VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

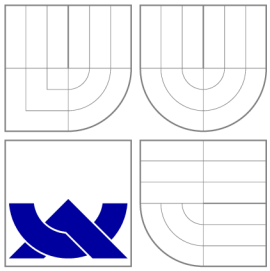
DESIGN OF DIGITAL CIRCUITS AT TRANSISTOR LEVEL

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

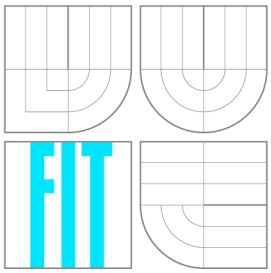
AUTOR PRÁCE
AUTHOR

FILIP KEŠNER

BRNO 2014



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

NÁVRH ČÍSLICOVÝCH OBVODŮ NA ÚROVNI TRANZISTORŮ

DESIGN OF DIGITAL CIRCUITS AT TRANSISTOR LEVEL

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — FILIP KEŠNER
AUTHOR

VEDOUCÍ PRÁCE — Ing. ZDENĚK VAŠÍČEK, Ph.D.
SUPERVISOR

BRNO 2014

Abstrakt

Práce se zaměřuje na návrh obvodů na úrovni tranzistorů, především za použití evoluční metody návrhu. Za tímto účelem je nutné volit rozumnou míru abstrakce a tak dosáhnout vyšší rychlosti ohodnocování kandidátních řešení pomocí fitness funkce. Práce probírá již vyzkoušené postupy návrhu obvodů na tranzistorové úrovni a z nich vybírá užitečné prvky pro vytvoření výkonějšího systému, který by byl schopen navrhovat komplexní logické obvody. Dále se práce zabývá implementací tohoto systému a probírá použitý přístup k řešení problémů návrhu a optimilizace tranzistorových obvodů použitím evoluce.

Abstract

This work aims to design process of integrated circuits on the transistor level, specially using evolutionary algorithm. For this purpose it is necessary to choose reasonable level of abstraction during simulation, which is used for evaluation candidate solutions by fitness function. This simulation has to be fast enough to evaluate thousands of candidate solutions within seconds. This work discusses already used techniques for transistor level circuit design and it chooses useful parts for new design of faster and more reliable automated design process, which would be able to design complex logic circuits. The thesis also discusses implementation of this system and used approach with regard to encountered problems in transistor-level circuit design and optimization by evolution.

Klíčová slova

evoluce, návrh obvodů, kartézké genetické programování, diskrétení simulace, SPICE, tranzistory, CMOS, MOSFET, integrovaný obvod, optimalizace návrhu

Keywords

evolution, evolotionary algorithms, transistor, integrated circuit, CMOS, MOSFET, discrete simulation, SPICE, cartesian genetic programming, design optimization

Citace

Filip Kešner: Design of Digital Circuits at Transistor Level, diplomová práce, Brno, FIT VUT v Brně, 2014

Design of Digital Circuits at Transistor Level

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Zdeňka Vašíčka, Ph.D.

.....
Filip Kešner
May 28, 2014

Contents

Chapter 1

Introduction

The aim of this thesis is to describe transistor-level technology and its usage in the integrated circuits design using evolutionary algorithm.

Some effort in area of usage evolutionary algorithms for transistor-level circuit design has been already done and also some level of success has been achieved. Usually quite small and relatively simple circuits were designed using artificial evolution. There were different aims in those attempts, some were experimental and try to prove, that in some way better innovative designs can be produced. Others selected the aim for transistor variability tolerance, lower power consumption, or just minimal transistor count.

This work is oriented to provide efficient evolutionary designing system which will produce transistor-level circuits defined by their input and output specification. But to achieve that, we need to design that system first.

Let's start with simplified description of what can be found in this document: in this technical report will be discussed problems and methods used in at least partially automated design of digital circuits at transistor level. In the first following chapter (2) transistor technology will be described and discussed. There will be mentioned not only the current state but also something relevant to development of this technology.

Next chapter (3) is dedicated to review what was already done in the area of conventional and unconventional transistor-level design of digital VLSI(Very Large Scale Integration) circuits.

In the chapter after that one, will be described evolutionary algorithms and their possible usage in this digital circuit design area. When this was mentioned, the important part of evolutionary algorithm will arise to our attention, the *fitness* evaluation of circuits during evolutionary process, with regard to computational time.

Implementation, chapter (6), is also included, just after previously mentioned one. There is described how the programming of the designing system went through time and what limitations have been encountered and how they have been overcome. In next chapter there are also provided information about computational time and efficiency of the implementation. To evaluate efficiency of implemented simulator and evolutionary designing system, there is chapter describing benchmarks and experiments.

Circuit which was created by unconventional automated design mechanism need to go through some more advanced testing and simulation before they can be used in real circuit chip. Some of transistor circuit designs, which were produced be earlier mentioned process are illustrated, verified and discussed at the end of benchmarking chapter.

The conclusion, chapter (9) at the end of this document, summarizes what have been done in this project and it also points out some ideas for possible future work.

Chapter 2

Transistor

Here we will discuss transistor technology itself, starting with description what exactly is the transistor, followed by different implementation technologies.

2.1 What is the transistor

Said in simplified manner, transistor is a semiconductor device. Actually there exist more than few variants of transistor and more about this will be described in the following section 2.3. But all transistors have things in common. Probably the most important such thing would be their ability to control electrical signals. By the control we can image for example amplification, switching „on/off“, or even generating electrical signals.

Deeply embedded in almost everything electronic, transistors have become the nerve cells of the Information Age [3].

2.2 History

The transistor was invented in 1947–48 by three American physicists, John Bardeen, Walter H. Brattain and William B. Shockley, at the American Telephone and Telegraph Company's Bell Laboratories.

The discovery was made during experimentation with current flowing

This discovery made electron tubes, which needed hundreds of volts and energetically expensive heating, obsolete. But it took some time, almost 20 years. And that made possible to design and produce integrated circuits. Which are the hearts (and brains) of modern electronics and computers.

2.3 Variants of transistor

2.3.1 Bipolar transistor

Bipolar transistors are one of the most important semiconductor devices. The *bi-polar* in the name means that both types of electrical conductive elements are used for function of this transistor. These elements are electrons and holes. The structure of the transistor is based on silicon substrate, where the base substrate is made of one type of conductivity and others two electrodes are made of complementary type. Possible combinations are PNP and NPN, where base substrate is usually connected to collector electrode.

The Bipolar Junction Transistor (BJT) is a three layer device constructed form two semiconductor diode junctions. These junctions are joined together in the one of following ways,

- NPN - base to emitter is forward oriented junction and base to collector reverse oriented
- PNP - base to emitter is reverse oriented junction and base to collector forward oriented

Let's take a look at structure of the bipolar transistor:

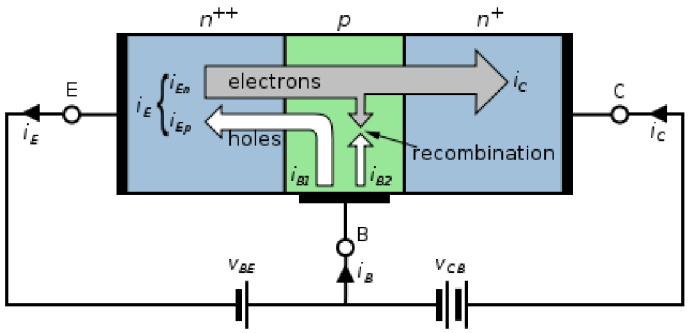


Figure 2.1: Bipolar Junction Transistors function structure [6]

To have a better idea of bipolar transistor usage it would be useful to have diagram which shows some relevant electro-physic values.

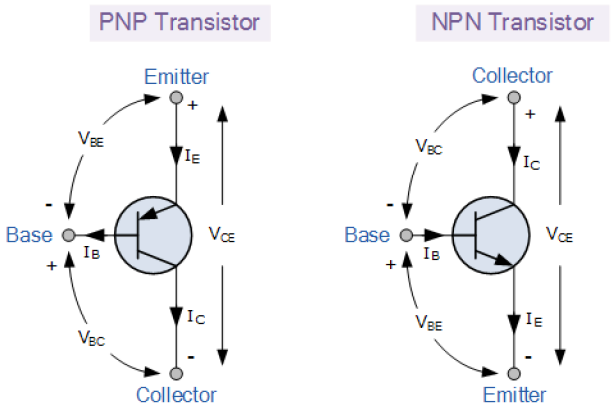


Figure 2.2: Bipolar Junction Transistors schematic voltages [6]

Bipolar transistors are **current controlled devices**, where relatively small base current I_B controls possibly bigger current flow through CE electrodes, this current is called I_E. The ration between these to is called **current amplification coefficient** and it is one of the most important characteristics of bipolar transistor

$$H_{21} = \frac{I_E}{I_B}$$

Disadvantages of the bi-polar transistors

These limitations are quite important with regard to construction of very high scale integration integrated circuits, where heat dissipation can be quite problematic and where at least efficiently cooling-able not even low power consumption is needed.

Although bipolar junction transistor technology have certain limitations, they were the first transistor technology, and they are still widely used for their qualities, such as more intuitive usage, current control, much higher immunity to high voltage static charges and etc.

Also in digital integrated circuits were commonly used in the past, TTL logic is actually based on them [7].

2.3.2 Unipolar transistor

Unipolar transistor, also commonly referred as field-effect transistor(FET) is a transistor, that uses an electric field to control the conductivity of channel with one type of charge carrier (therefrom *unipolar*).

Unipolar transistor technology is practically state of the art. Which is quite interesting, because the concept of the FET actually predates bipolar junction transistors [8].

These types of transistors are used widely in integrated circuits, but also in many other applications, where power efficiency and low heat production is required.

Unipolar transistor can be implemented in several ways, where each of them can be quite differ from others. By international standardization these symbols are commonly used to describe field effect transistors in schematics:

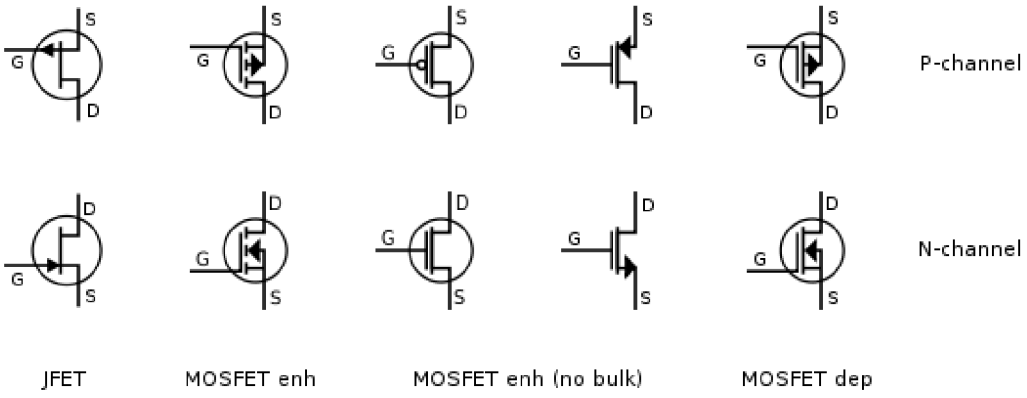


Figure 2.3: Field Effect Transistor schematic symbols [8]

JFET - junction field effect transistor

The JFET transistors are not so relevant to digital integrated circuits. Therefore we will not discuss them in more detail here.

2.4 MOSFET - metal oxide semiconductor field effect transistor

The MOSFET, which largely superseded the JFET and had a more profound effect on electronic development, was invented by Dawon Kahng and Martin Atalla in 1960 [5].

The growth of digital technologies like the microprocessor has provided the motivation to advance MOSFET technology faster than any other type of silicon-based transistor. A big advantage of MOSFETs for digital switching is that the oxide layer between the gate and the channel prevents DC current from flowing through the gate, further reducing power consumption and giving a very large input impedance. The insulating oxide between the gate and channel effectively isolates a MOSFET in one logic stage from earlier and later stages, which allows a single MOSFET output to drive a considerable number of MOSFET inputs.

Bipolar transistor-based logic (such as TTL) does not have such a high fanout capacity. This isolation also makes it easier for the designers to ignore to some extent loading effects

between logic stages independently. That extent is defined by the operating frequency in the way witch increasing frequency, the input impedance of the MOSFETs decreases.

But lets now take a look at MOSFET transistor and how it actually works.

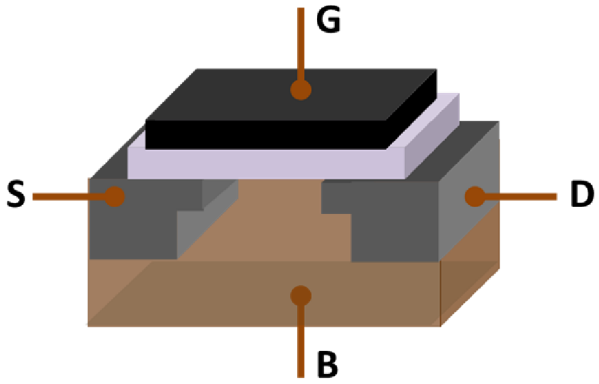


Figure 2.4: MOSFET structure S(source) G(gate) D(drain) B(body) [8]

The MOSFET is actually 4 electrode device, but quite often the Body electrode is connected together with the Source electrode(short-circuited internally) and only 3 electrodes or terminal are there to be connected within circuit, the gate, the source and the drain.

It is practically voltage controlled device, where the width of a conducting channel is reduced or enlarged by voltage applied on gate against the source electrode it is abbreviated as V_{GS}. The effect is different in P-channel and N-channel transistor but we will discuss that later.

The width of the channel is directly affecting conductivity between drain(D) and source(S) electrodes. Conductivity is an inverted value of the resistance and for this drain-source resistance we will use abbreviation R_{DS}. So with some simplification we can say, that rising V_{GS} is reducing R_{DS}. If V_{GS} is higher than V_T the threshold voltage transistor channel is conducting, when it is bellow the threshold voltage the transistor channel is mostly insulating drain and source electrodes with some minor problems, which we will discuss later. This is the primary function of MOSFET transistor as switch inside digital circuits.

The MOSFET can be used for amplifying or switching electrical signals. The usage as an analog amplifier require more linear input output characteristics and they usually differs from switching oriented MOSFET transistors. In this work we aim at MOSFET usage in the digital circuitry design, so we will concentrate on switching function of the MOSFET.

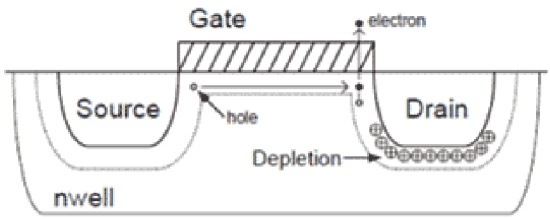


Figure 2.5: Field Effect Transistor structure [8]

2.4.1 MOS characteristics

In the area of electronic engineering there are lot parameters and characteristics which describe construction and behavior of a MOS transistor.

Those mentioned here are those of designer's interest.

Electrical Specifications

Parameter	Conditions	Symbol	Min	Typ	Max	Unit
Static						
Drain-Source Breakdown Voltage	V_{GS} = 0V, I_D = 250μA	BV_{DSS}	20	--	--	V
Gate Threshold Voltage	V_{DS} = V_{GS}, I_D = 250μA	$V_{GS(TH)}$	0.65	0.95	1.2	V
Gate Body Leakage	V_{GS} = ±8V, V_{DS} = 0V	I_{GSS}	--	--	±100	nA
Zero Gate Voltage Drain Current	V_{DS} = 16V, V_{GS} = 0V	I_{DSS}	--	--	1.0	μA
On-State Drain Current	V_{DS} □ 5V, V_{GS} = 4.5V	$I_{D(ON)}$	6	--	--	A
Drain-Source On-State Resistance	V_{GS} = 4.5V, I_D = 2.8A	$R_{DS(ON)}$	--	40	65	mΩ
	V_{GS} = 2.5V, I_D = 2.0A		--	50	95	
Forward Transconductance	V_{DS} = 5V, I_D = 2.8A	g_{fs}	--	6.5	--	S
Diode Forward Voltage	I_S = 1.6A, V_{GS} = 0V	V_{SD}	--	0.76	1.2	V
Dynamic[b]						
Total Gate Charge	V_{DS} = 6V, I_D = 2.8A, V_{GS} = 4.5V	Q_g	--	3.69	--	nC
Gate-Source Charge		Q_{gs}	--	0.7	--	
Gate-Drain Charge		Q_{gd}	--	1.06	--	
Input Capacitance	V_{DS} = 6V, V_{GS} = 0V, f = 1.0MHz	C_{iss}	--	427.12	--	pF
Output Capacitance		C_{oss}	--	80.56	--	
Reverse Transfer Capacitance		C_{rss}	--	57	--	
Switching[c]						
Turn-On Delay Time	V_{DD} = 6V, R_L = 10Ω, I_D = 1A, V_{GEN} = 4.5V, R_G = 6Ω	$t_{d(on)}$	--	6.16	--	nS
Turn-On Rise Time		t_r	--	7.56	--	
Turn-Off Delay Time		$t_{d(off)}$	--	16.61	--	
Turn-Off Fall Time		t_f	--	4.07	--	

Figure 2.6: N-MOS FET TSM2302 characteristics (from datasheet)

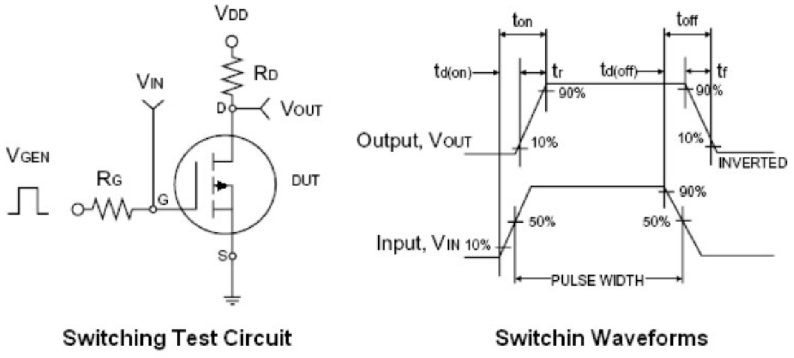


Figure 2.7: N-MOS FET TSM2302 switching characteristics(from datasheet)

Electrical Characteristics Curve (Ta = 25°C, unless otherwise noted)

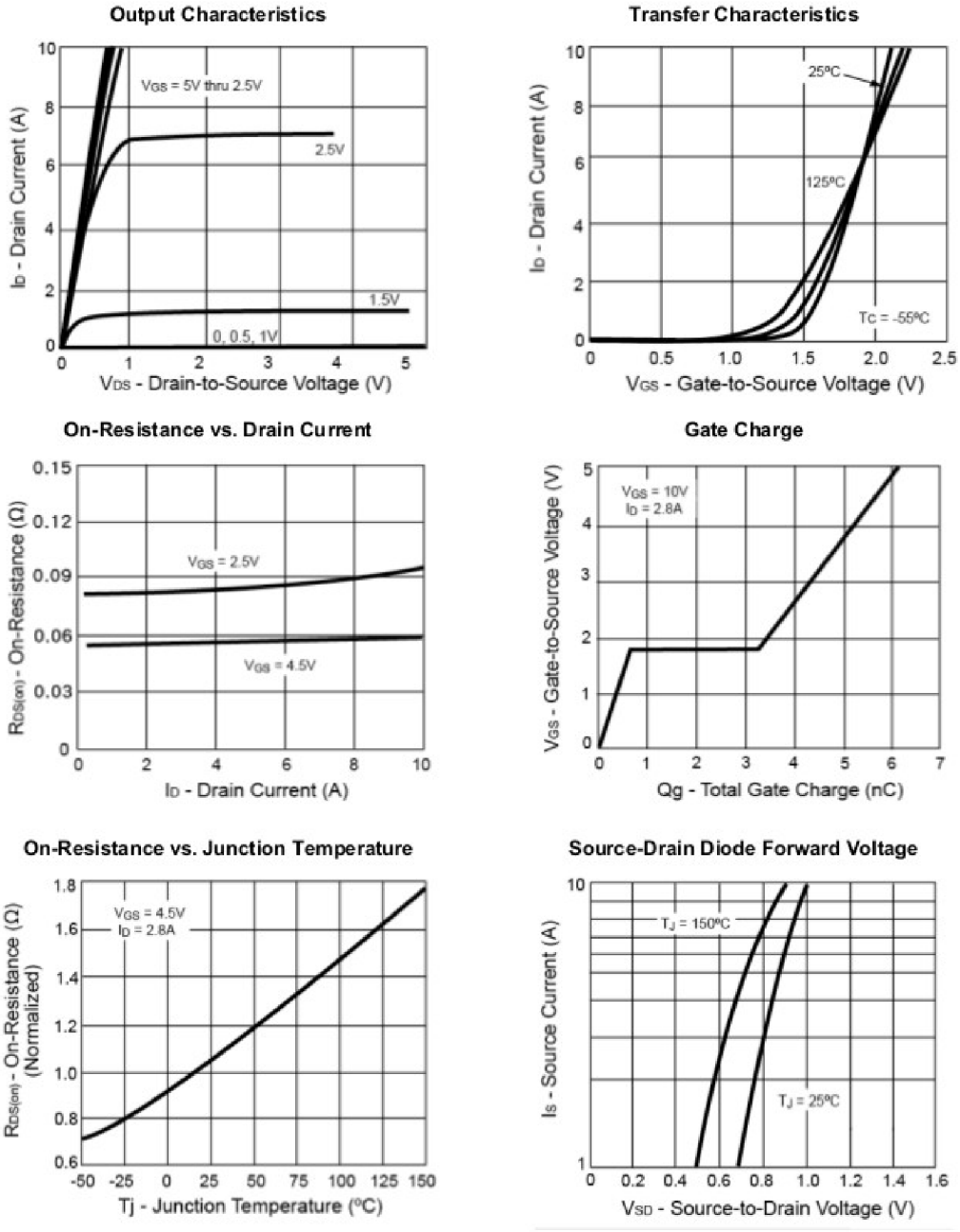


Figure 2.8: N-MOS FET TSM2302 graph characteristics (from datasheet)

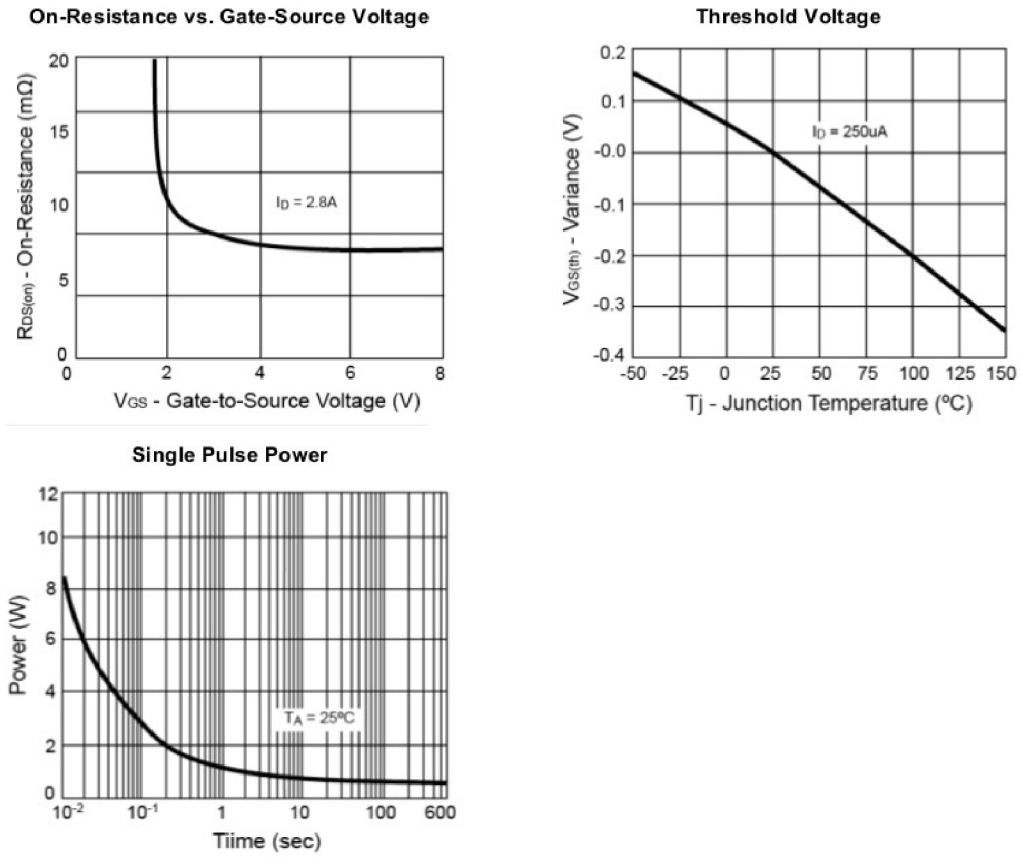


Figure 2.9: N-MOS FET TSM2302 graph characteristics (from datasheet)

But as long as the goal of this work is not in the selection of the best transistors for the integrated circuits, we will use abstract model of MOSFET transistor, which will be more than sufficient for the automated design process.

Quite complex model taking into consideration more specific characteristics is used by electronic circuit simulator mentioned in the simulation chapter.

2.4.2 CMOS

CMOS is abbreviation which stands for complementary metal-oxide-semiconductor which is state of the art at the digital integrated circuitry. CMOS logic use p-channel(PMOS) and n-channel(NMOS) MOSFETs as building blocks.

Overheating is a major concern in integrated circuits since ever more transistors are packed into ever smaller chips. CMOS logic reduces power consumption because no current flows (ideally), and thus no power is consumed, except when the inputs to logic gates are being switched. CMOS accomplishes this current reduction by complementing every NMOS FET with a PMOS FET and connecting both gates and both drains together. A high voltage(logical one) on the gates will cause the NMOS to conduct and the PMOS not to conduct and a low voltage(logical zero) on their gates causes the reverse. During the switching time as the voltage goes from one state to another, both MOSFETs will conduct briefly. This arrangement greatly reduces power consumption and heat generation. But as we will mention in following chapters, there can be made in some way better arrange-

ments than complementary MOS switch, such as pass-trough transistor used instead, saving transistor count, and even decreasing shortcut leakage during switching. [17]

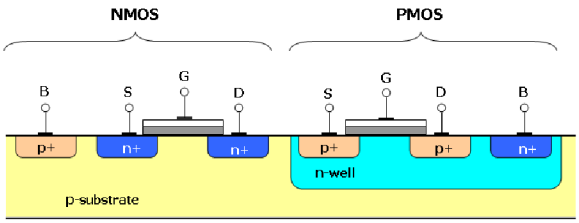


Figure 2.10: PMOS and NMOS transistors in CMOS gate [9]

Dual-type (CMOS) MOSFET switch

This „complementary“ or CMOS type of switch uses one P-MOS and one N-MOS FET to counteract the limitations of the single-type switch. The FETs have their drains and sources connected in parallel, the body of the P-MOS is connected to the high potential (V_{DD}) and the body of the N-MOS is connected to the low potential (GND). To turn the switch on, the gate of the P-MOS is driven to the low potential and the gate of the N-MOS is driven to the high potential. For voltages between V_{DD}–V_{tn} and GND–V_{tp}, both FETs conduct the signal. For voltages smaller than GND–V_{tp}, the N-MOS conducts alone, and for voltages greater than V_{DD}–V_{tn}, the P-MOS conducts alone.

The voltage limits for this switch are the gate–source, gate–drain and source–drain voltage limits for both FETs. Also, the P-MOS is typically two to three times wider than the N-MOS, so the switch will be balanced for speed in the two directions.

Tri-state circuitry sometimes incorporates a CMOS MOSFET switch on its output to provide for a low-ohmic, full-range output when on, and a high-ohmic, mid-level signal when off. [8]

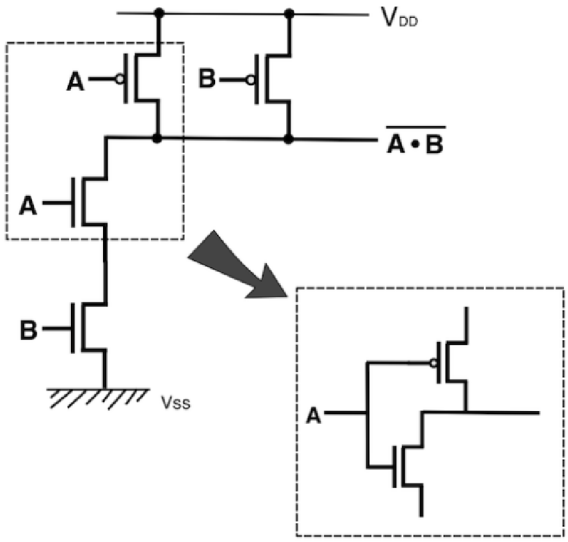


Figure 2.11: Convetional CMOS NAND gate, CMOS switch illustrated

2.5 Abstraction of transistor

There a lot of attributes which can define specific transistor, but in the design on upper level, for example gate level we do not care about that. It is because otherwise it would be overcomplicated to design anything. Here we will also need certain level of abstraction to work with, if we want to design something more complex than logical gates.

We could look on transistor as if it would be a switch. It is actually the role of transistor inside digital circuits. The voltage controlled switch, where drain and source electrodes are connected or disconnected based on voltage applied on the gate electrode.

Considering this, we could look at the PMOS and NMOS transistors as if their actually the same and only difference is the inverted function of their gate. Where NMOS transistor is opened by logical level HIGH (sufficient voltage applied on the gate) and PMOS transistor is closed at the same condition. PMOS transistor is opened at the logical level LOW (zero or low enough voltage on the gate) where NMOS is closed by that.

Using such high level of abstraction could possible work for small and simple enough circuits as the two input NAND gate (which can be created from 4 transistors).

But it would certainly encountered failure, when we would attempt to evolve more complex circuits. In this model they would work fine, but in real or realistic enough simulation they would fail at least because V_T loss problem.

V_T voltage loss problem

The voltage loss problem is characteristic thing of all electronic devices, nothing is ideal. Metal-Oxide-Semiconductor shows much higher efficiency here than older bipolar-junction transistor technology, but still certain voltage loss remains.

V_T - The threshold voltage of a field-effect transistor is the value of the gate - source voltage when the conducting channel just begins to connect the source and drain contacts of the transistor, allowing significant current.

Practically speaking the threshold voltage is the voltage at which there are sufficient electrons in the inversion layer to make a low resistance conducting path between the MOSFET source and drain.

If the gate voltage is below the threshold voltage, the transistor is turned off and ideally there would be no current from the drain to the source of the transistor.

In the real world, there is a current even for gate biases below the threshold called subthreshold leakage current, it is small and varies exponentially with gate bias, but it is there.

If the V_{GS} is higher than V_T we can say that transistor is conducting. How well depends in reality if it is PMOS or NMOS transistor and also it also depends if it should conduct the logical zero(LOW level) or logical one(HIGH level).

About P-channel MOS transistor, which is opened by logical zero at the gate, we can say, that it degrades logical zero value when it passes through drain-source.

On the other hand about N-channel MOS transistor, which is opened by logical one we can say, that it degrades logical one value when it passes through drain-source.

This behavior is quite the same as would happen on poorly opened transistors. Those which does not get the exact logical one or logical zero on the gate, but some voltage level between.

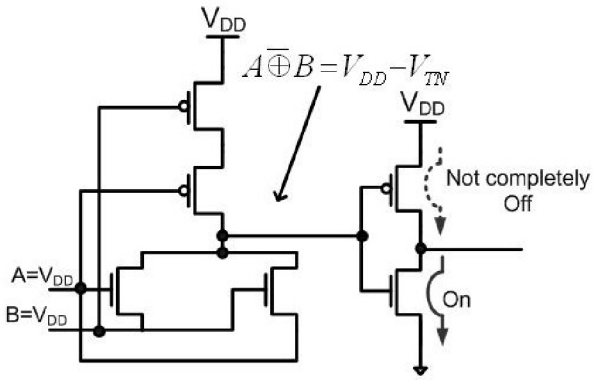


Figure 2.12: logical circuit illustrating problem [17]

This would not be so much of issue in the past, when relatively high voltage levels were used, such as 5V in TTL(Transistor - Transistor Logic).

But in these days, voltage levels used in complex integrated circuits such as CPUs are much smaller. For example ultra-low-voltage CPU Intel Core i7-620UM operates on 0.725 - 1.4 V at frequency 1.07 GHz, while it is manufactured by 32 nm technology.

PMOS abstraction

- opened by logical zero at the gate
- closed by logical one at the gate
- in fully opened state degrades logical zero (LOW)
- if logical level on gate is degenerated above selected threshold transistor is switched into partially opened state
- if the gate is disconnected (high impedance state = Z) transistor is set into X state, which is unknown but may behave in similar way as partially opened
- in the partially opened state degrades both logical one and logical zero

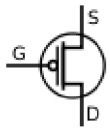


Figure 2.13: PMOS schematic symbol

NMOS abstraction

- opened by logical one at the gate
- closed by logical zero at the gate
- in fully opened state degrades logical one (HIGH)
- if logical level on gate is degenerated above selected threshold transistor is switched into partially opened state
- if the gate is disconnected (high impedance state = Z) transistor is set into X state, which is unknown but may behave in similar way as partially opened
- in the partially opened state degrades both logical one and logical zero

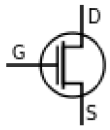


Figure 2.14: NMOS schematic symbol

Historical progress, integration scale rising

As the progress in the manufacturing of integrated circuits continue, the devices(transistors) are becoming smaller and smaller. When we are reaching the molecular level, the problems such as variation in transistor attributes can become more important. Using conventional methods it would be quite hard to design circuits, which can be more tolerant to these arising problems. But using automated design, simulation and evaluation of the circuits for example by evolutionary algorithms can bring new circuit designs more tolerant to intrinsic variation of devices [18].

- 1959 - single transistor : 1
- 1960 - logic gate : cca 4-10 transistors
- 1964 - SSI - small scale integration : up to 10 gates
- 1967 - MSI - medium scale integration : 10 - 100 gates
- 1972 - LSI - large scale integration : 100 - 1000 gates
- 1978 - VLSI - very large scale integration : 1000 - 100 000 gates
- 1989 - ULSI - ultra large scale integration : 100 000 and more gates
- late 1990s - SLI/SOC - more than 10 million gates

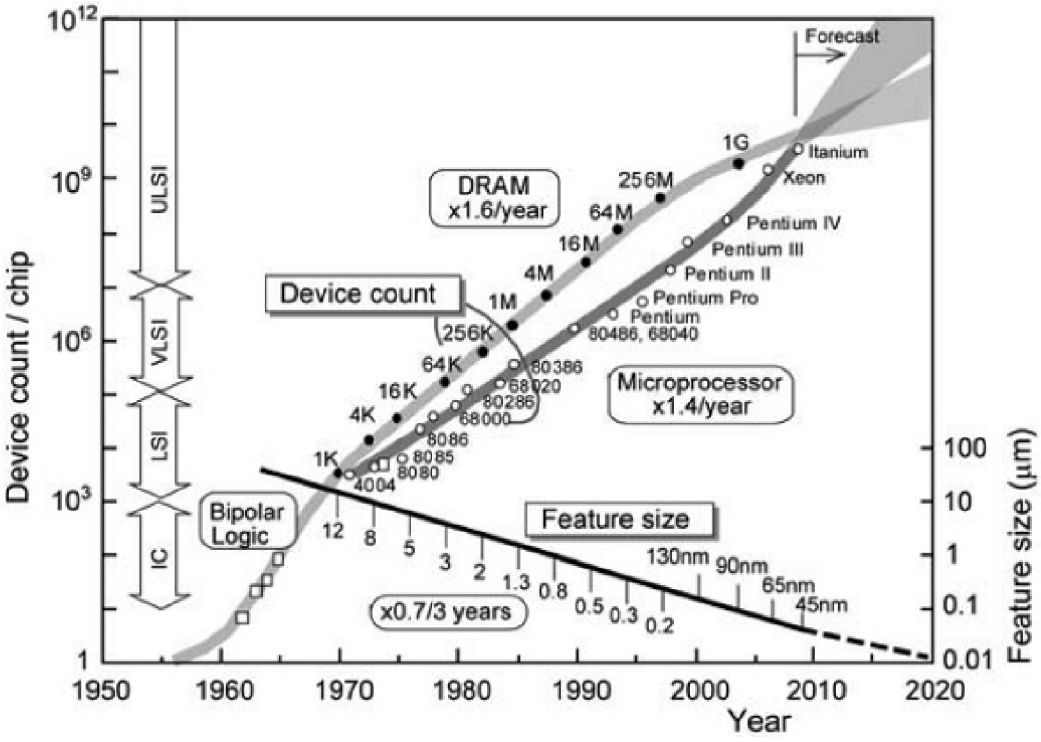


Figure 2.15: Integration scale progress [5]

Chapter 3

Review of previous work

Here we will review what was already done in the area of conventional and unconventional transistor-level design of VLSI digital circuits before this work. Albeit it is unusable to include everything what was done in this area, significant techniques and approaches are reviewed here, concentrating on CMOS technology. As far as is known to this day, conventional or even unconventional transistor-level design was used only to create or optimize relatively small digital circuits. Let's mention some of these circuit types:

- invertor (NOT)
- AND, NAND
- OR, NOR
- XOR, XNOR
- HA - half adder
- FA - full adder
- RCA - ripple carry adder
- CSA - carry save adder
- combinational multipliers
- and many others

When a design or optimization of certain circuit on transistor level is started, there are usually some constraints, which are to be met. Let's call them criteria of optimization. These criteria can be quite simple, such as minimal transistor count, or high reliability, variability tolerance and etc. [18].

On the other hand they can be also multi-objective, where fitness of proposed transistor circuit should met several requirements at the same time [14].

Let's list the important ones:

- minimal transistor count
- minimal area usage
- full output voltage swing
- variability tolerance
- low dynamic power consumption
- low static power consumption
- high maximal operation frequency
- minimal delay in critical path
- reliability
- hazard free solution
- and some others

minimal transistor count

This is quite simple criteria, usually it has direct influence on some other characteristics, conventional designs have usually certain almost minimal number of transistors, but they have few more than is achievable minimum, for several reasons such as design method technique or stabilizing circuit behavior and etc..

minimal are usage

Area or even volume covered by specific circuit, we can say that this metric directly correlates with transistor count, but it is definitely not the same one. For example due pathways interconnecting transistors.

full output voltage swing

PTL - Pass Through Logic - this is technique or let's say design approach where is signal passed through transistor, instead of applying signal on the gate of transistors controlling interconnection from V_{dd} or GND to an output (typical CMOS switch). This creates circuits which does not provide full output voltage swing, but can achieve for example lower power consumption and even eliminate shortcuts which can be experienced in conventional inverter design, when both transistors are switching at the same time, they can for very short amount of time interconnect V_{dd} directly to GND.

variability tolerance

When the manufacturing of transistor devices is reaching molecular level, differences between transistors in circuit are becoming more and more important. Here we can mention for example random dopant fluctuations, oxide layer thickness, line edge roughness and etc.. Variability tolerance means, that circuits designed with variability tolerance in mind, will work correctly even at the circumstances where transistor variability is higher.

low dynamic power consumption

Dynamic power consumption is experienced, when transistors are switching, usually it is result of input value change, clock line change and etc. The quite important here is elimination of brief shortcut leakage typical for CMOS switches.

low static power consumption

Static power consumption is experienced, when transistors are in a stable state, opened or closed. Higher power consumption at this state can be caused by control voltages too close to voltage threshold causing transistors to be poorly closed or poorly opened creating a leakage.

high maximal operation frequency

It is quite typical requirement. There are many things which have influence on maximal operating frequency of a circuit. Including power dissipation, capacitance, physical layout, signal delays and etc.

minimal delay in critical path

Minimal delay in critical path enables circuit to achieve lower latency and in some cases higher achievable operating frequency.

reliability

Reliability is quite wide requirement, it should be further specified to concentrate on described special cases. We can mention reliability handling thermal differences, input voltage variation, input signal variation, EM interference and etc.

hazard free solution

Logic hazards are manifestations of a problem in which changes in the input variables do not change the output correctly due delay caused by logic elements or at lower level, transistors. This results in the logic circuit not performing its function properly. The most common kinds of hazards are usually referred to as static, dynamic and function hazards.

others

After these optimization parameters of course there can be special requirements, such as

- fault tolerance
- limited radiation immunity
- wider range of operation temperature
- etc.

but this is an area exceeding this work, and this will not be discussed here in detail.

3.1 Conventional design of digital VLSI circuits

To meet some of earlier mentioned requirements by conventional way of designing integrated circuits it was great challenge for circuit designers, especially at the transistor level of design.

Conventional designs are usually optimized only for one or very few criteria at the same time. For example the work oriented on design of low power adder, which were meant to be used in combinational multipliers [17] mentions designs which size is varying from 10 to 12 transistors. Optimization criteria were aimed on low power consumption, minimal transistor count. After designs were finished the comparison between designed circuits also included delay and output voltage swing additionally to original 2.

It presented interesting approach called PTL(Pass Transistor Logic) where pass transistor is used instead of CMOS switch.

Also identification of V_T loss problem was helpful for realization what causes failures of evolutionary designed transistor level circuits using naive switch level fitness evaluation.

Although it achieved some level of success, it illustrates the need of hierarchical approach in the conventional design. The mentioned work aimed higher in complexity than typical conventional design on transistor level, which usually end at gate circuit design. But it reaches its limits at the complexity designing 1-bit full adders.

3.2 Transistor-Level Evolution of Digital Circuits Using a Special Circuit Simulator

An evolutionary algorithm was used to design digital circuits at the transistor level. Various static CMOS circuits which used up to 4 inputs were evolved. The usage of specialized circuit simulator, which worked with quite abstract representation of a transistor allowed to search through space of solutions faster than it would be achieved by conventional SPICE simulator [19].

In order to quickly evaluate candidate circuits, new simulation tool was developed, allowing speed up at two orders of magnitude compared to conventional SPICE simulator. But at the end, where the best candidate solutions were chosen, SPICE was needed to evaluate circuit behavior in more realistic manner. During this it was found that some of designs were not functioning properly.

Due this problem, some special expert knowledge was used to restrict search space for candidate solutions also by special representation of candidate solutions. Restrictions which should provide more reliable solutions were following:

- gates of transistors are connected only to circuit input
- at least 1 connection to Vcc
- at least 1 connection to GND
- source terminals can be connected to other sources, drains, Vcc or GND
- drain terminals can be connected to other sources, drains, Vcc or GND
- it is not possible to connect PMOS to GND

- it is not possible to connect NMOS to Vcc

But it is questionable, if these restrictions are not too limiting in the aim for more complex circuitry designs.

It also was found, that it can be useful to provide inputs in direct way accompanied by their complements (double size of input vector).

The scalability of representation and scalability of fitness calculation were identified as major problems of evolutionary approach.

Proposed simulation method

The specialized proposed simulator uses quite abstract representation of MOS transistors, where NMOS

- has 3 electrodes
- has infinite impedance between drain and source when logical zero at the gate
- has zero impedance between drain and source when logical one at the gate

and for the PMOS last two are inverted in the manner:

- has infinite impedance between drain and source when logical one at the gate
- has zero impedance between drain and source when logical zero at the gate

Due further investigation we find this model too much abstract and we decide to apply some more details into design of our own transistor representation.

Proposed circuit representation

The representation was inspired by Cartesian Genetic Programming. Candidate is represented as a string of integers with direct genotype-phenotype mapping [13] [11].

Using following encoding:

- GND 0
- Vcc 1
- First Circuit Input 2
- Second Circuit Input 3
- n-th Circuit Input n+1
- k-th transistor terminal G (electrode) n+1 + k*3
- k-th transistor terminal D (electrode) n+1 + k*3 + 1
- k-th transistor terminal S (electrode) n+1 + k*3 + 2

This representation does not show any major flaw, so it could be adapted by this work as potential candidate solution representation.

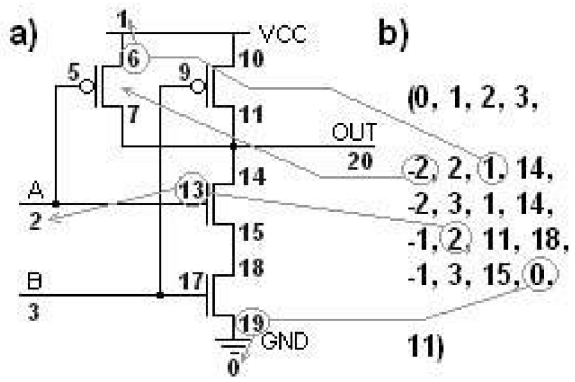


Figure 3.1: Circuit CGP representation a) Typical 2-input NAND gate. b) Chromosome representation of 2-input NAND where the quadruples in the middle represent individual transistors - grey circles connect selected genes and their corresponding transistor terminals while arrows indicate the connection of the terminals according to genes values [19].

Fitness function

Candidate circuit is evaluated for all possible binary combinations of primary inputs. The results are compared to the reference output defined by a designer [10].

If generated output matcher the reference output, the fitness value is increased by 1.

If the value matches only partially for example weak 1 instead of strong 1 only 0.75 is added to fitness value.

High impedance (Z) at the output adds only 0.5 to fitness value.

And undefined (X) value subtracts 0.5 value from fitness.

Rare output combinations

In some special cases it is hard to reach solution using only this standard fitness evaluation.

For example we will take a look at 4-input NOR gate. This circuit is special in the way that only for one input combination output is different then for rest 15 input combinations.

Evolution usually finds the partial solution in case of example 4-input NOR gate. Therefore we need to encourage the evolutionary algorithm into finding the last part, for example by specially adjusted fitness function which put more weight into this rare situation (0000 resulting into 1).

In original paper there was also mentioned usage of some virtual/fake constant input supporting the search for full solution of the rare combination problem.

Chapter 4

Simulation of circuits

Precision of physical simulation versus computational time, that is one of the trade-offs which we will discuss here. In the past due memory and computational power limitations gate-level simulations have been usually performed for larger circuits. Although some other methods have been developed. We do not live in times when computational power was a big issue anymore, but still we look for efficient way to simulate digital transistor circuits. When we would need to simulate only few (dozens) digital circuit at a transistor level we could use very precise simulations such as are those provided by SPICE software. So why we care here so much about efficiency and computational time ? Of course everybody likes fast and efficient programs but the point here is, that if we want to use automated design process, the algorithm needs to evaluate circuit practically after each change in design. And now imagine, that we use evolutionary algorithm, more precisely genetic algorithm or Cartesian genetic programming. It means, that computer performing the algorithm is not very far from blind search inside possible solutions space. There is very little of expert knowledge put inside design algorithm and it produces even hundreds of candidate solutions each generation(iteration). Where generations needed to achieve proper and optimality close solution can go to thousands and even tens of thousands.

Simulation at a gate-level for MOS(metal oxide semiconductor) technology is not very appropriate level of abstraction. MOS technology circuits may contain ratio-ed logic and pass transistor experience bidirectional signal flow, transistors may also exhibit charge sharing effect and etc.. Gate-level simulation will show completely unacceptable at modeling MOS technology failures adequately.

Therefore switch level simulation presented by Bryant was created and used, but in 1993 was encountering computing effeciency problems due high memory requirements.

4.1 Idealistic simulation

In the case of gate-level simulation, we would be able to perform great part of evaluation just using binary states(0/1). And when we would like to go little further, we could use (X) as representation of uninitialized, unknown or unconnected value.

4.2 Simulation used by Žaloudek

In order to quickly evaluate a candidate circuit, a circuit simulator was developed which works at the level of simplified models of PMOS and NMOS transistors [19].
The simulator operates with six logic levels (ordered by voltage to GND):

- strong 1
- weak 1
- Z
- weak 0
- strong 0
- X

The simulator works directly with the proposed circuit representation which was mentioned earlier in the chapter (3).

Propagation of the signal

At the beginning of simulation, all values of source/drain/output terminals are set to high impedance and the primary inputs are set according to a given training vector. First, the path of the strong 1 signal is followed from Vcc node through all the connections.

The values on the terminals on the way are updated to strong 1. Drains and sources are considered to be identical from the function point of view. Both sides (drain, source) work identically from the viewpoint of microelectronics.

When the signal reaches a transistor source or drain the algorithm checks the transistor state (logic signal on the gate) and updates its state according to the rule table developed for the simulator. The rules reflect the fact, that strong 1 degrades on NMOS or poorly open transistors. On the other hand, strong 0 degrades on PMOS or poorly open transistors.

If the algorithm recognizes that the transistor is open in some way, it propagates the signal to the other side if it is possible. Note that there is no need to propagate it when the signal on the other side is stronger or identical. If the algorithm finds the opposite value or undefined valued on the other side, a short circuit is encountered.

That means, that the output is set to undefined value and the process proceeds with another training vector. Otherwise, the signal is followed until it propagates. Then, other signals are processed in the same way (of course, logic 0 propagates from GND) in the following order: weak ones, strong zeros and weak zeros. At the end, the values on the output terminals are updated.

Problem of this method

The proposed method of simulation is practically event based one. Event base simulation cannot handle cycles in the circuit by itself. It cannot even identify them. Because of this the method can effectively works only using restrictions mentioned earlier, such as that gates of transistors can be connected only to primary inputs and etc.

4.3 Multi-level simulation - hierarchical switch level

Due high memory requirements of switch-level simulation, creating a slowdown in 1993. Authors mentioned that circuit regularity and hierarchy must be exploited [15]

In a bottom-up design system, identification of regularity and its exploitation is a difficult problem.

In a top-down design, regularity and connectivity of circuit are given. Exploiting those is similar or even equivalent to exploiting the design inherent hierarchy. This can be useful when large and designs too complex to be handled in one level. Just try to imagine simulating the whole processor at a transistor-level. Quite a nightmare, isn't it ?

Therefor transistor-level blocks are simulated at the switch-level. Using the notation of Bryant, set of discrete values $S_T = \{s_k, s_{k+1}, \ldots, s_{k+l-1}\}$ is used to describe transistor strength levels.

Two types of nodes: input nodes - V_{DD}, Gnd, primary circuit inputs - assumed to have infinite drive capacity s_I

storage nodes - all other nodes, which get strengths assigned from $S_N = \{ s_0, s_1, \ldots, s_{k-1}\}$ reflecting their capacitance to ground values.

Each node gets assigned state. State consists of two values

- logic value - element from set {0, 1, X}
- signal strength - value from ordered set $S = \{s_0, s_1\}$

Also two operators are defined

- - strength of node
- - strength of transistor

A switch level circuit can be described as the directed graph, where: vertices - corresponds to nodes in a circuit edges - corresponds to transistor channels

Each component of a graph can be solved independently of other components N-type transistor is understood to be conducting when its gate has logical 1 state. P-type transistor is understood to be conducting when its gate has logical 0 state. D-type transistor is understood to be conducting always. N/P-type with X state at its gate is understood to be potentially conducting.

Some ideas from this simplified simulation method description are used in following methods.

4.4 Event based simulation

Intuitively we could come up with idea to use event-based simulation. Idea is relatively simple. We can represent circuit as a set of nodes. Nodes are interconnected with each other through transistors. Input nodes comes with specified values(0/1) or for fault testing they can be tested with unspecified value(X).

When input nodes are initialized, algorithm creates a list of transistors connected to the first node by its gate. We can spread the value of the node to everything connected to it. And after that we spread the value again and again, updating all nodes within reach of signal.

And if simulated circuit uses unlucky interconnections algorithms will end up in loop. That is very unfortunate, because detection of such situation is quite complicated. We would be capable of detecting relatively simple loops, by retracing steps of the algorithm, searching for repeated steps. But it is not very systematic and also not very computationally effective. Not speaking about encountering and dealing with more complicated loops which would slow down algorithm to unusable levels.

This type of simulation can be successfully used for many areas, but keeping in mind, that we want to simulate and evaluate circuits practically generated by random(evolution), where loops can occur quite often it would be actually very unfortunate to use it. It could be usable for very small circuits less than 10 transistors but even there, it would be reaching its limits.

4.5 Switch-level simulation

Typical switch level simulation, where MOS transistor is replaced by the voltage level controlled switch between drain and source electrodes, can be used as acceptable simple simulation method for certain cases. But for evaluation of something more complex, and possibly problematic such as automatically generated unconventional circuits it would need some enhancements.

4.6 Circuit signal path-finding simulation

We propose relatively simple simulation method based on graph circuit representation.

We can use abstraction look at the transistor and see it as interconnection of two segments, where one segment of circuit(graph) is connected to the drain and other is connected to the source electrode. Segments are connected when the transistor is open and disconnected when transistor is closed.

If everything goes fine and circuit is electronically correct, practically all conventional designs of logical circuits, we start witch identifying possible paths from output into any signal source.

Signal source can be:

- VCC
- GND
- primary (circuit) input

When we search for the paths at first, we see all transistors as opened, interconnecting their source and drain segments.

4.6.1 Electronic validity

We should try to evaluate if circuit is valid from electronic point of view, for this moment ignoring input vectors and reference output. We need to find out if there is possible shortcut inside the circuit. To do that, we will search for:

- path from VCC to GND
- path from any primary input to VCC
- path from any primary input to GND

When such path is found all transistors which are on this way should be putted into list coupled with this path. For each transistor we need to evaluate his condition, if it is opened or not. That can be done only by tracing connection from the transistor gate to any signal source. When the source.

At the end when all transistor gate paths are found, and there is no cycle. For example all transistor gates are connected directly to the primary inputs of the circuit(as is shown on following scheme).

We need to check if there is combination of variable states(inputs) which would lead to shortcut. To do that, equations need to be formulated from the circuit path. The serial connection of transistors put their gate variables into logical AND relation and the parallel one into logical OR relation.

Equation formulated for showed circuit possible shortcut would be following:

$$shortcut = A.B.C \tag{4.1}$$

If all combinations of input variables are possible or at least if combination A=1, B=1, C=1 is possible than this circuit should be disqualified completely or there should by strong penalty for this candidate solution when evolution design process is used.

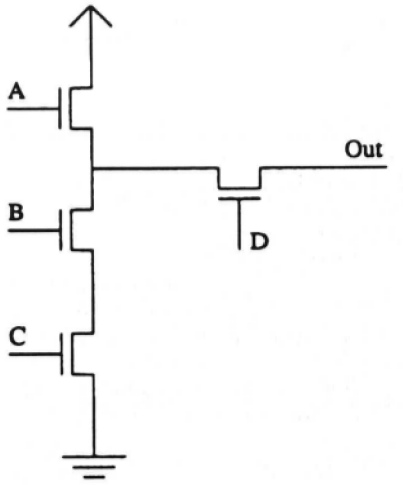


Figure 4.1: primitive shortcut able circuit

4.6.2 Signal path evaluation

If everything went fine and circuit does not contain shortcut from VCC to GND and is otherwise electronically correct, we start witch identifying possible paths from output into any signal source.

For the signal path finding for example BFS(Breadth First Search) algorithm can be used.

Where there is no path to any signal source from the circuit output, there output will be evaluated as high impedance (Z) for all input combinations.

For example lets say, that input $B = notA$, it means, that circuit cannot experience shortcut now. So to evaluate signal path we now build equations for each path to signal source.

- shortcut = A . not A . C => impossible
- out = 1 if D . A = 1
- out = 0 if B . not A = 1
- out = Z if not D + not A . not C = 1

When the first path is found, we could try to go deeper and check if there is another path connecting output into different signal source.

When the cycle/loop is identified during signal path search the signal has to be set as undefined (X) for the resulting output.

When this paths are found and circuit is evaluated using ideally conducting transistors and for each input vector is computed the result of the circuit output. The output of circuit is compared witch requested referential one. For each match fitness value of the circuit candidate is increased.

4.6.3 Imperfection consideration

When we have equations and paths found in the circuit. and we also have idealistic fitness value of the circuit. It is time to start take imperfect conductivity of each type of transistor into consideration.

For each NMOS leading to VCC or logical one from primary input there should be penalty applied and the same should happen for each PMOS leading to GND or logical zero from primary input there should be penalty applied too.

This method allows us also quickly evaluate circuit for shortcuts and even power leakage by finding the way from Vcc to GND, or from Vcc, GND to primary input.

4.7 Specialized SPICE simulator

For high precision simulation quite complex simulator should be used. High precision but usually brings also relatively long computational time. Because of this, a complex electronic circuit simulator such as SPICE is, should be used only there, where it is computationally affordable. Using SPICE for the fitness evaluation of each generated circuit by evolution algorithm is actually possible but it is far from computationally efficient solution.

Therefor SPICE (i.e. ngSPICE) should be used only for subset of generated circuits. This subset should be determined by faster simulation method proposed earlier in this document. Only circuit solutions which were successfully evaluated by faster more abstract simulation should be placed into this subset.

Chapter 5

Evolution

5.1 Evolutionary computation

Evolutionary computation can be described as usage of evolution principle for solving computational problem. Solving computational problems which are not characterized as easily solvable just by applying deterministic algorithm may be solved by artificial intelligence. Artificial intelligence is something what can be tricky to define. Of course we have Turing test definition and many other definitions, which usually correlate with each other, but they also differ one from another. For the simplicity sake, we will describe artificial intelligence trough its observable behavior. We may consider to look at certain activity which is usually described by sequence of actions, as manifestation of artificial intelligence. If such activity would be performed by human and we would in that case consider it as expression of intelligence. This leas us to question, if we should consider evolution to be part of artificial intelligence. It is usually categorized as such and there are some reasonable arguments for this choice [16].

Now we will take short look at natural inspiration for evolutionary computing.

Natural evolution

Evolution as itself in natural conditions may have some goal or it may not. It depends on point of view. But this is something we will not discuss further and we will concentrate on evolution as inspirational process for solving tasks, which were usually solved by engineers or by other approaches from artificial intelligence area.

According to evolutinary theories based on neo-darwinism, natural evolution chooses such individuals. Which are offspring of parents which have successfully accomplished two tasks, to survive long enough to reach reproduction age, and then reproduce itself [12]. How well they accomplish this task is something we mark as their *fitness*.

The offspring are almost exact copies of their parent in case of asexual reproduction, e.g. bacterial reproduction. Or they share part of each of both parents. Introducing *crossover* as useful genetic operation. Even so in both cases, process of copying is never perfect and some errors are always present. We call them *mutations*. If such mutations do too much damage to organism, it will usually die before it reaches its reproduction age. If it brings some beneficial feature, for example organism will grow for example about 10% faster, it will reach reproduction age earlier, and may have more offspring it increases the organism's fitness. Usually many mutations are mostly neutral. It means, they do not

affect the organism's fitness in any way. But together with some other mutations maybe few generations later it might.

This is very simplified description but as a principle introduction of evolution it should be sufficient. So now we know something about natural evolution. Lets take a look how it can be used for solving engineering tasks.

5.2 Evolution as a design process

Evolutionary computation can be used in several areas where usually engineer do the work but using very conventional design methodologies, which can produce designed systems far from the optimal solution.

That is the goal of this work, to provide system, which would be able to optimize and design digital circuits on lower level, the transistor level, where engineers encounter their limits and usually return back to design on the gate-level. When finished, gates are just replaced with their conventional transistor designs each alone, missing the possibility to optimize circuit in whole picture as a transistor circuit.

Algorithm

Evolutionary algorithm is term, which covers set of stochastic search algorithms. This set of algorithms has following features in common:

- they use population of candidate solutions
- they use approach for creating candidate solutions which is inspired by biology

At the beginning of of evolutionary algorithm initial population is created. This population contains preset number of candidate solutions.

The initial population can be created by random process or it can be created by proper heuristics. For example it can contain already known solutions or partially correct ones.

In each step (generation) of evolutionary algorithm are all candidate solutions evaluated by fitness function.

Fitness value represents how good candidate solution is. In evolutionary biology it represents its ability to survive and reproduce itself.

Each new population is created by following approach. At first proper candidates are selected from current population. Then those candidates will form set of parents for new population. New candidates are created from parents by applying genetic operators such as crossover and mutation. Then new populations is selected from parents and new offspring. How it will be done depends on used selection algorithm. If this algorithm is designed properly, then average fitness of population will be increasing.

Due usage of fitness function selection pressure occurs in the process. This leads search into more convenient areas of search space.

The application of selection algorithms and genetic operators are not deterministic. They works with embedded randomness, which can be set or controlled by user.

Algorithm is terminated if solution with sufficient fitness was found or if maximal number of generations were exceeded.

```
Generic evolutionary algorithm

t = 0;
P(t) = create_initial_population;
evaluate P(t)

while(not finished){

  Q(t) = select_parrent( P(t) );
  Q_new(t) = procreate_new_offpring( Q(t) );
  evaluate_population( Q(t) );
  P(t + 1) = select_individuals_for_new_population( P(t), Q_new(t) );

}
```

Fitness value can be expressed in different ways. The most generic is raw fitness, which is expressed in numbers natural for specific problem domain [13].

Fitness value can be also standardized, in a way that smaller number means better solution. It can be also normalized into interval $< 0, 1 >$ When normalized fitness is used, than fitness value of better candidate is always higher than worse one. And summary over all candidates is equivalent to 1.

The quality of evolutionary algorithm is strongly influenced by problem encoding, design of fitness function and by used genetic operators.

In the typical optimization problem parameters of optimized function represent directly candidate solution. Because we will not use evolutionary algorithms only for optimization but also in design process. We should take a look at possible problem representations. Encoded candidate solution in area of evolutionary computing is usually called *genotype* or chromosome. Chromosome is composed from sequences of genes. If we use binary encoding, then chromosome is practically sequence of bits. Where each bit has a function of gene. The most common is usage of binary, integer or floating point representation or even its combination. In genetic programming are usually used more complex structures such as graphs for example.

For evolutionary design it is appropriate to distinguish between genotype space and phenotype space. Phenotype is directly equivalent with the object of evolution design. Where genotype may represent it in indirect way, for example in sequence of instructions how to construct such object. But it is also possible to use direct representation where genotype and phenotype are equivalent.

5.2.1 Variants of evolutionary algorithms

Here we will mention some of commonly used variants of evolutionary computation:

- evolutionary algorithms
- evolutionary strategies
- evolutionary programming
- genetic programming

As long as we need only introduction to evolutionary computation here, the following text will concentrate mainly on methods useful for digital circuit design.

For more information about evolutionary computation in general, for example this publication can be consulted [10].

Genetic algorithm

Genetic algorithm was designed by John Holland during process of artificial system adaptation research. Its basic variant have the same structure as generic evolutionary algorithm, which was mentioned earlier. Candidate solution is represented by chromosome of constant length. The initial population is generated randomly. Candidates selected by selection algorithm are then based on certain probability modified with genetic operators: crossover and mutation. It can work in generation oriented variant, where only new candidates are selected to new population. Or it can also work in generation overlapping mode, if candidates from previous and current population are both present in the new population.

Evolutionary strategies

Evolutionary strategy is method which was first used for optimization of complicated engineering tasks in area of aerodynamics. For this method is typical, that it optimizes vector of floating point parameters. The basic variant of evolutionary strategy uses only mutation operator. Mutation here uses Gaussian layout of probability and values generated according to this layout are added to parameter values of selected parent. Crossover is rarely used in this method.

Evolutionary programming

Evolutionary programming have many features in common with evolutionary strategies, although it was designed independently by Lawrence Fogel which used mutation based approach for evolution of predictors, which were implemented as finite state machines. Evolutionary programming typically uses application specific representation of problem, auto-adaptation and tournament selection. Crossover is usually not implemented.

Genetic programming

Genetic programming was invented as one of evolutionary algorithms and it was further developed mainly by John Koza. The purpose of this method is not only to find optimal parametric values, which are encoded in a chromosome, but automatically generate whole programs.

The original work was done on programs implemented in functional programming language LISP, which is very suitable for work witch tree-like structures.

Algorithm of genetic programming is in principle the same as one mentioned as generic evolutionary algorithm. But the representation is something very different. Genetic programming works with executable structures, most commonly programs represented like tree graphs.

Genetic operators, mutation and crossover are both used here. Also some new problem specific operators can be used, for example subroutine construction.

For the fitness evaluation candidate program is executed for defined set of inputs. The program outputs are compared to referential output definitions and fitness is computed.

5.2.2 Cartesian genetic programming

Cartesian genetic programming was first time mentioned by Miller and Thomson in 1999. It is variant of genetic programming, where candidate solutions are represented as generic or oriented graphs.

It uses usually direct genotype-phenotype representation of the circuit elements and their connections. This is in contrast to building a tree based on indirect genotype representation of the result, which is common for genetic programming.

According to [11] CGP can be used for many tasks, such as: Machine Learning, Neural Networks, Artificial Intelligence, Data Mining, Financial prediction, Function optimization, Classification, Electronic circuit design, medical diagnostics, evolutionary art and music, etc., the list is endless.
But it was invented by Julian Miller and was developed from a representation of electronic circuits devised by Julian Miller and Peter Thomson developed a few years earlier [4]. Which makes it perfectly tailored for our transistor-level designing system.

To encode transistor level circuit we could use several possible representations for evolutionary algorithm. Encoding could be using:

- direct representation
- constructive representation - instruction how to build a circuit
- hybrid representation - combinations of direct and constructive representation

Really used representation of circuit is in detail described in following chapter about implementation.

It should be mentioned that also evolutionary process can be done by very different ways with regard to, what type of simulation is done. Evolution can be as:

- extrinsic - circuits are be simulated by software
- intrinsic - circuits are represented and simulated in FPGA or FPTA
- mixtrinsic - combination of both

We will focus on evolution in software, which have certain benefits, such as scalability, selectable level of precision but it also have certain drawbacks such as possibly longer simulation time or too high abstraction, and it can invent solutions, which may not work in real world.

Chapter 6

Implementation

The main features of implementation will be described here. We will not discuss every detail but we will focus on important parts. The goal of this project was to design and implement evolutionary based designing system for transistor circuitry design. The implementation consists of several important parts.

- path search algorithm
- circuit simulation
- shortcut detection
- fitness evaluation
- evolutionary algorithm

Because of computational efficiency was one of the primary requirements on the system C++ programing language was chosen for the implementation. The C language was also considered, but due high complexity of the system it would not be proper choice for code maintainability.

It is maybe not so obvious, but circuit simulation is probably the most complex and the biggest part of implementation. Circuit simulator implementation can be quite challenging task just by itself and when we need to make it efficiently cooperating with evolution it is even more challenging. Circuit simulator which does not need to cooperate with evolution can be slow and precise. But both of these, slowness and precision, work against efficient evolution process.

The reason against slowness is quite obvious, we need to simulate and evaluate hundreds of thousands candidate circuits, so the time consumption is critical.

The reason against precision is little more intriguing, precision usually goes hand-by-hand with slowness, this is true, but it is not what we have in mind. When high precise simulation is used the area of possible solutions is highly limited. Of course it is limited to circuits with the highest probability for correct behavior in reality. It is something we need, but we need it usually near the end of evolutionary process. When we use this precise approach lot of possible candidates are discarded. Lot of those, which could lead by mutations to efficient and in reality correct solution. Where higher precision have tendency to push evolution algorithm to local minimum and sometimes even bring evolutionary process closer to random search like behavior.

Also high precise simulation can bring us solutions, which will work correctly in simulation and even in reality with technology used in simulation, but it may not work with different physical size of transistors, different physical layout, capacitance or frequency.

So now we should have some idea why we will work more with heuristics and abstraction than real world simulation. The proper level of abstraction is the critical part which determines if the evolution will find solutions in acceptable time but also if found solutions will work in chosen MOS technology.

6.1 Circuit representation

Circuit representation could have been done by many ways, we could choose to use direct simple encoding, where circuit representation would be exactly equivalent to its chromosome representation.

Let's say it would be possible to encode each transistor as four integers or even bytes, where numbers would be equivalent to:

```
transistor[4] = {t_type, t_drain, t_source, t_gate };
circuit[transistor_count * 4] = {t0_type, t0_drain, t0_source, t0_gate,
  t1_type, t1_drain, t1_source, t1_gate, ...
```

This simple encoding can be quite efficient for simple numeric operations, random access, and practically all parts of evolutionary algorithm, except for the candidate fitness evaluation.

Typical operations done with chromosome, such as copy, mutation, or in some cases crossover are usually quite cheap when we count computational time. The fitness evaluation is the time consuming part of the process. This mean, that we should use representation more suitable for fitness evaluation in other words, for circuit simulation.

We have decided that it would be wise to implement circuit as object instead of using 1:1 chromosome - circuit mapping. There are several arguments to support this approach.

If we would not chose this object oriented approach there would have to be many structures mapped to each other simply by indexes. This does not sound too bad, but with rising complexity of evaluator it would make upgrades or different regimes quite difficult to implement and maintain.

When we have circuit represented as object, we can also efficiently store everything what was already computed: referential function table, input output table, shortcut paths, output signal paths, inside nodes signal paths, fitness, used transistor count, number of output bit degenerations, etc.

With those informations already computed and stored we can save some computational time when we will need them in future. We can even take this approach further, we can concentrate on creation of next generation but more importantly evaluation of that generation and maybe we can save some computational time there. Let's see why this may be possible. Based on earlier mentioned evolutionary algorithm - Cartesian genetic programing, system creates next generation as mutated offspring of the fittest from the last generation. Practically each candidate in new population is copy of the fittest candidate. After copy is done, then mutation is done. The impact/influence of mutation is based on currently chosen maximum number of mutations. But basically only minor part of circuit is changed by mutation, it is the idea of evolution. This characteristic feature can be exploited. Let's take a look on how is each new offspring created and evaluated:

```
new_offspring = this->copy();

new_offspring->mutate( mutations );

new_offspring->compute_paths();

new_offspring->compute_fitness();
```

When circuit object is being copied, already computed signal and shortcut paths are copied also, or not, based on internal setting variable/compiler directive. Copying the paths is quite computationally cheap, but without cooperation with mutate method it would be pure wasting of computational time, because after that computed paths would be dropped after mutation and computed again, even if almost all of them would be the same. This leads to optional upgrade of mutation method, which would mark transistors which have been mutated, store their previous connections and efficiently alter only paths which are affected by mutation. The implementation of this upgrade is more challenging than it may look like, it is because of plenty evolution modes implemented in system. Sometimes `compute_paths()` from scratch can be faster than copying them and trying to handle differences, this may occur quite rarely but in such situations or in situation which could result in looping dependencies differential approach may be replaced by standard `compute_paths()` from scratch. More details about mutations and path computing is mentioned in following sections. Now we will take a look at transistor, the only building block of digital circuits, and its representation in designing system.

6.1.1 Transistor representation

Transistor by itself is not particularly interesting anymore, but it is more interesting as part of circuit. This is the point of view which is taken in this designing system. Each transistor is represented by object `transistor` and it is associated with circuit trough pointers. Transistor object has several methods but the important core of the representation is:

```
transistor number
transistor type  {N, P, DISABLED, WIRE }
drain node
source node
gate node
```

This is the necessary part for circuit construction. Internally is stored little more information, mainly to achieve more efficient simulation, for example these informations:

```
transistor state  {OPEN, CLOSED, X_STATE, ...}
last gate state {0, 1, Z, X, ...}
```

Now would be probably the right time to explain, how are transistors connected and how they form circuit.

6.1.2 Nodes

In many implementations of CGP in digital circuit design area, gates or transistor are connected to each other directly. It means that each electrode of each transistor or gate is directly connected to special 'nodes' such as Vcc, Gnd, In, Out or directly to electrode of another transistor/gate. This results into situation, where complexity of search space is directly derived from number of transistors. Although it may be modified by use of CGP grid and limited connection possibilities.

We have come with different approach. The circuit is also represented by interconnected transistors, but not directly with each other. Transistors are connected to circuit nodes. The Vcc, Gnd, In and Out are also nodes but there are also regular circuit nodes without specialized predefined function. The regular nodes can be useful for several reasons. The amount of regular nodes can be exactly specified independently of number of transistors. This brings possibility to help evolution by specifying for how complex solution we are looking for. The number of nodes can be even increased during evolution, which can be useful to limit search space from start and expand it later. This can be efficient time saver.

Another benefit of this representation is that we can easily store node status(present logical level). We can also easily mark which node is being evaluated right now. This can be useful when circuit contains gate connection loops or node evaluation dependencies which could bring down simulator into program loop. This looping dependency is in our simulator detected and evaluation is done in way which will avoid repeated recursive evaluation of such node in other words looping followed by stack grow and segmentation fault. In our approach CGP grid may not be necessary, but it may be beneficial and it is optional upgrade. CGP grid option means, that we does not have only these 3 layers:

```
input layer { VCC, GND, IN_0, IN_1, ...}
circuit layer { NODE_0, NODE_1, ... }
output layer { OUT_0, OUT_1, ... }
```

But `circuit layer` is divided into more sub-layers for example:

```
circuit layer {
  L0 { NODE_0, NODE_1, ... }
  L1 { NODE_3, NODE_4, ... }
  L2 { NODE_6, NODE_7, ... }
}
```

It is then necessary to associate each transistor with some layer. When CGP grid is used, then we need to specify interconnection rules to get some benefit of it. The rules can be for example following:

- transistor's gate - can be connected only to lower level or input layer
- transistor's drain - can be connected only to the equal/lower level or input layer
- transistor's source - can be connected only to the equal/upper level or output layer

These rules construct circuit as forward signal propagating one, it can be beneficial, because we have more exact idea about solutions which can be produced by evolution. But it can also prevent occurence of new innovative circuit solutions. It is always trade-off which needs to be considered.

When CGP grid is not used, then rules for possible interconnections are quite simple:

- transistor's drain - can be connected to any node
- transistor's source - can be connected to any node

If `g_evolution_any_gate_input = 1` then:

- transistor's gate - can be connected to any node

Otherwise (`g_evolution_any_gate_input = 0`):

- transistor's gate - can be connected only to input layer

This is extremely benevolent set of rules for circuit construction. It brings to us very wide search area, where new solutions can be found, but also lot of circuit mutants will possibly contain looping dependencies, which simulator have to identify and discard such paths with `X_CONNECTED` state of path. This can possibly discard candidate solutions which does not necessarily have to be wrong, some of them may even work in SPICE simulator, but this is the price we pay for speed. But it does not have to be bad thing. It can slightly reduce group of possible solutions, but it can increase the probability that evolved circuits will work in real world. Which was and still is one of the critical problems of transistor-level evolution [19] [11].

6.2 Path searching algorithm

This embedded evolutionary circuit simulator is based on innovative signal paths identification and evaluation method. It makes path-searching algorithm one of the most important parts of this system.

The purpose of this simulator in the most simplified manner is to compute output vector for each input vector. The way how to do it in path-oriented simulation is to find all paths from output to all reachable signal sources. As was earlier mentioned, signal sources are VCC, GND and primary inputs. Important thing is, that simulator needs to find **all** such signal paths.
Searched paths for shortcut detection:

- VCC GND
- IN_0 VCC
- IN_0 GND
- IN_0 IN_1
- ...

Searched paths for signal propagation:

- OUT_0 VCC
- OUT_0 GND
- OUT_0 IN_0
- OUT_0 IN_1
- ...

The good question is, how it should be done. When we need all existing paths. After few experiments and analysis, only possible solution we found is *Breadth First Search* algorithm. It is probably only way, how to cover all existing paths between two different nodes.

BFS is well known algorithm, but we will mention the principle behind. BFS starts in selected node and it asks for all transistors connected directly to this particular node. For each transistor found is started/created new separate path which is stored on stack. Then path from top of stack is pulled out. Then BFS asks again for connected transistors to last reached node, which are not already present in the path. For each of these transistors current path is copied and transistor is added at the end of path. Then each of these paths are pushed into stack. Paths which reaches dead end are pulled from stack and because there are no other transistors which could connect them further, they are not returned to stack. The paths which reaches target node are copied into vector of complete paths and after all paths are searched and finished, they are returned. It should be noticed that paths, which are exactly the same except for even 1 transistor, are stored fully separately.

Path is basically vector of transistors, more particularly transistor object pointers. It can be easily transformed into vector of nodes. In this implementation path is actually represented not just by vector of transistor, but as an object. It allows to call useful methods, such as `path->pass( logic_value )`, which returns properly decreased/degenerated logic value according to type and states of path transistors.

```
NAND circuit example

T0 pMOS  VCC,     out_0,   in_0
T1 pMOS  VCC,     out_0,   in_1
T2 nMOS  out_0,   node_0,  in_0
T3 nMOS  node_0,  GND,     in_1

shortcut paths
VCC GND {T0, T3, T4} {T1, T3, T4}
in_0 VCC {}
in_0 GND {}
in_1 VCC {}
in_1 GND {}
in_0 in_1 {}

signal paths
out_0 VCC {T0} {T1}
out_0 GND {T3, T4}
out_0 in_0 {}
out_0 in_1 {}
```

It may seem that BFS is far from efficient algorithm because of its computational complexity. But as was mentioned, it is only way how to find all possible interconnections from start node to target node.

The breadth first search In evolution can be useful to limit depth of search for this purpose it is possible to set

```
BFS_round_limit = 30 .. 1000
```

and all circuits which does not match each of its BFS calls under this limit is automatically discarded, and paths which have been looked for are returned as none was found.

BFS implementation in this simulator have to respect certain special situations, for example VCC and GND are from program point of view equivalent with nodes, they have even node numbers GND = 0 and VCC = 1. But when BFS is searching for path, and even if VCC or GND is not `target_node` the search should not proceed through them, because they are sources of *hard* signal. It can be just small detail but when we search for signal paths to output, we start search always from output node to signal source. In typical transistor circuits it may save some time.

```
BFS(start_node, target_node, circuit)
```

The paths are stable, they do not change during simulation. The conductivity of those paths is the thing, that is different for different input vectors. This means, that paths should be precomputed first and stored, because they are the same for all possible input combinations.

The conductivity of these paths, is something that should be evaluated separately for each input vector. In other words it leads us towards circuit simulation.

6.3 Simulation : signal paths evaluation

Simulator can work in two main modes/regimes. In idealistic mode, where transistor is considered to be perfectly conductive voltage controlled switch. Or in more realistic mode, where transistor V_t loss technological flaw is taken into consideration.

```
g_degenerative_logic = .. // 0 = idealistic // 1 = realistic
```

6.3.1 Logical value degeneration

Logic/voltage state/value is represented in simulator by integer variable, which is computationally much more efficient on traditional CPUs than floating point representation, which could on other hand represent almost exactly voltages in fractions of volts.

To respect V_t loss technological feature of MOS transistors the simulator can represent logical value between full voltage(logical one) and zero voltage(logical zero) into range of right now 20 values:

```
STRONG_ONE                  30
(ONE with 1 degeneration)   29
(ONE with 2 degeneration)   28
..
STRONG_X                    20
..
(ZERO with 1 degeneration)  12
(ZERO with 2 degeneration)  11
STRONG_ZERO                 10
```

This range have been proven to be more than sufficient, in simulation of circuits, which should work in real world in voltage range from 5.0 V down to 1.3 V.

In situation where none of above mentioned node logic states are suitable to represent node situation, there are also special node states for this purpose:

```
X (undefined value)
Z (high impedance)
S (shortcut)
```

The logic state degeneration can be caused by several situations. At first we will consider MOS technology feature and degeneration with valid logic state on transistor gate:

```
pMOS leading ZERO with valid ZERO on gate -> ZERO +1
pMOS leading ONE  with valid ZERO on gate -> ONE

nMOS leading ZERO with valid ONE  on gate -> ZERO
nMOS leading ONE  with valid ONE  on gate -> ONE -1
```

When already degenerated logic value is conducted trough drain-source channel it may of course degenerate further.

But it is not only possible situation, which will result into logic value degeneration. The situation where degenerated logic value occurs on transistor gate electrode is another case.

```
pMOS leading ZERO with  degenerated ZERO on gate -> ZERO -1 -DEG
pMOS leading ONE with   degenerated ZERO on gate -> ONE +DEG
pMOS leading ZERO with  degenerated ONE on gate  -> partialy closed
pMOS leading ONE with   degenerated ONE on gate  -> partialy closed

nMOS leading ZERO with  degenerated ZERO on gate -> partialy closed
nMOS leading ONE with   degenerated ZERO on gate -> partialy closed
nMOS leading ZERO with  degenerated ONE on gate  -> ZERO +DEG
nMOS leading ONE with   degenerated ONE on gate  -> ONE -1 -DEG

(for each logic gate value degeneration level DEG coeficient is increased)
```

When transistor encounters highly degenerated logic level it is switched into `X_STATE` which represents partially possibly opened state somewhere between `OPEN` and `CLOSED`. This behavior can differ widely based on used transistor technology and applied V_{cc} voltage, so this is something that should be checked and correlated with chosen V_{cc} and technological parameters.

Current settings looks valid for $V_{cc} = 1.8$ V according to precise analog-like simulation.

Special situations which more than classical degeneration cause hardly predictable or unpredictable logic value propagation trough transistor are following:

```
pMOS leading ANYTHING with X on gate -> X
nMOS leading ANYTHING with X on gate -> X

pMOS leading ANYTHING with Z on gate -> X
nMOS leading ANYTHING with Z on gate -> X

(all these situations switch transistor into X_STATE}
(anything != Z)
```

From theoretical point of view, high impedance on gate can cause, that transistor may stay in last state, which was present on gate before, thanks to its capacitance. But it would not be wise to rely on it in design process, so it is treated equivalently with situation, where logic state of gate is unknown → propagating `X`.

6.3.2 Path evaluation

We have seen that signal degeneration can be quite complex problem and it is necessary to take it into consideration, when paths conductivity is evaluated.

When we want to obtain output logic states the simulator has to evaluate paths from each output to all possible signal sources. Those paths, starting from circuit output and ending in signal source, are precomputed according to previously mentioned algorithm.

This simulator will evaluate one output after another in rising order. For each combination of output and signal source are paths stored separately.

```
output_paths[out][signal_source]
```

Signal sources are selected one after another, first VCC, GND and then inputs. For `output` → `signal_source` combination the paths are checked for conductivity. Simulator asks if the path `is_conduction()`. If it does, then simulator will `pass( logic_signal )` trough that path, and will receive signal, which may be degenerated according to path properties, if degenerative_logic mode is set. If all transistors in the path are in `OPEN` state then it can be simply done by counting nMOS or pMOS types of transistors, based on passed logic value.

It is possible to speed up simulation with sacrificing some level of precision by taking the first conductive path, passing logic level available from signal source and taking it as final result and state of output bit for selected input combination.

```
g_simulator_take_first_path =  0 / 1
```

But if path is degenerative for passed logic value then result of simulation will choose the first one found instead of the best one found. More paths leading and conducting at the same time to one point are correctly simulated if the result is the least degenerated value.

Situation, where output would be conductive to the `ZERO` signal source and at the same time to the `ONE` signal source, is the shortcut situation. But this something what simulator detects and solves by separate paths and functions. In path computing:

```
compute_shortcut_paths()
```

In simulation:

```
evaluate_shortcuts()
evaluate_current_shortcuts()
```

This may not seem so important, but this separation allows to discard circuits with shortcuts even without computing and simulating signal paths which in evolutionary process can be quite beneficial.

6.4 Evolution

Let's take a look on the implementation of evolution.

Evolution is process which can be done in many ways we will concentrate on genetic programming variant of evolution where the fittest candidate (circuit). In this variant and our implementation, the fittest candidate is chosen from population, and is stored in special, slot which is not encountered in population size.

This fittest candidate is chosen purely based on fitness value until first correct solution is found. After that there are two conditions for the fittest candidate selection. The candidate has to have higher or equal fitness and have to be also correct solution. Even if circuit candidate has higher fitness than fittest in this phase it will not be chosen if it is not fully correct solution. Determination if the solution is correct is done by method

```
circuit->is_correct_solution()
```

Which takes into consideration how strong degenerations we allow and also how many of them in sum.

If the current fittest has equal fitness value as the new candidate from younger population, implemented system always prefer the younger candidate. It is quite important because of accumulating neutral mutations which can lead later to better solution leaving the local minimum.

After first full solution of circuit defined function, the bonus for each `DISABLED` transistor is allowed, leading the evolution process from solution search to optimization.

How is fitness computing and rewarding done, is described in one of the following subsections.

Now we will move to description of basic parameters of evolutionary process.

6.4.1 Evolution parameters

Evolution process can be parametrized with several basic attributes, such as:

```
population_size
generations_count
max_mutations
```

But this evolutionary system can be parametrized also with more specialized attributes relevant for transistor circuit evolution. These following parameters defines borders of the search space.

```
transistors =  1 .. UNLIMITED
inputs      =  1 .. MAX_INPUTS
outputs     =  1 .. MAX_OUTPUT
nodes       =  0 .. MAX_NODES
```

How many transistor will be interconnected within each circuit is quite important parameter. It has probably the highest influence on time needed for circuit evaluation.

Inputs and outputs are quite obviously specific for each circuit definition, which we would like to evolve, but sometimes may be useful to add some extra inputs, which will represent inverted values of the circuit real inputs. And after evolution is finished, this

secondary inputs can be created by manual or automatic addition of CMOS inverters (2 transistors each). This may help to solve certain problematic situation which can arise with complex circuits [18]. We have used this approach also in gate-level circuit evolution in our previous work.

The amount of internal circuit nodes is something which is useful for extending or on the contrary limiting complexity of search space.

Circuit function specification

The function which we want to solve by evolved circuit have to be somehow specified. This can be done simply by using digital function table, where for each input combination is specified referential output combination, which should be set on output pins.

Some evolutionary systems generate all possible input combinations leading to 2^{inputs} input vectors. Our system uses only input combinations which were specified and requested for circuit compliance.

```
# function table in .tct file format (left)
# and in the source code format (right)
FUNCTION TABLE
00 -> 1               # IV({0, 0}); OV({1});
01 -> 1               # IV({0, 1}); OV({1});
10 -> 1               # IV({1, 0}); OV({1});
11 -> 0               # IV({1, 1}); OV({0});
END
```

It may happen that in certain specific functions evolved circuits fails to solve one or very few input-output vector specifications. It happens for example in case of 3-input AND, 4-input AND and similar circuit's evolution. Evolution usually easily finds solution for first 15 vectors, but fails to find the last 1.

```
AND 4 function table
0000 -> 0
0001 -> 0
0010 -> 0
  ...
1110 -> 0
1111 -> 1
```

To help evolution with solving this problem it has proven to be useful encourage system to solve problematic input-output earlier or even at cost of disrupting some of already solved input-output combinations [19]. It can be done by specifying fitness bonus for such problematic vectors. In this system, it can be easily done by duplicating input-output combination (line in function table).

The attributes, which we have specified until now, are absolutely necessary in evolutionary search for solution. Now we can take a look at evolution/simulator options which can tell evolution something more about circuit characteristics which we would like, beyond the function table.

6.4.2 Evolution options

Circuit designer may want to find solution for some specified function, but it may be necessary not only to find any correct solution, but also to find solution with full voltage swing. In other words solution without any degenerated output bits. Or in other situation it may be allowed to find circuit with some level of degeneration on outputs so it can be utilized to find solution with lower amount of transistors.

List of evolution options/modes:

```
g_evolution_any_gate_input   = 0 / 1
g_evolution_allow_degenerative_inside = 0 / 1
g_evolution_allowed_inside_degeneration_per_bit = 0 .. 10
g_evolution_allowed_inside_degenerations = 0 .. UNLIMITED
g_evolution_allow_degenerative_output = 0 / 1
g_evolution_allowed_output_degeneration_per_bit = 0 .. 10
g_evolution_allowed_output_degenerations = 0 .. UNLIMITED
```

any gate input

If we want very fast and efficient search it is wise to disable any gate input. Then all transistor gates can be connected only to input layer. It may seem very restrictive but it actually is sufficient for lot of circuit functions. In some cases for cost of 1 level signal degeneration.

It can be also effective in combination with this type of solution stored, and then loaded, with decreased amount of transistors, nodes, etc. and allowed any gate input option.

allow degenerative output

This option affects the evolutionary process only in situation when all outputs are correct by defined function table, but some may be degenerated. When degenerative output is allowed then according to following two options is determined, when the optimization for transistor count will start.

Allowed output degenerations limit sets how many degenerations can occur on output in summary, it does not count only degenerated bits but also level of their degeneration.

Allowed output degeneration per bit limits maximal level of degeneration on output. None of output bit in any situation may pass it, if the circuit should be considered to be correct solution.

allow degenerative inside

If this option is set, internal node states connected to gates of transistors will switch transistor into `X_STATE` (partially opened) if their state logic level degeneration is higher than allowed degeneration per bit.

If number and levels of nodes degenerations in summary will be higher than allowed inside degenerations circuit will not be considered to be correct solution. Or it may be even discarded directly.

6.4.3 Simulator options

The following options are also important for evolution but in little indirect way. They are closer to be circuit simulation or circuit evaluation options.

```
g_degenerative_logic = 0 / 1
g_any_gate_input = 0 / 1
g_X_shortcut_check = 0 / 1
g_take_first_conducting_path = 0 / 1
```

The degenerative logic switch is the main mode selector which determines if the simulation is going to be idealistic or more realistic(voltage level degenerations).

Any gate input has practically similar function as mentioned above, but if circuit simulator knows, that there is not allowed connection from non-input layer to gate, it can save some computational time, because it does not have to check this option for each transistor, mark it and try to search paths for it.

`X` shortcut check tells simulator, how should be possibly partially conductive shortcut paths perceived. If set they will be perceived as regular shortcuts. Otherwise they will be ignored.

Another attribute which can limit simulator from spending too much time on one circuit, when there may be many others, is the BFS search limit.

```
g_BFS_round_limit = BFS_ROUND_LIMIT
```

Circuits which cannot find all signal paths in limited number of rounds are directly discarded in this phase.

The question how evolution can create non-identical offspring is answered in following text here.

6.4.4 Mutation

The offspring in evolution process can never all be identical with parent. They may have the same fitness and exactly the same qualities and problems but at least neutral mutations have to be always present. Neutral mutations are those, which does not affect the fitness value or correctness of candidate circuit.

Mutation is done by random selection of transistors, which should be mutated. Then selected transistors generate random chance which decides what should be changed from following list:

```
transistor type  = {N, P, DISABLED}
drain node = { VCC, GND, IN, .. , OUT .. , NODE, .. }
source node = { VCC, GND, IN, .. , OUT .. , NODE, .. }
gate node = { VCC, GND, IN .. (OUT, .. , NODE, ..)}
```

Transistor type is selected from P-channel, N-channel, disabled and also pure drain-source inter-connector(wire) was considered.

The rest of attributes have different sets to pick from, which depends on selected evolution options.

Drain and source electrode are considered to be equal in used transistor model.

Probably only problematic thing in implementation of mutation can be mutation of gate node. Because if there are no limits, to where it can be connected it will create all types of problematic and looping situations in the circuit. The proposed solution such as CGP grid may improve chances, that this will not happen so often. Another proposed approach is to connect gate primarily to input layer but with smaller chance allow limited amount of transistor gates, to be connected to some other node. The perfect solution for this problem is hard to find, also thanks to allowed pass-trough logic (PTL). It is quite reasonable for more complex circuits and wider search space, where is much more internal nodes than inputs.

6.4.5 Logical value distance/difference

When circuit is simulated and output vector is computed, it is time to compare it with referential output vector stored in function table. The first implementation used to compare vector to vector for full width of output and then compute vector distance.

The better approach implemented after this one compares vectors bit after bit and computes distance between bits instead of distance between vectors.

Let's take a look at simplified bit distance table:

output bit	ref	dist
ANY ONE	0	10
STRONG_ZERO	0	0
STRONG_ZERO +1	0	1
STRONG_ZERO +2	0	2
STRONG_ZERO +3	0	3
STRONG_ZERO +4	0	4
X	0	4
Z	0	5
STRONG_ONE	1	0
STRONG_ONE -1	1	1
STRONG_ONE -2	1	2
STRONG_ONE -3	1	3
STRONG_ONE -4	1	4
X	1	4
Z	1	5
ANY ZERO	1	10
ANYTHING	X	0
Z	Z	0
ELSE		99

It appears to be beneficial for evolution treat Z or X logic value as closer ones, than the opposite strong logic bit.

Alternative to computing bit distance would be directly computing fitness addition. But we consider this bit-bit distance to be more generic.

After bit-bit distance is computed then according to distance result fitness value addition is selected and added to circuit fitness.

Fitness value

Bit-bit distance is something that can be practically constant all the time. This separation of distance and fitness allows to change rules of fitness addition based on bit-bit distance when it is needed.

Fitness can be also increased for different reasons such as input-output vector with increased importance is correct or evolution found correct candidate with lower number of transistors.

But fitness can be also decreased for each shortcut causing input combination, or even each conducting shortcut path, by strong penalty.

Fitness bonus can be added when bit of duplicated input-output vector is correct.

Another fitness bonus is added for each disabled transistor, decreasing number of transistors needed for proper circuit function.

6.4.6 Stopped, modified and continued evolution

Evolution is quite complex process. It can be very innovative in one area and very conservative in another.

It was empirically observed that evolution with certain parameter settings can be quite successful in developing almost correct circuit solution with one or two bits wrong or degenerated. Drop all the work which it has done would be wasting of development time.

Because it may easily finish the job just with slightly modified settings.

For example a designer may set wide search space with many transistors and internal nodes but disable any gate input. Run the evolution for some time. Evolution then can bring solution which is almost correct. Then a designer can edit function table of stored 'solution' in progress to make evolution think, that it found correct solution. It will switch evolution into optimization mode and as many redundant transistors as was found will be disabled. Designer have now almost minimal size and almost correct evolution. Then some disabled transistors may be erased from stored candidate file reducing search space. Any gate input mode can be set and evolution started again from this point on tinier search space. It can speed up design process by few orders of magnitude.

For this purpose, our own circuit representation file format was designed. We call it TCT - Transistor Circuit formaT. Following functions for this format were implemented:

```
load_circuit( filename )
store_circuit( filename )
```

It was designed to be able to do something more than just stop evolution, save result, shutdown computer, boot up computer, load and continue but even this can be quite useful.

File format was designed to be human readable and should be able to provide also something like a log of evolutionary parameters. It can also be easily written by circuit designer or automatically generated from another electronics designing tool.

6.4.7 Circuit representation file format

To illustrate simplicity and usefulness of internal circuit format we included this automatically generated TCT file. This file contains 2bit AND transistor circuit and it have been generated automatically after this solution was found.

Illustrated circuit is fully operational even in SPICE simulation.

```
CIRCUIT AND
inputs 2
outputs 1
nodes 5

# ------------------------------------------------

# used_transistors 5
# degenerated_output_bits 0
# output_degenerations 0
# fitness 65

# ------------------------------------------------
EVOLUTION_OPTIONS    # (OPTIONAL)

g_population_size = 30
g_generations = 100000
g_transistor_space = 30
g_max_mutations = 20

g_evolution_any_gate_input = 1
g_degenerative_logic = 1
g_evolution_allow_degenerative_inside = 0
g_evolution_allowed_inside_degeneration_per_bit = 0
g_evolution_allowed_inside_degenerations 0
g_evolution_allow_degenerative_output = 0
g_evolution_allowed_output_degeneration_per_bit = 0
g_evolution_allowed_output_degenerations 0
g_X_shortcut_check = 1
g_BFS_round_limit = 50

END

# ------------------------------------------------
# first_solution_found_after 33.9163 s
# this_solution_found_after  43.1254 s
# ------------------------------------------------

FUNCTION_TABLE

00 -> 0
01 -> 0
10 -> 0
11 -> 1

END

# ------------------------------------------------
INPUT_OUTPUT_TABLE     # (OPTIONAL)

# input output degeneration distance
00 -> 0 # 0 : 0
01 -> 0 # 0 : 0
10 -> 0 # 0 : 0
11 -> 1 # 0 : 0

END

# ------------------------------------------------
TRANSISTOR_TABLE

T1   P 7 1 2
T7   N 3 4 2
T13    N 4 0 7
T14    N 0 7 2
T18    P 4 3 7

END
```

6.4.8 Verification of evolved solution

When evolution founds a solution for requested function, it should be checked with more precise analog-like simulator, which is usually much slower but should be precise enough to find out if the circuit is capable of correct function in real world.

For this purpose we use ngSPICE simulator. If we want to use it for simulation, it is necessary to represent circuit in ngSPICE compatible format. It may be possible to do it manually but it could be source of mistakes and also it can be unpleasant routine work, which nobody would want to do. To avoid this, automated creation of ngSPICE's netlist was implemented.

Automated SPICE netlist generation

It can requested by circuit method

```
store_netlist( filename )
```

Shortened version without transistor parameters definition is here for illustrative purpose.

```
CIRCUIT ''
.global vdd vss
.param supply=1.8
V0 vss 0 0V
V1 vdd 0 'supply'
*
V2 in.0 vss pulse(0 'supply' 5n 10p 10p 5n 10n)
V3 in.1 vss pulse(0 'supply' 10n 10p 10p 10n 20n)
* circuit
X1 net.2 in.0 vdd pMOS
X7 in.1 in.0 out.0 nMOS
X13 out.0 net.2 vss nMOS
X14 vss in.0 net.2 nMOS
X18 out.0 net.2 in.1 pMOS
Cload9a out.0 vss 20f
* models
.subckt pMOS in.s in.g out
  .global vdd vss
  Mp1 in.s in.g out vdd CMOSP l=0.25u w=1.00u ad=0.5p pd=3u as=0.5p ps=3u
.ends
.subckt nMOS in.s in.g out
  .global vdd vss
  Mn1 in.s in.g out vss CMOSN l=0.25u w=0.50u ad=0.25p pd=2u as=0.25p ps=2u
.ends
.subckt JUNC in.s in.g out
  .global vdd vss
  R1 in.s out 0R
  R2 in.g out 0R
.ends

...  here is included transistor model ...

* analysis
.TRAN 0.1n 20n
.save v(in.0) v(in.1) v(out.0)
.end
*
* plot: vdd v(in.0) v(in.1)
* plot: vdd v(out.0)
```

The netlist parameters such as voltage, transistor parameters and technological model mentined here are also used in same form in experiments chapter.

We will not describe SPICE netlist format in detail here, its description can be found in for example ngSPICE documentation [2].

But it should be mentioned, that for proper and adequate simulation we need appropriate transistor model for selected layout implementation technology. We use transistor parameters stored in specific file `transistor.model` which is then included into all generated netlists.

Another important thing is definition of input signal sequence. Where all possible input combinations have to occur in order.

Here we can take a look on graphical output of SPICE simulation. Simulated circuit was AND gate with 2 inputs, implemented on 5 transistors. This circuit was found by evolution and also its TCT file was shown earlier.

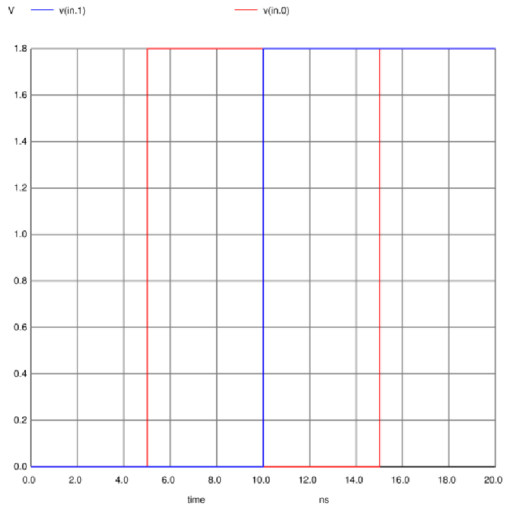


Figure 6.1: input signals

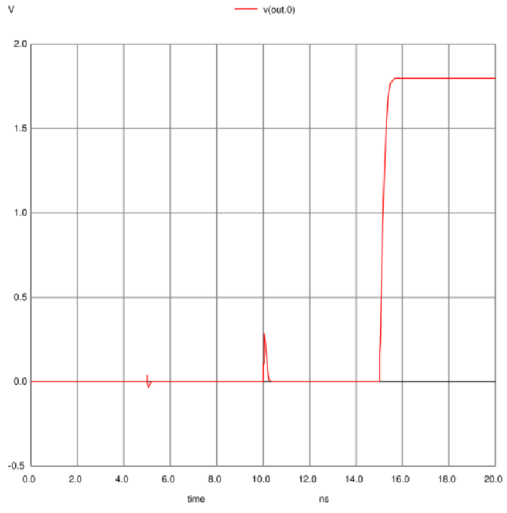


Figure 6.2: output signal

Simulation can be also simply run remotely using ngSPICE web interface [1].

Automated schematics generation

When circuit was successfully simulated in SPICE-like simulator designer will want to see how this circuit really looks like.

It may be possible to read TCT file and manually create scheme. It will highly likely happen anyway due need for different symbols or scheme format. But in case, when designer would like to have an idea how circuit really looks like, system have implementation of method

```
store_scheme( filename )
```

It will automatically generate scheme in `gschem` format which can be then manually repositioned. For illustration how it could look like we include following scheme opened in `gschem tool`:

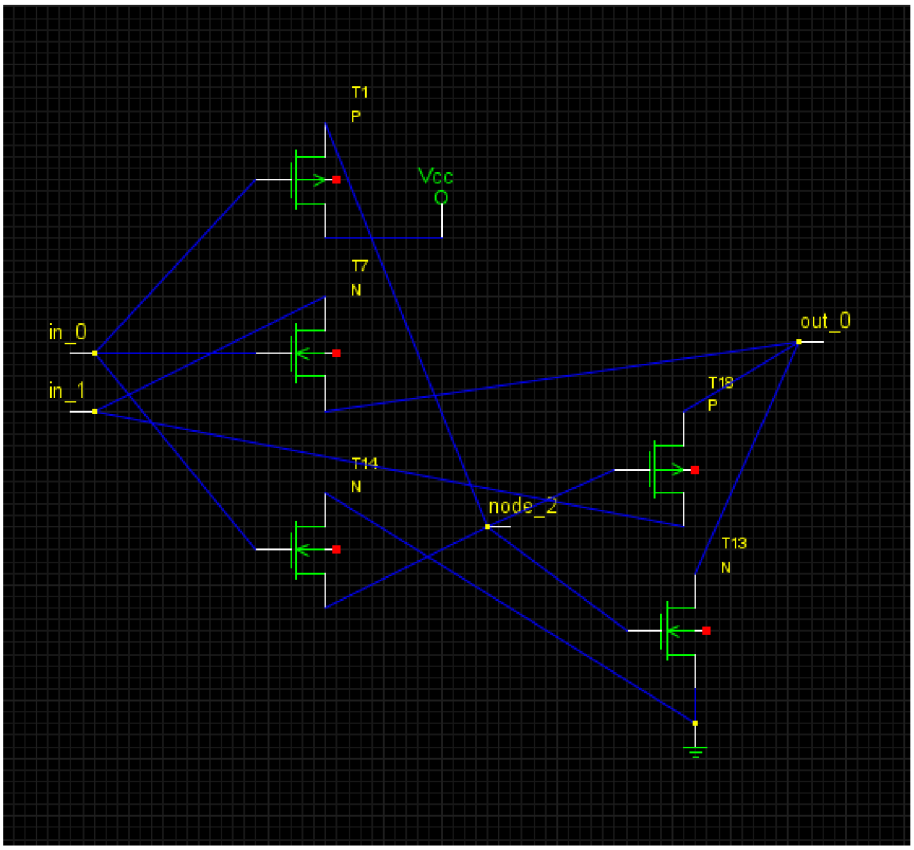


Figure 6.3: Automatically generated scheme in gschem format (AND 5T circuit)

Chapter 7

Experiments and benchmarks

In this chapter we will go trough experimenting with implemented designing system. We have seen in previous chapters that evolutionary process can be quite complex and can be controlled by many parameters. Each of these parameters have very strong influence on efficiency of evolutionary process, but also on results which will arise from the process. This leads us to situation, where we have to set the same conditions for evolution of each circuit, if we want to be able to compare efficiency results.

The first benchmark is done on defined search space with 20 transistors and 10 internal circuit nodes with disabled possibility for transistor gate connection to any internal node. The `deg bits` parameter specifies how many degenerated output logic levels were acceptable in found solution.

The simulator benchmark illustrates the simulation efficiency/speed on conventional circuit designs, from the smallest circuits towards very complex circuit structures reaching almost two hundred transistors. It also compares simulation speed with SPICE.

Another benchmark shows the efficiency of candidate solutions evaluation, based on evolution parameters.

All benchmarks and time measurements were done on computer with parameters:

```
CPU: Intel(R) Core(TM)2 Duo T8300 2.40 GHz
RAM: 2 GB DDR2 667 MHz

OS: Linux FL90 3.10-3-amd64 SMP Debian x86_64
```

7.1 Evolution Benchmark

Time measurements presented in this table are medians from repeated measurements. It seems more appropriate to use medians instead of arithmetical averages. Because they are not affected by distant values which represent more than anything lucky hit or unlucky local minimum.

```
population_size = 50
max mutations = 20

transistors =  20;
internal nodes = 10

g_evolution_any_gate = 0
g_degenerative_logic = 1;
g_simulator_take_first_conducting_path = 0;
g_any_gate_input = g_evolution_any_gate_input;
g_BFS_round_limit = 1000;
g_X_shortcut_check = 1;
```

measured 10x for each circuit

circuit	inputs	outputs	vectors	deg bits	transistors	first found	best found
NOT	1	1	2	0	2	0.038 s	1.793 s
NAND	2	1	4	0	4	1.145 s	1.285 s
NOR	2	1	4	0	4	0.415 s	1.777 s
AND	2	1	4	3	2	1.447 s	4.994 s
AND	2	1	4	2	3	2.052 s	2.384 s
XOR	2	1	4	1	4	0.623 s	3.889 s
ADD	2	2	4	4	6	3.035 s	6.851 s
ADD	2	2	4	3	7	11.516 s	11.868 s

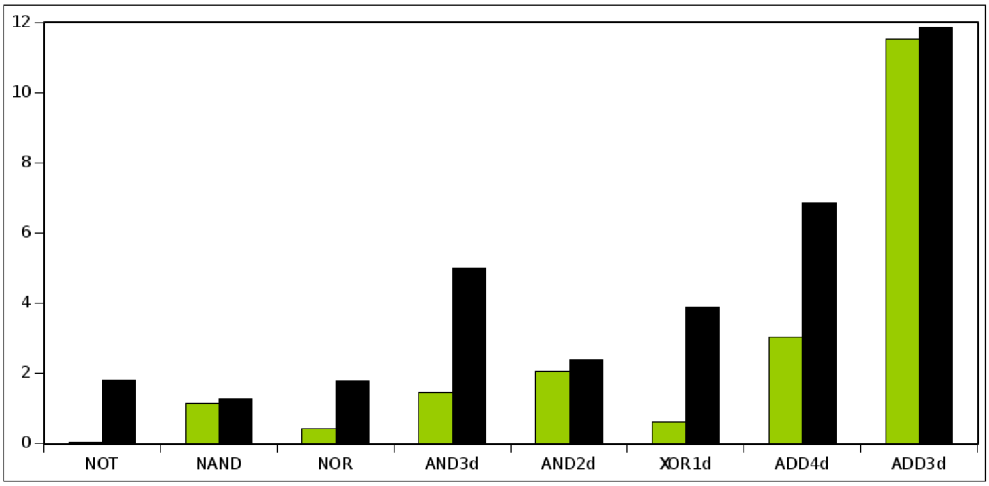


Figure 7.1: Evolution benchmark - first and best solution time [s]

7.2 Simulator Benchmark

Now let's take a look on simulator performance.

7.2.1 Simulation of conventional circuits - time requirements

circuit	in	out	tr.	sum t[μs]	sim. t[μs]	paths t[μs]	SPICE t[s]	speed-up
NOT	1	1	2	15	3	13	0.024 s	1600x
NOT A	2	1	2	19	4	15	0.024 s	1263x
NAND	2	1	4	33	5	27	0.029 s	878x
NOR	2	1	4	34	9	25	0.032 s	941x
AND	2	1	6	58	12	46	0.034 s	586x
OR	2	1	6	42	7	35	0.030 s	714x
XNOR	2	1	8	113	11	102	0.032 s	283x
XOR	2	1	8	112	11	101	0.036 s	321x
ANDNOR 4	4	1	16	175	63	112	0.111 s	634x
B1	3	2	30	309	53	256	0.123 s	398x
C17	5	2	28	393	182	211	0.356 s	905x
FA 1b	3	2	48	1306	206	1100	0.173 s	132x
FA 2b	5	3	94	1598	636	962	1.243 s	777x
FA 3b	7	4	174	10982	6927	4055	9.794 s	891x

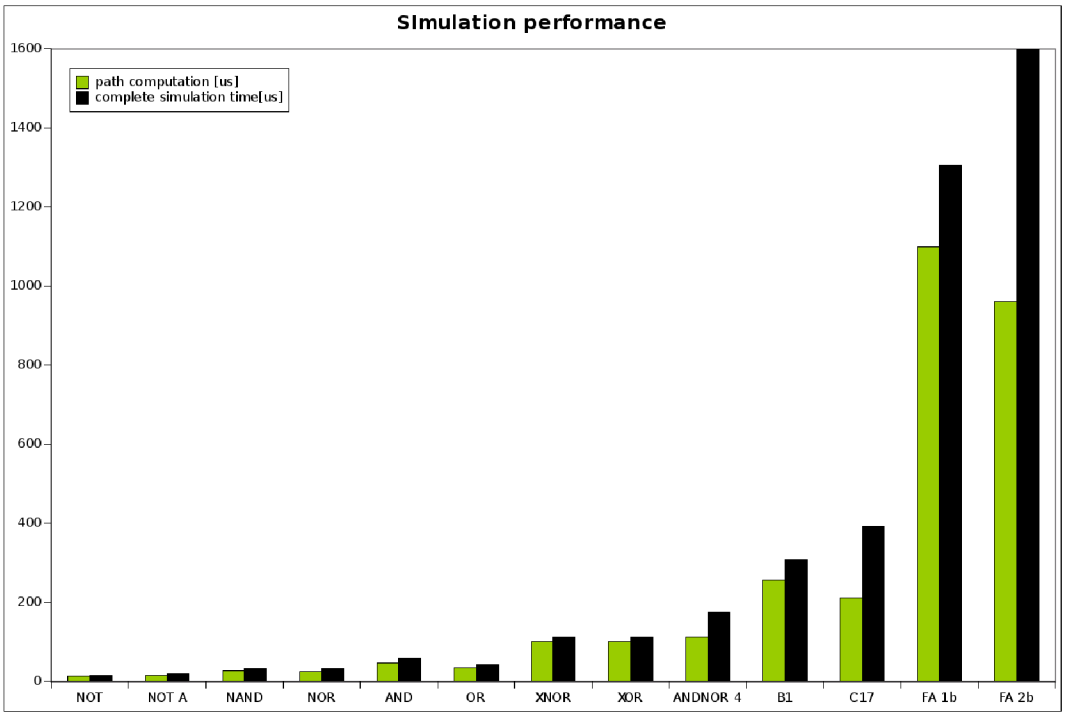


Figure 7.2: Simulation benchmark - time [us]

The simulator was tested on several conventional circuit designs. Its performance was measured and also compared with SPICE simulation. Our implementation is on average 737.76x faster than SPICE. In other words, our simulation method can on average evaluate fitness of about 700 circuits where SPICE simulator would evaluate just one.

It can also be seen that major time consumption of our simulator is caused by paths computation. But paths are computed just once and stay stored in memory. Then simulation of each input vector is much faster. This is the major source of varying speed-up in comparison with SPICE.

7.2.2 Path computation time requirements

circuit	in	out	tr.	all [μs]	shortcut [μs]	output [μs]	node [μs]
NOT	1	1	2	13	5	7	1
NOT A	2	1	2	15	6	8	1
NAND	2	1	4	27	11	15	1
NOR	2	1	4	25	9	15	1
AND	2	1	6	46	18	10	18
OR	2	1	6	35	11	8	15
XNOR	2	1	8	102	49	7	46
XOR	2	1	8	101	49	7	45
ANDNOR 4	4	1	16	112	30	20	61
B1	3	2	30	256	98	25	133
C17	5	2	28	211	54	33	124
FA 1b	3	2	48	1100	329	33	737
FA 2b	5	3	94	962	278	70	614
FA 3b	7	4	174	4055	1171	134	2750

Evolutionary candidates simulation

This table represents time of candidate evaluations computed by average from 10000 generations with population size = 100. It illustrates how evolution parameters affects simulation speed.

situation	transistors	nodes	any_gate	avg. candidate t[μs]	candidates/s
FULL ADDER	20	20	0%	338.6	2953
FULL ADDER	20	20	30%	437.8	2288
FULL ADDER	50	50	30%	1088.2	919
FULL ADDER	100	100	30%	4785.8	209
FULL ADDER	100	100	100%	7283.2	135

Fitness value modification according simulation time

Circuit object method `compute_paths()` and also `compute_fitness` have time measurement implemented inside. It may also be beneficial to try embedding computational time

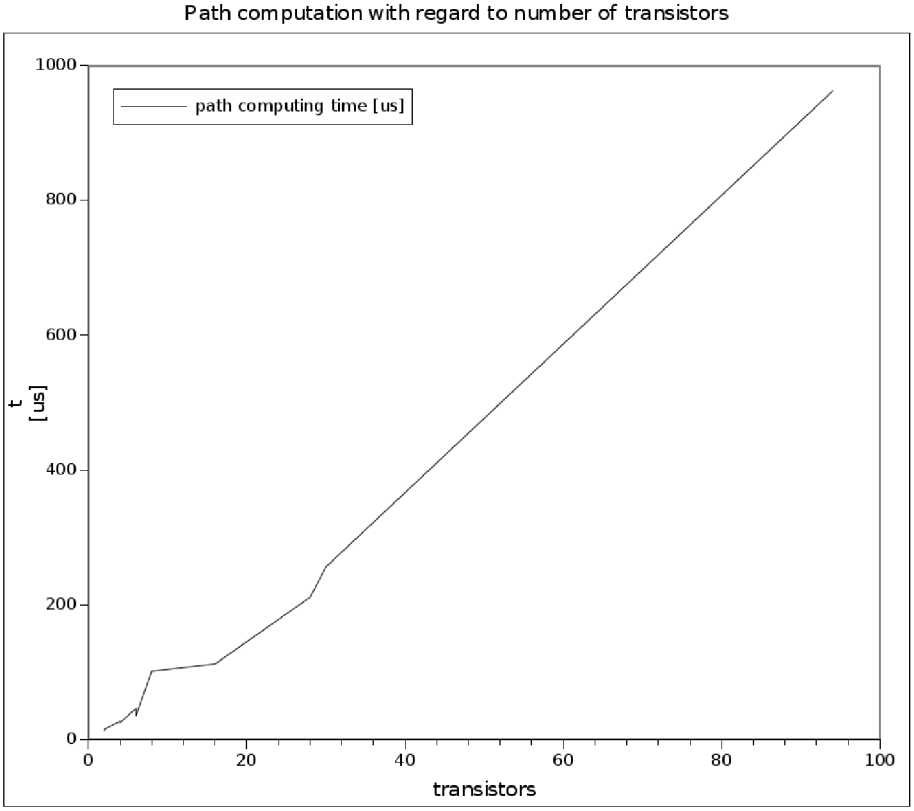


Figure 7.3: Path computation benchmark - conventional designs - time [us]

needed for circuit simulation into fitness rewarding system. It may be tricky, but it may help to navigate evolution to developing circuits which can be simulated faster. That would mean more circuits evaluated per second and that would worth to try. We will leave this idea for the future work.

Chapter 8

Solutions

Here we will review some of circuit solutions which have been evolved during experiments with implemented designing system. For each circuit we will provide also voltage characteristics from ngSPICE simulation.

It should be mentioned, that evolutionary design here uses PTL(pass-trough logic) very often. It brings more interesting solutions, which are rarely created by design engineers.

On the other hand, typical conventional designs use lot of inverters and cascade of control nodes, where transistor in the next cascade level is controlled by an output of transistor in the previous level.

Our system can use this also, but when there are many transistor gate inputs from internal circuit nodes very noticeable slowdown occurs. It is due path computation. For each such node, there have to be exactly the same type of path-search and also logical level evaluation as is done for each primary output. It can also very rapidly increase memory requirements.

Those are the primary reasons for which is usage of path nodes limited by probability. But evolution even without that prefers PTL approach because it is not so fragile, such as the gate control from not always stable signal source.

Schematics have been manually recreated for easier function understanding from those automatically generated by designing system. We used also transistor rotation for distinguishing conventional connection and PTL connection. Transistors with their channel in horizontal position represents PTL and those with channel in vertical position represents connection to VCC or GND signal source.

Simulation in SPICE was done with frequency of input signal equal to 100 MHz for all possible input combinations to verify circuit's proper functionality.

We present here only few of evolved circuits with short comment. But also many other circuit were evolved, for example 2-bit binary coder for 8 transistors. Or even 3-bit binary coder for 21 transistors. Both without output voltage degenerations. We can also mention 2-bit binary decoder for 11 transistors with few bits degenerated also full adder with some degenerated outputs on 11 transistors. We will present some of them later in our future work.

8.1 AND 2 transistor solution

This circuit was evolved with 3 allowed output bit voltage degenerations. According to implemented simulator there should be all 3 degenerations present, but according to more precise simulation, 2 of these degeneration are very short and cases only small spikes. This solution may not be universally usable in all situations. But it may be very useful in situations, where very limited chip area is available and used technology can handle limited output voltage swing. This solution uses only 33% of transistors of conventional AND gate design which is build on 6 transistors.

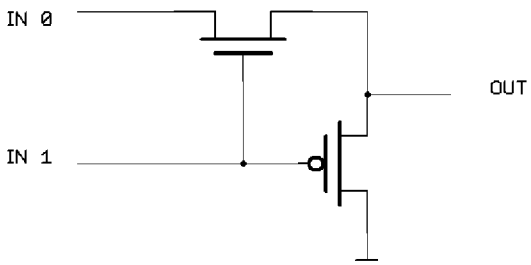


Figure 8.1: AND schematic 2 transistor solution

```
INPUT OUTPUT DEG
 00 -> 0   #   1
 01 -> 0   #   0
 10 -> 0   #   1
 11 -> 1   #   1
```

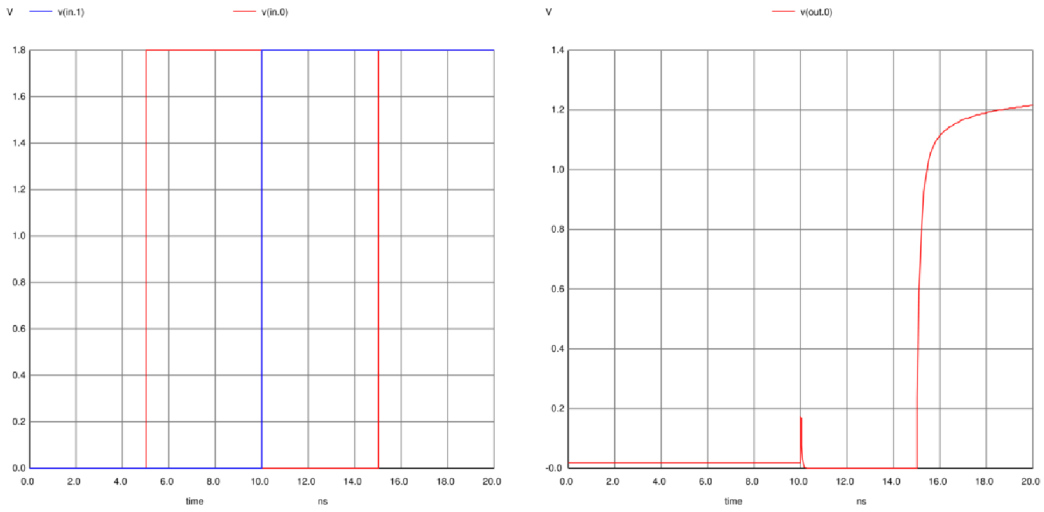


Figure 8.2: input and output voltage diagram

8.2 AND 5 transistor solution

This solution also uses PTL, but it provides full output voltage swing when there is full voltage swing on primary input. Conventional solution uses 6 transistors. It is practically constructed as NAND + NOT in cascade. The interesting thing is, that evolution cannot find solution with full output swing without transistor gate controlled from the inside of circuit(internal node). When allowed evolution used it to build up input signal inverter.

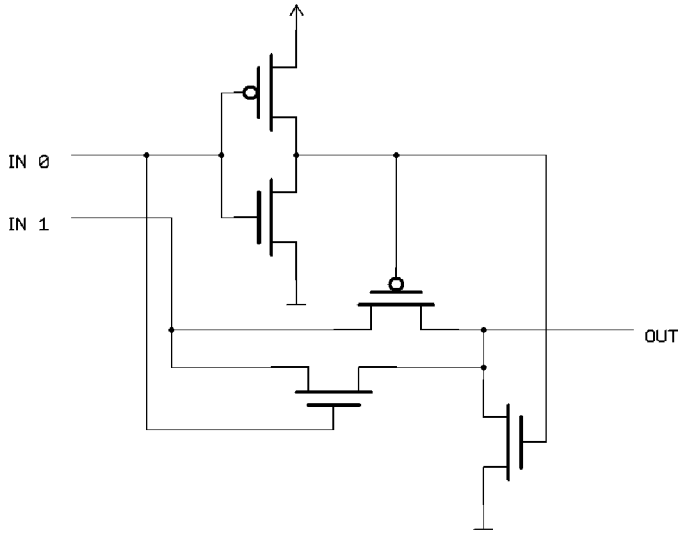


Figure 8.3: AND schematic 5 transistor solution

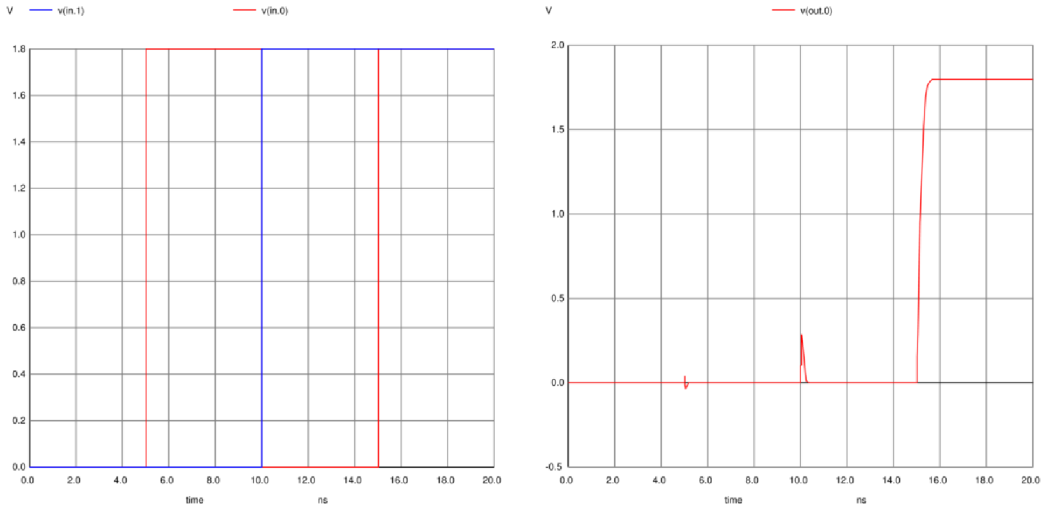


Figure 8.4: input and output voltage diagram

8.3 XOR 4 transistor solution

XOR solution found by evolution uses only 4 transistors. It has degenerated output for only 1 input combination. So it can be used as part of complex circuits only there, where other circuits will work correctly with one V_t loss on input. Conventional solution of XOR circuit is implemented on 8 transistors. So this solutions provides 50% transistor/space reduction for small price.

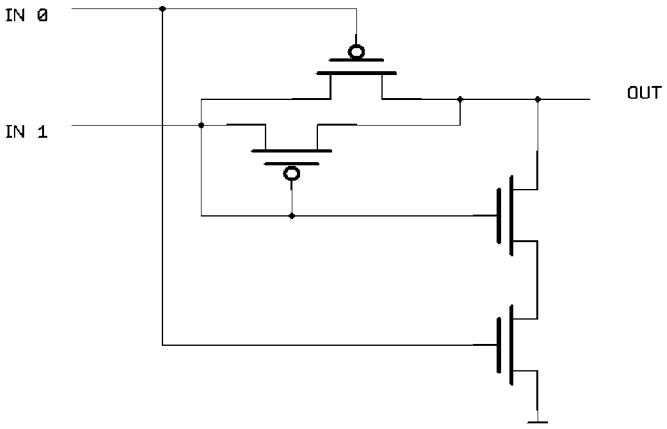


Figure 8.5: XOR schematic 4 transistor solution

```
INPUT OUTPUT DEG
 00 -> 0   #   1
 01 -> 1   #   0
 10 -> 1   #   0
 11 -> 0   #   0
```

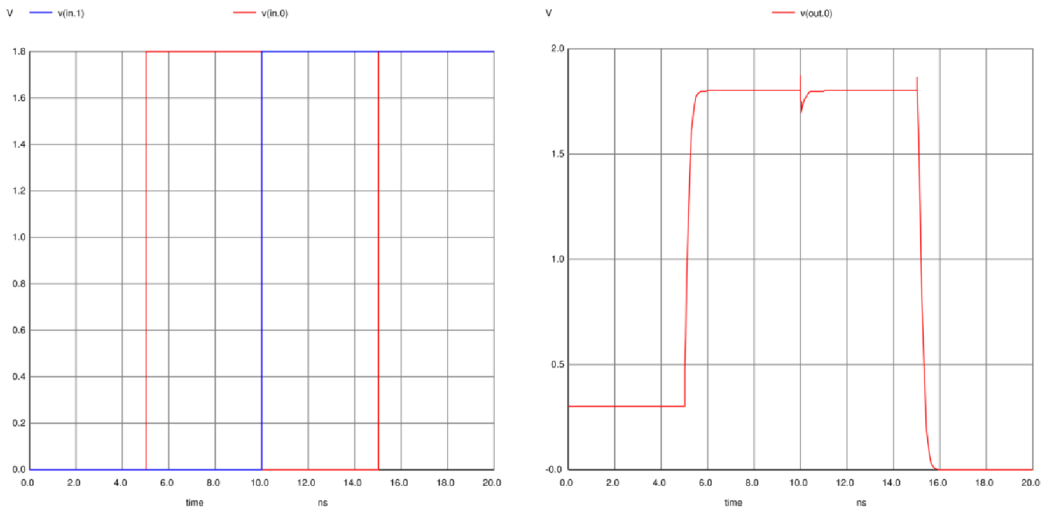


Figure 8.6: input and output voltage diagram

8.4 Half Adder 6 transistor solution

From more complex circuits exceeding size of one gate, half-adder circuit have been evolved. It has small output voltage swing limitations for some input combinations, but it is designed by using only 6 transistors, where conventional gate-level design uses XOR(8 transistors) with AND(6 transistors) gates together. This solution is then more than twice smaller. Exactly it uses only 42% transistors of original solution.

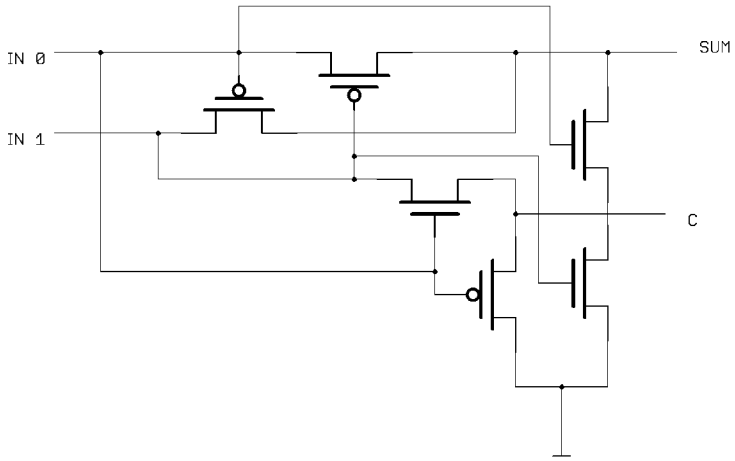


Figure 8.7: Half Adder schematic 6 transistor solution

```
INPUT OUTPUT  DEG
 00 -> 00    #  11
 01 -> 10    #  01
 10 -> 10    #  00
 11 -> 01    #  01
```

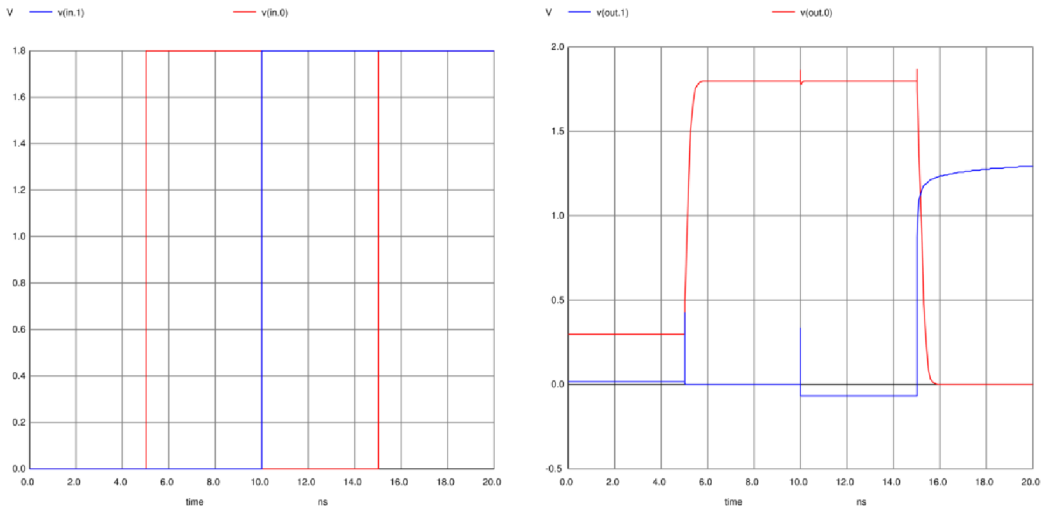


Figure 8.8: input and output voltage diagram

8.5 AND OR NOT - 8 transistor solution

Function `NOT(A.B + C.D)` nicely illustrates, that direct transistor-level solutions can bring great savings when it comes to transistor count. It is because conventional design on gate level requires 16 transistors for its implementation. This solution, also found by evolution, uses only 8 transistors, exactly 50% of conventional one. This solution also have full voltage swing on its output.

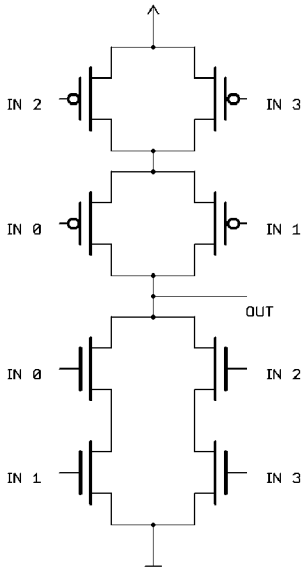


Figure 8.9: AND OR NOT schematic 8 transistor solution

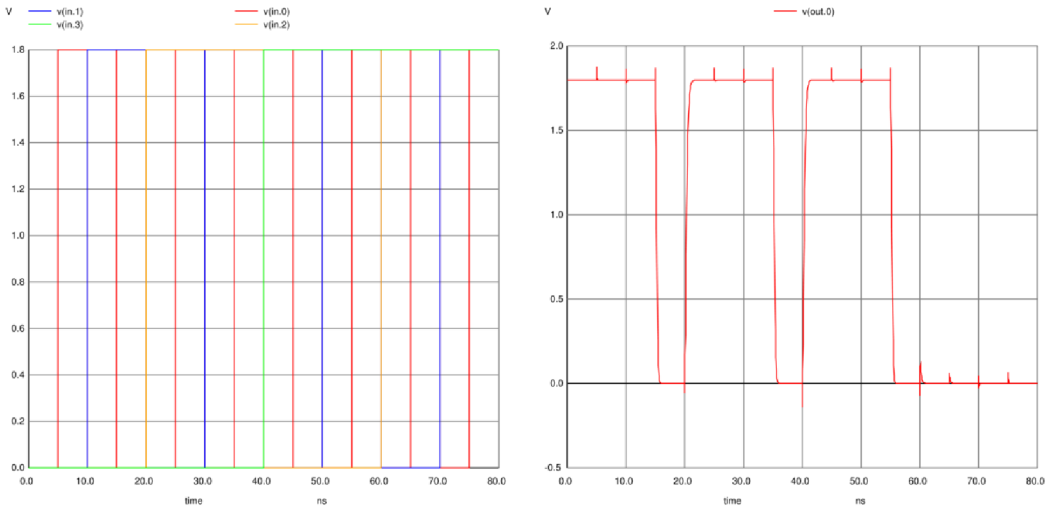


Figure 8.10: input and output voltage diagram

Chapter 9

Conclusion

In this work have been discussed several approaches, that can be used for each phase during transistor-level circuit design process. It was shown that, important thing is to find the ideal trade-off between precision of simulation and evaluation time of candidate solution during the evolution process. We saw the problems which were ignored by too much idealistic approaches. We also understood that simulation by complex reality close simulator such as SPICE cannot simply be used for efficient candidate evaluation because of its high computational time requirements.

We also saw restrictions which allowed evolution algorithm to find some working circuit solutions using even very simplified simulation. But those restriction are practically unusable, when complex circuits need to be evolved or optimized.

Knowing this we proposed and implemented reasonable simulation method based on graph circuit representation, where the path is to be searched starting from the output node and ending in signal source(VCC, GND, primary input). This method allowed us not only efficiently compute output logic levels for each input vector but also evaluate circuit for shortcuts and it can even identify possible power leakage by finding paths possible partially opened shortcut paths.

Based on this circuit evaluation method fast and efficient evaluation simulator have been implemented, which is the critical part of whole transistor level circuit designing system. In comparison with SPICE simulation it is on average 700 times faster.

Future work

There is always potential for upgrades based on knowledge and experience gained from experiments and research. Some ideas were mentioned in the implementation chapter, but for recapitulation we can mention them also here:

Mutation can be more restrictive to speedup circuit simulation. It could prohibit mutations which would certainly cause shortcut. This may brings some speed up for cost of clean separation source code functions.

Any gate input can be limited in little more intelligent way. How to do it efficiently without strictly limiting possible search space but avoiding unsolvable signal loops is interesting area of future research.

This work also created an idea of separating complex evolution goals into several circuit modules where first module would be connected to primary inputs and last module to primary circuit outputs. The modules between would be perceived as independent circuits but their input-output function tables would be dynamically chosen according to fitness of complete solution. It may remind gate-level evolution process but this approach could be more flexible.

Another idea which could also extend the possibilities of this system is to integrate it together with gate-level evolutionary design system, but not just replace all gates with their transistor representation, but also take into consideration possibly allowed gate output signal degeneration based on gate level interconnections.

Final words

This work brought new circuit simulation approach for design process into practise. It was shown that it is very efficient for circuit simulation by itself for conventional circuits but also for innovative solutions and even more as a critical part of evolutionary designing system in a role of the fitness evaluator.

Bibliography

[1] Ngspice simulation web interface [online]. http://www.ngspice.com/, 2013-02-23 [cit. 2013-04-28].

[2] Ngspice documentation [online]. http://ngspice.sourceforge.net/docs.html, 2013-02-23 [cit. 2014-02-23].

[3] Encyclopedia britanica [online]. http://www.britannica.com/EBchecked/topic/602718/transistor, 2013-11-27 [cit. 2013-11-27].

[4] Cartesian genetic programming [online]. http://www.cartesiangp.co.uk/, 2013-11-28 [cit. 2013-11-28].

[5] Computer history museum - the silicon engine — 1960 - metal oxide semiconductor (mos) transistor demonstrated [online]. http://www.computerhistory.org/semiconductor/timeline/1960-MOS.html, 2013-11-28 [cit. 2013-11-28].

[6] Wikipedia, the free encyclopedia [online]. http://en.wikipedia.org/wiki/Bipolar_junction_transistor, 2013-11-28 [cit. 2013-11-28].

[7] Wikipedia, the free encyclopedia [online]. http://en.wikipedia.org/wiki/Transistor_transistor_logic, 2013-11-28 [cit. 2013-11-28].

[8] Wikipedia, the free encyclopedia [online]. http://en.wikipedia.org/wiki/MOSFET, 2013-11-28 [cit. 2013-11-28].

[9] Wikipedia, the free encyclopedia [online]. http://en.wikipedia.org/wiki/CMOS, 2013-11-28 [cit. 2013-11-28].

[10] A.E. Eiben and J.E. Smith. *Introduction to Evolutionary Computing*. Springer, 2007. ISBN: 978-3-540-40184-1.

[11] Miller Julian F. *Cartesian Genetic Programming*. Springer, 2011. ISBN 978-80-200-1729-1.

[12] Kenneth Kardong. *An Introduction to Biological Evolution*. Washington State University, 2010. ISBN: 978-0073050775.

[13] Sekanina L., Vašíček Z., et al. *Evoluční hardware: Od automatického generování patentovatelných invencí k sebemodifikujícím se strojům.* Academia Praha, 2009. ISBN 978-80-200-1729-1.

[14] S. Mishra, A. Agrawal, and R. Nagaria. *A comparative performance analysis of various CMOS design techniques for XOR and XNOR circuits.* International Journal of Emerging Technologies (1), 2010.

[15] R. Mueller-Thuns, J. Rahmeh, J. Abraham, J. Wehbeh, and D. Saab. *Concurrent Hierarchical and Multi-level simulation of VLSI circuits.* Simulation Councils. Inc., 1993.

[16] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach.* Pearson Education, 2003. ISBN: 0137903952.

[17] Fartash Vasefi and Z. Abid. *Low Power N-bit adders and multiplier using lowest-number-of-transistor 1 bit adders.* University of Western Ontario Canada, 2005.

[18] Walker, Hilder, and Tyrrel. *Evolving Variability-Tolerant CMOS Designs.* University of York UK, 2008.

[19] Žaloudek Luděk and Sekanina Lukáš. *Transistor-Level Evolution of Digital Circuits Using a Special Circuit Simulator.* Faculty of Information Technology, Brno University of Technology, 2008.