Fast Navigation in Hierarchical Circuits

Björn Schieffer

Report – 14/1992

Fachbereich 14 – Informatik
Universität des Saarlandes
W 6600 Saarbrücken, Germany

Sonderforschungsbereich 124 – TP B1

Contents

Abstract

In this paper, three demands of a *good* logic simulator for hierarchical circuits are postulated: stack oriented signal assignment, hierarchical circuit representation and linear runtime. It is shown that the intuitive idea of a hierarchical simulator does not hold condition 3 having a runtime of $\Theta(n^2)$ in the circuit size. May this be improved or is there a contradiction in these conditions? Not in the way that there are two conflicting groups, as for each pair of these conditions an algorithm may be presented. But it is shown that there is no way to fulfill all of them at once!

Introduction

To deal with complex systems, hierarchical representations are often needed. They shorten memory usage and allow single computation for several instances of the same object. So hierarchy might be such a sweat thing if it had not such a big disadvantage compared to flat representations: the hierarchy borders separate objects and may result in expensive information transports. A way to avoid this – called *fast navigation* – is presented in this paper. As an example of a complex system and a tool working on it, integrated circuits and the task of logic simulation is used.

In the first chapter a simple simulator for flat circuit representations and its recursive extension for hierarchical circuits are presented. This extension increases the runtime from $\Theta(n)$ to $\Theta(n^2)$ in the circuit size. To decide the question whether this may be improved or not, in the second chapter three conditions of a *good* simulator are postulated. The method of *fast navigation* is used to combine hierarchical aspects with linear runtime. Last, a fully hierarchical simulator with linear runtime is shown to be impossible. Formal definitions of the terms used are given in appendix A.

Acknowledgements

I want to thank Prof. Dr. Günter Hotz for his encouragements and Dr. Uwe Sparmann for helpful comments and discussions.

Chapter 1

A simple simulator

Formal definitions are unavoidable for exact proofing. To make the paper better readable, I banish them into appendix A. I will give an informal explanation of the terms at their first use that should make it able to understand the work.

A *circuit* consists of a set of *modules*. A module has inputs and outputs, assigned with values out of $\mathbf{B} = \{0, 1\}$. The assignment of the outputs is determined by the assignment of the inputs and the *behavior* of the module. Modules are connected to each other by *signals*. There is a special module τ called *border* which contains the primary inputs and outputs of the circuit. Therefore a circuit may be seen as a module again. Such modules are called *box cells* to discriminate between the not disintegratable *basic cells*.

You may get a *flat circuit* consisting only of basic cells by *disintegrating* all of the box cells of such a *hierarchical circuit*.

The behavior of the the whole circuit depends of the behavior of the basic cells and their connections. To determine it – that means to determine the assignment of the primary outputs given the assignment of the primary inputs – is the task of *logic simulation*.

1.1 A simulator for flat circuits

Many simulators [AS87, BHKS91, HSS92] need the circuit to be flat. Hierarchical circuits have to be disintegrated first. Because simulation is easier for flat circuits I also restrict to them in the first approach.

And I only consider circuits with a topological order on their modules. A circuit must not have any cycle. Such circuits are called *combinational*. (One may reduce simulation of *sequential* circuits to simulation of combinational circuits by cutting the cycles and introducing so called *secondary* inputs and outputs.)

The topological ordering of a combinational circuit guarantees, that the inputs of a module m_1 do not depend on the outputs of a module m_2 when

$m_1 \leq_{top} m_2$. Thus, the assignment β of the circuit may be calculated from the assignment of its modules in topological order. A formal description of that is given by $simulator_1$.

$C = (T, M, S, \tau, \rho, type)$;
$\omega_{in} \in \mathbf{B}^{indegree(\tau)}$;

$simulator_1(C, \omega_{in})$
    $\beta : S \rightarrow \mathbf{B}$;
    calculate topological ordering $(M \backslash \{\tau\}, \leq_{top})$;
    foreach $m \in M \backslash \{\tau\}$ in topological order
    begin
        $\beta(\rho^{out}(m)) := behave(type(m))(\beta(\rho^{in}(m)))$;
    end;
return $\beta(\rho^{out}(\tau))$;

1.1.1 Runtime

As it is usual for logic simulation, we take the size of a circuit as the number of its signals.

The topological ordering $\leq_{top}$ is independent of the circuits inputs assignment ω_{in}. It may be calculated in linear time [Meh84]. So it is possible to calculate the topological ordering in a preprocessing phase. So we do not have to take it into account to determine the runtime of $simulator_1$.

$simulator_1$ calculates the behavior of each module of the circuit. The computation costs of one single module are given with the size of its border. This is motivated by the reading and writing of the assignment of its inputs and outputs. The real calculation consists of simple operations, which have not to be taken into account related to the expensive read and write operations.

With exception of the module τ describing the border of the circuit, the size of all modules is limited – it makes no sense packing arbitrary large functions in basic cells. The size of the border of τ and the number of modules are limited by the number of signals $|S|$. So we get runtime $cost_1$ for $simulator_1$:

$$cost_1(T, M, S, \tau, \rho, type) = \sum_{m \in M} |\rho(m)| = O(|M| + |\rho(\tau)|) = O(|S|)$$

Flat combinational circuits may be simulated in linear time!

1.2 A simulator for hierarchical circuits

A recursive extension of $simulator_1$ leads to a simple simulator for hierarchical circuits [App88, WM89, Mau91]. Beginning at the top-level, the assignment of

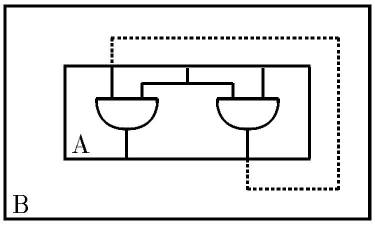

Figure 1.1: virtuelles Feedback

the outputs of each module is determined in topological order. The calculation of box cells is done by a recursive call of the simulation routine.

Therefore a topological ordering is needed for all circuits C_i. In contrast to the flat circuit this is **not** equivalent to the circuit being combinational. There may be *virtual feedbacks* [Bee92], that means cycles in the hierarchical circuit representation, that will vanish by disintegration of the circuit! The dotted line in figure 1.1 leads from one output of module A to one of its inputs in circuit B. But this output is independent from that input. So there exists a topological order of the modules of the flat circuit. (first the right and-gate then the left one.) The feedback of the dotted line is only virtual. It is shown in [KS91], that to overcome all virtual feedbacks the size of the circuit representation may exponentially increase. The search for a minimal disintegration of the circuit that overcomes all virtual feedbacks is a NP-complete problem [KS91, WM89]. In [Sch91] it is described how to simulate circuits in presence of virtual feedbacks. So I may now consider only circuits with none of them.

$simulator_2$ is a formal description of the recursive extension of $simulator_1$:

$H = (C_1, C_2, \cdots, C_k)$;
$C_k = (T_k, M_k, S_k, \tau_k, \rho_k, type_k)$;
$\omega_{in} \in \mathbf{B}^{indegree(\tau_k)}$;

$simulator_2(H, \omega_{in})$
    $\beta : S_k \to \mathbf{B}$;
    calculate topological ordering $(M_k \backslash \{\tau_k\}, \leq_{top})$;
    foreach $m \in M_k \backslash \{\tau_k\}$ in topological order
    begin
        if $type_k(m) = t_j$
            then $\beta(\rho_k^{out}(m)) := simulator_2((C_1, C_2, \cdots, C_j), \beta(\rho_k^{in}(m))$;
            else $\beta(\rho_k^{out}(m)) := behave(type_k(m))(\beta(\rho_k^{in}(m)))$;
        fi;
    end;
return $\beta(\rho_k^{out}(\tau_k))$;

1.2.1 Runtime

The nodes of the *structure tree* of a hierarchical circuit are *instances of modules* where the root is the instance of the circuit itself. The children of an instance are instances of the modules of the circuit describing the instance. So, basic cells are the leafs of the structure tree. For formal details see appendix A.

The runtime of $simulator_2$ is given by the function $cost_2 : IoM_k \rightarrow \mathbf{N}$. As for $simulator_1$ the time needed to calculate the topological ordering is left out of discussion.

The runtime of $simulator_2$ is determined by the size of the circuits border – representing the parameters in the recursive procedure call – and the work in the **if**-statement inside the loop over the modules out of M_k. Like in $simulator_1$ the costs of the **else**-case are given with the size of the border of the current module which is a basic cell. In the *then*-case the current module is a box cell and stands for a whole circuit. We have to take the runtime of $simulator_2$ for this circuit as the costs of the module. So we have the simulation cost of a basic cell as its border size and the simulation costs of a box cell as its border size plus the simulation costs of all its modules.

Lemma 1.1 *The runtime of a module instance is the sum of the border size of all nodes in its subtree of the structure tree.*

The proof is made by induction over the size of the structure tree

$|V| = 1$: m is a basic cell and the assertion is true.

$|V| > 1$: m stands for a box cell. Its simulation costs are its border size plus the simulation costs for its modules. With the assumption for the corresponding subtrees the assertion is fulfilled.

□ (Lemma 1.1)

The runtime of $simulator_2$ is influenced by the structure of the simulated circuit.

Definition 1.1 *A circuit* $H = (C_1, C_2, \cdots, C_k)$ *is said to be* reasonable *iff it contains no* package circuits *– that are circuits containing only one box cell and no basic cells. Only basic cells may be packed in a circuit.*

Formal: For all subcircuits C_i*:*

$$M_i = \{\tau_i, m\} \Rightarrow type_i(m) \in T_1$$

With this definition of reasonable circuits we have the following theorem:

Theorem 1 *Let* $n := size(flat(H))$ *be the size of an arbitrary circuit* H*. Then the runtime is not limited in a function of* n*. But if* H *is reasonable,* $simulator_2$ *has runtime* $\Theta(n^2)$*.*

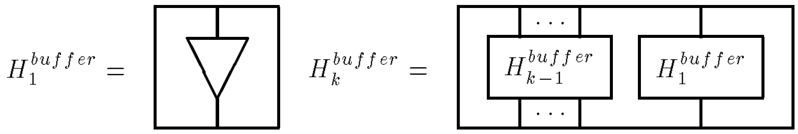

Figure 1.2: buffer circuit

Proof:
The proof of that theorem is divided into three parts. First I want to show the existence of circuits with runtime $\Omega(n^2)$ by a parameterized family of circuits H_k^{buffer} (for $k = 1, 2, \cdots$). Then, I explain with the help of another family of now unreasonable circuits $H_k^{babushka}$ how the runtime for a circuit may be arbitrary increased without changing its size. Last, the upper limit n^2 is given for reasonable circuits.

to 1:
Let the family of circuits H_k^{buffer} (for $k = 1, 2, \cdots$) be given by figure 1.2. The simulation costs for H_k^{buffer} are the sum of the simulation costs for H_{k-1}^{buffer} with the costs for H_1^{buffer} and the size of the border of H_k^{buffer}. The costs for H_1^{buffer} are the sum of its border size and the border size of the basic buffer cell.

$$cost_2(H_1^{buffer}) = 2 + 2 = 4$$

$$\begin{aligned} cost_2(H_k^{buffer}) =& |\rho_k(\tau_k)| + cost_2(H_1^{buffer}) + cost_2(H_{k-1}^{buffer}) \\ =& 2k + 4 + cost_2(H_{k-1}^{buffer}) \\ =& 4 + \sum_{i=2}^{k}(2k+4) \\ =& 4 + 4(k-1) + 2(\tfrac{k(k+1)}{2} - 1) \\ =& k^2 + 5k - 2 \end{aligned}$$

The simulation costs of these circuits increase quadraticly with the size of the circuits which is $size(H_k^{buffer}) = size(flat(H_k^{buffer})) = 2k$.
Comment:
To transport the value of the leftmost input of the circuit to its corresponding buffer, it has to cross $k-1$ circuit borders. Each crossing stand for a parameter in a procedure call. As the size of the whole circuit itself is only $2k$, that is very much for a single transport. To get hierarchical algorithms with linear runtime we demand in the next chapter a single transport only to have constant costs!

to 2:
Let me introduce another family of circuits $H_k^{babushka}(k = 1, 2, \cdots)$ to show that the runtime of $simulator_2$ may not be limited by a function in the circuit size n for unreasonable circuits. Thereby each circuit of the family has the same

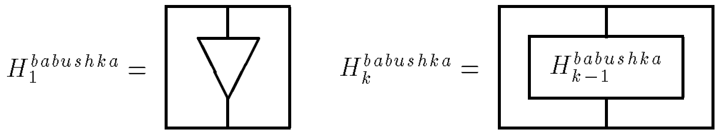

Figure 1.3: babushka circuits

size. But the simulation costs increase linear in k. Again the circuit $H_1^{babushka}$ is a buffer. The circuit $H_k^{babushka}$ is build by *packing* the circuit $H_{k-1}^{babushka}$. See also figure 1.3. Because $flat(H_k^{babushka}) = H_1^{babushka}$ the circuit size as the number of signals of the disintegrated circuit is $n = size(H_k^{babushka}) = size(flat(H_k^{babushka})) = 2$. Therefore the runtime $cost_2$ of $simulator_2$ is:

$$cost_2(H_1^{babushka}) = 4$$

$$cost_2(H_k^{babushka}) = |\rho_k(\tau_k)| + cost_2(H_{k-1}^{babushka}) = 2 + cost_2(H_{k-1}^{babushka}) = 2k + 2$$

to 3:
The circuits $H_k^{babushka}$ in figure 1.3 are not *reasonable* in the sense of definition 1.1. It is left to show, that the limit of $\Omega(n^2)$, shown in part 1, may not be exceeded by reasonable circuits – that means the runtime is limited in $O(n^2)$.

Lemma 1.2 *The number of nodes $s = |V|$ of a structure tree $G = (V, E, root)$ of a reasonable circuit is limited by the triple number of its leafs b: $s \leq 3b - 1$.*

The proof of the lemma is made by structural induction.

$degree(root) = 0$**:**
$\Rightarrow s = 1, b = 1 \Rightarrow s \leq 3b - 1$

$degree(root) = 1$**:**
Now we need the *reason* of the circuit. It implies that if m is the only module of a circuit C_i, then m has to be a basic cell. So the child of a root with degree 1 has to be a leaf!
$\Rightarrow s = 2, b = 1 \Rightarrow s \leq 3b - 1$

$degree(root) > 1$**:**
The structure tree $G = \bigcup(B_1, B_2, \cdots, B_k)$ corresponds to a circuit with $k > 1$ modules. Let $B_1, B_2, \cdots, B_k$ be their structure trees, $b_1, b_2, \cdots, b_k$ and $s_1, s_2, \cdots, s_k$ the number of leafs and number of nodes of these trees. With the assumption for them follows:

$$s = 1 + \sum_{i=1}^{k} s_i \leq 1 + \sum_{i=1}^{k} (3b_i - 1) = 1 + 3b - k \leq 3b - 1$$

□ (Lemma 1.2)

We are now able to proof the last point of theorem 1. According to Lemma 1.1 the simulation costs of $simulator_2$ are given with the sum of the border size of each node of the structure tree. Each border size is limited in the number of signals of the disintegrated circuit. The number of nodes of the structure tree is also limited in it. That's because the number of nodes are in order of the number of leafs (Lemma 1.2) which correspond to the modules of the disintegrated circuit which are limited in the number of signals. So we have the square of the number of signals of the disintegrated circuit – what is exactly the size of the circuit – as an upper bound for the runtime of $simulator_2$.

□ (Theorem 1)

Chapter 2

Fast navigation

We have seen two algorithms for circuit simulation so far and will meet two others. To compare these algorithms I introduce three claims of a *good* simulator.

S_1: **stack for signal assignment:** $simulator_1$ needs the assignment β of all the signals of the disintegrated circuit while $simulator_2$ only uses the assignment of the signals of the current circuit and the ones whose simulation is interrupted for the simulation of a subcircuit. After the simulation of such a subcircuit it drops its inner signals and only stores the assignment of its outputs. This may be regarded as the implementation of a stack for the signal assignment. Before starting a simulation of a subcircuit, memory is allocated for the local signals and it is deleted afterwards.

S_2: **hierarchical circuit representation:** Not only the *dynamic* memory needed for the signal assignment, but the *static* memory space to store the description of the circuit may be hierarchical or flat. Surely, a good simulator should only use hierarchical circuit representations.

S_3: **linear runtime**

$simulator_1$ fulfills condition S_3 but not S_1 and S_2. $simulator_2$ in contrast does it with conditions S_1 and S_2 but not S_3. Is there a contradiction in these conditions? The answer is: not in the way that the conditions may be divided into two conflicting groups. For each pair of conditions an algorithm may be found that fulfills both. I will show later that there exists none for all three claims.

There are three pairs of conditions. For the pair $\{S_1, S_2\}$ we met $simulator_2$. This chapter is dedicated to the algorithms $simulator_3$ and $simulator_4$ that fulfill $\{S_2, S_3\}$ and $\{S_1, S_3\}$ respectively.

Differences in the algorithms lie in the storage of the signal assignment and the transport of signal values from the point of creation – the buffer – to the points of usage – the access.

To argue in visual terms see figure 2.1: The value of a signal is given by the calculation of the source and, for the computation of another gate, has to be transported to one of its inputs. I want to visual the information flow in the different algorithms by arrows. An arrow in the direction of the information flow should illustrate a write operation at the source of the arrow to its destination. An arrow against the flow describes a read operation at the destination from the source. The arrows may be realized with pointers and represent indirect read and write operations.

The claims for the algorithms may now be written in terms of arrows:

P_1**: stack for signal assignment:** the realization of the signal assignment as a stack implies a *visibility* of instances of modules which is passed to their corresponding signals. An instance does not *see* all others but only its ancestors, that means all instances on the path from the root to itself in the structure tree.

The destinations of the arrows have to be visible from the sources as a reading or writing operation to the destination is made there. A stack realization is only possible when all the pointers are directed on valid memory.

That means, that the arrows may not show *into* boxes. At best, they may show to the border of a box as the border is visible from the father of a box.

P_2**: hierarchical circuit representation:** All arrows together built a representation of the circuit. For a hierarchical storage, the arrows of one module have to be independent of its instances. If the arrows would depend on the instance, we had to store them for each instance and that would mean a flat circuit representation.

So the arrows may not *leave* boxes in the figures as the *outside* of a box depends on the instance while the inside does not.

P_3**: linear runtime:** The sum of all arrows in a figure has to be linear in the number of signals of the disintegrated circuit. A sufficient criterion for that is that all transports from a buffer to an access are built of a constant amount of arrows.

In $simulator_1$ the driving and access signal are identical. Imagine figure 2.2 without the border lines. As the buffer is not visible from the access this arrow violates condition P_1. It also violates P_2: in another instance of the module containing the access gate it may be driven by another buffer. Condition P_3 is surely fulfilled, the number of arrows is constant for each transport!

Figure 2.3 shows the transport in $simulator_2$. At each circuit border the value of a signal is propagated by a copy operation as a procedure parameter. Therefore each arrowhead is visible from the start point (condition P_1). The

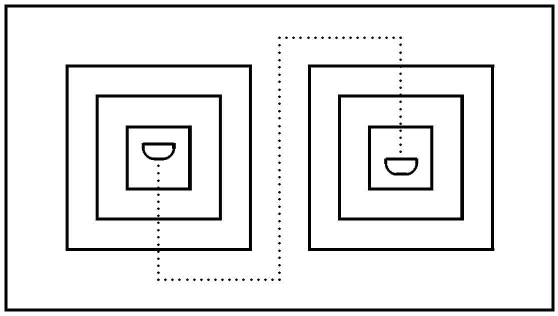

Figure 2.1: buffer and access of a signal

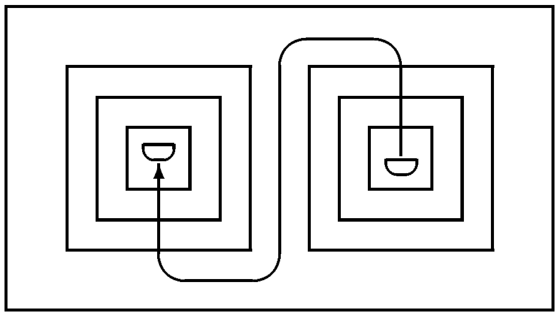

Figure 2.2: transport in $simulator_1$

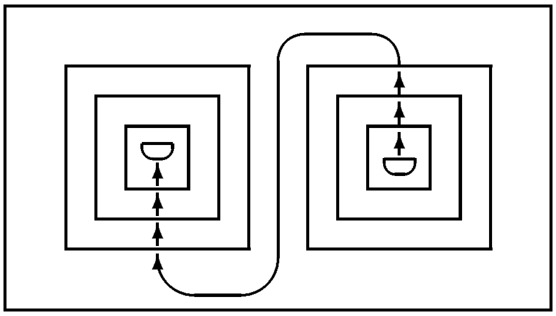

Figure 2.3: transport in *$simulator_2$*

arrows reflect the signal structure of the hierarchical circuit and therefore are independent of the instances (condition P_2). Unfortunately condition P_3 is violated: We have seen in the last chapter, that in the circuit family H^{buffer} the length of such a transport may reach the size of the circuit and the sum over all transports may be quadratical in it.

Having a close look at the last figures you will see that each signal value first has to past some circuit outputs and then some circuit inputs. Surely each of this two sequences may be empty, but there will never be an output after an input in the signal transport. This observation is only true when the circuit does not contain any *feed through*. That are signals from an input to an output of the circuit. From the point of simulation such signals seem to be stupid, but they are very useful to get a regular design of parametric circuits [BHK+87]. In the following, I only consider circuits with no feed throughs. This is justified in the ease of deleting feed throughs for the purpose of simulation [KS91].

The above observation allows us to find algorithms that fulfill condition P_1 in parallel with P_3 and condition P_2 in parallel with P_3. So, let me first formalize and proof it.

I will denote the father and son relation between modules in the structure tree by $\gamma_1 \rightarrow \gamma_2$ when γ_1 is the son of γ_2 and $\gamma_1 \leftrightarrow \gamma_2$ when either $\gamma_1 \rightarrow \gamma_2$ or $\gamma_1 \leftarrow \gamma_2$. This may be extended to signals with the additional condition that they are driven by the same buffer. $\gamma_1.s_1 \rightarrow \gamma_2.s_2$ means that γ_1 is a instance of a module and the son of γ_2 and the signal s_1 of module γ_1 is connected with the signal s_2 of γ_2. The transitive closures of $\rightarrow$ and $\leftrightarrow$ are denoted by $\rightarrow^\star$ and $\leftrightarrow^\star$ respectively.

Theorem 2 *If the instance of a signal $\gamma_1.s_1$ is source of $\gamma_2.s_2$ in a circuit C_i, in formal terms: $source_i(\gamma_2.s_2) = \gamma_1.s_1$, then there are an* increasing *and a* decreasing *sequence of instances of signals that connect $\gamma_2.s_2$ and $\gamma_1.s_1$:*

$$\exists \gamma_0.s_0 : \gamma_2.s_2 \rightarrow^\star \gamma_0.s_0 \leftarrow^\star \gamma_1.s_1$$

Thereby, the intermediate *signal instance* $\gamma_0.s_0$ *is unique.*

Proof:

1. Because $\gamma_1.s_1$ is the source of $\gamma_2.s_2$ there has to be a connection of the way $\gamma_2.s_2 \leftrightarrow^\star \gamma_1.s_1$. We have to proof that it contains no *inversion.* That means, after a $\leftarrow$ in the sequence may not appear a $\rightarrow$.

   Suppose there is such an inversion: Then there is point where a $\rightarrow$ directly follows a $\leftarrow$:

   $$\exists \gamma_{\mu-1}.s_{\mu-1} \leftarrow \gamma_\mu.s_\mu \rightarrow \gamma_{\mu+1}.s_{\mu+1}$$

   That means, γ_μ is the son of $\gamma_{\mu-1}$ and also the son of $\gamma_{\mu+1}$ in the structure tree. That implies that $\gamma_{\mu-1} = \gamma_{\mu+1}$. So we have found a (forbidden) feed through in the module γ_μ.

2. It is left to proof the uniqueness of $\gamma_0.s_0$. There is only one sequence of the form $\gamma_2.s_2 \leftrightarrow^\star \gamma_1.s_1$. If there would be several the signals had to build a *fan in.* As the relations $\leftarrow$ and $\rightarrow$ are disjoint it exists only one $\gamma_0.s_0$ with $\gamma_2.s_2 \rightarrow^\star \gamma_0.s_0 \leftarrow^\star \gamma_1.s_1$.

$\square$ (Theorem 2)

2.1 Hierarchical signal assignment

We now are searching for an algorithm with a stack oriented signal assignment and constant costs for each signal value transport. The problem is the mutual *invisibility* of the source and any drain of each signal.

Definition 2.1 (visibility)
Visible *for a signal instance are only signals of module instances in the path from the root to its own module instance.*

$$visible : IoS_k \rightarrow \wp(IoS_k)$$

$$visible(\gamma.s) := \{\gamma_0.s_0 | \gamma.s \rightarrow^\star \gamma_0.s_0\}$$

This mutual invisibility prevents us from writing the signal value from the location of the source to the location of the drain. The solution to the problem is shown in figure 2.4. Theorem 2 tells us the unique existence of a signal instance for each pair of source and drain that is visible from both of them. The source may write its value to it and the drain may read it from there later on. We see in figure 2.4 that condition P_1 holds, as the arrows point not inside a module. But the arrows violate condition P_2 leaving the modules. Each transport is build out of three arrows, so condition P_3 holds.

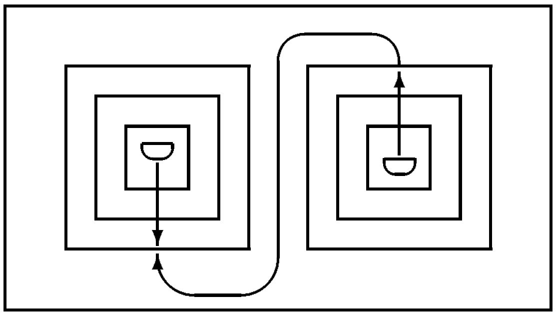

Figure 2.4: fast transport via visible signals

A formal description is given with the two functions ν^{out} and ν^{in}. ν^{out} gives us the set of signal instances $\gamma_0.s_0$ with the source $\gamma_2.s_2$ and that are used by any drains of $\gamma_2.s_2$.

$$\nu^{out} : IoS_k \rightarrow \wp(IoS_k)$$

$$\nu^{out}(\gamma_2.s_2) := \{\gamma_0.s_0 | \exists \gamma_1.s_1 \in drain_k(IoS_k), \gamma_2.s_2 \rightarrow^{\star} \gamma_0.s_0 \leftarrow^{\star} \gamma_1.s_1\}$$

The *middle* signal instances given their values by ν^{out} have to be found by the corresponding drains. This is the task of ν^{in}. It assigns $\gamma_1.s_1$ that signal instance $\gamma_0.s_0$, which has the same drain and knows it value. Theorem 2 assures us that ν^{in} is well defined.

$$\nu^{in} : IoS_k \rightarrow IoS_k$$

$$\nu^{in}(\gamma_1.s_1) := \{\gamma_0.s_0 | \exists \gamma_2.s_2 \in source_k(IoS_k), \gamma_2.s_2 \rightarrow^{\star} \gamma_0.s_0 \leftarrow^{\star} \gamma_1.s_1\}$$

With the help of that flat circuit description functions ν^{out} and ν^{in}, we now may give the algorithm $simulator_3$ including a stack for the signal assignment β with constant transport costs.

$H = (C_1, C_2, \cdots, C_k)$;
$1 \leq i \leq k : C_i = (T_i, M_i, S_i, \tau_i, \rho_i, type_i)$;
$\omega_{in} \in \mathbf{B}^{indegree(\tau_k)}$;
$\omega_{out} \in \mathbf{B}^{outdegree(\tau_k)}$;
$\nu^{out} : Ios_k \rightarrow \wp(IoS_k)$;
$\nu^{in} : Ios_k \rightarrow IoS_k$;

$simulator_3(H, \omega_{in}, \nu^{in}, \nu^{out})$
    $\beta \leftarrow \epsilon$; (initialize stack)
    $calculate_3(t_k)$;
return ω_{out};

$\gamma \in IoM_k$;

$calculate_3(\gamma)$
$\quad t_i \leftarrow type(\gamma)$;
$\quad \beta \leftarrow \beta \cup (\{\gamma.s | s \in S_i\} \rightarrow \mathbf{B})$; (allocate memory for local signals)
$\quad$ **if** $(i = k)$ **then** $\beta(t_k.\rho_k^{in}(\tau_k)) := \omega_{in}$; **fi**;
$\quad$ **foreach** $m \in M_i \backslash \{\tau_i\}$ in topological order
$\quad$ **do**
$\quad\quad$ **if** $(\exists j, type_i(m) = t_j)$
$\quad\quad\quad$ **then** $calculate_3(\gamma m)$;
$\quad\quad\quad$ **else** $\beta(\gamma.\rho_i^{out}(m)) := behave(type_i(m))(\beta(\nu^{in}(\gamma.\rho_i^{in}(m))))$
$\quad\quad\quad\quad$ **foreach** $s \in \rho_i^{out}(m)$
$\quad\quad\quad\quad$ **do**
$\quad\quad\quad\quad\quad \beta(\nu^{out}(\gamma.s)) := \beta(\gamma.s)$;
$\quad\quad\quad\quad$ **od**
$\quad\quad$ **fi**
$\quad$ **od**
$\quad$ **if** $(i = k)$ **then** $\omega_{out} := \beta(t_k.\rho_k^{out}(\tau_k))$; **fi**;
$\quad \beta \leftarrow \beta \backslash (\{\gamma.s | s \in S_i\} \rightarrow \mathbf{B})$ (deallocate memory for local signals)

In $simulator_3$ the procedure $calculate_3$ is called for each module instance γ. For *non reasonable* circuits in the sense of the last chapter this may arbitrary increase the runtime. This may easily be repaired by changing the procedure call $calculate_3(\gamma m)$ into $calculate_3(\gamma m \mu(t_j))$. The function $\mu : T_{k+1} \rightarrow \bigcup_i IoM_i$ helps to jump over the packing modules and go directly to that module that contains the basic cell next dealed with.

$$\mu(t_j) = \omega :\Leftrightarrow$$

1. $\forall m, \omega = \gamma_1 m \gamma_2, type_j(\gamma_1) = t_i \Rightarrow m = \min_{\leq_{top}} M_i$
2. ω has maximal length with property 1.

With that change we receive a linear runtime even for *non reasonable* circuits, what I now want to proof for *reasonable* circuits:

Theorem 3 *The runtime of simulator*$_3$ *for a* reasonable *circuit* H *is* $O(size(H))$.

Proof:
$simulator_3$ uses the procedure $calculate_3$ to prevent the expensive parameter variables of the circuit border in the recursion. We now have a stack organization but its costs are given with the constant procedure overhead. Another difference to $simulator_2$ lies in the **else** statement: There is a loop to write the output values of the corresponding basic cell to signal instances described by ν^{out}. But the costs of this loop are included in the border size as the costs of a

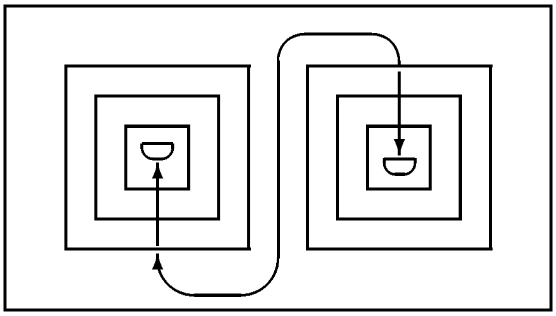

Figure 2.5: fast navigation in $simulator_4$

basic cell. So we again have the sum over all basic cell borders which has been shown to be in the order of the circuit size. It is left to deal with the **then** statement: As we now only have one parameter the costs are in the order of the size of the structure tree. That is according to lemma 1.2 in the order of the circuit size. □ (Theorem 3)

2.2 Hierarchical circuit description

The last of the promised algorithms should use a hierarchical description of the circuit (S_2) while having linear runtime (S_3). In figure 2.5 you find the arrow description of its signal transports. The circuit description, given by a representation of the arrows, is hierarchical, as according P_2 the arrows only point inside of modules. Each transport consists of three arrows, so condition P_3 holds. Pointing inside the modules, the head of the arrows are not visible from the starting points and condition P_1 is violated.

Figure 2.5 describes the possibility of *fast navigation* through hierarchical descriptions of integrated circuits. But the idea of that *fast navigation* is not restricted to circuits.

Definition 2.2 (fast navigation)
A fast navigation $\alpha = (\alpha^{in}, \alpha^{out})$ *of a hierarchical circuit* $H = (C_1, C_2, \cdots, C_k)$ *describes the connection of the border of each module with the corresponding inner signal instances without the roundabout of other module borders.*

$$\alpha_i^{in} : \{\rho_i^{in}(\tau_i)\} \rightarrow \wp(IoS_i), \quad \alpha_i^{in}(s) := drain_i(t_i.s)$$

$$\alpha_i^{out} : \{\rho_i^{out}(\tau_i)\} \rightarrow IoS_i, \quad \alpha_i^{out}(s) := source_i(t_i.s)$$

To use the *fast navigation*, we *press* the signal value via α^{in} to its destination. We get it via the arrow described by α^{out}. This is done by an extension of the signal assignment β to $\widetilde{\beta}$ that distinguish between outputs of basic cells that are handled with just like before and outputs of box cells that are indirectly addressed by α^{out}.

$$\widetilde{\beta} : drain_k(IoS_k) \rightarrow source_k(IoS_k)$$

$$\widetilde{\beta}(\gamma.s) := \begin{cases} \beta(\gamma m \alpha_j^{out}(\rho_j^{out}(\tau_j)_\mu)) & : \begin{array}{l} type(\gamma) = t_i, \exists m \in M_i, \\ type_i(m) = t_j, s = \rho_i^{out}(m)_\mu \end{array} \\ \\ \beta(\gamma.s) & : \text{else} \end{cases}$$

So we now are able to write down $simulator_4$.

$H = (C_1, C_2, \cdots, C_k)$;
$1 \leq i \leq k : C_i = (T_i, M_i, S_i, \tau_i, \rho_i, type_i)$;
$\omega_{in} \in \mathbf{B}^{indegree(\tau_k)}$;
$\alpha_i^{in} : \{\rho_i^{in}(\tau_i)\} \rightarrow \wp(IoS_i)$;
$\alpha_i^{out} : \{\rho_i^{out}(\tau_i)\} \rightarrow IoS_i$;

$simulator_4(H, \omega_{in}, \alpha^{in}, \alpha^{out})$
    $\beta : source_k(IoS_k) \cup drain_k(IoS_k) \rightarrow \mathbf{B}$;
    $\beta(t_k.\rho_k^{in}(\tau_k)) := \omega_{in}$;
    $calculate_4(t_k)$;
return $\widetilde{\beta}(\rho_k^{out}(\tau_k))$;

$\gamma \in IoM_k$;

$calculate_4(\gamma)$
    $t_i \leftarrow type(\gamma)$;
    foreach $m \in M_i \backslash \{\tau_i\}$ in topological order
    do
        if $(\exists j, type_i(m) = t_j)$
            then foreach $s \in S_i, s = \rho_i^{in}(m)_\mu, s \notin \rho_i^{in}(\tau_i)$
                do
                    foreach $\gamma_1.s_1 \in \alpha_j^{in}(\rho_j^{in}(\tau_j)_\mu)$
                    do
                        $\beta(\gamma\gamma_1.s_1) := \widetilde{\beta}(\gamma.s)$;
                    od
                od
                $calculate_4(\gamma m)$;
            else $\beta(\gamma.\rho_i^{out}(m)) := behave(type_i(m))(\widetilde{\beta}(\gamma.\rho_i^{in}(m)))$;
        fi

od

For the running time of $simulator_4$ it is decisive whether the simulated circuit is *reasonable* or not. That is – just like for $simulator_3$ – caused by an unrestricted call of procedure $calculate_4$. So it may be eliminated the same way as shown for $simulator_3$ and I may restrict on *reasonable* circuits.

Theorem 4 *The runtime of* $simulator_4$ *for* reasonable *circuits* H *is* $O(size(H))$.

Proof:
The runtime of $simulator_4$ is dominated by the one of $calculate_4$. Its work may be divided: into the **then**-case and the **else**-case of the **if** statement.

In the **else**-case you get the usual sum over the border size of the basic cells which is several times mentioned to be in order $O(n)$ of the circuit size $n = size(H)$.

In the **then**-case we have the recursive procedure call and two interlaced loops. With lemma 1.2 about the size of the structure tree of *reasonable* circuits the costs of the procedure calls are limited by $O(n)$. So we are left with the loops: because $s \notin \rho_i^{in}(\tau_i)$ the signal instance $\gamma.s$ is visible for $\gamma_2.s_2$, that means $\gamma_2.s_2 \rightharpoondown^\star \gamma.s$. As on the other side $\gamma.s \leftharpoondown^\star \gamma_1.s_1$, we have $\gamma_2.s_2 \rightharpoondown^\star \gamma.s \leftharpoondown^\star \gamma_1.s_1$. Since the signal instance $\gamma.s$ is unique for $\gamma_1.s_1$ and $\gamma_2.s_2$ according to theorem 2, we are sure that the inner signal instance $\gamma_1.s_1$ is not multiple used. So the costs of the loops are limited by the drains what are the inputs of the basic cells. Summarized over all inputs we again have costs $O(n)$. $\square$ Theorem(4)

2.3 Is there an optimal algorithm?

I postulated three demands for a *good* hierarchical circuit simulator: **signal assignment as a stack**, **hierarchical circuit description** and **linear runtime**. For each pair of conditions an algorithm may be found that fulfills both of them. What we need for total contentment is an algorithm that holds all three conditions. The destructive task left in this paper is to show that such an algorithm does not exist.

Theorem 5 *There is no simulator for combinational circuits with stack oriented signal assignment, hierarchical circuit representation and linear runtime in the circuit size.*

To proof this theorem I use the signal value transport conditions in terms of arrows from page 12. According to condition P_1 arrows may not point into a module and according P_2 not out of it. The combination of these condition implies that arrows do not cross borders of modules. So we have to use the inputs and outputs of each module to propagate the signal values as we are not able to leave one out in the transport path. But this is a contradiction to the third condition P_3 as according to theorem 1 the sum over the border size of each module may increase unlimited for arbitrary circuits and quadraticly for *reasonable* ones. □ (Theorem 5)

Appendix A

Definitions

Definition A.1 (tree)
A tree $B = (V, E, root)$ *is a tripel consisting of a set of knode* V*, a set of edges* $E \subseteq \binom{V}{2}$ *and a special knode* $root \in V$*. Thereby it has to exists a unique connecting path for each two elements* $v_1 \in V$ *and* $v_2 \in V$*.*

A connecting path for two knodes $v_1 \in V$ *und* $v_2 \in V$ *is a sequence of knodes* $(u_1, u_2, \cdots, u_k)$ *with:* $v_1 = u_1, v_2 = u_k, \{u_i, u_{i+1}\} \in E\ \forall 1 \leq i < k$*.*

The degree *of a knode* $v \in V$ *is defined as* $degree(v) := |\{(v, w)|w \in V, (v, w) \in E\}|$*.*

By connecting the roots of several subtrees with a single knode, you get a new tree.

Definition A.2 (Conjunction of trees)
The conjunction $\bigcup$ *of* k *disjoint trees* $G_i = (V_i, E_i, root_i)$ *for* $1 \leq i \leq k$ *again is a tree:*

$$\bigcup(G_1, G_2, \cdots, G_k) := (V, E, root)$$

$$V := \{root\} \cup \bigcup_{i=1}^{k} V_i$$

$$E := \bigcup_{i=1}^{k} (\{root, root_i\} \cup E_i)$$

Sometimes a function is not only used once, but simultaneously for a sequence of objects. To increase readability I do not want to split the sequence but write the function applicated to the whole sequence.

Definition A.3 (simultaneous function application)
For a function $f : A \rightarrow B$ *the* simultaneous function application $f : A^\star \rightarrow B^\star$ *is defined as:*

$$f(a_1 a_2 \cdots a_l) := f(a_1) f(a_2) \cdots f(a_l)$$

A *circuit* consists of a set of *modules*. Modules are connected to each other by signals. There is a special module called *border* which contains the primary inputs and outputs of the circuit. Therefore a circuit may be seen as a module again. Such modules are called *box cells* to determine against the not disintegratable *basic cells*. Modules are *typed* to specify their behavior.

Definition A.4 (circuit)

1. *A* circuit C *is a tupel* $(T, M, S, \tau, \rho, type)$, *where* T, M *and* S *are finite sets of types, modules and signals.* $\tau \in M$ *is a special module representing the border of the circuit* C. $\rho = (\rho^{in}, \rho^{out})$ *with* $\rho^{in} : M \rightarrow S^*$ *and* $\rho^{out} : M \rightarrow S^*$ *connects the border of each module, that means its inputs and ouputs, with signals.* $type : M \setminus \{\tau\} \rightarrow T$ *types a module. There has to hold some syntactical conditions:*
   - *There exist two functions* $indegree : T \rightarrow \mathbf{N}$ *and* $outdegree : T \rightarrow \mathbf{N}$ *with*
   $$\forall m \in M : indegree(type(m)) = |\rho^{in}(m)|$$
   $$\forall m \in M : outdegree(type(m)) = |\rho^{out}(m)|$$
   They may be combined as $degree : T \rightarrow \mathbf{N}$ *by*
   $$\forall m \in M : degree(type(m)) = |\rho(m)|$$
   - $\forall s \in S :\ \exists$ *exactly one* $m \in M$ *and* $\mu \in \mathbf{N}$ *with*
   $$s = \begin{cases} \rho^{in}(m)_\mu & : \quad m = \tau \\ \rho^{out}(m)_\mu & : \quad else \end{cases}$$
   Each signal has exactly one source.
   - $\forall s \in S :\ \exists$ *at least one* $m \in M$ *and* $\mu \in \mathbf{N}$ *with*
   $$s = \begin{cases} \rho^{out}(m)_\mu & : \quad m = \tau \\ \rho^{in}(m)_\mu & : \quad else \end{cases}$$
   Each signal has at least one drain.
   - $\exists$ *no* μ, ν *with* $\rho^{out}(\tau)_\mu = \rho^{in}(\tau)_\nu$
   there are no feed throughs.
2. *A* hierarchical circuit H *is given by a sequence of circuits* $(C_1, C_2, \cdots, C_k)$ *with:* $C_i = (T_i, M_i, S_i, \tau_i, \rho_i, type_i)$ *and* $\forall i > 1 : T_i := T_{i-1} \cup \{t_{i-1}\}$. *The elements in* T_1 *are called* basic types, *the ones in* $t_1, t_2, \cdots, t_{k-1}$ box types. *The circuit* C_i, *for* $i < k$ *sometimes called* box, *may also include modules that represent circuits of the type* C_1 *to* C_{i-1}. *to make sure that the border*

of a module of type t_j in a circuit C_i is correctly connected to the inputs and outputs of C_j the following must hold:

$$\forall i > j : indegree_i(t_j) = |\rho_j^{in}(\tau_j)| =: indegree(t_j)$$

$$\forall i > j : outdegree_i(t_j) = |\rho_j^{out}(\tau_j)| =: outdegree(t_j)$$

That implies

$$\forall i > j : degree_i(t_j) = |\rho_j(\tau_j)| =: degree(t_j)$$

In a hierarchical description we have to determine between the description of a thing, in our terms a subcircuit or box, and any occurrences of it, called *instances*.

Definition A.5 (module instance)
The instances of modules *IoM_i of the i-th subcircuit C_i of a hierarchical circuit $H = (C_1, C_2, \cdots, C_k)$ is a recursively defined set. Simultaneously a function $type : IoM_i \rightarrow T_i$ is created.*

1. $t_i \in IoM_i, type(t_i) := t_i$
2. $\omega \in IoM_i, type(\omega) = t_j, m \in M_j \Rightarrow \omega m \in IoM_i, type(\omega m) := type_j(m)$
3. *IoM_i minimal under 1. and 2.*

From a instance of a module m in the circuit C_i and a box instance from **type** t_i in the circuit C_j a instance of the module m in the circuit C_j may be build:

Definition A.6 (Concatenation of module instances)
The concatenation *$\omega_1 t_i \omega_2$ of module instances ω_1 and $t_i \omega_2$ is the module instance $\omega_1 \omega_2$, if ω_1 is from type t_i.*

$$type(\omega_1) = t_i \Rightarrow \omega_1 t_i \omega_2 := \omega_1 \omega_2$$

Definition A.7 (signal instances)
The set of instances of signals *IoS_i of the i-th subcircuit C_i of a hierarchical circuit $H = (C_1, C_2, \cdots, C_k)$ are build from the module instances of C_i.*

$$IoS_i := \{\omega.s | \omega \in IoM_i, type(\omega) = t_j, s \in S_j\}$$

To describe the structure of a hierarchical circuit the *structure tree* is used. Its knodes are module instances where the root is the instance of the circuit itself. The children of an instance are instances of the modules of the circuit witch describes the instance. So, basic cells are the leafs of the structure tree.

Definition A.8 (structure tree)
The structure tree S_H *of a hierarchical circuit* $H = (C_1, C_2, \cdots, C_k)$ *is the conjunction of the structure trees of all modules* $M_k = \{m_1, m_2, \cdots, m_l\}$ *in* C_k*. Thereby the structure tree of a module of type* t_j *is the structure tree of the subcircuit* $(C_1, C_2, \cdots, C_j)$ *and the structure tree of a basic cell consists only of one knode.*

$$S_m := \begin{cases} S_{(C_1, C_2, \cdots, C_j)} & \exists j,\ type_k(m) = t_j \\ (\{root\}, \emptyset, root) & else \end{cases}$$

$$S_H := \bigcup (S_{m_1}, S_{m_2}, \cdots, S_{m_l})$$

It is important for a signal where its source is and where are the accesses on it. In a flat circuit this information is encoded into the connections ρ of the module borders with the signals. In hierarchical circuits things are a little more difficult, the functions *source* and *drain* are used.

To find the *source* of a signal instance you have to navigate through the circuit. We have to determine four cases:

1. You get to a primary input. Then you have found the source.
2. You get to the output of a basic cell. Again, the source is found.
3. You get to the output of a box cell. You have to go *down* into the box and continue the search there.
4. You get to an input of a box, but the box corresponds to a subcircuit and the input is not, like in case one, a primary one. You now have to leave the box and continue the search at the outside.

This is reflected in the following definition:

Definition A.9 (source of a signal)
The source $source_i : IoS_i \rightarrow IoS_i$ *of a signal instance* $\omega.s \in IoS_i$ *with* $type(\omega) = t_j$ *for a* $1 \leq j \leq i$ *of a circuit* $H = (C_1, C_2, \cdots, C_k)$ *is defined as:*

$$source_i(\omega.s) := \begin{cases} \omega.s & \omega = t_i, s \in \rho_i^{in}(\tau_i) \\ \omega.s & \exists m \in M_j, type_j(m) \in T_1, \\ & s \in \rho_j^{out}(m) \\ source_i(\omega m.s_1) & \exists m \in M_j, type_j(m) = t_l, \\ & s = \rho_j^{out}(m)_\mu, s_1 = \rho_l^{out}(\tau_l)_\mu, \\ source_i(\omega_1.s_1) & \omega = \omega_1 m, type(\omega_1) = t_l, \\ & s = \rho_j^{in}(\tau_j)_\mu, s_1 = \rho_l^{in}(m)_\mu, \end{cases}$$

The set of accesses of a signal instance contains all signal instances for witch it is the source and that are connected to the input of a basic cell or a primary output.

Definition A.10 (signal accesses)
The set of signal accesses $drain_i : IoS_i \to \wp(IoS_i)$ *is defined as:*

$$\begin{aligned} drain_i(\omega.s) := \; & \{\omega_1.s_1 | \omega_1.s_1 \in IoS_i, source_i(\omega_1.s_1) = \omega.s\} \cap \\ & (\{\omega_1.s_1 | type(\omega_1) = t_j, \exists m \in M_j, type_j(m) \in T_1, s_1 \in \rho_j^{in}(m)\} \\ & \cup \{t_i.s_1 | s_1 \in \rho_i^{out}(\tau_i)\}) \end{aligned}$$

You may get a *flat circuit* consisting only of basic cells by *disintegrating* all of the box cells of such a *hierarchical circuit.*

Definition A.11 (Disintegration of hierarchical circuits)
The disintegration $flat(H)$ *of a hierarchical circuit* $H = (C_1, C_2, \cdots, C_k)$ *is a flat circuit defined as:*

$$\begin{aligned} flat(H) &:= C = (T_1, M, S, \tau_k, \rho, type) \\ M &:= \{\omega | \omega \in IoM_k, \; type(\omega) \in T_1\} \cup \{\tau_k\} \\ S &:= source_k(IoS_k) \\ \rho^{in}(\omega m) &:= source_k(\omega.\rho_j^{in}(m)), \; type(\omega) = t_j \\ \rho^{out}(\omega m) &:= source_k(\omega.\rho_j^{out}(m)), \; type(\omega) = t_j \end{aligned}$$

As the size of a flat circuit usually the number of signals is used. To get a fair comparison between the flat and the hierarchical circuit description, I use the size of the disintegrated circuit as the size of a hierarchical circuit.

Definition A.12 (circuit size)
The size *of a circuit* C *and of hierarchical circuit* H *is defined as:*

$$size(C) := |S| \text{ if } C = (T, M, S, \tau, \rho, type)$$

$$size(H) := size(flat(H))$$

I only consider circuits with a topological order on their modules. A circuit must not have a cycle. Such circuits are called *combinational.* (One may reduce simulation of *sequential* circuits to simulation of combinational circuits by cutting the cycles and introducing so called *secondary* inputs and outputs.)

Definition A.13 (combinational circuit)

1. *A* combinational circuit *is a circuit* $C = (T, M, S, \tau, \rho, type)$ *with a topological ordering* $(M \backslash \{\tau\}, \leq_{top})$ *on its modules with:*

$$m_1 \leq_{top} m_2 \Rightarrow \forall \mu, \nu : \rho^{in}(m_1)_\mu \neq \rho^{out}(m_2)_\nu$$

2. *A hierarchical circuit* $H = (C_1, C_2, \cdots, C_k)$ *is called* combinational, *if* $flat(H)$ *is a combinational circuit.*

A module has inputs and outputs, assigned with values out of $\mathbf{B} = \{0, 1\}$. The assignment of the outputs is determined by the assignment of the inputs and the *behavior* of the module. The behavior of the the whole circuit depends of the behavior of the basic cells and their connections.

Definition A.14 (semantics of combinational circuits)
Let $\mathbf{B} := \{0, 1\}$ *be the domain of a signal value. The* semantics *of a combinational circuit* $H = (C_1, C_2, \cdots, C_k)$ *is then defined as:*

1. *The behavior of a basic cell is given with the function* $behave : T_1 \rightarrow (\mathbf{B}^* \rightarrow \mathbf{B}^*)$

2. *A* signal assignment *of a circuit* $C = (T, M, S, \tau, \rho^{in}, \rho^{out}, type)$ *is a function* $\beta : S \rightarrow \mathbf{B}$. *A signal assignment* β *is called* stable, *iff*

$$\forall m \in M \setminus \{\tau\} : \beta(\rho^{out}(m)) = behave(type(m))(\beta(\rho^{in}(m)))$$

3. *the behavior of box cell of type* t_i *is defined as an extension of the behave function to all types of the hierarchical circuit* $H = (C_1, C_2, \cdots, C_k)$*:*

$$behave : T_{k+1} \rightarrow (\mathbf{B}^* \rightarrow \mathbf{B}^*)$$

$$behave(t_i) := f \text{ with } f(\omega_{in}) = \omega_{out} :\Longleftrightarrow$$

$$\exists \text{ stable signal assignment } \beta_i \text{ with } \beta_i(\rho_i^{in}(\tau_i)) = \omega_{in}, \beta_i(\rho_i^{out}(\tau_i)) = \omega_{out}$$

The semantics of H *is the behavior of a box of type* t_k.

$$behave(H) := behave(t_k)$$

The semantics is welldefined as there is one unique stable assignment for each input assignment ω_{in} in a combinational circuit. To determine it – that means to determine the assignment of the primary outputs given the assignment of the primary inputs – is the task of *simulation*.

Definition A.15 (simulation of combinational circuits)
A simulator *of a combinational circuit* H *is a program, that determines for a given assignment* ω_{in} *of the primary inputs the corresponding assignment of the primary ouputs* $\omega_{out} = behave(H)(\omega_{in})$.

Example

To finish the Appendix with the formal definitions I want to introduce a full adder as an example for all that stuff.

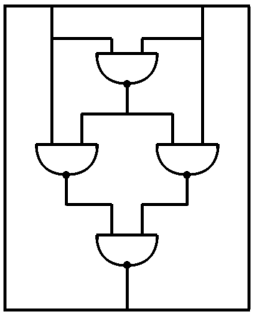

Figure A.1: circuit C_1 (Xor)

Types and behavior

$$T_1 := \{nand, and, or\}$$

$$T_2 := T_1 \cup \{t_1\}$$

$$T_3 := T_2 \cup \{t_2\}$$

$behave(nand) \quad := \quad \begin{pmatrix} 00 \rightarrow 1 \\ 01 \rightarrow 1 \\ 10 \rightarrow 1 \\ 11 \rightarrow 0 \end{pmatrix}$
$behave(and) \quad := \quad \begin{pmatrix} 00 \rightarrow 0 \\ 01 \rightarrow 0 \\ 10 \rightarrow 0 \\ 11 \rightarrow 1 \end{pmatrix}$
$behave(or) \quad := \quad \begin{pmatrix} 00 \rightarrow 0 \\ 01 \rightarrow 1 \\ 10 \rightarrow 1 \\ 11 \rightarrow 1 \end{pmatrix}$

Xor – figure A.1

$$M_1 := \{\tau_1, nand_1^1, nand_1^2, nand_1^3, nand_1^4\}$$

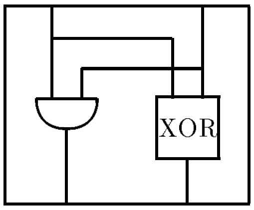

Figure A.2: circuit C_2 (half adder)

$$S_1 := \{s_1^1, s_1^2, \cdots, s_1^6\}$$

$$\rho_1^{in} := \begin{cases} \tau_1 & \rightarrow s_1^1 s_1^2 \\ nand_1^1 & \rightarrow s_1^1 s_1^2 \\ nand_1^2 & \rightarrow s_1^1 s_1^3 \\ nand_1^3 & \rightarrow s_1^3 s_1^2 \\ nand_1^4 & \rightarrow s_1^4 s_1^5 \end{cases} \rho_1^{out} := \begin{cases} \tau_1 & \rightarrow s_1^6 \\ nand_1^1 & \rightarrow s_1^3 \\ nand_1^2 & \rightarrow s_1^4 \\ nand_1^3 & \rightarrow s_1^5 \\ nand_1^4 & \rightarrow s_1^6 \end{cases}$$

$$type_1 := \begin{cases} nand_1^1 & \rightarrow nand \\ nand_1^2 & \rightarrow nand \\ nand_1^3 & \rightarrow nand \\ nand_1^4 & \rightarrow nand \end{cases}$$

$$C_1 := (T_1, M_1, S_1, \tau_1, \rho_1, type_1)$$

The half adder – figure A.2

$$M_2 := \{\tau_2, and_2^1, xor_2^1\}$$

$$S_2 := \{s_2^1, s_2^2, \cdots, s_2^4\}$$

$$\rho_2^{in} := \begin{cases} \tau_2 & \rightarrow s_2^1 s_2^2 \\ and_2^1 & \rightarrow s_2^1 s_2^2 \\ xor_2^1 & \rightarrow s_2^1 s_2^2 \end{cases} \rho_2^{out} := \begin{cases} \tau_2 & \rightarrow s_2^3 s_2^4 \\ and_2^1 & \rightarrow s_2^3 \\ xor_2^1 & \rightarrow s_2^4 \end{cases}$$

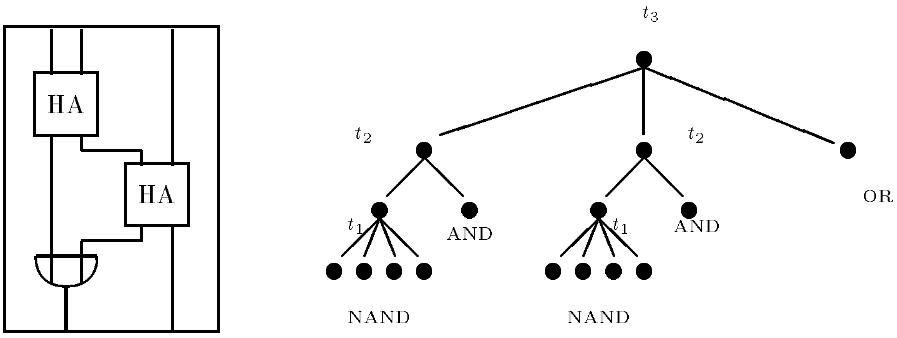

Figure A.3: full adder (C_3) with structure tree $S_{(C_1,C_2,C_3)}$

$$type_2 := \begin{cases} and_2^1 & \rightarrow and \\ xor_2^1 & \rightarrow t_1 \end{cases}$$

$$C_2 := (T_2, M_2, S_2, \tau_2, \rho_2, type_2)$$

The full adder – figure A.3

$$M_3 := \{\tau_3, ha_3^1, ha_3^2, or_3^1\}$$

$$S_3 := \{s_3^1, s_3^2, \cdots, s_3^8\}$$

$$\rho_3^{in} := \begin{cases} \tau_3 & \rightarrow s_3^1 s_3^2 s_3^3 \\ ha_3^1 & \rightarrow s_3^1 s_3^2 \\ ha_3^2 & \rightarrow s_3^5 s_3^3 \\ or_3^1 & \rightarrow s_3^4 s_3^6 \end{cases} \qquad \rho_3^{out} := \begin{cases} \tau_3 & \rightarrow s_3^8 s_3^7 \\ ha_3^1 & \rightarrow s_3^4 s_3^5 \\ ha_3^2 & \rightarrow s_3^6 s_3^7 \\ or_3^1 & \rightarrow s_3^8 \end{cases}$$

$$type_3 := \begin{cases} ha_3^1 & \rightarrow t_2 \\ ha_3^2 & \rightarrow t_2 \\ or_3^1 & \rightarrow or \end{cases}$$

$$C_3 := (T_3, M_3, S_3, \tau_3, \rho_3, type_3)$$

So we got the hierarchical description of the full adder circuit $H := (C_1, C_2, C_3)$.

Bibliography

[App88] A. Appel. Simulating digital circuits with one bit per wire. *IEEE Transactions on Computer-Aided Design*, pages 987–993, 1988.

[AS87] Kurt J. Antreich and Michael H. Schulz. Accelerated fault simulation and fault grading in combinational circuits. *IEEE Transactions on Computer-Aided Design*, CAD-6(5):704–712, September 1987.

[Bee92] John F. Beetem. Hierarchical topological sorting of apparent loops via partioning. *IEEE Transactions on Computers*, pages 607–619, 1992.

[BHK+87] B. Becker, G. Hotz, R. Kolla, P. Molitor, and H.G. Osthof. Hierarchical design based on a calculus of nets. In *Proceedings of the 24th ACM/IEEE Design Automation Conference (DAC87)*, pages 649–653, June 1987.

[BHKS91] B. Becker, R. Hahn, R. Krieger, and U. Sparmann. Structure based methods for parallel pattern fault simulation in combinational circuits. In *Proceedings of the 2nd European Design Automation Conference (EDAC91)*, 1991.

[HSS92] J. Hartmann, B. Schieffer, and U. Sparmann. Cell oriented fault simulation. In *Proceedings of the European Simulation Multiconference 92*, pages 424–429, 1992.

[KS91] R. Kolla and B. Serf. The virtual feedback problem in hierarchical representations of combinational circuits. *Acta Informatica*, 28:463–476, 1991.

[Mau91] P. Maurer. Scheduling blocks for hierarchical compiled simulation of combinational circuits. *IEEE Transactions on Computer-Aided Design*, pages 184–192, 1991.

[Meh84] K. Mehlhorn. *Data Structures and Algorithms 2: Graph Algorithms and NP-Completeness.* EATCS Monographs on Theoretical Computer Science. Springer-Verlag, Berlin Heidelberg New York Tokio, 1984.

[Sch91] B. Schieffer. Hierarchische Fehlersimulation mit Nicht-Standard-Fehlermodellen. Master's thesis, Fachbereich Informatik, Universität des Saarlandes, 1991.

[WM89] Zhicheng Wang and Peter M. Maurer. Scheduling high-level blocks for functional simulation. In *Proceedings of the 26st ACM/IEEE Design Automation Conference*, pages 87–90, 1989.