A GaAs electrical rule checker

P. Remy*, P. Royannez**, A. Amara***

* Laboratoire MASI/CAO-VLSI
Université Pierre et Marie Curie
4, place Jussieu, 75252 Paris Cedex 05, France.
Email: pascal.remy@masi.ibp.fr

** Siemens AG HL MI E EM
St. Martin Str. 76, 81 541 Munich, Germany.
Email: royannez@hl.siemens.de

*** MASI laboratory member
Laboratoire de Microélectronique
Institut Supérieur d'Electronique de Paris
21, Rue d'Assas, 75270 Paris cedex 06, France.
Email: amara.amara@isep.fr

N.B. this work was done at Laboratoire MASI/CAO-VLSI.

Abstract

This paper presents electrical verification in GaAs DCFL VLSI circuits. A set of about thirty rules that deal with both static and dynamic electrical aspects has been especially defined. Moreover, because of the complexity of GaAs circuits, a software tool has been developed to perform this verification automatically. VITESSE HGaAsII process has been targeted but the software which uses a parameter file, is process independent.

1 Introduction

The design of high speed GaAs digital circuits requires a very well-defined methodology because of the noise and power limitations of such technologies. More precisely, beyond the fact that bad transistor sizing leads to circuits with under-optimal delay performance, it sometimes leads to circuits which don't work at all. This is a very important difference between DCFL and CMOS. It follows that in addition to careful design, an automatic electrical rule checking step must be included in the validation procedure.

The first part of the paper presents and explicits the eight most significant rules that need to be verified in order to guarantee static functionnality. The most important electrical limitations are highlighted and then translated

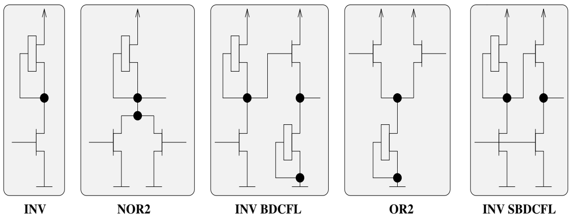

Figure 1: DCFL and buffer gates

into rules that can be verified afterwards. Most of these rules deal with noise immunity, the others are defined for dynamic and timing considerations. The second part of the paper describes the software aspects and their implementation. The automated electrical verification takes a transistor/capacitance netlist as input and returns a list of warnings and the corresponding locations for each violated rule. The last part of the paper is dedicated to benchmarks and CPU cost considerations.

2 Set of rules

VITESSE GaAs E/D MESFET process provides several logical families of gates [1]. To get the best compromise between speed, area, and power consumption, DCFL logic has been chosen. Figure 1 shows gates used in this approach. Apart from DCFL gates we also use BDCFL (Buffered DCFL), and SBDCFL (Super BDCFL) gates. For each of them, we successively describe the main electrical verifications which have to be done.

DCFL

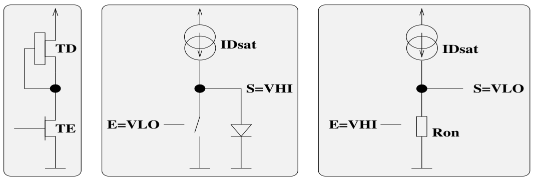

Figure 2: Logical levels

As shown on figure 2, DCFL logical levels V_{HI} and V_{LO} are respectively determined by the forward biased gate-source diode at the input of the next stage and the resistive drop R_{on} in the E-MESFET. Consequently, those levels strongly depend on transistor sizing, whereas CMOS logical levels mostly depend on power supply voltages. The low logical level V_{LO} must be lower than V_{TE}, the threshold voltage of the E-MESFET if we want the latter to be turned off. To provide $V_{LO} \leq 100mV$ we use the Statz et al [2] model in which the D-MESFET is in saturation mode and the E-MESFET switch is in linear mode. We obtain the following expression [3]:

$$\boxed{\underline{\text{Rule1}}:\ \beta = \frac{W_E/L_E}{W_D/L_D} \geq \beta_0}$$

Where W_E, L_E, W_D and L_D are respectively the length and the width of the E and D-MESFET. β_0 depends on the process used and can be verified with a spice simulation [4]. In what follows, we consider that L_D and L_E are constant values.

Concerning V_{HI}, a MES junction is forward biased around 0.7V. Let us consider that V_{HI} is correct only if it is close to this value, if it is not the case then two problems may occur. The first one occurs if V_{HI} is too low (under-drive), in which case the noise margins of the current gate are reduced. The second one occurs if V_{HI} is too high (over-drive), in which case the noise margins of the next gate are reduced. Indeed, the static transfer curve of a DCFL inverter shows that an overdriven input gives an output voltage that is too high (see figure 3). Let W_{Eload} be the length of the driven E-MESFET. The Schottky model gives $I_{GS} = \alpha W_{Eload} e^{\frac{q(V_{GS}-V_{bi})}{nkT}}$ for the gate-source current. So, if $V_{GS} = V_{HImax}$ or if $V_{GS} = V_{HImin}$, we have respectively $I_{GS} = \alpha_{max} W_{Eload}$ and $I_{GS} = \alpha_{min} W_{Eload}$. V_{HImax} and V_{HImin} are defined by the designer according to the noise immunity he wants. We could choose for instance the values 640mV and 730mV.

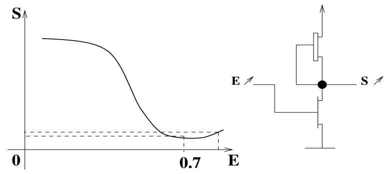

Figure 3: DCFL inverter transfert curve

Furthermore, the current which flows into the gate-source junction comes from the D-MESFET. The gate-source voltage is constant and equal to zero. The drain-voltage is also constant and equal to $V_{TT} - V_{HI}$. We easily verify that this transistor is always in linear mode and that it provides the following drain current $I_D = \alpha_d W_D$ regardless of the logical level. Finally, to avoid both, over and under-drive, we have to forward bias the driven gate-source junction properly. This is verified if:

$$\boxed{\underline{\text{Rule2}}:\ \alpha_{min} W_{Eload} \leq \alpha_d W_D \leq \alpha_{max} W_{Eload}}$$

Noise immunity plays an important part in GaAs digital design [5]. According to [6], static noise margins can be defined by drawing the largest possible square between an inverter's characteristic curve and its mirroring image or by the maximal distance between those two characteristics (see figure 4).

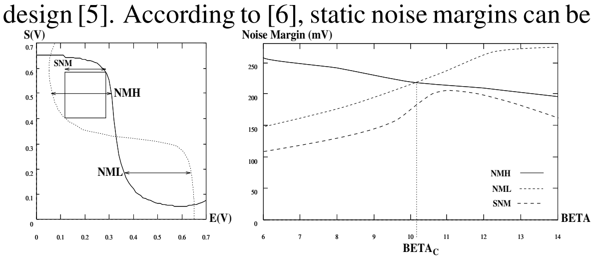

Figure 4: definition and analysis of noise margins

Following the methodology described in [7], we have developed a custom software to extract those noise margins from DC Spice simulations and report them on figure **??**. It appears that the NMH (noise margin high) and the NML (noise margin low) are well-balanced for an optimal β_c ratio value. Consequently, the next rule is:

<u>Rule3</u>: $\beta_c - tol \leq \beta \leq \beta_c + tol$

Where **tol** is a user defined tolerance factor.

NOR gates are intensively used in DCFL logic. Like NMOS, delay performance is not affected when Fan-In increases. However, if Fan-In is greater than four, the noise margin high is greatly reduced. Figure 5 shows a static transfer curve for an N-input NOR gate. Furthermore, the importance of leakage current grows as the number of switch down E-MESFET is increased. This gives us the rules:

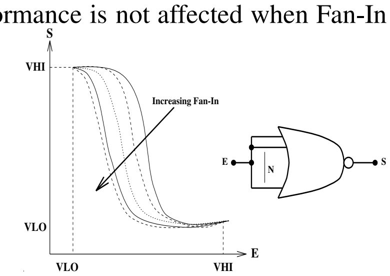

Figure 5: DCFL NOR transfer curve

<u>Rule4</u>: Never use NOR gates with a Fan-In greater than five

<u>Rule5</u>: Four and five input NOR gates must be driven by buffered gates (BDCFL or SBDCFL) to provide a better low logical level

BDCFL

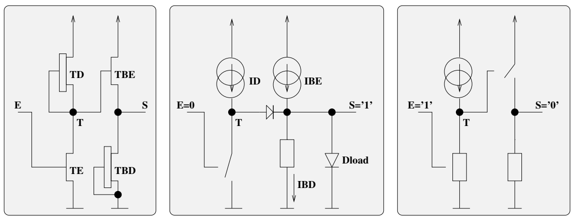

Figure 6: BDCFL inverter gate and its static configurations

Figure 6 sumerizes the behaviour of a BDCFL inverter. Unlike DCFL, value of the low logical level V_{LO} is always around 30mV that is an excellent value to properly turn off the switch down E-MESFET of the next gate. V_{HI} is now determined by the current given by **TBE**, the switch up E-MESFET of the buffer stage and **TD**, the active load of the inverter stage. If we assume that $S = V_{HI}$ and $T = 2V_{HI}$ (see figure 6). Spice simulations and curve fitting give $I_D = \alpha_D W_D$ and $I_{BE} = \alpha_{BE} W_{BE}$. The resulting current $I_D + I_{BE}$ flows into **Dload** but also into **WDB** (see figure 6). The current into the latter is approximately $I_{DB} = \alpha_{DB} W_{DB}$. So, to avoid both under and over-drive we use:

Rule6: $\alpha_{min} W_{Eload} \leq I_D + I_{BE} - I_{DB} I_D + I_{BE} - I_{DB} \leq \alpha_{max} W_{Eload}$

Since the buffer stage is the dual configuration of the inverter stage, we can easily design OR gates with a set of parallel E-MESFET switch-ups (see figure 1). But, once more this opportunity is limited due to electrical reasons. Indeed, a worst case configuration consists in forcing every input with a high level. Each active Pull-up provides additional currents and leads to over-drive. To avoid this situation:

Rule7: Or gates have no more than two inputs

SBDCFL

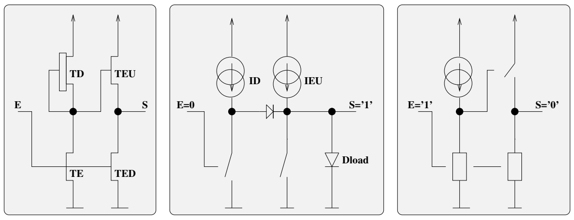

Figure 7: SBDCFL inverter gate and its static configurations

The following approach is very close to the previous one. As indicated on figure 7 no rule is necessary to provide a good V_{LO}. Good V_{HI} can be achieved with the following rule:

Rule8: $\alpha_{min} W_{Eload} \leq I_D + I_{EU} \leq \alpha_{max} W_{Eload}$

Where $I_D = \alpha_D W_D$ and $I_{EU} = \alpha_{EU} W_{EU}$ come from Spice simulations.

3 Software implementation

Thus we have seen so far which electrical rules need to be checked and what input parameters are at our disposal for that task. What follows presents the software aspects and implementation of the automated electrical rules check. We will start by describing how the extracted electrical parameters are formatted and then move on to the workings of the verification software itself.

For obvious flexibility reasons, it was decided to have the verification software parse the set of extracted electrical parameters which hence had to be gathered in a file. This file was chosen to be an ASCII file so that it may easily be read, maintained and modified. It is to be noted that the contents of this file can be incorporated into any ASCII technological file a CAD system might posses thus forming a section of that file. The format of this file or section is now detailed. This file or section begins with the starting identifier ERC_RULES. Then, for each parameter a pertinent keyword is given. An equal sign separates that keyword from it's numerical value so that each parameter may be represented in the following format: *keyword* = *value*. That format represents a given parameter. Each parameter representation as well as the starting identifier may be separated with a space, tabulation or line feed or any combination of these. Finally, the ending identifier END_ERC_RULES closes the list of parameter representations and is separated from the last parameter representation by the same possible separator(s) as above. A simplified example of the described section follows:

```
/****************************************
 * properties for vitesse 0.8            *
 * gallium arsenide technology: 04/04/95  *
 ****************************************/
ERC_RULES
/* rule 1 : good low level (dcfl)         */
WEonWD_MIN = x.x /* minimum WE/WD allowed */
/* rule 2 : good high level (dcfl) */
ALPHA_d = xx.x /* alpha d value    */
...
...
RULE22_MAX = xxx.x /* maximum ratio for
                  dcfl driving a follower */
END_ERC_RULES
```

The automated electrical rules check is but one of the various verification steps in the design of a circuit and in our context one of the later ones. Indeed, the electrical verification is done at gate level, once the input transistor/capacitance netlist has been "dissassembled" into gates and that shorts, unconnected elements, conflicts and high impedance points have been identified and solved. The verification mainframe in which the electrical rules check is done is called VEGA (Verification Environment for GAtes). Running VEGA with the -erc option enables the electrical rules check module called EVM which stands for Electrical Verification Module to process the input circuit and report any electrical violations. Before going through EVM in more detail, it is necessary to give a quick description of the internal gate structure which is that of VEGA. VEGA's gate structure is called a cone. A cone has incones and outcones which are it's input and output cones. It also has lists of branches made out of lists of links [8]. A branch is nothing more than an electrical path towards a connector or a power supply. A cone will have as many lists of branches as it has different types of branch termination. The different types of possible branch endings are an external input connector, or one of the possible identified power supply connector types. A link may be a transistor or a connector with a node capacitance. A detailed description of the transistor structure used is given in [9]. That structure carries all the information that is needed for the electrical verifications. At this point, we then have a structurally correct cone netlist defining our input circuit and the electrical verification process may start. Its second task after the parsing of the electrical parameters is to prepare each cone for the coming rules verifications. The justification for this preparation step will now be made clear.

If we take a close look at the electrical rules defined in the previous parts of this paper and if we anticipate eventual future rules, we may notice two things; one, that most rules are equations or inequations and two, and most important of all, that most rules deal with transistor sizes, sums of transistor widths, minimum and/or maximum transistor width values. So it was decided that these values would be calculated once and for all for each cone and that they would be stored in each one of them in such a way that they may be easily available. But since all these values are not systematically needed depending on the type of each cone (dcfl, bdcfl or sbdcfl), only the necessary ones are stored. Furthermore, since it might also be convenient to be able to access quickly a specific input or output cone, these are also stored in some cones. Therefore and considering the two possible branch lists per cone which are branches leading to the high level power supply that will be referred to as "up" and branches leading to the lower level power supply that will be referred to as "down", let us now look at the values stored in each cone depending on it's type.

For DCFL cones the values that are needed are:

- The sum of the widths of the pull up transistors (there usually is only one pull up).
- The minimum switch down width amongst all the transistors leading down
- The sum of the widths of the driven transistors contained in branches leading up.
- The sum of the widths of the driven transistors contained in branches leading down.

For BDCFL cones the values that are needed are:

- The sum of the widths of the pull down transistors (there usually is only one pull down).

- The sum of the widths of the switch up transistors (there should only be one switch up).
- The minimum switch up width (there should only be one switch up).
- The driving cone.

For SBDCFL cones the values that are needed are:

- The sum of the widths of the switch up transistors (there should only be one switch up).
- The minimum switch down width amongst all the transistors leading down.
- The sum of the widths of the driven transistors contained in branches leading up.
- the sum of the widths of the driven transistors contained in branches leading down.

These calculations being quite straight forward, there is no need to go deeper into the matter.

Now that the cones have all the required information the actual rules checking may take place. The strategy adopted here is the same as before i.e. depending on each cone's type, a separate function is called to check for any rule violation. These functions are dcflerc(), bdcflerc() and sbdcflerc(). The three working alike, let's chose the first one to describe the method for testing a dcfl gate. Lets then suppose that the cone being tested is a dcfl gate. The first thing dcfl() does is to retrieve the information stored in the cone and then it checks each and every rule that are sequentially listed. Checking a rule basically means fitting the extracted parameters and the stored cone information discussed previously into an equation or an inequation. To illustrate this, let us see a typical example. The simplified software implementation for testing the left hand side of dcfl rule number 2 follows:

```
 if (VEGA_ALPHA_MIN*WEL_SUM=<VEGA_ALPHA_d*WD_SUM_VDD)
warn(2);
```

where VEGA_ALPHA_MIN and VEGA_ALPHA_d are respectively the internal representations of the parsed parameters ALPHA_MIN and ALPHA_d. Where WEL_SUM and WD_SUM_VDD are respectively the calculated, stored then retrieved cone values of WEload and WD. And where finally warn(2) calls a warning function that writes the warning message in the report file and that eventually prints that message on the screen. If the test is negative meaning that the given rule has been violated, the corresponding warning is generated and the next rule is checked. The warning information given by the program in the report file enables the designer to quickly locate the elements at fault so as to take the appropriate measures.

If the automated electrical rules checking presented here is in its first version and that improvements such as the ones stated here after will enhance future versions, this program is nevertheless quite efficient and has certainly proven itself to be a useful tool for designers working at our laboratory. The considered useful future improvements are those of including the complete set of rules (in terms of the equations and inequations) into the parsed file or section described earlier so as to make the program even more flexible and of tackling the more subtle dynamic electrical rules. But lets not forget that one of the most interesting features offered here is that the electrical verification is process independent through the use of a parametrized electrical rules file. The next and final part of this paper presents significant results obtained with our approach.

4 Results

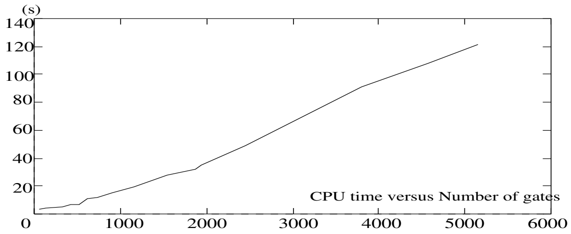

Figure 8: CPU time for several circuits with a increasing number of gates (On SPARC IPC)

The bench illustrated above has been done on various circuits, such as adders, shifter and multipliers. The number of gates contained in these ranged approximately from 10 to 5000 gates: These results show that the algorithm employed is roughly linear and that it gives reasonable CPU times. This tool has proven itself to be very useful to the designers working at MASI as they have extensively used the program for their designs with a significant time gain. VEGA is now part of the Alliance CAD chain [10].

Acknowledgements

Thanks to the whole CAO-VLSI team at MASI laboratory, and especially to A. Hajjard.

References

[1] S. I. Long, S. E. Butner, *Gallium Arsenide Digital Integrated Circuit Design,* McGraw-Hill, 1990.

[2] H. Statz, P. Newman, I. Smith, R. Pucel, H. Haus, “GaAs FET Device and Circuit Simulation in SPICE”, *IEEE Transactions on Electron Devices, Ed. 34, pp. 160-169, 1987.*

[3] S. J. Harrold, *GaAs IC Design,* Prentice Hall, 1993.

[4] *Foundry design manual-Version 6.0,* Vitesse Semiconductor Corporation.

[5] S. I. Long, M. Sundaram, “Noise-Margin limitation on Gallium-Arsenide VLSI”, *IEEE Journal of solid-state circuits, Vol. 23, No 4, August 1988.*

[6] C. F. Hill, “Noise margin and noise immunity in logic circuits”, *Microelectron, vol. 1, pp. 16-21, Apr. 1968.*

[7] E. Seevinck, F. J. List and J. Lohstroh, “Static-Noise Margin Analysis of MOS SRAM cells”, *IEEE Journal of solid-state circuits, Vol. SC-22, No 5, October 1987.*

[8] M. Laurentin, “Abstraction fonctionnelle de circuits intégrés CMOS”, *PhD thesis, Université Pierre et Marie Curie, Laboratoire MASI, December 1994.*

[9] F. Pétrot, “Outils d’aide au développement de bibliothèques VLSI portables”, *PhD thesis, Université Pierre et Marie Curie, Laboratoire MASI, October 1991.*

[10] A. Greiner, F. Pêcheux, “ALLIANCE: A Complete Set of CAD Tools for Teaching VLSI Design”, *3rd Eurochip Workshop on VLSI Design Training, pp. 230-237, Grenoble, Sept. 1992, ftp.ibp.fr available in /ibp/softs/masi/alliance, 1993.*