Designing Asynchronous Circuits in Gallium Arsenide

José A. Tierno
Department of Computer Science
California Institute of Technology
Pasadena, CA 91125

March 24, 1993

Contents

Chapter 1

Introduction

1.1 Asynchronous circuits

Most of the digital design done today is "synchronous", that is, a global synchronization signal is used to get the different parts to work in lockstep. It is simple and elegant, and requires little circuit overhead.

Yet, as VLSI circuits increase in size and complexity, distributing global signals becomes more delicate, timing assumptions are harder to guarantee; at the system level, the global clock has to be slowed down to accommodate the slowest parts. It is interesting, at this point, to consider asynchronous circuits, where the time can be eliminated from the specification.

A circuit is speed-independent when its correct operation is independent of delays in operators. A circuit is delay-insensitive when its correct operation is independent of the delays in operators and wires, except that the delays be finite [Sei80] [vdS85]. No global synchronization signal, or knowledge about delays is used. As a subclass of asynchronous circuits, delay-insensitive circuits are very interesting for formal design methods; we can reason about the correctness of such circuits independently of timing.

Our research group has developed a synthesis method to compile a high-level description of a circuit down to a gate level description [Mar86] [Mar90]. This compilation is largely technology-independent; only at the stage of sizing transistors for better performance do we have to look at the actual transistor network.

1.2 Gallium Arsenide

Gallium Arsenide, or GaAs, has traditionally been used for microwave circuits, where it far outperforms silicon devices. However, its use for digital circuits has always been marginal or very specialized because of high costs, low yields, and limited integration compared to CMOS.

GaAs fabrication technology has evolved and matured. Today, yields are up, and density is highly improved: 100,000 transistor chips are commercially available, and 1,000,000 transistor chips are feasible.

The MESFET, or metal-semiconductor field effect transistor, is the natural GaAs transistor. It is a junction FET, where the gate connects to the channel through a Schottky junction. Its main advantage is easy to manufacturing, with high density and good yield. An isolated gate FET would be highly desirable, but no stable GaAs oxides have been discovered so far. Complementary logic is also un-practical, since p-type transistors are ten times slower than the n-type.

MESFET circuits have the simplicity of NMOS, the speed of ECL, and VLSI integration. It is still far from being a competitor for CMOS, low RAM density being one of the most notorious shortcomings of GaAs.

1.3 Asynchronous GaAs

At the higher speed achieved by GaAs circuits, problems like clock skew, transmission delays, and the increasing penalty of off-chip communication become critical. This makes the technology a good target for asynchronous design; it can also let us discover new problems in asynchronous circuits at increased speeds.

The purpose of this thesis is to propose a series of techniques to translate the basic building blocks ("production rules") of the synthesis method into GaAs circuits.

Traditional GaAs MESFET logic families are not adequate for this purpose. Optimized for synchronous circuits, they provide us with nor gates, latches, and a few other specialized gates.

We first look for a straightforward implementation of *production rules.* Production rules are programs of the type $G_i \rightarrow A_i$ where G_i is a boolean expression and A_i sets or resets a given variable.

We also derive a timing model to predict performance, a fan-in and fan-out model to predict electrical compatibility between the different gates, and a power consumption model. As much as possible, we want to be able to predict the correct functioning of the circuit without going to detailed simulations.

1.4 Results

All of the techniques presented in this work have been tried on actual circuits. The circuits tested have provided a lot of data on the different models, as well as indications on which type of circuits to use where, better sizing techniques, what to expect in terms of power consumption and yield, etc.

Although any circuit can be implemented directly using these techniques, some gates are so much faster than others that there is a lot to be gained by using this fact early on in the design. In this sense, the architecture of the circuit is highly influenced by the underlying technology.

The experiments done so far have given us a basic understanding of how digital GaAs circuits work. Many problems are still to be solved in the area of power consumption, speed, and reliability. The data we have accumulated so far gives a good indication on where improvements are possible.

1.5 Outline of this Thesis

Chapter 2 presents a brief introduction to Gallium Arsenide technology. A model is given for the MESFET, some of the second order effects are discussed, as well as a few circuits.

Chapter 3 presents several GaAs logic families used today. A brief description of advantages and disadvantages is given for each of them.

Chapter 4 introduces a new family of logic gates that will be used to implement asynchronous circuits, discusses their properties, and shows how to use them.

Chapter 5 presents a linear timing model for some of the gates that were introduced in the previous chapter, and shows how to use it to optimize performance.

Chapter 6 presents an example of asynchronous design in GaAs, a register array. The circuit was designed, fabricated and tested, and the results are included.

Chapter 2

Gallium Arsenide Basics

Introduction

In this chapter we give an introduction to MESFET based GaAs technology. The MESFET is, at this moment, the transistor of choice for GaAs digital circuits. It is simple to manufacture in small geometries, it is reasonably fast, and can be integrated to LSI and VLSI levels at a price that competes very well with silicon.

MESFET, which stands for MEtal Semiconductor Field Effect Transistor, is a junction FET (*JFET*), where the *p-n* junction between the gate and the channel has been replaced by a Schottky junction (see Figure 2.1).

To predict correctly the behavior of MESFET circuits it is essential to have an accurate DC model. The models and equations used for JFET's can be used for MESFET's; however, second order effects have to be taken into consideration, which make real devices behave differently from the first order approximation. Back-gating, subthreshold currents, and drain current lag affect the predicted operation of circuits, and have to be modeled. Also, GaAs MESFET's operate *velocity saturated* at much lower electric fields than silicon transistors. Throughout this work we will use the *Raytheon* model [SNS+87]. All simulations were done with *HSPICE*, which uses an in-house modification of the Raytheon model, specially tuned for the *Vitesse* fabrication process.

Because of their complexity, full models, however important to verify designs, give us little insight on the general properties of the circuits we are using. Therefore we use greatly simplified models, like the small signal

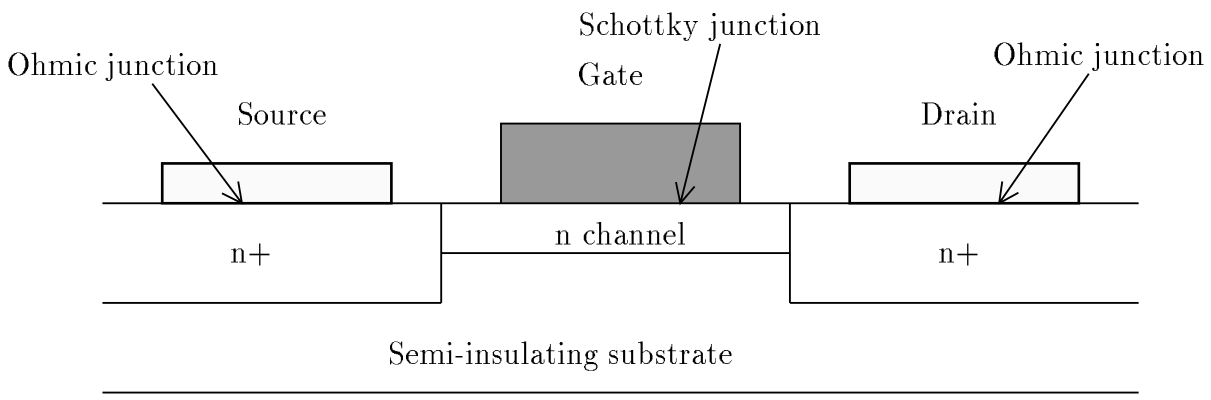


Figure 2.1: Self aligned MESFET.

model, that give us a qualitative idea of the characteristics of the circuit.

2.1 Equivalent circuit for the MESFET

Figure 2.2 shows an equivalent circuit for the MESFET that will be the basis for the equations presented later on. The diodes simulate the effect of the junction; C_{gs} and C_{gd} represent the gate-to-source-and gate-to-drain capacitance and charge stored in the channel. The gate and drain resistances take into account the diffusion resistance. Ids is a current source controlled by the gate to drain voltage. Finally, D and S are the internal drain and source respectively.

2.1.1 Equations

These equations are based on the Raytheon model [SNS$^+$87], developed by Satz et al.

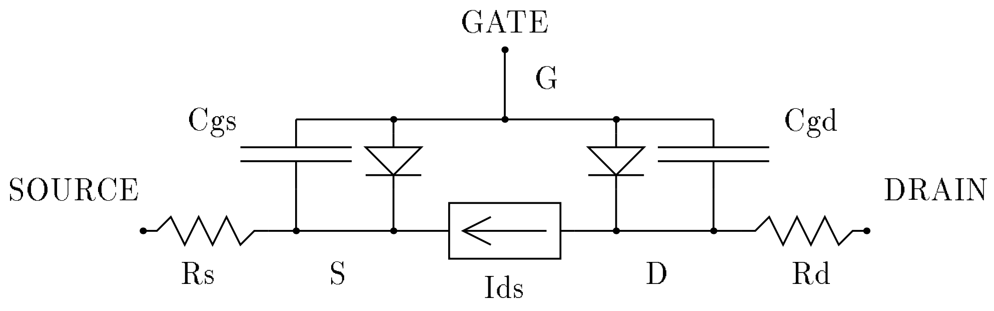


Figure 2.2: Equivalent circuit for the MESFET.

DC model

For the DC model, we have:

$$I_{ds} = \frac{\beta(V_{gs} - V_T)^2}{1 + b(V_{gs} - V_T)} \left\{ 1 - \left(1 - \frac{\alpha V_{ds}}{3}\right)^3 \right\} (1 + \lambda V_{ds}),$$

for $0 < V_{ds} < \frac{3}{\alpha}$

$$I_{ds} = \frac{\beta(V_{gs} - V_T)^2}{1 + b(V_{gs} - V_T)} (1 + \lambda V_{ds}),$$

for $V_{ds} \geq \frac{3}{\alpha}$, where α, β, b, and λ are model parameters.

Subthreshold model

In the subthreshold region, there will still be some current flowing through the channel [CVFC87]. If V_{ds} is low enough, this will be mostly a diffusion current, and will have an exponential dependency on V_{ds} and V_{gs}:

$$I_{ds} = I_0 \left\{ 1 - \exp\left(-\frac{cqV_{ds}}{kT}\right) \right\} \exp\left(\frac{bqV_{ds}}{kT}\right) \exp\left(\frac{aqV_{gs}}{kT}\right)$$

where I_0, a, b, and c are model parameters, k is Boltzman's constant, and T is the absolute temperature.

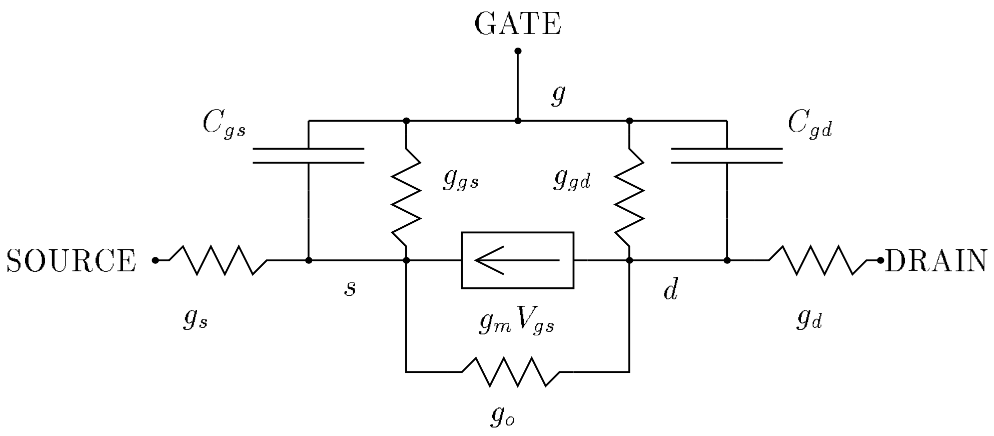


Figure 2.3: Small signal model for the MESFET.

Backgating

This is similar to the body effect in *MOSFET*'s. Regions close to a transistor that are more negative than its source will cause a reduction in the drain-source current. This can be modeled as a shift in the threshold voltage of the transistor:

$$V_T = V_{TO} + K_{bg}(V_S - V_{SS})$$

Where V_T is the new threshold voltage, V_{T0} is the threshold voltage with no backgating, K_{bg} is a constant, V_S is the source voltage, and V_{SS} is the most negative voltage in the neighborhood of the transistor.

2.1.2 Small signal model

To get a small signal model for a transistor circuit, we linearize the circuit equations around the DC quiescent point. If input signals are small enough, this new circuit will be a good approximation of the behavior of the more complex model. Figure 2.3 shows the linear circuit that replaces each transistor. Parameters g_m, g_d, g_s, g_o, g_{gs}, g_{gd}, C_{gs}, C_{gd}, are obtained by differentiation of the transistor equations. In particular, conductances come from the DC model, and capacitors come from the *charge storage model.*

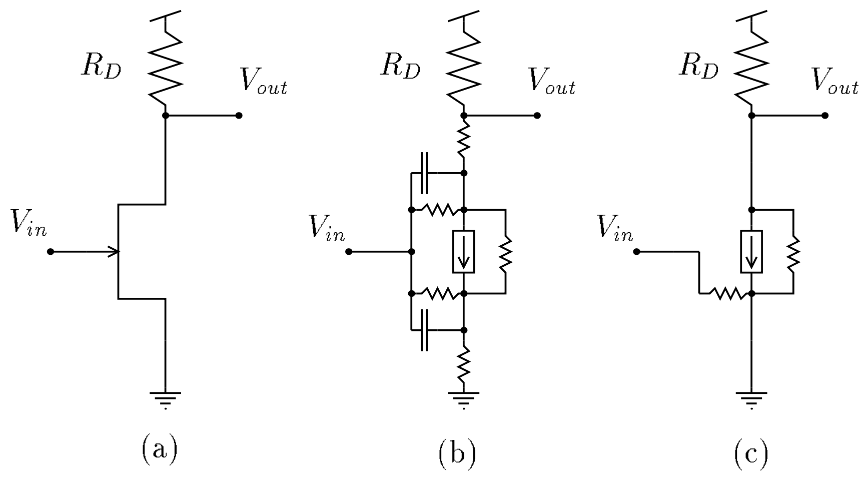


Figure 2.4: Common source amplifier, transistor schematic (a), small signal model (b), simplified small signal model (c).

2.2 Some MESFET circuits

We can arrange transistors in a number of ways to create logic gates. A simple study of the small-signal behavior of these circuits will help us to later justify some of those arrangements. Specifically, we will discuss the *common-source* and *source-follower* configurations [MH72]. Because of the actual values of the parameters, it is often possible to simplify the small signal model to get simpler expressions for the circuit gain, input impedance, etc. This model will be used in the following circuits

2.2.1 Common source amplifier

Figure 2.4 shows a schematic of the common source amplifier. We derive the small signal gain from the current equation for the node V_{out}:

$$g_m V_{in} = -(g_o + \frac{1}{R_D})V_{out}$$

Therefore,

$$A_v = -\frac{g_m R_D}{1 + g_o R_D} \approx -g_m R_D$$

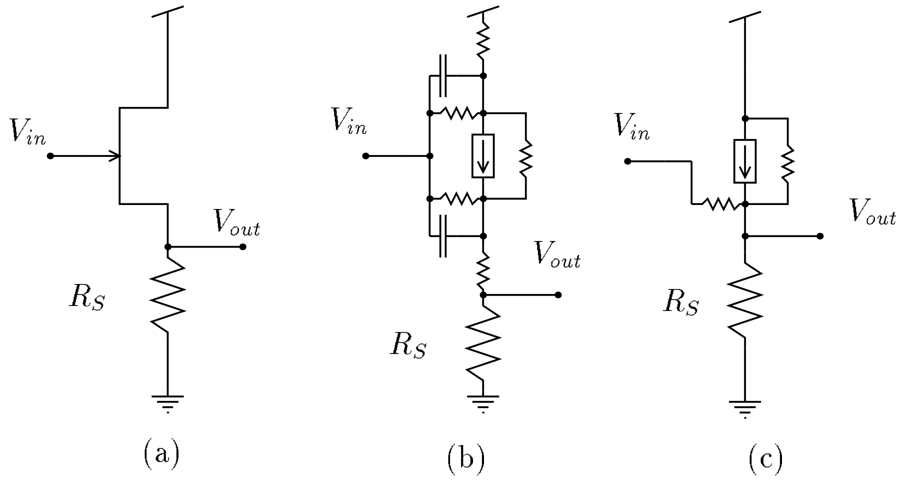


Figure 2.5: Source follower amplifier, transistor schematic (a), small signal model (b), simplified small signal model (c).

And the input impedance:

$$R_{in} = \frac{1}{g_{gs}}$$

The main characteristics of this amplifier are large and negative voltage gain (between -10 and -50), and poor input impedance. It can be used for restoring signals.

2.2.2 Source follower amplifier

Figure 2.5 shows a schematic of the source follower amplifier. As in the previous case, we derive gain and input impedance:

$$(V_{in} - V_{out})g_m = (V_{out} - V_{in})g_{gs} + V_{out}(g_o + \frac{1}{R_S})$$

Therefore,

$$A_v = \frac{g_m + g_{gs}}{g_m + g_{gs} + g_o + \frac{1}{R_S}}$$

$$R_i = \frac{1}{1 - A_v}\frac{1}{g_{gs}}$$

For practical values of the parameters, $g_{gs} \gg g_o$. We choose $R_S \gg 1/g_m$, and we get, $A_v \approx 1$ and the input impedance becomes very large. This stage can be used to drive big loads.

Chapter 3

Logic Families in GaAs

Introduction

This chapter introduces several design styles currently in use for gallium arsenide logic circuits. Some try to optimize delays, or power consumption, or noise margins, but all, in some way or another, try to make up for the fact that the MESFET is far from being an ideal switch.

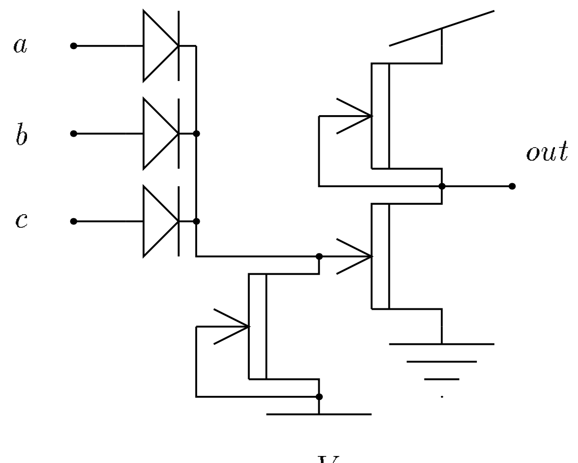
The gate is connected to the channel through a Schottky junction; in most configurations this causes an interaction between the input and the output, or limits the excursion of the input signal, or adds current paths that have to be considered carefully to make sure they are not the cause of any problem.

GaAs technology is inherited from the microwave circuit domain, where lower densities are sufficient. To get LSI or even VLSI integration levels, the depletion-mode-only process has to be improved to include enhancement mode transistors.

3.1 Depletion Mode Only

Early GaAs processes had only depletion mode MESFET's; this saves one mask and one step in fabrication, increases yield, and lowers costs.

Since depletion mode transistors are normally on, we need a negative power supply to turn off gates, not unlike depletion mode only NMOS circuits.



$--V_{ss}$
Figure 3.1: SDFL nor gate.

3.1.1 Schottky Diode Fet Logic

Using diode circuits plus a voltage restoring gate we can create complex logic functions with little circuit overhead.

Figure 3.1 shows a three-input nor-gate of this type. Several other diode configurations can be used.

Using diodes for the logic function offers significant gains in speed, area and power over FET only logic, and admits a large fanout. However, to prevent the noise margins and switching speed from deteriorating, the gate has to be "buffered"; that is the function of the two extra transistors in the output stage.

3.1.2 Fet Logic

Figure 3.2 shows a fet-logic nor gate. Here the logic function is performed by transistors; the output is fully restored.

The output stage can produce a limited amount of current and frequently needs to be buffered. Since the input transistors are always on, they dissipate a significant amount of static power.

Nor gates are the primitive building blocks for this family. Two input nand gates can be implemented, but they are problematic and have very small noise margins.

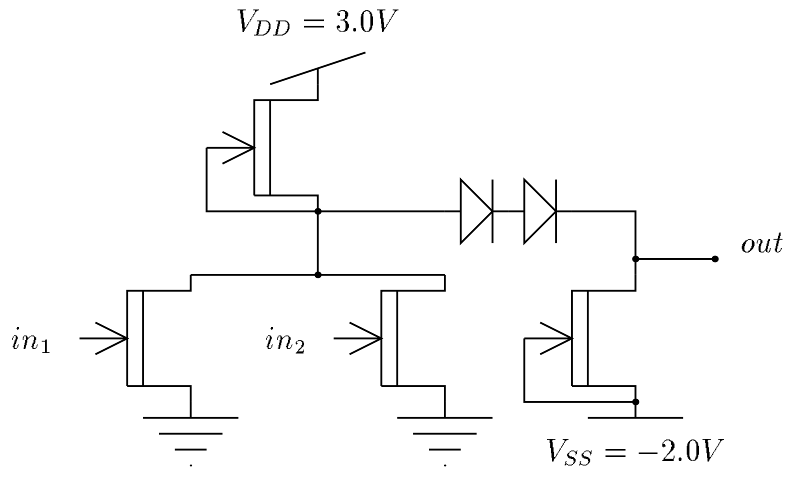


Figure 3.2: Fet logic nor-gate.

3.1.3 Current mode logic

This is a differential amplifier based logic. All gates generate differential signals, like ECL. This is the fastest gate style, and uses only one power supply. On the down side, it requires a lot of area and current, and a higher power supply. Also, several reference voltages have to be generated.

Because all signals are dual ended, circuits like a D-latch become very simple (see Figure 3.3).

3.2 Enhancement-Depletion Mode Logic

The second power supply can be eliminated by adding transistors with a positive threshold voltage. Since these transistors can be totally cut, the static power dissipation also goes down.

In particular, large loads can be driven with high efficiency, using a *push-pull* super buffer (see Figure 3.6).

3.2.1 Direct Coupled Fet Logic

Direct Coupled Fet Logic, or DCFL, is a direct equivalent of DCFL in NMOS. We have an example of an inverter and a nor gate in Figure 3.4.

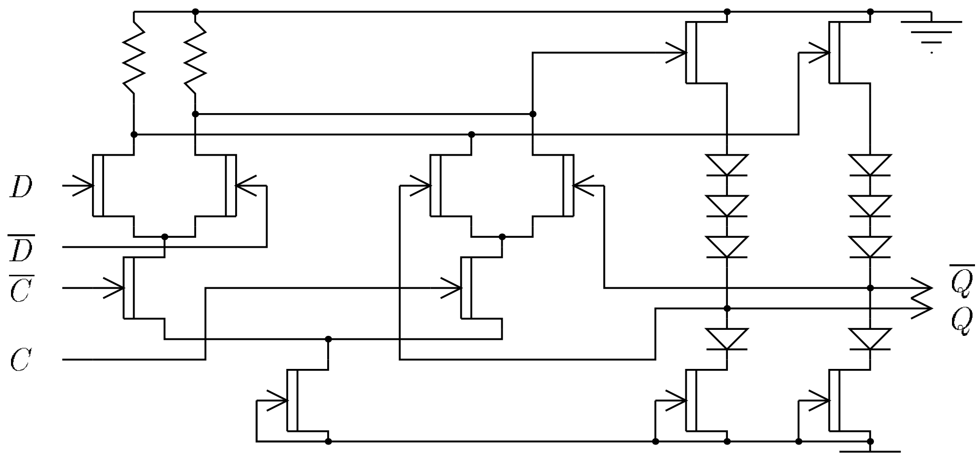


Figure 3.3: D-type latch in Current Mode Logic.

This is the most compact design style, offering a good tradeoff between power and speed. However, noise margins are small, and extra circuitry is required to drive loads bigger than 3 or 4 gates. Also, nand-gates have too many problems to be useful.

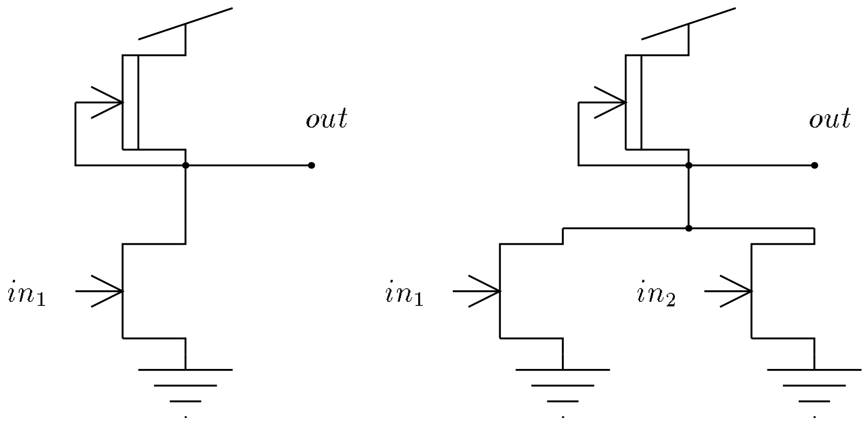
For VLSI applications, DCFL is certainly the gate of choice, because of density and power consumption. Noise margins can be enhanced with a number of techniques, and some circuit overhead. It is the most widely used design style, and probably the one that will allow us to design 1,000,000 transistor chips.

However, DCFL is not always adequate for self-timed circuits. Direct implementations of C-elements with nor gates have hazards; we either have to carefully design around them, make timing assumptions, or add circuitry that will eliminate them. Figure 3.5 shows a c-element implemented with nor-gates. A glitch can occur at the output for the sequence $a\uparrow; a\downarrow; b\uparrow; b\downarrow$.

3.2.2 Super Buffered Fet Logic

For larger loads, a *super buffer* stage can be used (see Figure 3.6 (a)). This style combines well with DCFL, and provides some savings in power for similar delays. A price is paid in the area used.

The buffering is achieved by a push-pull stage, that does not require static power. However, there is a moment in the switching transient when



(a) (b)

Figure 3.4: DCFL circuits, (a) inverter, and (b) nor-gate.

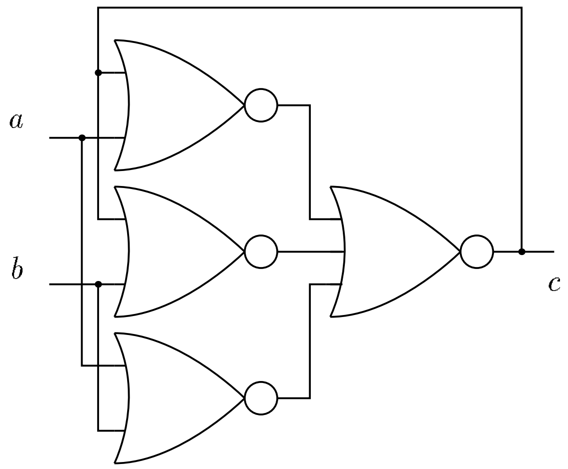


Figure 3.5: Nor-gate C-element. Observe that it has a race condition for the sequence $a\uparrow$; $a\downarrow$; $b\uparrow$; $b\downarrow$. Normally, this sequence should not produce any output. Initially, c is low

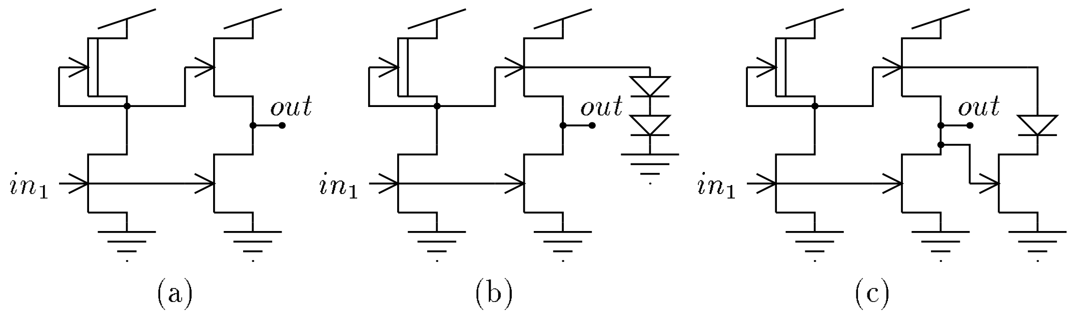


Figure 3.6: SBFL inverters.

both transistors are turned on, and care has to be taken that this current spike does not cause any problem in neighboring circuits.

When the load is highly capacitive, and the fanout is small, the pull up transistor will end up being too large and force the next voltage well above the maximum input voltage to the next gate. This has to be prevented, since it will cause the output gate to malfunction. Figure 3.6 (b) and (c) show two ways of achieving this objective.

Chapter 4

A New Logic Family

Introduction

In this chapter we present a novel way of implementing logic circuits in GaAs. This logic style is specifically designed for asynchronous circuits. The special needs of these circuits make usual nor-gate based technologies to be less than ideal, specially for the lack of C-elements, complex gates, completion trees, etc.

Although all basic elements can be implemented with nor gates, this introduces extra nodes in the circuit that have to be taken care of. The delay insensitivity would be greatly compromised without making extensive timing assumptions on those intermediate nodes, assumptions that we would be unable to guarantee if, for example, we use a standard cell place and route program to generate the layout.

Also, noise margins and fanout have to be increased to simplify the task of mapping logic equations into actual circuits. It is an important feature that the designer need not be overly concerned with the electrical characteristics of the target circuitry.

4.1 Input Stage

Several input stages have been designed, depending on the logic function to be implemented. Even though they have different characteristics with respect to load, they share the same logic levels, and can be arbitrarily mixed.

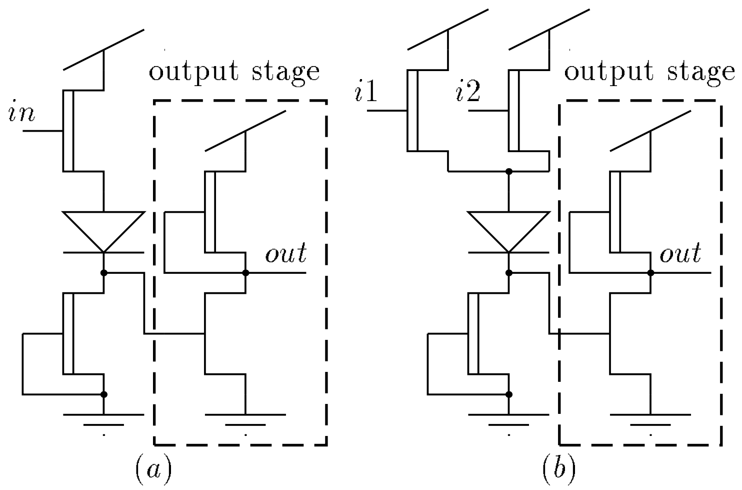


Figure 4.1: SFFL inverter (a) and nor gate (b).

This stage allows the input signal to switch rail to rail, implements the logic in the gate, and reduces the coupling between the input and output signals. However, a price is paid in some extra delay and power consumption.

4.1.1 Inverter

An inverter illustrates how the basic gate works (see Figure 4.1 (a)). The input stage is a source follower. The input signal is shifted in level and isolated from the output circuitry. This allows the input to switch rail to rail. The power supply is selected so that the Schottky junction of the input transistor is barely forward biased over V_D (threshold voltage of the schottky diode), so that there is very little current going into the gate (see Figure 4.2)

The correct operation of this circuit depends on the ratio between the pull-up transistor and the passive pull-down. The constraint on the ratio is two sided, to guarantee a good logic low and high for the output stage. A spice simulation puts this ratio between 1 and 2. However, if the threshold voltage of the d-mode transistors is approximately the same as the threshold voltage of the diode, the logic low for the output stage is guaranteed irrespective of the ratio.

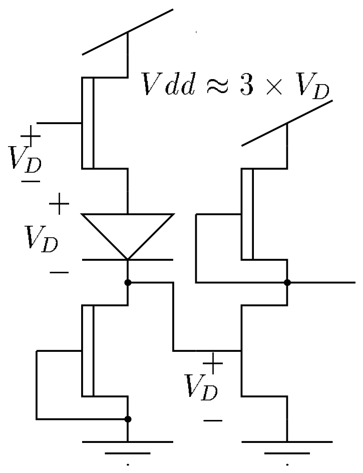


Figure 4.2: Vdd has to be less than $3 \times V_D$ to prevent forward conduction on the gate of the input transistor.

4.1.2 Nor gate

We make the inverter into a nor gate by adding another input transistors (see Figure 4.1 (*b*)). The input stage looks now like a wired-or gate.

Very wide nor gates are possible. However, as the intermediate node becomes larger, the asymmetry between the up-going and down-going transitions grows; the passive pull-down is weaker than the active pull-up.

Even wider nor gates can be implemented by using a DCFL nor gate as the output stage, thus using both stages of the gate for logic.

4.1.3 Nand gate

The obvious way of implementing a nand function in this design style would be to stack several transistors in the pull-up chain. However, this will not work, since the current flowing into the gate of the lower transistor may be enough to force the gate on, even if the top transistor is cut. This effect can be easily verified with a SPICE simulation of the circuit in Figure 4.3. The input transistor is now working as a diode, and the input signal has to provide the current to maintain the high level. Depending on the strength of the input signal, either the gate will switch or it will pull the input down to $\approx 2 \times V_D$.

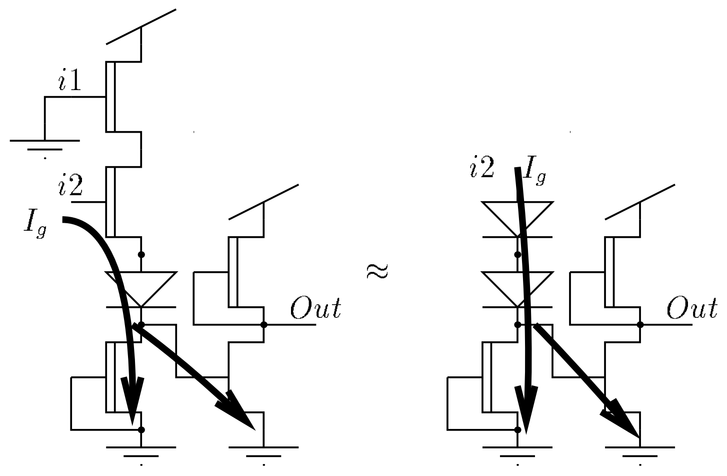


Figure 4.3: Current leakage into the gate of a nand gate. When the top transistor is cut, the gate will behave as the circuit on the right.

Protection Circuitry

To circumvent this problem we can use the protection circuitry shown in Figure 4.4. The extra input transistor limits the current going into the gate of the next transistor, and is cut-off when the top input transistor cuts the path to V_{dd}.

This protection circuit can be used with more inputs, to get bigger nand gates. The practical limit seems to be around 5 inputs, as derived from SPICE simulations.

Sneak Paths

The protection circuitry introduces a conductive path to ground when the input is low. If we have a complex pull-up, we have to be careful that we are not creating a Vdd to ground path through forward conducting diodes (see Figure 4.5). The safest way to get around this problem is to use one pull-up chain, including an extra diode, for each minterm in the sum-of-minterms representation of the logic function we are implementing. Simplifications are possible in particular circuits, but we have to make sure that all sneak paths have been considered.

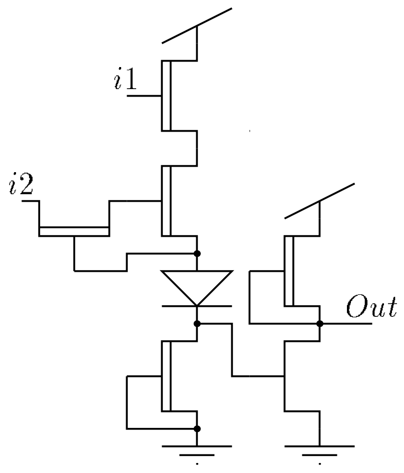


Figure 4.4: Nand gate with protection circuit.

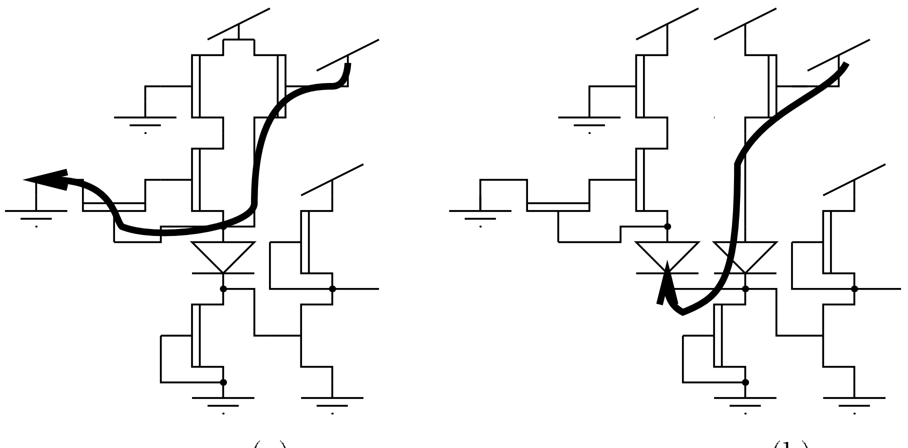


Figure 4.5: Sneak path in an And-Or-Invert gate (a) and how to fix it (b).

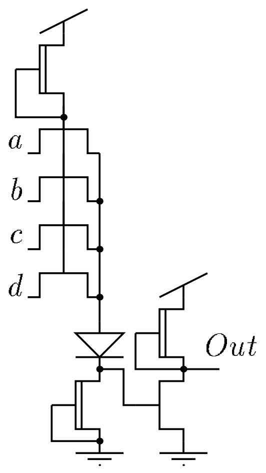


Figure 4.6: Parallel input nand gate.

4.1.4 Parallel configuration Nand gate

An alternate way of implementing nand gates is shown in Figure 4.6. In this case the forward conduction on the Schottky junction is used to create an input stage similar to the TTL multi-emitter transistor. Since these transistors are in parallel, we can build large gates, like 32-input completion trees. This will be the main use of this configuration.

As a disadvantage, it requires a large pull-down transistor to get the input down to logic low. This is specially important if that signal has to go to other inputs of different type.

4.2 Output Stage

The input stage generates a signal with DCFL logic levels. This signal has to be restored to rail-to-rail voltage swing.

4.2.1 DCFL output stage

The simplest output stage we can use is a DCFL inverter. Since the output will not be tied to another DCFL gate, it will be able to have a wider output swing, depending on the type of input it goes to.

The strength ratio between the output transistors can be calculated to maximize the noise margins, as in DCFL gates. This results in much weaker ratios than are typical for DCFL, between 6 and 8 instead of 10 to 14.

4.2.2 SBFL output stage

For bigger loads it is advantageous to use a super-buffer configuration for the output. The small d-mode transistor in the pull-up is added to increase the noise margins, at the expense of some power.

The design and sizing of this super-buffer is very different from the DCFL super buffer. The load is mostly the capacitive load of the wires. Some extra current is necessary to drive the parallel type nand inputs in the fanout.

This output circuitry is preferred in most cases for capacitive loads, since the passive pull-up is not very fast driving a signal all the way to the Vdd rail.

4.3 C-elements and Completion Trees

We have two different strategies for building C-elements.

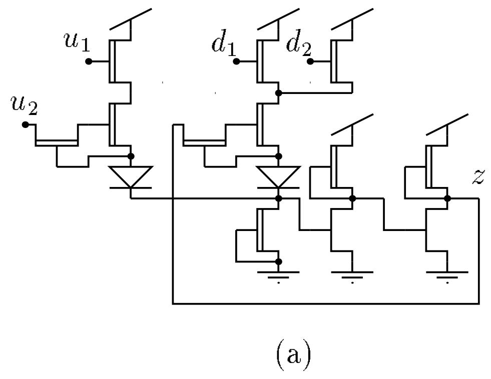
4.3.1 Majority Gates

Given the pair of production rules,

$$\begin{array}{lcl} u1 \wedge u2 \wedge \ldots & \rightarrow & z\uparrow \\ \neg d1 \wedge \neg d2 \wedge \ldots & \rightarrow & z\downarrow \end{array}$$

we can transform them into $z = u1 \wedge u2 \wedge \ldots \vee z \wedge (d1 \vee d2 \vee \ldots)$. This can be implemented as a complex gate. Figure 4.7 shows what a two-up two-down C-element would look like.

As before, there are two alternatives for implementing the *and* part of the equation. The series chain is preferred for two or three inputs, the parallel chain being used above that.



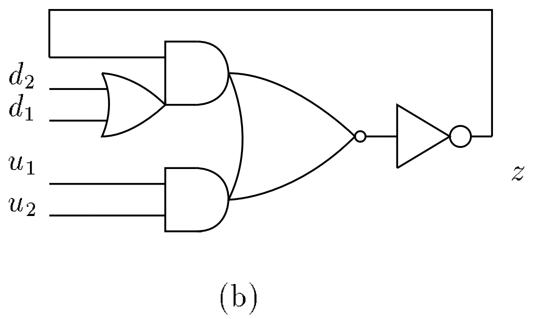


Figure 4.7: Two-up two-down C-element, transistor network (a), and gate schematic (b).

4.3.2 Cross-coupled Operators

The same production rules can be implemented in a different way; we provide two versions for each signal z, zf and zt, and the production rules for this new set becomes,

$$\begin{array}{lcl} zt \vee u1t \wedge u2t \wedge \ldots & \rightarrow & zf\downarrow \\ \neg zt \wedge \neg(u1t \wedge u2t \wedge \ldots) & \rightarrow & zf\uparrow \end{array}$$

$$\begin{array}{lcl} zf \vee d1f \wedge d2f \wedge \ldots & \rightarrow & zt\downarrow \\ \neg zf \wedge \neg(d1f \wedge d2f \wedge \ldots) & \rightarrow & zt\uparrow \end{array}$$

These circuits are combinational, and can be directly implemented with complex gates. Also, the negative logic version of this circuit is realizable, and might be more convenient in some cases, since it requires wide nor gates instead of wide and gates.

4.3.3 Completion Trees

We build a completion tree as shown in Figure 4.8. Note that instead of using an extra inverter in each majority gate we use the fact that each stage

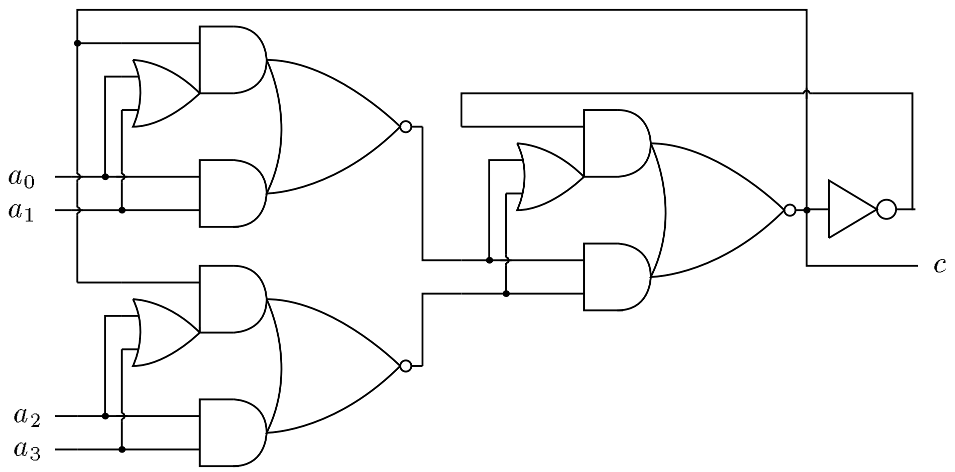


Figure 4.8: Four input completion tree implemented with majority gates.

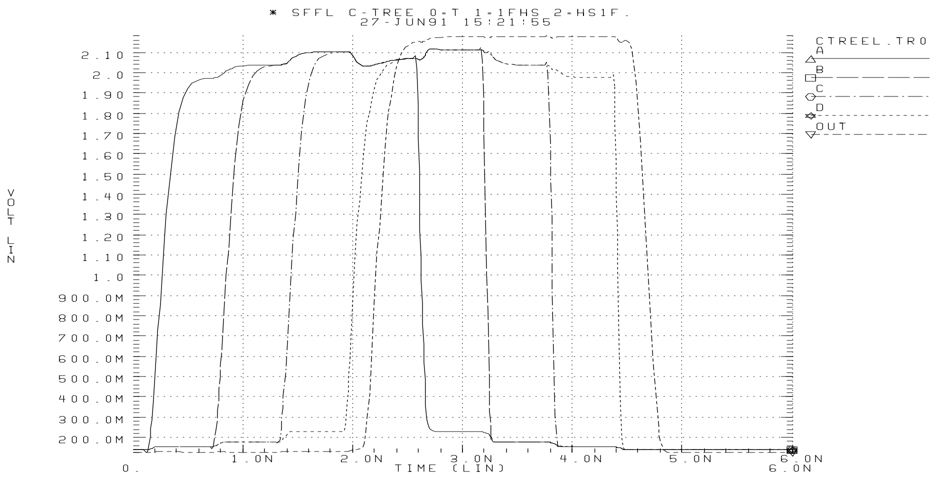


Figure 4.9: Spice simulation of a 4-input C-element with super-buffered output

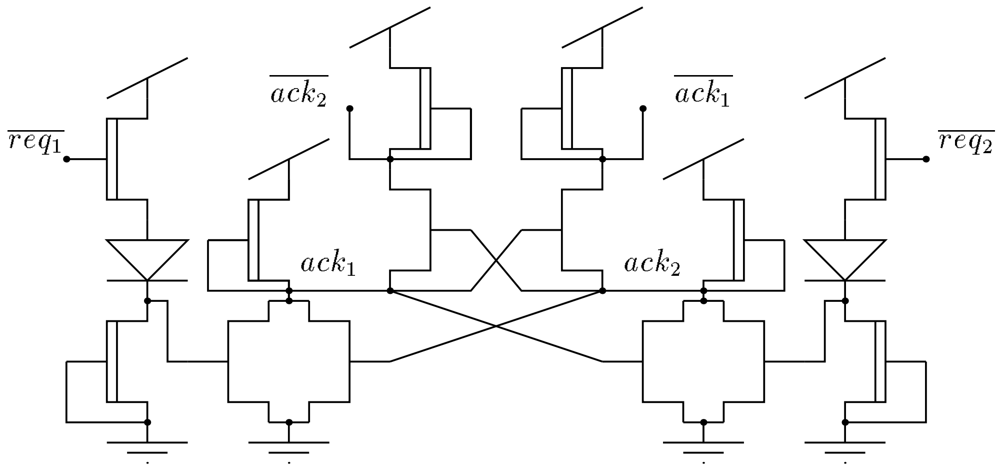


Figure 4.10: Two input arbiter.

is inverting and we get the feedback signal from the following stage. We need an inverter in the last stage to staticize the whole tree.

Using combinations of 2-input and 3-input elements, we can build an 8 input tree with 2 levels of logic, a 16 input tree with 3, and a 32 input tree with 4.

This circuit does not work as a general purpose C-element, because of races in the intermediate nodes.

4.4 Arbiter

Figure 4.10 shows an implementation of an arbiter circuit [Sei80]. This is a mutual exclusion element; it takes possibly concurrent requests and generates mutually exclusive acknowledge signals.

Notice that this is the same arbiter we would use for DCFL logic, with level conversion in the input signals. The output signals can go rail to rail.

Figure 4.11 shows an HSPICE simulation of the arbiter circuit, with both inputs arriving at the same time. The numerical noise in the circuit simulator breaks the symmetry to produce a winner in the arbitration. Notice that even with very low noise level the arbiter will make a decision in a very short time.

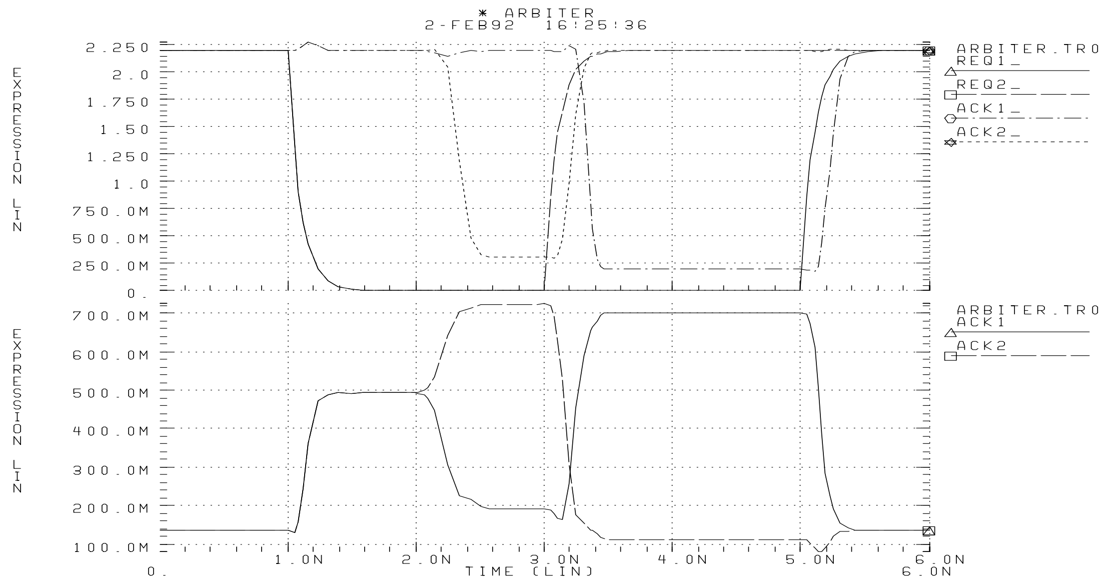


Figure 4.11: Spice simulation of the arbiter of Figure 4.10

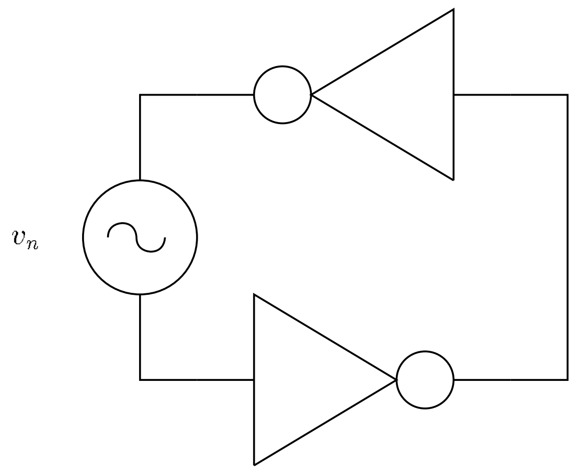


Figure 4.12: Circuit used to determine the noise margins.

4.5 Noise margins

We define the noise margins as the amplitude of the noise signal necessary to switch the output of a latch [LB90]. This can be measured using the circuit shown in Figure 4.12. Because signals switch rail to rail, noise margins are greatly improved with respect to other design styles. For the simple inverter, we get approximately $0.8V$; noise margins in DCFL are about $0.2V$ for the same power supply.

Chapter 5

Timing Model

Introduction

In this chapter we characterize a timing model for the family of logic gates that we have just introduced. The target technology is Vitesse's enhancement-depletion mode gallium arsenide process (Hgaas II). A linear timing model analogous to the τ-model of CMOS is necessary to estimate circuit performance and transistor sizes [Bur91]. However, this is difficult to achieve due to the nature of the gates used. Some analysis of the circuits is necessary to get such a model, with some loss of accuracy.

The circuits designed using this timing model are close in performance to those obtained by very computationally intensive methods, hence the model represents a good way of giving an initial guess for the transistor sizing. Other applications include choosing between several buffer styles, making power estimates, finding critical paths, and cycle times.

Since the *Hspice* model for Vitesse's process was highly accurate in several chips that were designed, tested and checked against the simulated figures, we used this program to get the data to fit the model.

5.1 Gate Speed

To characterize the speed of an inverting gate, we define the following parameters (see Figure 5.1).

The first two parameters indicate how fast the output stage is.

- Rise time, t_{rise}: the time it takes for the output of the gate to go from 10% to 90% of its high value (0% is its original value).
- Fall time, t_{fall}: the time it takes for the output of the gate to go from 90% to 10% of its final value.

The next two parameters represent the delay we get per gate in a long chain.

- Low to high gate delay, t_{dlh}: Time from input reaching switching voltage to output reaching switching voltage, with the output going low to high.
- High to low gate delay, t_{dhl}: Same as above, with the output going high to low.

The next two parameters represent how long it takes for the gate to switch after a change in one of the inputs. They can be used as a worst case for gate delay.

- Input up to output low time, t_{ul}: Time from input going up to output is low.
- Input down to output high time, t_{dh}: Time from input going down to output high.

5.2 Simplified gate model

We use a simple switch model for all transistors in the gate, to get a model for the delays. Transistors will be modeled as an ideal switch plus a series (drain) resistance. The model will be checked later against simulations to adjust the theoretical parameters [LB90].

5.2.1 Input stage

The input stage can be modeled as a fixed delay independent of the load conditions, plus a variable part dependent on the size of the output stage. The fixed delay is technology dependent, and related to the internal capacitances

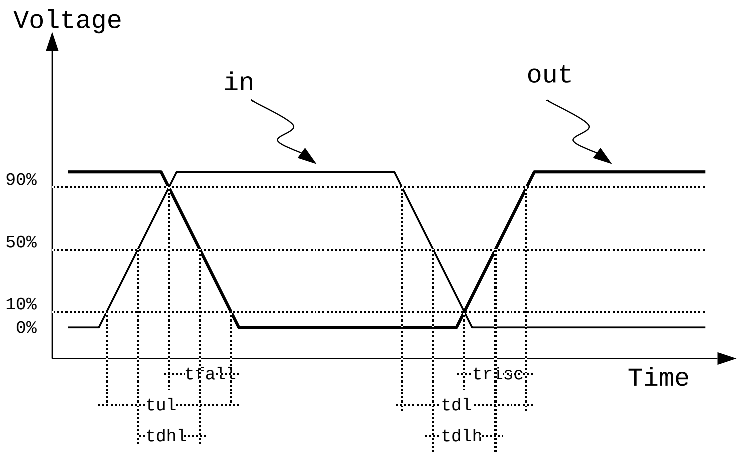


Figure 5.1: Delay model for an inverting gate.

of the MESFET's in the input stage. The variable part depends primarily on the gate to substrate capacitance of the output transistor. Since we are charging this capacitance with current sources, we will assume this dependency to be linear.

This stage will not affect the rise time of the input signal, since it has unit voltage gain.

The transfer from input voltage to current in this first stage is roughly linear, from 0 to I_i (see Figure 5.2). The maximum current is limited by the size of the pull-down transistor. If we assume that the fraction of time that the input is high is δ and that the fraction of time that the input spends switching from low to high or high to low is γ, then the power consumption of this stage is:

$$P_{is} = V_{dd} \times I_i \times (\delta + \frac{\gamma}{2})$$

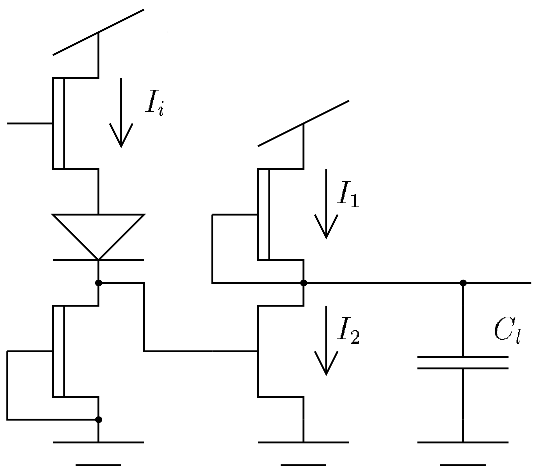


Figure 5.2: Inverter currents.

5.2.2 Output stage

The output stage acts as a current source that can provide an output current I_1 in the high state, and an input current $I_2 - I_1$ in the low state. I_1 is the saturation current of the pull-up transistor, I_2 is the saturation current of the pull-down transistor turned on (see Figure 5.2). These two currents have to charge the load capacitance. The time to do so is:

$$t_{rise} = \frac{(V_h - V_l) \times C_l}{I_1} + t_{0l}$$

$$t_{fall} = \frac{(V_h - V_l) \times C_l}{I_2 - I_1} + t_{0h}$$

Where C_l is the load capacitance, V_l and V_h are the logic low and high voltages respectively, t_0 is a fixed delay due to the internal capacitances of the model, and t_{rise} and t_{fall} were defined previously (see Figure 5.1).

The t_{0l} and t_{0h} parameters depend on the ratio of strengths of the two output transistors, as well as their gate lengths. Since the gate lengths are fixed by the technology, and the strength ratio is determined by maximizing the noise margin of the gate, these parameters are constant; the currents I_1 and I_2 scale with the output stage.

We separate the power consumption of this stage in two parts, as usual. One contribution comes from the static current in the low state, I_1. This is provided by the power supply at V_{dd} Volts, dissipating

$$P_s = V_{dd} \times I_1 \times \delta$$

where δ is the fraction of time that the gate spends on the output low state. There is no static power dissipation in the high state. The rest of the power comes from charging and discharging the load capacitance. Since we are assuming that this capacitor is charged by a current source, the energy dissipated every cycle is

$$P_d = C_l \times V_{dd}^2 \times f_s$$

where f_s is the switching frequency.

5.2.3 Delay model and power-delay product

We can now put everything together to get a model for delays and power consumption. Taking into account all terms, we have,

$$t_{rise} = \frac{k_1 \times C_l}{S} + k_2 + \frac{k_3 \times S}{S_i}$$

$$t_{fall} = \frac{k_4 \times C_l}{S} + k_5 + \frac{k_6 \times S}{S_i}$$

The load capacitance C_l is formed by the wiring capacitance and the gate capacitance of the transistors the node is connected to. We add those two terms explicitly and we get:

$$\begin{aligned}
t_{dlh} &= k_1^1 + k_2^1 \times \frac{S_o}{S_i} + k_3^1 \times \frac{C_l}{S_o} + k_4^1 \times \frac{f_{out}}{S_o} \\
t_{dhl} &= k_1^2 + k_2^2 \times \frac{S_o}{S_i} + k_3^2 \times \frac{C_l}{S_o} + k_4^2 \times \frac{f_{out}}{S_o} \\
t_{rise} &= k_1^3 + k_2^3 \times \frac{S_o}{S_i} + k_3^3 \times \frac{C_l}{S_o} + k_4^3 \times \frac{f_{out}}{S_o} \\
t_{fall} &= k_1^4 + k_2^4 \times \frac{S_o}{S_i} + k_3^4 \times \frac{C_l}{S_o} + k_4^4 \times \frac{f_{out}}{S_o}
\end{aligned}$$

The parameters S_o and S_i are size factors for the output and the input stage respectively, relative to the size of the minimum inverter. These times

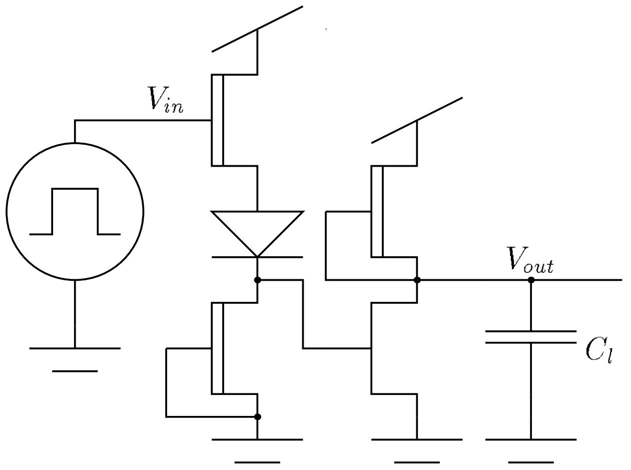


Figure 5.3: Circuit used to determine the delay model parameters.

can be obtained from measurements or simulations. For this linear model, we can use *least squares analysis* to determine the parameters. The correlation coefficient indicates how well the model fits the data.

We put together a similar model for the power consumption.

$$P_T = P_i \times (\delta + \frac{\gamma}{2}) \times S_i + (P_o \times \delta + P_f \times C_l) \times S_o$$

Where δ is the duty cycle, γ is the transition time as a fraction of the period. Again, this model is linear, and can be calculated from simulations or measurements. Coefficient P_i represents the contribution to power from the input stage, P_o comes from the output stage, and P_f is the frequency dependent part.

5.3 Inverter parameters

We first calculate a set of parameters for a simple inverter, using the technology parameters from VITESSE's 1.2μ process. Since these parameters depend somewhat on the shape of the input voltage, we use a standardized input that represents reasonably well the signals on-chip (see Figure 5.3). Other parameters like δ and γ can be fixed to some standard value in the model.

This set of parameters was calculated using *Hspice* simulations on a standard inverter . These simulations were repeated for a total of 480 different combinations of input stage size, output stage size, load capacitance, and fanout. The values were:

$$S_i \in \{1.0, 2.0, 3.0, 4.0\}$$

$$S_o \in \{1.0, 2.0, 3.0, 4.0, 6.0, 8.0\}$$

$$C_L \in \{5.0, 10.0, 20.0, 40.0\}$$

$$fanout \in \{1, 2, 3, 4, 8\}$$

Where $S_i \times 2.8$ is the width in μm of the transistors of the input stage, $S_o \times 2.0$ is the width in μm of the pull-up transistor of the output stage and C_L is the load capacitance in *ff*. The lengths of all transistors is $1.2\mu m$. The width of the output pull-down is 7 times the width of the output pull-up, as determined by noise-margin maximization.

From each of these simulations we extract seven parameters, t_{dlh}, t_{dhl}, t_{rise}, t_{fall}, P_i, P_o, P_f as defined previously. Low and high limits were taken to be 30% and 70% respectively.

A *Least Squares Fit* is used to extract the model parameters and get an estimate of the correlation of the fit. The correlation coefficient between estimated and measured parameters was better than 99% in all cases.

5.4 Multi-input gates

Once we have the model parameters for the inverter, we would like to see what effect the number of inputs has on the gate delay. For nor gates with a small number of inputs (less than four) the effect is minor, and absorbed by the model errors. For two-input nand gates, the largest we are considering in this style, we get the same model parameters by choosing properly the relative sizes of the transistors in the input stage.

In general, small corrections can be added to take into account the number of inputs and asymmetries, as in the case of nand gates, trading accuracy for simplicity.

5.4.1 Other delay models

There are other types of gates with slightly different topologies, that require that the delay model be specially fitted for them. One such gate is the super buffer stage, that can be used to increase the drive capability of any of the other gates. This stage can be modeled independently as a load to a standard gate, or it can be modeled as a simple gate. The second alternative is better, since the relative sizing of the different stages has to be altered for optimum speed and power.

5.5 Numerical results

The data for the fit was obtained from the following *Hspice* file.

```
* Calculate tdhl tdlh tr tf and power.
.param vdd=2.2 vlo=0.0 hi=vdd mid=1.1 ll=0.3 hh=1.8
.global vdd vddp % Vdd for circuit and environment.
*
* Load circuit. This is the unit fanout.
.subckt load in
j1 vddp in   sfdl  dep1.2 L=1.2u W=2.8u
j2 sfdc sfdl sfdc ddep1.2 L=1.2u W=2.0u
j3 sfdc gnd  gnd   dep1.2 L=1.2u w=2.0u
.ends load
*
* Input Signal to the test circuit
vin in gnd dc 0.0 exp(vlo hi 0 80ps 1000ps 80ps)
vdd  vdd  gnd dc vdd
vddp vddp gnd dc vdd
*
* Define the test circuit with parameters.
* si = size of the input stage
* so = size of the output stage
j1 vdd  in   sfdl  dep1.2 L=1.2u W='2.8u*si'
j2 sfdc sfdl sfdc ddep1.2 L=1.2u W=2.0u
j3 sfdc gnd  gnd   dep1.2 L=1.2u w='2.0u*si'
j4 out  sfdc gnd   enh1.2 L=1.2u W='14.0u*so'
j5 vdd  out  out   dep1.2 L=1.2u W='2.0u*so'
```

```
*
xl out load M=fanout
cl out gnd cload
*
* Measure delays, rise and fall time, power dissipation.
.meas TRAN tdhl TRIG v(in) VAL=mid RISE=1 TARG v(out) VAL=mid FALL=1
.meas TRAN tdlh TRIG v(in) VAL=mid FALL=1 TARG v(out) VAL=mid RISE=1
.meas TRAN trise TRIG v(out) VAL=ll RISE=1 TARG v(out) VAL=hh RISE=1
.meas TRAN tfall TRIG v(out) VAL=hh FALL=1 TARG v(out) VAL=ll FALL=1
.probe par('so/si') par('si/so')
.measure TRAN pd AVG par('-p(vdd)')
*
.tran 10ps 2000ps SWEEP data=datan
.include 'delays.data'
.include '/arpa/jat/cad/hspice/vscmodels/models'
.lib '/arpa/jat/cad/hspice/vscmodels/corners' tt
.end
```

These are the numerical results for the inverter simulations:

$$\begin{aligned}
t_{dlh} &= 35.0 + 49.6 \times \frac{S_o}{S_i} + 4.1 \times \frac{C_l}{S_o} + 11.7 \times \frac{f_{out}}{S_o} \\
t_{dhl} &= 98.6 + 18.3 \times \frac{S_o}{S_i} + 0.7 \times \frac{C_l}{S_o} + 0.3 \times \frac{f_{out}}{S_o} \\
t_{rise} &= 29.0 + 65.3 \times \frac{S_o}{S_i} + 7.4 \times \frac{C_l}{S_o} + 21.6 \times \frac{f_{out}}{S_o} \\
t_{fall} &= 109.0 + 18.7 \times \frac{S_o}{S_i} + 0.9 \times \frac{C_l}{S_o} + 0.0 \times \frac{f_{out}}{S_o} \\
P_d &= 0.25 \times S_i + 0.4 \times S_o
\end{aligned}$$

Times are in *ps*, capacitances in *ff*, f_{out} in number of equivalent gates, and power is in mW. The P_f parameter was found to be too small to be meaningful compared with P_i and P_o, and was left out.

5.6 Example: D-element

Consider the circuit of Figure 5.4. We will try to size the different gates for optimum speed, as defined by the oscillation frequency. In this case, the metric is:

$$T = t_{dlh1} + t_{dhl1} + t_{dlh2} + t_{dhl2} + t_{dlh3} + t_{dhl3} + t_{dlh4} + t_{dhl4}$$

We will assume a standard wiring capacitance of 10.0*ff* on all wires. We constrain the search space by limiting the static power dissipation to $5mW$. In this circuit all gates work with a duty cycle of $\frac{1}{3}$; we modify the power equation accordingly, that is, $\delta = 0.3$, $\gamma = 0$.

Making all gates equal, we get the following equation to minimize,

$$\frac{T}{4} = 133.6 + 67.9 \times \frac{S_o}{S_i} + 4.8 \times \frac{10.0}{S_o} + 12.0 \times \frac{S_i}{S_o}$$

subject to the constraint,

$$5 \geq 1.25 \times S_i + 2.0 \times S_o$$

There is a minimum for $S_o = 1.2$, $S_i = 2.1$, for which the period is $T = 936ps$. A direct simulation of this circuit yields a period of $T = 998ps$. The same circuit was optimized directly with *Hspice*; the same power equation was used as a constraint, with the following results: $S_o = 1.6$ $S_i = 1.4$ $T = 920ps$.

5.7 Conclusions

A linear timing model was generated for this family of logic gates. This model was made linear, with the objective of using it to optimize delays, cycle times, critical paths, since tools for such a model are available. In general, this model has given a good approximation of the optimum for the circuit, and with low computational complexity. This optimum can be used as a starting point for a more elaborate procedure, or used directly if the complexity of the circuit is too high.

More elaborate models can be generated in the same way if the required accuracy has to be higher. However, simpler models can be used to design very large circuits, which was one of the motivations for this experiment.

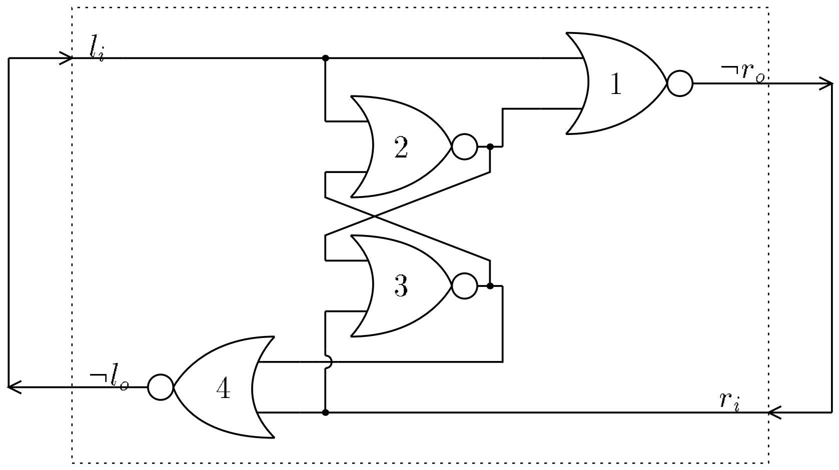


Figure 5.4: Free running D-element.

Chapter 6

Design Example

Introduction

The ideas presented in this document were used to build a number of circuits. Among them, several memories and register files, and a simplified version of the Caltech asynchronous processor.

The process used was VITESSE's HGaAs II, offered through the MOSIS foundry brokerage service.

In this chapter, we show the design of a static RAM as an example of an asynchronous circuit built with the logic family presented earlier.

6.1 Self timed memory

This section describes an implementation in GaAs of a Self-Timed random access memory chip. This memory chip has a capacity of 64 words of 4 bits, a simulated read time of 2.6 ns and a simulated write time of 3.0ns. The chip is fully delay insensitive, except for some isochronic forks and the circuitry to convert to and from dual rail encoding. The original circuit for a *CMOS* version of the memory was derived by H. P. Hofstee[Hof91].

6.1.1 Decoder

The decoder takes as input the address encoded in dual rail, and generates a decode signal:

$*[[v(Address)]; s(Address)\uparrow; [z(Address)]; s(Address)\downarrow]$

This is implemented as an array of C-elements:

$$\begin{array}{lcl} a0? \wedge a1? \wedge \ldots & \rightarrow & sa\uparrow \\ \neg a0? \wedge \neg a1? \wedge \ldots & \rightarrow & sa\downarrow \end{array}$$

where each question mark represents either a 1 or a 0, to fully decode all addresses.

These C-elements are implemented in *SFFL* style, with parallel input nand gates. The output is super-buffered, with the sizes optimized for a row of 4 bits. The feedback inverter has an extra transistor to take into account the shift in threshold in the memory cells. The output of these C-elements goes to *DCFL* nand gates in the memory arrays.

Alternate outputs go on alternate sides of the decoder, so as to reduce the height of the columns to half in number of bit-cells. A total of 4 decoders take care of the 8 memory arrays.

6.1.2 Bit-cell

The bit-cell was implemented with 12 transistors. This is the basic 10 transistor cell used in the CMOS version of the memory, where two extra transistors were added to ensure that every signal goes to the same level in all pull-down chains (Figure 6.1). The data lines have passive pull-ups, which are reasonably fast due to the small height (8 words) of each memory array.

6.1.3 Ram arrays

The memory has 8 8-word ram arrays, for a total of 64 words. The ram arrays are paired with nand gates on the data-buses, to reduce by a factor of two the load on the passive pull-ups of the bit lines. This nand gate also converts the *DCFL* level of the bit line into *SFFL* level to be processed by the output circuitry.

The output circuitry determines whether the operation was a read or a write, and generates the output data, again in *DCFL* level.

The 4 resulting pairs are connected together via an output data bus, pulled up with passive pull-ups. This bus goes to the pads and the completion detection circuitry.

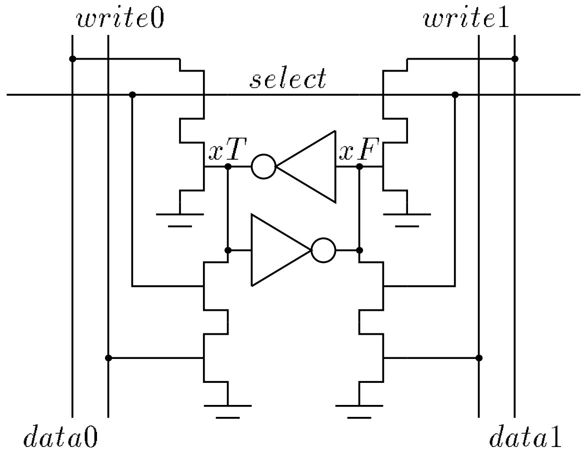


Figure 6.1: Transistor schematic of a 12T bitcell

All buses in this design were implemented in *DCFL* logic levels. Due to the limited voltage swing, they operate faster and can be pulled up with a passive pull-up. Whenever possible, third level metal was used to minimize the capacitance to ground. Otherwise, second level metal was use; there are no buses in first level metal.

6.1.4 Single rail to dual rail conversion

This conversion was done at *DCFL* levels, which seemed to offer the best figures in power dissipation and speed.

The address buffers had to generate very strong signals to drive the decoders, and with rail to rail voltage swing. This required very careful electrical design, and uses a good part of the total power dissipation (about 30%). For a bigger decoder, some pre-decoding needs to be done so that the fanout of the address bus does not become too large.

6.1.5 Completion signal.

The completion signal was derived from the data output. An *SFFL* C-element was used. The data was converted to *SFFL* level with nand gates taking both rails of each bit signal as input. There is no separate completion for read and write, though they can be generated by duplicating the C-element and adding one input for the corresponding control signal.

6.1.6 Sizing.

Sizing was done with the Hspice optimizer on each individual circuit, using nominal loads. The loads were taken from a preliminary sizing and layout, or from already sized circuits.

The circuits were optimized for speed, but the power consumption was added as a soft constraint to get a reasonable result.

The first circuit to be sized was the bitcell. The first parameter to be determined was the maximum height of the array that could be handled by passive pull-ups on the bit lines. Though 16 seemed to be possible, it required that the transistors on the bitcell itself be too big. Eight by four arrays were therefore used.

The transistors on the bitcell came next. The sizes were estimated, and then optimized for speed on a write cycle.

With the bitcell fixed, the next circuit to be sized was the decoder. The actual load, the bitcell array, was abstracted and used in the process. The optimization goal was to get about the same delay on the decoder as on the array.

Address single-to-dual rail converters and drivers were sized next. The load had to be carefully calculated, since there is interaction between the different inputs to the decoder's C-elements. It turned out that the load represented by the 32 inputs was roughly the same as 11 equivalent gates with no interaction.

Data drivers and output circuitry was sized next. This represented a small percentage of the total delay, since it is done concurrently with other operations.

The total breakdown in delays is as follows: data conversion 25%, decoding of the address 25%, memory array 25%, pad delays 25%.

6.1.7 Simulation results

We show here the simulation results for the memory(Figure 6.2). This was done with Hspice on the whole circuit. The simulation consists of two write actions, followed by two read actions.

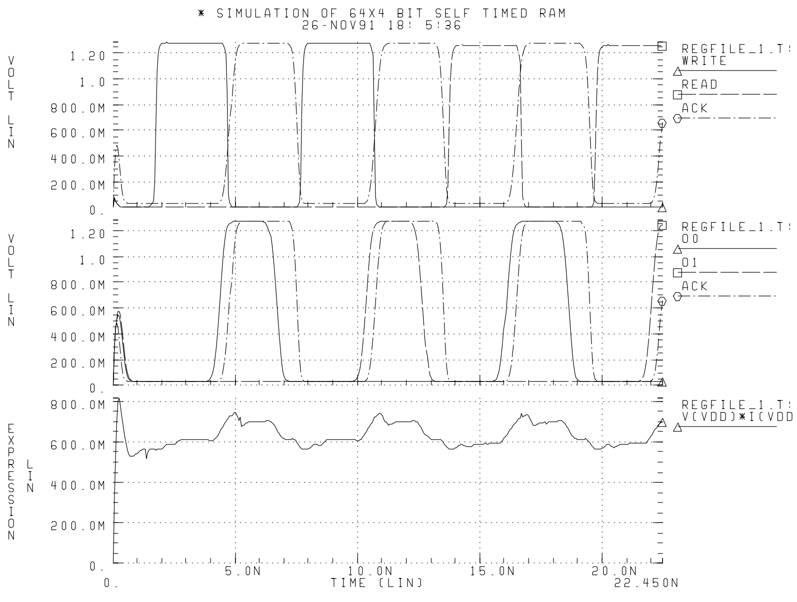


Figure 6.2: Simulation results.

The read operation access time is 2.7ns; the write operation takes 3.0ns; cycle time is 6ns. The total power dissipation for this memory is 700mW.

6.1.8 Experiment results

The memory was fabricated and tested. Results agree with the simulation to a high degree, in timing and power consumption. However, part of the circuitry does not work at the nominal voltage. This is attributed so far to inaccuracies of the spice model, or the way the circuit parasitics are extracted. The contact resistances in the input stage could play an important role, and are ignored by the extractor.

Chapter 7

Conclusions

So far, we have achieved the following objectives:

We designed a number of circuits that implement directly arbitrary production rules. Some are specialized for some specific functions, like completion trees or arbiters.

We derived a simple timing and power model that allows us to get a quit estimate of delays and power consumption, for a subset of the circuits introduced earlier.

A standard cell library was implemented, that is used by the standard cell place and route program *Vgladys*. This library was used in several circuits that were fabricated.

A program was written to do several syntactic and static checks on the transistor network extracted from the circuit layout, to verify as much as possible that the circuit is legal, and create a new circuit that can be simulated using *Cosmos*. Cosmos is a ternary switch level simulator.

A number of circuits were fabricated to test different aspects of this design style. Among them, divide-by-two counters, D-elements, register files, RAM arrays, and am asynchronous microprocessor. The last one is not fully functional, but has provided some useful data.

On the down side, we noticed the following problems:

These circuits require more power than equivalent circuits implemented in DCFL, and sometimes extra delay. However, it is very hard to implement arbitrary production rules in DCFL, without compromising the delay-insensitivity of the circuit.

Some gates become extremely complex and slow. In these cases, it is always a good idea to look for an alternative circuit, by transforming the production rules.

These gates turned out to be rather sensitive to sizing, specially the more complex ones. So far they have been fixed at the expense of extra power.

Future work

We need a better sizing strategy, that gives robust gates with respect to parameter variation, and doesn't have a big price in power dissipation.

A more elaborate timing model, that takes into account a wider variety of gates would be useful for complex designs.

DCFL circuits should be used a lot more, to gain in density and power dissipation. This would require in general that the productions rules for the circuit be changed so that they can be implemented with combinational gates; an alternative would be to implement the datapath in DCFL, reserving our gates for the control circuitry.

Acknowledgments

The research described in this paper was sponsored by the Defense Advanced Research Projects Agency, DARPA Order number 6202, and monitored by the Office of Naval Research under contract number N00014-87-K-0745.

Many thanks to Steve Long for very inspiring lectures in GaAs electronics.

Sam Reynolds, Wes Hansford, and Terry Dosek of ISI provided invaluable help with the submission of all circuits fabricated.

Cindy Hibbert, of Meta Software, for keeping HSPICE running for us.

This work would not have been possible without the help of Alain J. Martin and his research group, Steve Burns, Pieter Hazewindus, Tony Lee, Dražen Borković, and Marcel van der Goot. They gave invaluable comments and help throughout the development of this project.

Bibliography

[Bur91] Steven M. Burns. *Performance Analysis and Optimization of Asynchronous Circuits.* PhD thesis, California Institute of Technology, 1991.

[CVFC87] C. T. M. Chang, T. Vrostos, M. T. Frizzel, and R. Carrol. A subthreshold current model for GaAs MESFET's. *IEEE Electron Device Letters*, EDL-8(2):69–72, February 87.

[Hof91] H. P. Hofstee. Deriving some asynchronous memories. Unpublished, 1991.

[LB90] S. I. Long and S. E. Butner. *Gallium Arsenide Digital Integrated Circuit Design.* McGraw-Hill, New York, 1990.

[Mar86] Alain J. Martin. Compiling communicating processes into delay-insensitive VLSI circuits. *Distributed Computing*, 1(4):226–234, 1986.

[Mar90] Alain J. Martin. The limitations to delay-insensitivity in asynchronous circuits. In William J. Dally, editor, *Sixth MIT Conference on Advanced Research in VLSI*, pages 263–278. MIT Press, 1990.

[Met90] Meta-Software, Inc., Campbell, Ca. *HSPICE User's Manual*, 1990.

[MH72] Jacob Millman and Christos C. Halkias. *Integrated Electronics: Analog and Digital Circuits and Systems.* McGraw-Hill Electrical and Electronic Engineering Series. McGraw-Hill, New York, 1972.

[Sei80] Charles L. Seitz. System timing. In Carver A. Mead and Lynn A. Conway, editors, *Introduction to VLSI Systems*, chapter 7. Addison-Wesley, 1980.

[SNS+87] H. Statz, P. Newman, I. W. Smith, R. A. Pucel, and H. A. Haus. GaAs FET device and circuit simulation in SPICE. *IEEE Transactions on Electron Devices*, ED-34(2):160–169, 1987.

[vdS85] Jan L. A. van de Snepscheut. *Trace Theory and VLSI Design*, volume 200 of *Lecture Notes in Computer Science*. Springer-Verlag, 1985.