  $v_{ss}$  liggen, kunnen gemaakt worden door verschillende onderliggende lagen met elkaar te verbinden, zodat een soort “sandwich” ontstaat. De waarden van deze capaciteiten per oppervlakte staan gegeven in appendix C. Omdat op deze manier slechts capaciteitswaarden tot enkele pF’s te maken zijn, zal het in de praktijk vaak voorkomen dat capaciteiten extern aangesloten moeten worden. Hierdoor is het tevens mogelijk om achteraf de waarde ervan nog aan te passen.

## E.4 Voorbeelden van implementaties

### E.4.1 Toetsenbord-uitlezing

In deze paragraaf zal niet diep worden ingegaan op het scannen van een toetsenbord, maar alleen op het omzetten van de impedantievariaties in een digitaal signaal. In figuur 46 is een simpel schema gegeven.



Figuur 46: Methode om een toetsenbord uit te lezen.

Een referentiestroom  $I_{ref}$  kan door middel van de ingangen E0, E1 en E2 worden geleid door één van de drie takken met een schakelaar. Het is de bedoeling dat de signalen op E0, E1 en E2 zo worden gekozen dat steeds slechts één weg voor de stroom mogelijk is.

Wanneer  $I_{ref}$  goed wordt gekozen, is het mogelijk om de spanningsvariaties als gevolg van het indrukken van de toetsen zodanig te krijgen, dat het signaal direct kan worden aangeboden aan een digitale schakeling. Omdat de flank van dit signaal meestal niet al te steil zal zijn, is het aan te raden om het digitale circuit een zgn. Schmitt-trigger ingang te geven. Deze schakeling is ook weergegeven in de figuur. Een Schmitt-trigger is een schakeling die vanwege meekoppeling een zeer abrupte input-output-karakteristiek krijgt. Overigens hebben de inputbuffers op de chip alle een Schmitt-trigger-ingang.

#### E.4.2 DA-converter

Het principe van de DA-converter wordt behandeld in [8]. Hier staat ook een voorbeeld van een DA-converter op basis van een ladder-verzwakker-netwerk. Andere methoden zijn bijvoorbeeld met behulp van ladingsdeling of geschaalde stroomspiegels. In deze paragraaf zal dieper worden ingegaan op een DA-converter gebaseerd op geschaalde stroomspiegels.

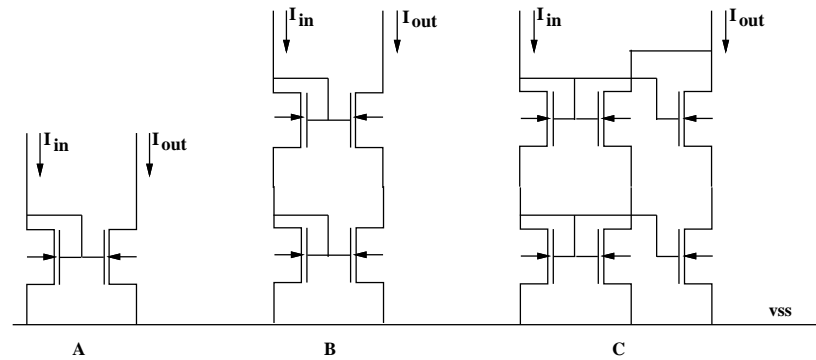
In figuur 47A is het schema gegeven van een eenvoudige stroomspiegel.

Bij een stroomspiegel is het de bedoeling dat uitgaande stroom gelijk is aan de ingaande. Eén van de oorzaken waarom dit nooit precies zo zal zijn, is de eindige uitgangsimpedantie van de uitgangstransistor. Deze gedraagt zich hierdoor niet als een ideale stroombron.

Als dus de drain-spanningen niet hetzelfde zijn, zal  $I_{out}$  niet precies gelijk zijn aan  $I_{in}$ .

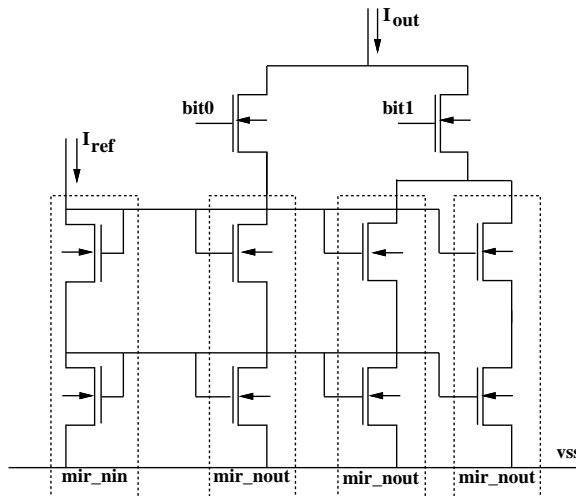
Bij de transistoren die op de Sea-of-Gates chip voorhanden zijn, kan de fout bij een verschil van één Volt al vele procenten bedragen.

Om dit effect te verkleinen kan voor een configuratie als in figuur 47B gekozen worden. De stroomspiegel is nu gecascadeerd. Hierdoor worden de drain-spanningen (van de onderste transistoren) beter aan elkaar gelijk gehouden. Meer informatie over stroomspiegels kunt u vinden in [8].



Figuur 47: A: Een eenvoudige stroomspiegel, B: Een gecascadeerde stroomspiegel, C: Een schalende (gecascadeerde) stroomspiegel

Het is ook mogelijk om met een stroomspiegel stromen te schalen. Door in figuur 47B de uitgangstransistoren dubbel te nemen, zal de stroom  $I_{out}$  twee keer zo groot zijn als  $I_{in}$ , zie figuur 47C. Op deze manier kunnen dus uit één referentiestroom stromen gemaakt worden die een geheel veelvoud hiervan zijn. Met deze techniek is het heel eenvoudig een DA-converter samen te stellen.



Figuur 48: Een 2-bits DA-converter gebouwd met stroomspiegels en stroomschakelaars.

In figuur 48 is een uitvoering van zo'n DA-converter gegeven.

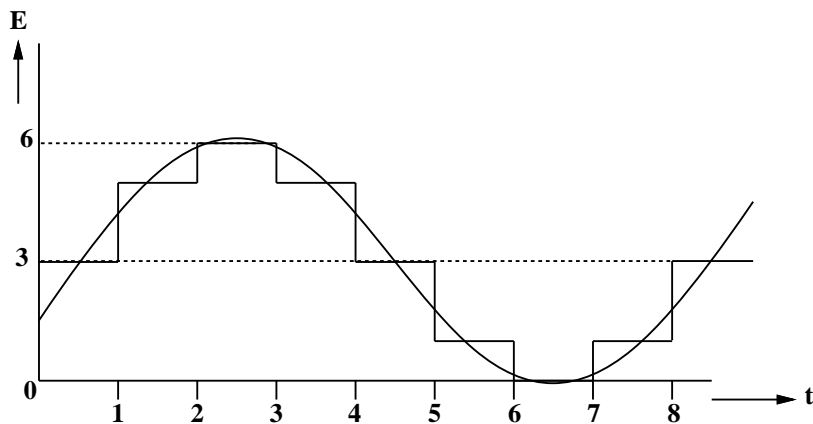
Met de (digitale) signalen bit0 en bit1 kunnen stromen aan- en afgeschakeld worden. Als bijvoorbeeld bit0 "hoog" is (5 V) en bit1 "laag", zal de uitgangsstroom  $I_{out}$  gelijk zijn aan  $I_{ref}$ . Wanneer beide signalen hoog gemaakt worden, zal  $I_{out}$   $3I_{ref}$  bedragen.

Vanwege het al eerder geschetste probleem met de (te) lage uitgangsimpedantie, is in dit voorbeeld gekozen voor de gecascadeerde stroomspiegel. De basiscellen `mir_nin` en `mir_nout` komen uit de bibliotheek.

### E.4.3 Sinus-generatie met een DAC

Eén van de mogelijkheden om een sinus te genereren is een DAC aan te sturen vanuit een digitaal circuit.

Stel dat we een sinus willen genereren in 8 stapjes met een 3-bits DAC. We krijgen dan een signaal dat er uitziet als in figuur 49.



Figuur 49: Benadering van een sinus met een DAC

In tabel 5 staan de signalen die toegevoerd moeten worden om deze "sinus" te krijgen.

Tabel 5: Signalen voor de DAC

t	E	bit2	bit1	bit0
0	3	0	1	1
1	5	1	0	1
2	6	1	1	0
3	5	1	0	1
4	3	0	1	1
5	1	0	0	1
6	0	0	0	0
7	1	0	0	1

Het is vrij lastig om de distorsie van dit signaal met de hand te berekenen. Programma's als SPICE doen dit via een Fourier-analyse. Uit simulaties met dit programma blijkt dat een sinusgeneratie in 16 stapjes met een 4-bits DA-converter een harmonische distorsie geeft die kleiner is dan -20 dB (Wanneer de DAC niet-lineair gemaakt wordt, kan bij juiste dimensionering ook volstaan worden met 3-bits).

### E.4.4 Eindversterker

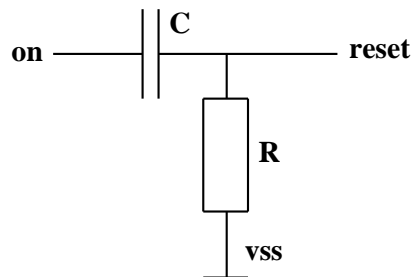
De luidspreker heeft een impedantie van ongeveer 50  $\Omega$ . Om hierin voldoende vermogen te kunnen ontwikkelen is er een eindversterker nodig.

Er kan gebruikt gemaakt worden van een geschaalde stroomspiegel zoals dit ook al is gedaan bij de DA-converters. De schaalfactor wordt zò gekozen dat er voldoende stroom door de luidspreker zal lopen. Het zal duidelijk zijn dat er op deze manier geen “high-performance” versterker ontstaat, maar op dit moment zal de schakeling kunnen voldoen.

#### E.4.5 Power on reset

Bij geheugenelementen is het vaak niet te voorspellen in welke toestand de schakeling komt wanneer de voedingsspanning aangesloten wordt. Omdat dit in veel gevallen ongewenst is, wordt vaak een zgn. “power-on-reset” aan de schakeling toegevoegd. Dit schakelingetje genereert een reset-puls op het moment dat de voedingsspanning aangesloten wordt. Hierdoor komt de schakeling altijd in een vaste begintoestand als hij aangezet wordt.

Een mogelijkheid die men vaak ziet, is die met een simpel RC-filttertje, zoals in figuur 50.



Figuur 50: Simpele implementatie power on reset.

Deze schakeling wordt volledig extern aangebracht.

Op dit moment wordt er gewerkt aan een bibliotheekcel die hetzelfde doet. Deze is echter nog niet voor het practicum geschikt.





## F Testing SOG chips with the Logic Analyzer

### F.1 Introduction

In this document we describe how a Sea-Of-Gates chip in a DIL40 package can be tested with the logic analyzer LA-5580. The purpose is to verify if, given a set of input patterns, the tested device produces the same output patterns as a set of reference output patterns.

A logic analyzer is an instrument that is similar to an oscilloscope: both instruments measure output signals (voltage) as a function of time after some trigger moment. However, important difference are:

- A logic analyzer usually only measures two different signal levels: high and low.
- A logic analyzer is capable of measuring a large number of output signals (e.g. 80).
- Besides measuring output signals, a logic analyzer can simultaneously apply input signals.

The logic analyzer LA-5580 is a hardware box that is controlled (via a USB connection) by a graphical interface program LA5000 that is running a PC. A picture of the interface is shown in Figure 51

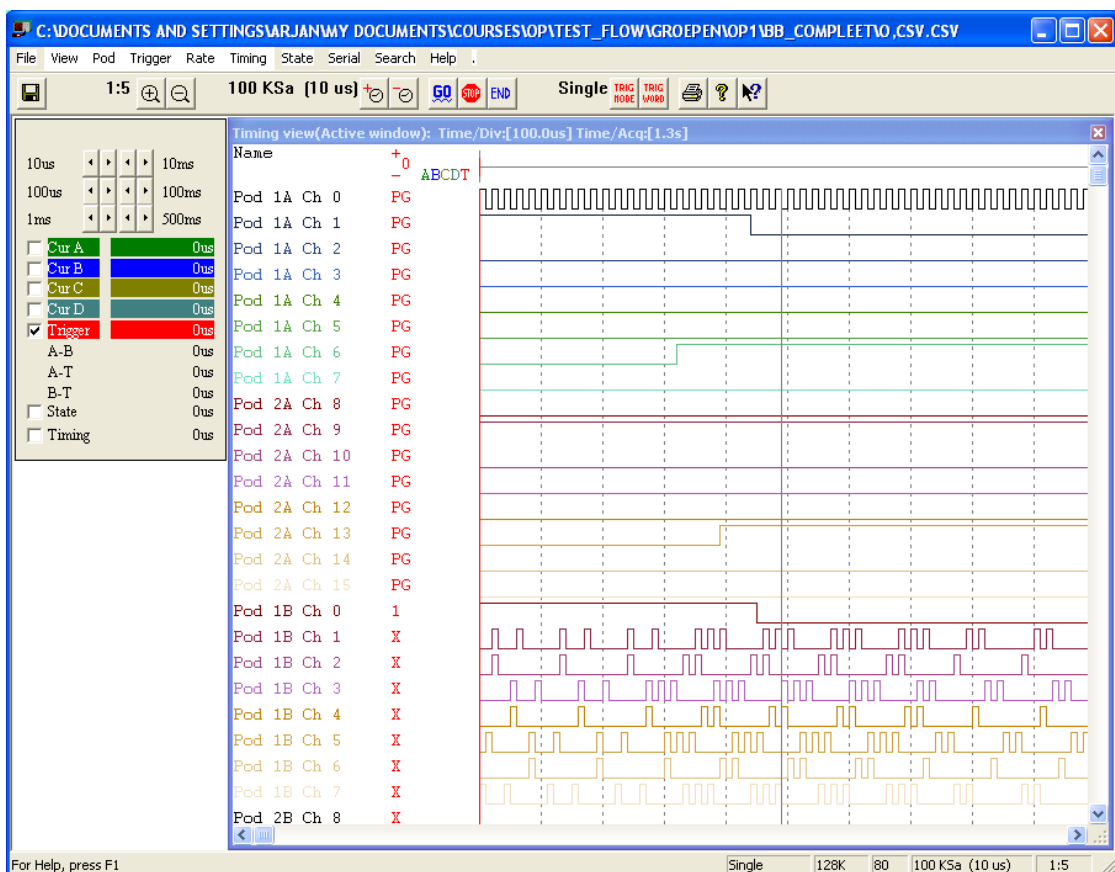


Figure 51: Graphical Interface for the Logic Analyzer LA-5580

## F.2 Overview

As input we need a file with reference input and output patterns (.ref file) and a file with information about how the pins of the design are connected to the pins of IO buffers of the chip (.buf file). Typically, both these files have been produced during the design of the chip. Using the program test\_flow, these files are converted into a pattern (.csv) file and an initialization (.ini) file that can be used as input for the logic analyzer program, and a scheme that shows how the pods of the logic analyzer should be connected to the pins of the package of the SOG chip. Then, via the logic analyzer program LA5000, the logic analyzer is used to send the input patterns to the tested SOG device and to collect the resulting output patterns. Finally, the program test\_flow is used to compare the measured output patterns against the reference output patterns.

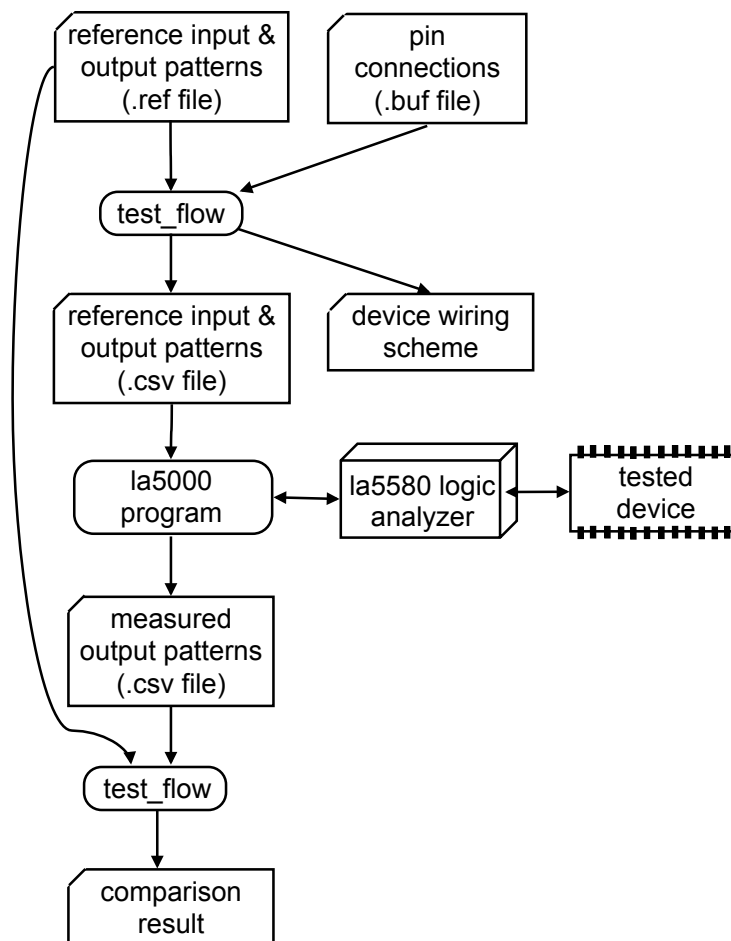


Figure 52: The different steps for testing

### F.3 Obtaining the Reference Files

Copy the relevant .ref and .buf files that you have created during the design phase to a writable directory on the local disk of your PC (e.g. C:\wpstmp). Note that it is important to store the files on the local disk because otherwise the logic analyzer software may have problems to read it. Note that the desktop of the PC is usually not on the local disk in our environment.

### F.4 Preparing the Input Data for the Logic Analyzer

Start the program TestFlows. Next click Commands → Read\_reffile and open the .ref file. If the same directory contains a .buf file, you will be asked if it is ok to update the information with this file. In that case, click yes. Otherwise click Commands → Update\_with\_buffile and open the appropriate .buf file. To create a window with input patterns and output patterns in csv format, click Commands → Make\_csvfile. In the new csv window, use the command File → Save\_csvfile to create the data files that can be used as input for the logic analyzer.

A wiring scheme for how to connect the clips of the wires of the pods to the pins of the SOG device can be obtained by first selecting the appropriate bondbar number for the package in the upper right corner of the test\_flow window and next using the command Commands → Show\_pod\_connections.

### F.5 Setting up the Logic Analyzer

Connect the USB cable of the logic analyzer LA-5580 to the upper USB port at the front of your computer. Plug in the pattern generation pods Pod 1A, Pod 2A, etc. in the lower connector rows (label "board 1") of the analyzer and the logic (measurement) pods Pod 1B, Pod 2B, etc. in the upper connector rows (label "board 2") of the analyzer. Note that the pods have numbers on them. Plug Pod 1 in the leftmost connector, Pod 2 in the second leftmost connector etc.

### F.6 Mounting and connecting the SOG chip

Mount the SOG package on the DIL40 connector box. From the separate IC testbox, connect GND (0 Volt) and VDD (5 Volt) to the appropriate connector holes on the DIL40 connector box, using cables with 4mm plugs. Connect the clips of the wires of the pods to the pins of the SOG package according to the device-wiring scheme that was generated using test\_flow. Connect at least one black wire of each pod to the GND signal. The pod input Pod 1B(0) at channel 0 will be used as a trigger signal for the acquisition of data and should directly be connected to an output of a pattern generation pod. This connection is given at the bottom of the wiring scheme that was obtained using test\_flow.

Switch on the power supply for the SOG chip and switch on the power supply for the logic analyzer. Be careful to only reconnect wires to the pins of the device when the power supply for the SOG chip is switched off.

## F.7 Using the Logic Analyzer

Start the Logic Analyzer control program LA5000 Logic Analyzer. If the Timing view window is not yet shown, click Timing → Timing window.

Besides the .csv file that was created by test\_flow, also a .ini file with the same base name was created by test\_flow. Click File → Open to first open the file .ini file that was created by test\_flow.

Next, click View → Clear Data Buffer to clear all signals. Then, click File → Open to open the .csv file that was generated using test\_flow. The option Type 2 CSV file must be enabled during this. After reading the .csv file you will see the reference output signals in the Timing window and you may inspect the input signals when you click Pod → Edit Pattern Generator Data (or button EDIT PAT). The reference output signals will be overwritten when capturing analysis data from the logic analyzer.

Then, click Trigger → Go or the Go button to start the analysis. The output of the analysis will appear in the Timing view window.

To prepare for a more exact comparison using the program test\_flow, export the output signals using File → Export. For the Data field, select "Group outputs" and select Binary, click OK, and then save the file as type "Comma Sep (.CSV)".

## F.8 Comparing the Analyzer Output with the Reference Output

In test\_flow, click on Commands → Compare. Next, in the compare window, click File → Compare csv and select the .csv file that was created using the Logic Analyzer. After the file has been read, the measured output signals will be compared against the reference output signals, just before every rising clock edge. On each line you will see from left to right: the time, the clock period number, the input signals, the reference output signals and the measured output signals. In case the reference outputs and measured outputs are different, they have a red colour. You can click on a line to list the different output vectors below each other. If you click on a column in this list, the name of the corresponding output terminal will be highlighted at the sub window at the right. You can use commands from the Commands button in the compare window to navigate through the errors.

Default, all output signals and all inout signals will be compared. By disabling the toggle button inout, only output signals will be compared.

## F.9 Trouble Shooting

When the output signals are not according to as expected, carefully check all pod connections. Further you can try to lower the sample rate. Thirdly, you can try another IC.

## F.10 Further Reading

LA 5000 Logic Analyzer Software Manual (see Course Documents on blackboard)  
Logic Analyzer LA-5000 series, Link Instruments Inc (<http://linkinstruments.com>)  
<http://duteela.et.tudelft.nl/tools/measure/LA5580.html>

## References

- [1] J. Rabaey, “Matlab models for transistors and invertors,” Last Visited: 2015-09-24. [Online]. Available: <http://icbook.eecs.berkeley.edu/resources/reader-student/device-models>
- [2] J. Chern, P. Chang, R. Motta, and N. Godinho, “A new method to determine MOSFET channel length,” *Electron Device Letters, IEEE*, vol. 1, no. 9, pp. 170–173, 1980.
- [3] IEEE, “Information for authors,” last visited: 2015-09-24. [Online]. Available: <http://www.ieee.org/documents/auinfo07.pdf>
- [4] Brown and Vranesic, *Fundamentals of Digital Logic with VHDL Design, 3rd Edition*. McGraw - Hill, 2009.
- [5] P. J. Ashenden, *The Student’s Guide to VHDL, 2nd Edition*. Morgan Kaufmann Publishers, 2008.
- [6] R. Grit, *Projectmanagement*. Noordhoff Uitgevers B.V., 2011.
- [7] J. M. Rabaey, *Digital Integrated Circuits, A Design Perspective*. Prentice Hall Electronics and VLSI Series, 1996.
- [8] R. Spencer and M. Ghausi, *Introduction to Electronic Circuit Design*. Prentice Hall, 2003.