AN ABSTRACT OF THE THESIS OF

Christopher Hanken for the degree of Master of Science in
Electrical and Computer Engineering presented on August 22, 2007.

Title: Simulation and Modeling of Substrate Noise Generation from
Synchronous and Asynchronous Digital Logic Circuits

Abstract approved:

Terri S. Fiez Karti Mayaram

Efficient methods for simulating the substrate noise generated by complex synchronous and asynchronous digital logic circuits are presented. By simulating digital logic at the gate level, and precharacterizing the gates, the substrate noise generation can be predicted and used in a transistor level simulation of the sensitive analog blocks. This approach is shown to have better than 20 percent peak-to-peak matching for both traditional CMOS logic and NULL Convention Logic (NCL) by correctly modeling critical gate characteristics. Synchronous and asynchronous versions of a pseudo-random number generator (PRNG) are implemented in a 0.25μm CMOS test chip. Simulations validate both a standard

transistor level setup and the predictive substrate noise approach against measurements. Simulations of an 8051 processor, with separate synchronous and asynchronous logic cores, are in good agreement with measurements from another 0.25μm CMOS test chip, and show validation with a large and complex circuit.

Simulation and Modeling of Substrate Noise Generation from Synchronous and Asynchronous Digital Logic Circuits

by

Christopher Hanken

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

Presented August 22, 2007
Commencement June 2008

Master of Science thesis of Christopher Hanken presented on August 22, 2007.

APPROVED:

Co-Major Professor, representing Electrical and Computer Engineering

---

Co-Major Professor, representing Electrical and Computer Engineering

---

Director of the School of Electric Engineering and Computer Science

---

Dean of the Graduate School

I understand that my thesis will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my thesis to any reader upon request.

Christopher Hanken, Author

ACKNOWLEDGEMENTS

I am very grateful for all the experiences and opportunities that I have had at Oregon State University. The staff and faculty have provided me an excellent education and foundation for my career, and life. In particular I would like to thank my advisors, Dr. Terri Fiez and Dr. Kartikeya Mayaram. I have learned so much from both of them, and sincerely appreciate their support throughout my degree.

I would like to thank the institutions which funded my research. In particular the DARPA TEAM and CLASS programs, and SRC under contract 2005-TJ-1299. I also appreciate all the work and technical support provided by the people at Theseus Logic, in particular Mike Hagedorn and Jeff Wilson.

I have had the honor of coming to know a multitude of intelligent students throughout my graduate studies. I would like to thank all of the people in the analog and mixed-signals group for their friendship and mentoring. In particular thanks to Robert Batten, Vova Kratyuk, Min Gyu Kim, Merrick Brownlee, James Ayers, and Thomas Brown for all their help. I would also like to thank everyone I worked with in the substrate coupling group. Martin Held, Kyle Webb, Sasi Kumar Arunachalum, Chenggang Xu, Kavitha Srinivasan, Arthi Sundaresan, Matt McClary, Andrew Tabalujan, Brett Peterson, Waileng Cheong, Jacky Wong, and Bob Shreeve all made working in this research group enjoyable

and enlightening.

I have made some very personal friends during my 3 years of graduate study. I would first like to thank Jim Le. Jim and I were together working on classes and research throughout my degree. We were so joined at the hip that, despite our lack of physical similarities, our advisors would continually mix up our names. I would like to thank all of the friends I was fortunate enough to hang out with both at school, and outside of school, including Erik Vernon, Sunwoo Kwon, Nema Talebbeydokhti, Nishant Chadha, Celia Hung, Dave Gubbins, and Jeongseok Chae to name a few. I would also like to give a special thanks to Pavan Hanumolu for his friendship and mentoring. Pavan is a great friend who can just as comfortably discuss a technical problem, or the latest sports news.

Without the support of my family, I would not have been able to accomplish any of this. I would like to thank my brothers Joel and Benjamin, my mother Diane, and my father David, for their love and encouragement. Thank you to Amy, my better half, for always being there. I would also like to thank Amy's family for all their support, including Eddie, Robert, Jen, Beth, and Ed.

I would finally like to thank God for allowing the pieces of my life to come together the way they have. I feel very blessed.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

LIST OF FIGURES (Continued)

LIST OF FIGURES (Continued)

LIST OF TABLES

LIST OF APPENDICES

LIST OF APPENDIX FIGURES

DEDICATION

To Amy, for always believing in me

Chapter 1 – Introduction

The prediction of substrate noise coupling effects in Systems-on-a-Chip (SoC) is becoming an increasingly important issue. As larger numbers of circuits are integrated on the same substrate, the differences between ideal simulations and the actual performance of the silicon can become pronounced. If coupling issues are discovered and addressed in simulation, appropriate and adequate steps can then be taken to reduce or isolate the noise.

This thesis describes methods and considerations for the simulation of substrate noise. This chapter examines the background and previous work on which this thesis is based. Chapter 2 describes the setup of the simulation tests. An improved method for modeling digital circuitry for substrate noise simulation is presented in Chapter 3. Results for the simulation tests are presented in Chapter 4. Chapter 5 analyzes different aspects of substrate noise simulation. Finally, conclusions and future work are described in Chapter 6.

1.1 Background

As IC technologies continue to scale down, high power consumption in traditional CMOS digital logic makes the exploration of alternative implementations more appealing. One such alternative is asynchronous digital logic, such as NULL Con-

vention Logic (NCL) [1]. This logic presents advantages in power consumption, synchronous switching, and subsequently, substrate noise [2]. It is, therefore, important to simulate the substrate noise generation from these alternative implementations.

The simulation of substrate noise is most accurate with complete knowledge of how the circuits will be laid out in silicon. Post-layout information provides a substrate network which is complete. By incorporating this substrate network with a full transistor and parasitic description of the circuitry, the substrate noise simulations can be accurately matched with measurements. Several tools are available to facilitate post-layout simulation, including the Cadence tool SNA [3] and Silencer! [4].

In contrast, pre-layout simulation is of particular interest for designers. Earlier discovery of substrate noise issues allows for adjustments not only at the layout level, but also at the schematic level. A methodology is presented which facilitates pre-layout substrate noise simulations. With a Hardware Description Language (HDL) netlist of the digital block, the digital block's substrate noise behavior is simulated using the characterization information generated from a standard cell library. Along with a schematic level netlist of the analog block, substrate noise is estimated without final layout information.

Similar methodologies for substrate noise prediction have been presented in [5, 6, 7, 8]. However, none of these have addressed the simulation of noise in asynchronous circuits. The method presented in this thesis accounts for key issues that are critical for predicting the noise generated in asynchronous circuits. Specifi-

cally, the effect of the delay modeling and state information in asynchronous logic is examined. An advanced delay model and an approach that accounts for the internal states of a sequential gate have been developed. These form the basis for an efficient and accurate approach for simulation of substrate noise.

1.2 Previous Work

The method for modeling the substrate noise behavior of digital blocks, or digital macro-models (DMM), presented in this thesis is based upon previous work [9, 10]. The basic concept behind these models is to simplify the simulation by deriving a model for the digital logic which retains the correct substrate noise injection characteristics. By simulating the digital logic with a HDL, the correct switching behavior of the logic is preserved. This is an idealized simulation which gives no substrate noise information. A mapping of the switching of a digital gate to the amount of current through the ports of the gate allows calculation of the substrate noise. The simulation is now a two step process, the simulation of the HDL for switching and the simulation of the current sources along with the sensitive circuits. The advantage of the two step process is the efficiency and speed increase over the equivalent single simulation at the transistor level. Much of the digital gate information preserved by a full transistor level simulation is not needed in substrate noise simulations.

The DMM presented in [10] requires two library characterization steps. Standard cell libraries for digital gates need to be characterized for the simulation of

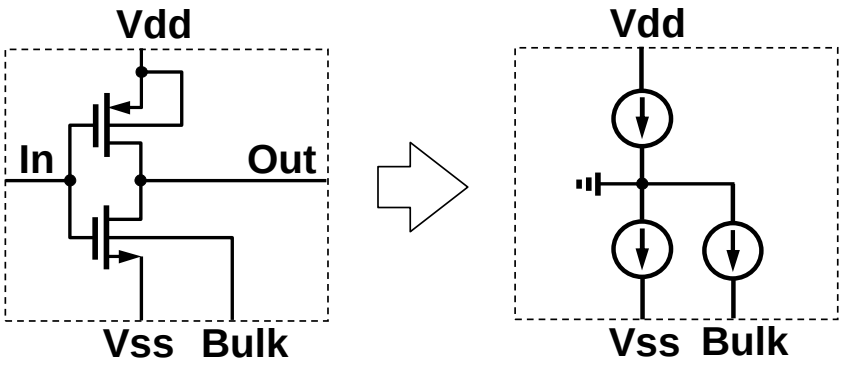

Figure 1.1: Model previously used to represent the digital gates.

substrate noise through this method. The first characterization determines the gate's local substrate network. The second characterization determines the current waveforms used when mapping switching information to the equivalent gate model for the final substrate noise simulation.

The gate model of an inverter for the DMM presented in [10] is shown in Fig. 1.1. In this model, the non-linear transistors are replaced with current sources. The sources model the current passing through the gate ports to the supply lines. The gate model has no input or output ports. The digital signal information passed between gates is represented only by the changes in the current sources.

1.3 Test Chips

Two test chips (Losco and Eris) were fabricated in TSMC's 0.25μm logic process. The test chips have clocked Boolean logic (CBL) and NCL versions of digital logic circuitry implemented on each. This section describes the circuits and implementations on the test chips. The lab setup and measurements are presented in more

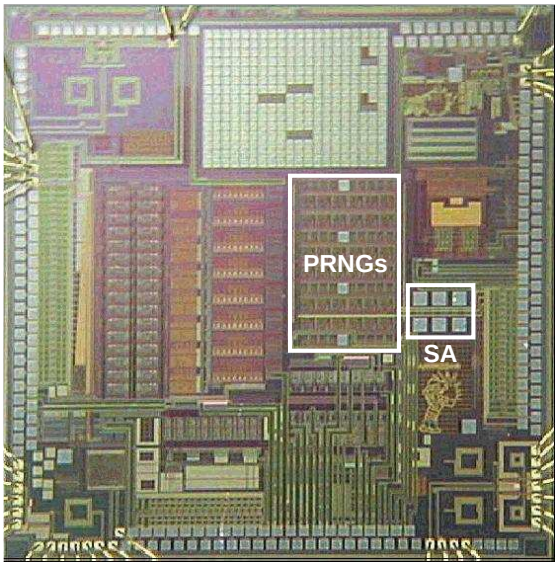

Figure 1.2: Die photo for the Losco test chip.

detail in [11].

1.3.1 Losco Test Chip

The die photo for the Losco test chip is shown in Fig. 1.2. The pseudo-random number generators (PRNGs) are the digital circuitry implemented on this chip in both CBL and NCL versions. There are 72 PRNGs, 36 CBL and 36 NCL, interdigitated in the region indicated in Fig. 1.2. The PRNG block is a combination of an adder, multiplier, and register. When correctly seeded the PRNG generates a pseudo-random number sequence. The sequence repeats every 256 cycles, and generates a digital word representing numbers between 0 and 255.

A sense amplifier was implemented on the Losco test chip in the region labeled

'SA' in Fig. 1.2. This circuit is approximately 200μm from the closest edge of the PRNGs. The sense amplifier is a differential low gain, wide bandwidth amplifier, which has one input capacitively coupled to a dedicated 'quiet ground' pin, and another input capacitively coupled to the substrate [12, 13]. The amplifier is designed to drive 50 Ω loads through ground-signal-ground (GSG) probes placed directly on the die.

The CBL PRNGs are clocked at the equivalent operating frequency of the NCL PRNGs. The equivalent operating frequency is defined as the rate the CBL circuitry must be clocked to complete the same functions as the NCL over the same time period. After testing, this was found to be approximately 50MHz for the PRNGs.

Each version of PRNGs is tested at separate times for comparisons between the CBL and NCL logic. The printed circuit board (PCB) used to test the chip has jumpers for disconnecting any unused pins, so all pins not used in a particular test are left floating. The signal generator used for the CBL clock has rise and fall times of 1.5ns. All circuits were supplied with on-PCB regulation of 2.5V except the buffers which required 3.3V.

1.3.2 Eris Test Chip

The die photo for the Eris test chip is shown in Fig. 1.3. An 8051 microprocessor was designed with two separate cores in the region indicated. One core was designed using CBL, and the other core with NCL. Both 8051 cores share the RAM,

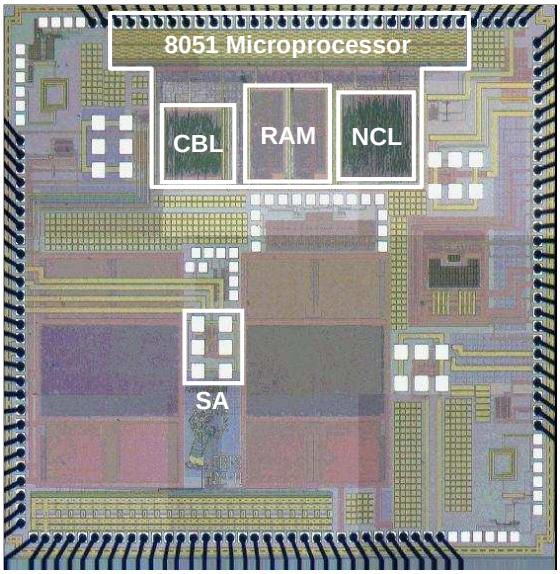

Figure 1.3: Die photo for Eris test chip.

IO buffers, and wrapper logic. The wrapper logic is a collection of digital gates used to connect and control the signals between the different sections of the microprocessor. The cores can operate separately for characterization of their substrate noise generation. The shared RAM is loaded with a program which generates a random number sequence in an effort to create random substrate noise with limited program memory.

The same sense amplifier implemented on the Losco test chip, is also implemented on the Eris test chip in the region labeled 'SA' in Fig. 1.3. This circuit is approximately 1mm from the 8051 microprocessor cores.

As done in the PRNG setup, the CBL core is clocked at the equivalent operating speed of the NCL core. This equivalent frequency was determined to be 33MHz for the 8051 microprocessor. It may be confusing to note that this frequency is no

longer the same as the PRNG blocks, while the technology and logic libraries are exactly the same. However, the NCL and CBL are not directly comparable. The number of gates and configurations are synthesized differently. For this reason, different configurations of digital logic offer slower or faster equivalent operating speeds.

Chapter 2 – Simulation Setup

Substrate noise is a deterministic noise source. The transitioning of digital gates injects noise into the substrate at levels much higher than random noise. For this reason, substrate noise can be accurately predicted in a transient simulation. This chapter describes the setup of the transient simulations presented in this thesis. The setup of the circuit blocks is described in the first section. The second section presents the parasitic modeling used in simulations. The final section examines the substrate and its modeling in simulations.

2.1 Circuit Blocks

Transistor level simulations are the most accurate method for transient simulation. Transistors, however, are non-linear devices, and require significant computational time to converge to a solution. Although the full transistor level simulation is accurate, there is a significant trade off in terms of simulation time and convergence issues. Replacing transistor level descriptions of digital blocks with an equivalent DMM reduces both the number of instances and nodes in a substrate noise simulation. However, only digital blocks composed of gates from a standard cell library can be replaced by an equivalent DMM. This section presents a top level description of the circuit blocks used in the simulations.

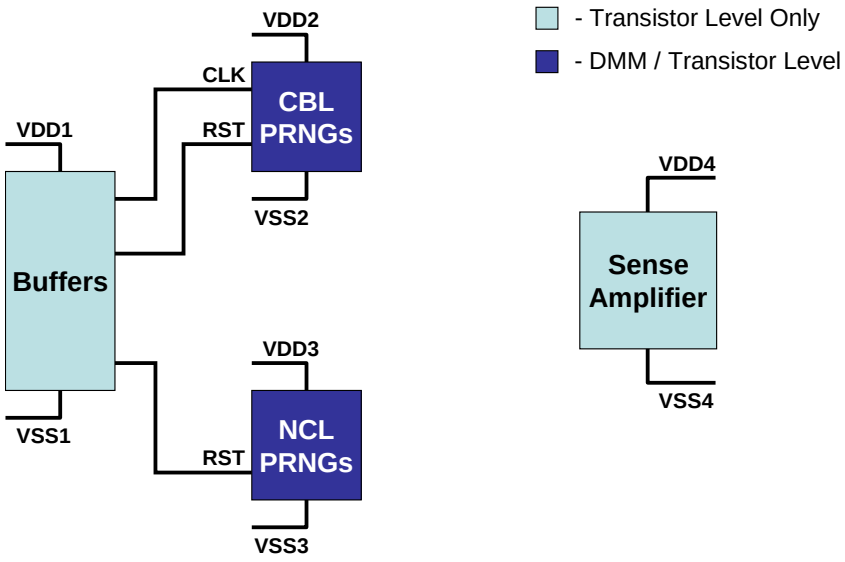

Figure 2.1: Block diagram showing the circuit setup for the simulation of the PRNG blocks.

2.1.1 Pseudo-Random Number Generators

A block diagram of the circuit setup for the PRNG simulations is shown in Fig. 2.1. The IO buffers and sense amplifier are simulated as transistor level blocks. Although the IO buffers are digital blocks, they are not composed of gates from a standard cell library, and cannot be replaced with an equivalent DMM. The CBL and NCL PRNG blocks have both transistor level and equivalent DMM versions. Simulation of the transistor level PRNG blocks is performed for comparison with the measurements. The DMM versions are simulated and compared with the transistor level versions for validation of the DMM methodology.

The connections in the circuit setup are identical to those in the test chip. The supply lines for each circuit block have separate pins for VDD and VSS. The clock

and reset lines are connected through the IO buffers to the PRNG circuits. The DMM versions of the PRNGs have the clock and resets aligned correctly with the outputs of the IO buffers. This alignment is required for correct timing of substrate noise injections from the different circuit blocks.

2.1.2 8051 Microprocessors

A block diagram of the circuit setup for the 8051 microprocessor simulations is shown in Fig. 2.2. The sense amplifier is simulated as a transistor level block. The DMM method was used on three separate blocks in the 8051: the NCL core, the CBL core, and the wrapper logic. Each block described as an equivalent DMM is characterized twice, once for the running of the NCL core, once for the running of the CBL core. Although one core is turned off during operation of the other, the equivalent gate parasitics are required for accurate simulation. The IO buffers and RAM blocks are not implemented with discrete gates and must be simulated at the transistor level. Simulation of the complete microprocessor at the transistor level is not possible due to convergence issues of such a large circuit. The circuit setup is simulated and compared with measurements.

Since the RAM is described at the transistor level, it must have the correct stimulus and initial conditions to operate correctly. An adjusted version of the DMM is used to find the input switching voltages to the RAM (described more in Appendix A). This facilitates the simulation of the RAM without simulating the rest of the microprocessor at the transistor level. The initial conditions for the

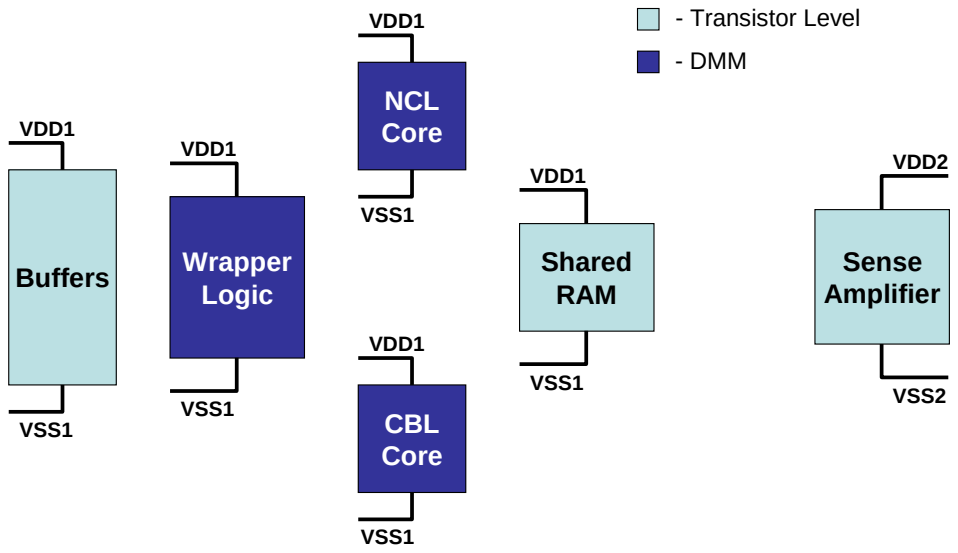

Figure 2.2: Block diagram showing the circuit setup for the simulation of the 8051 microprocessor cores.

RAM are found by a separate simulation. By simulating the RAM block alone, and with a simpler transistor model, the program memory is loaded. The final conditions for this simulation are used as the initial conditions for the RAM in all 8051 microprocessor simulations.

The p-tap connections of other circuits on the Eris test chip have their ground pins always connected, and must be modeled for the 8051 microprocessor simulations. Two large digital signal processing (DSP) blocks, a delta-sigma modulator (DSM), and the buffers for the DSPs all have low resistance paths from the single node backplane out through their ground pins. All of these connections to the substrate network are included in the simulations.

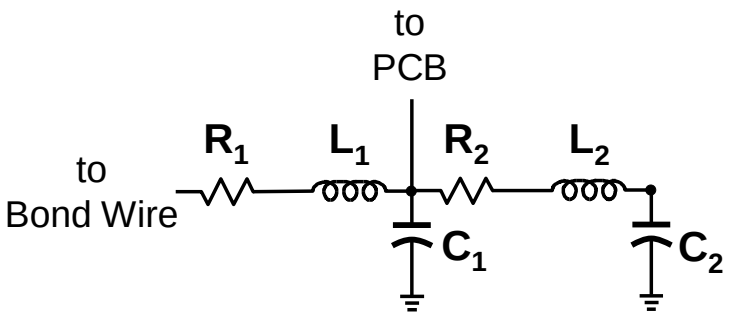

Figure 2.3: Model used to represent the parasitics of the package.

2.2 Parasitics

The modeling of off-chip parasitics is critical for the simulation of substrate noise. These off-chip parasitics include bond wires, packaging, socket, and PCB. All the simulations in this thesis pay careful attention to accurately modeling these parasitics.

The bond wires are modeled with the approximation that a gold bond wire of 1 mil diameter has 1 nH/mm of inductance and 50 mΩ/mm of resistance. The length of the bond wires are determined for each setup, and used in simulation.

The package is modeled based on the manufacturer's data. The package for the 8051 microprocessors (Eris) is a Kyocera ceramic PGA132. The package for the PRNG circuits (Losco) is a Kyocera ceramic PGA257. However, there is no available manufacturer's characterization for the PGA257. The next closest size was the PGA132 package, so an extrapolation of the parasitics is used. The package model given on the data sheet is shown in Fig. 2.3. The ground in the model is assumed to be an ideal ground for the simulations.

The socket and PCB parasitics are the most difficult to model for simulation.

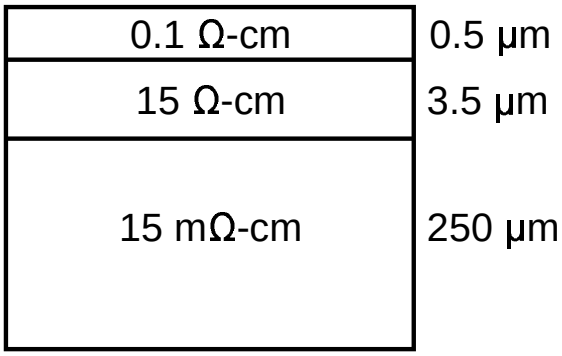

Figure 2.4: Approximated 3-layer profile for the TSMC 0.25μm heavily doped substrate.

There are not many socket models readily available, and PCB parasitics require the use of more complicated physical modeling to extract a SPICE/Spectre model. For the test cases presented in this thesis, large decoupling capacitors are placed very close to the chip on the PCB. Large ground planes make it possible to assume ideal supply and ground at these decoupling capactors. An inductor in series with a resistor is therefore used as a simple model for the socket and PCB trace. By using the method described in [14], both values are determined and used in simulation.

2.3 Substrate Modeling

Heavily doped substrates with a high resistivity epitaxial layer are commonly used to avoid latch-up issues in SoCs. The profile for the TSMC 0.25μm logic process (a heavily doped process) was approximated as the 3 layer profile shown in Fig. 2.4. In this substrate, the bottom layer has a much lower resistivity and is considered to be single node for the entire chip [15].

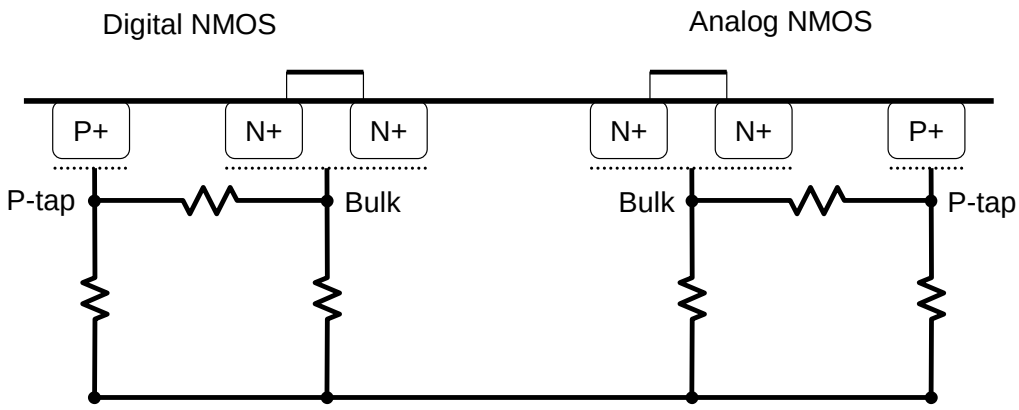

Figure 2.5: Model used to represent the substrate.

Using this single node approximation, the digital and analog block's substrate networks are extracted separately. As long as the blocks have adequate separation (4 times the epitaxial layer and channel stop layer thicknesses [15]), the cross-coupling can be assumed to be negligible compared to the coupling through the single backplane node.

For frequencies below 1GHz, the substrate network is modeled as a resistive network [16]. Since the PMOS transistors are isolated in n-wells, their substrate noise coupling is significantly reduced and, consequently, their connections to the substrate need not be modeled very accurately. A capacitor in series with a resistor from the power supply to the backplane is used. The NMOS transistors are connected to a π-model substrate network as shown in Fig. 2.5. Both the p-taps and the transistor bulk terminals are shown.

The resistive substrate network can be extracted using finite difference [16, 17] or boundary element methods [18, 19]. The extractions for this thesis have used EPIC, a Green's function based boundary element solver [20].

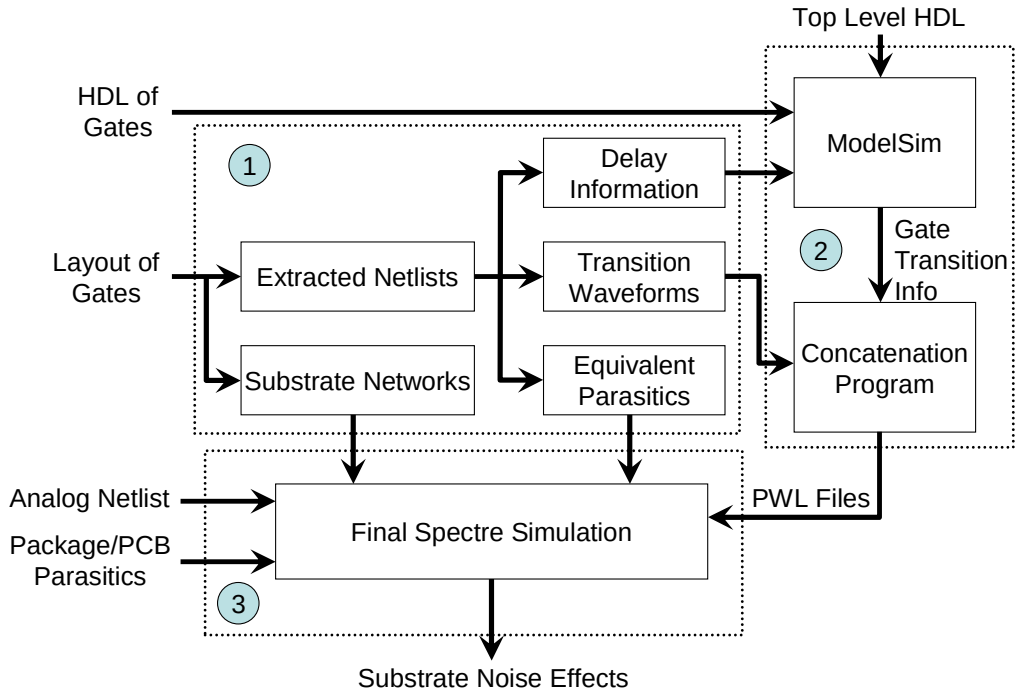

Figure 3.1: Flow chart showing all the steps in the digital macro-modeling (DMM) method.

Chapter 3 – Digital Macro-Model (DMM)

The improved DMM method presented in this thesis is described with the flow chart shown in Fig. 3.1. The flow is divided into three distinct parts. The first part is the characterization. The second is the digital noise current generation. The third part is the substrate noise simulation. All parts are described in this chapter, with the digital noise current generation and substrate noise simulation both being incorporated in the simulation flow section.

3.1 Characterization

Before the creation of a DMM for a digital block, the standard cell library used must be characterized. The standard cell library implemented in a specific technology is characterized gate by gate. Both a transistor and HDL netlist of each gate in the library is required for characterization.

There are four types of characterization for each gate. These are displayed, along with the extraction of netlists (described briefly in the transition waveforms section), as Part 1 in Fig. 3.1. The following subsections will describe the different characterizations. More specific details on coding and how the characterizations are done is described in Appendix A.

3.1.1 Substrate Networks

The substrate network for the complete digital block is a combination of locally derived gate substrate networks. The local substrate networks for each gate are π-networks representing the resistances between the bulk contact (active region) of the transistor, the p-tap contacts, and the single-node approximated backplane. As in the previous method [10], all bulk regions are approximated as a single node, and an equivalently sized contact is created.

The contact information for each gate is found from the layout information in the Cadence environment (Contact Lister, see Appendix B). EPIC simulations for each gate are run to find the resistive substrate network. A text file containing the resistor network of each gate in the library is saved and used in the simulation

flow, described in Section 3.2.

Combining the local substrate networks into a complete substrate network is a valid approximation for several reasons. Since the substrate is a heavily doped substrate, the cross coupling resistances become negligible, compared to the coupling to the single node backplane, for distances larger than 4 times the thickness of the epitaxial and channel stop layers [15]. For the profile used, this equates to distances of approximately 16μm. This means only directly adjacent gates will be close enough to be significant. The circuit level connections of the contacts also play a role. The p-tap contacts are all connected together to the digital on-chip ground, making the cross coupling between these contacts of adjacent cells negligible. The other type of contacts, the bulk contacts, are positioned close enough to the p-taps to provide a low coupling resistance. The cross coupling to other contacts outside of the gate will be negligible compared to this low resistance.

3.1.2 Equivalent Parasitics

One of the characteristics of the digital gates which must be preserved in the DMM representation is the equivalent parasitics between the supplies (Vdd and Vss). The equivalent parasitics are formed by a combination of routing capacitances, channel resistances, and junction capacitances. Many digital macro modeling methods presented by other sources appear to neglect these important parameters as well [5, 6, 9, 10]. One reason for this may be that these parasitics do not affect substrate noise simulations in smaller digital circuits. Another reason may be that

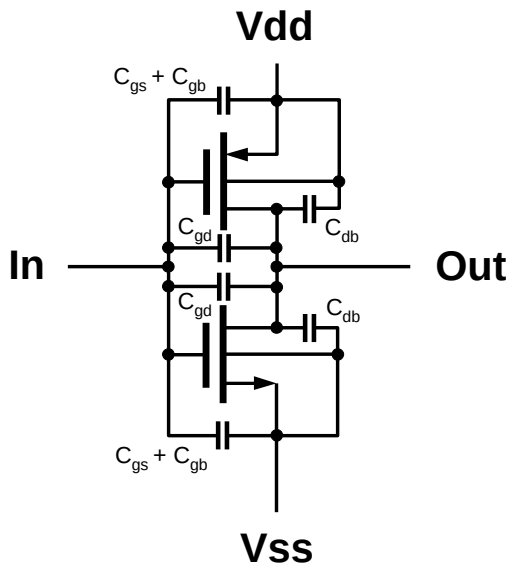

Figure 3.2: The parasitics of a simple inverter.

decoupling capacitances added on-chip can be much larger than some digital circuit's equivalent parasitics. However, as the size of a digital circuit increases, these parasitics become important for an accurate estimation of supply ringing, and subsequently, substrate noise.

To determine the type of equivalent model that should be used, an examination of a simple inverter's parasitics will first be conducted. Neglecting any routing capacitances, an inverter's parasitics are shown in Fig. 3.2. In this figure, the resistance through the substrate is assumed to be zero, making the source and bulk the same node for each transistor. The gate-to-bulk and gate-to-source capacitances are combined under this assumption.

By assuming an input voltage to the inverter, the parasitics can be simplified further. The input to the gate is tied low through a previous logic gate's NMOS

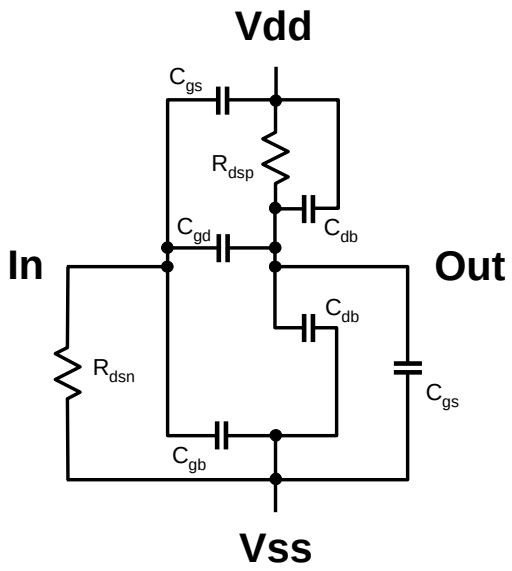

Figure 3.3: Equivalent parasitics of an inverter when the input is held low and the overlap capacitances are ignored.

transistor, and the output is tied to the C_{gs} of the next logic gate's NMOS transistor. The equivalent parasitics are now represented by Fig. 3.3. The overlap capacitance is assumed to be negligible in this figure. The NMOS transistor of this gate is replaced by an open circuit since it is in cutoff. The PMOS transistor is replaced with an equivalent resistor since it is in deep triode. From this figure it can be seen that a dominant pole between the rails is formed by $\frac{1}{R_{dsn}\ C_{gs}}$. Another similar sized pole between the rails is formed by $\frac{1}{R_{dsp}\ (C_{db}+C_{gs})}$. As this analysis is expanded to larger gates, the problem becomes much more complex. However, the dominant pole or poles will continue to come from the deep triode resistance which holds the output of a gate to either supply, and the capacitances from that output to the opposite supply.

The preceding analysis justifies the characterization of each gate's equivalent

parasitics as a resistor in series with a capacitor between the supply lines. The values of these equivalent parasitics vary depending on the digital state of the gate being modeled. Each gate is characterized for all input combinations and internal states. The parasitics are characterized after settling, with no input or output transitioning occurring. The parasitic information is stored and used in the simulation flow (described in Section 3.2) to construct a complete set of equivalent rail parasitics.

3.1.3 Delay Information

The correct delay information is important in making sure the gate level simulation's timing, and subsequently, the substrate noise waveforms are equivalent to the noise generated in a transistor level simulation. For clocked logic, all combinational logic following the flip-flops needs to be triggered at the correct time after a clock edge. For clockless logic this becomes an even larger issue, since any small error in delay can cause differences not only locally for a proceeding gate, but globally and cumulatively, since the logic is never re-synchronized with a clock.

For these reasons each cell must be correctly characterized to generate a Standard Delay Format (SDF) file that is back annotated during the gate level simulation. The gate level delays are then combined to form a global delay file. One assumption made with this method is that the majority of the delays for the overall digital block will come from the switching of the gates themselves. The delays for the switching of the gates are usually orders of magnitude larger than the delays

added from interconnects between gates. A global SDF file is used in the DMM method presented in [10]. This file is not always available or accurate for a specific process.

Multiple SDF files are generated for each gate from an extracted netlist. The delays for different paths through a gate can depend heavily on the number of other gates connected to each output. Therefore, the gates are simulated multiple times and a set of SDF files are created which can be selected when simulating a specific configuration. The gates are loaded with an average-sized gate, since specific loading would require either a pre-known configuration or unacceptable amounts of simulation and storage.

The delays through a gate vary due to internal effects, not just external effects like output loading. The delay from an input to an output can be affected by the value of other inputs or the internal state of the gate. These delay paths are called conditional delays. The best SDF file is one which not only characterizes all delay paths, but also all conditional delays of each of these paths. The characterization first produces this type of SDF file.

Even with the best SDF file, the delay information must be correctly represented in the gate's HDL level model. If the HDL models do not have conditional delay paths described in them, the information in the SDF file will not be used in the simulation. The back annotation will only adjust the delay paths, not add new ones. Some HDL gate models contain conditional delays which may be simplified from several specific conditional delays. To account for all of the preceding issues, the complete SDF file must be tailored to correctly match the delays provided in

the HDL model. This adjusted SDF file is saved for later use in the simulation.

The delays are timed from the switching voltage of an input to the switching voltage of an output. The switching voltage can be a difficult parameter to determine. If the switching voltage is defined as the point when the output equals the input, then the switching voltage from high-to-low and low-to-high is equal. However, this voltage can change based on the input and output branches and the sizing of the transistors. Finding when an input switches is based only on the gate being characterized. Determining when an output switches is based on the switching of the connected gate(s). Therefore, the switching voltage is not recorded as a gate by gate specification, but determined as an average switching voltage for a particular gate library.

3.1.4 Transition Waveforms

Perhaps the most important characterization is that of the transition waveforms. When a given gate input transitions, there is a combination of switching and capacitively injected current through the terminals of the gate. This current is recorded and later used along with the switching information to form piece-wise linear (PWL) current sources which represent the active components of the gate in a substrate noise simulation.

A gate is defined as having an internal state if it's outputs are dependent on the current inputs as well as previous inputs. The DMM presented in [10] did not account for the internal states of gates. For circuits where the only gates

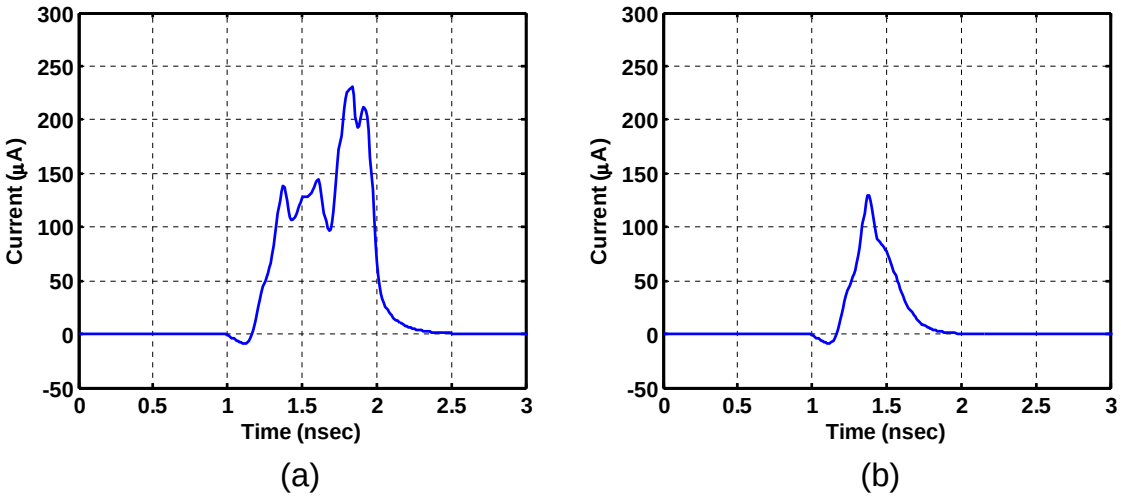

Figure 3.4: Supply current of a flip-flop for a given input pattern when the output (a) changes state, and (b) remains the same.

with internal states are a small number of flip-flops, and most of the logic is combinational, this is an acceptable approximation. However, in logic circuits where many of the blocks have multiple internal states, the differences in injected substrate noise from different states is too significant to ignore. For NCL, multiple states exist for most of the logic gates to enable the passing of NULL and DATA values.

To demonstrate the concept of internal states changing transition waveforms, a simple flip-flop is considered. The simulated supply current is compared for the same input transitions, but with different cell states. The current waveforms are shown in Fig. 3.4. The waveform in Fig. 3.4(a) shows a larger supply current waveform when the inputs and previous state cause a change in the output state. The waveform in Fig. 3.4(b) shows the supply current when the inputs and previous state cause no change in the output state.

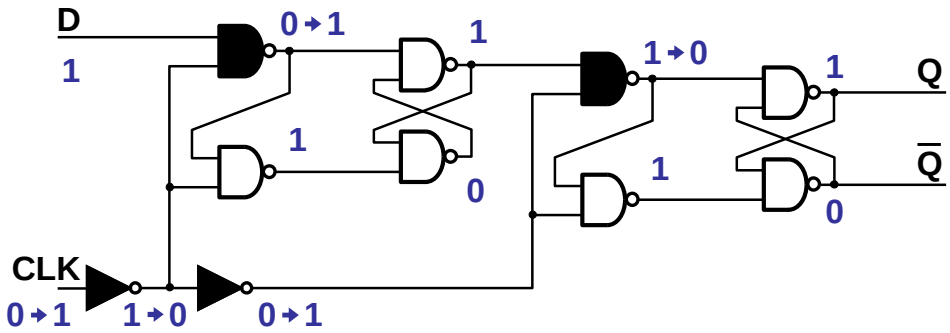

Figure 3.5: Internal logic branches switching when a D-type flip-flop does not change states.

The differences in the two waveforms are due to the multiple logic branches in the flip-flop. It can be observed from Figs. 3.4(a) and (b) that the peak current flow is the same at approximately 1.4ns. This initial current peak is due to logic branches switching in response to the incoming clock. The following current peaks found in Fig. 3.4(a) but absent from Figs. 3.4(b) are caused by the output and other internal branches switching as the flip-flop changes states.

In order to visually show the internal branches switching, a combinational logic implementation of a D-type flip-flop is examined. Fig. 3.5 shows the implementation of a D-type flip-flop with its internal state remaining the same after input switching events. The logic levels of each node are shown, and the internal gates, or branches, which switch are shadowed. Fig. 3.6 shows the implementation of a D-type flip-flop with its internal state changing after input switching events. The logic levels and internal branches switching are shown. In this example, the differences in transition waveforms are caused by the output logic branches switching when the internal state changes.

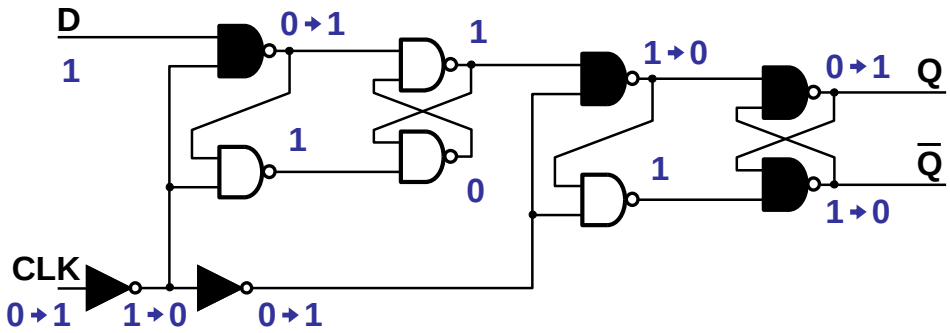

Figure 3.6: Internal logic branches switching when a D-type flip-flop changes states.

Since the internal state can change the transition waveforms, each gate is characterized not only for all possible input switching conditions, but for all possible input switching with different initial states. All of this switching information is stored in files and used in the simulation flow (described in Section 3.2).

Internal states for gates must be clearly defined in this methodology. To keep track of this information, a state map of each gate is created during characterization. The state map defines the internal state that the gate is in, based on its current outputs, and previous input values. The majority of internal states are defined only by the outputs. The previous input values are used in situations where the output values are identical for different internal states. The state map also helps to signal when the gate has entered an undefined state. There are some situations where the gate simulation may generate output or previous input values which do not describe any characterized state of the gate. In these cases, the state map adds some error checking to the simulation flow.

The waveforms are saved in constant time increments, with a fixed time length set in the characterization script. This time length must be long enough for all

gates in the library to settle. The length of all gate waveforms must be the same after the characterization. During the simulation flow, the lengths are truncated to the minimum allowable for a particular energy loss, as described in [21]. Performing the truncation later allows for different energy losses to be selected for different simulations, without having to re-characterize the standard cell library. It also alleviates the need to save the length of each truncated waveform. The disadvantages are larger file sizes from the excess data and a slight addition to the simulation flow time because of the truncation.

The time increments with which the waveforms are saved can also be adjusted. This change of resolution increases the file size, but also increases the accuracy of the simulation. The effect of increased file size is actually two-fold. Increasing the transition waveform files is only the first part. If the resolution is increased, then over the same simulation time, the final PWL files describing the gate's switching noise will be proportionally increased. If the length of these PWL files are too long, and the number of gates too large, Spectre may run out of memory when trying to load them. The increased resolution allows for the saving of higher frequency noise which will not be seen in simulations with a lower resolution. This resolution should be adjusted depending on the frequency range of interest in the substrate noise simulation. For the simulations in this thesis, a 100ps time step gave adequate resolution while allowing for reasonable file sizes.

The gates are all simulated with netlists extracted from layout information. The extraction also includes interconnect coupling capacitances for more accuracy. The bulks of all NMOS transistors are disconnected and added together as another

pin for the gate. This allows for the saving of the current waveforms through the Vdd, Vss, and bulk connections.

It is important to transition the inputs of the gate correctly. A PWL source is used to model the rising or falling waveforms. The exact waveform exciting the gate will depend on the previous gate, and that gate's output load. It is not reasonable to characterize and store this level of detail for all gate configurations. For this reason, the gates in the standard cell library are each simulated, and an averaged output transition waveform is developed that is used for all input transitions.

3.2 Simulation Flow

After the standard cell library characterization is complete, the DMM simulation of any digital block utilizing that library can be performed. This simulation flow is seen in Fig. 3.1, indicated by 2 and 3. These two parts will be described in the following subsection. For more specific information on coding of these parts refer to Appendix A.

3.2.1 Digital Noise Current Generation

The digital noise current generation is the process used to create the DMM of a specific digital block for use in the substrate noise simulation. The digital block is analyzed, and individual gate information combined to form the top level substrate network and the global SDF file. The delay information is used in an HDL

simulation to determine the gate transitions. The input transitions for each gate are saved during the simulation. For cells which have multiple states, the outputs are also saved when there are input transitions.

One issue confronted during the HDL simulation is matching the gate transitions with the transition waveforms. The delay characterization, as described previously, is timed in reference to switching voltages of the gates. Therefore, the transitions occurring in the gate level simulation are ideal changes that happen when the gate input/output would pass the switching voltage. However, the transition waveforms do not begin when the switching voltage is crossed, but as soon as the input begins to change. To account for this difference, an offset is be subtracted from the gate transition times. This time is a constant for each standard cell library characterization, based on the input switching waveforms and when they pass the determined switching voltage.

The input transitions, output values, and current waveform information is then used in a concatenation program. This program selects the correct current waveform to use at the designated input switching times, and concatenates them together for each gate. If the gate has multiple states, it also uses the output information and the state map of that gate to determine the current waveforms.

The output of the digital substrate noise generation is a set of PWL data that is stored in files. These files describe the current waveforms generated by the gates in the block. By linking these files through PWL current sources, the switching currents of the transistor level digital blocks are correctly modeled.

3.2.2 Substrate Noise Simulation

The final substrate noise simulation is performed after the digital noise current generation has been completed. This simulation incorporates the models for the digital gates with the rest of the substrate noise setup. The simulation is performed in the Cadence environment with Spectre. For specific coding and details on the digital model placement script, refer to Appendix B.

Part of the DMM for the digital block is the substrate network. The substrate network created in the digital noise current generation is added into the schematic view with the correct connections.

The equivalent parasitics are also added into the schematic view as part of the digital gate model. The gate transition information of each gate is needed. The input transition information (along with any output information for gates with internal states) is used to determine the equivalent parasitics at different times throughout the simulation. The equivalent parasitics for each gate are averaged over time and placed as constant values.

After the final addition of the PWL current sources, the simulation generates the correct amount of substrate coupling from the modeled digital block to the sensitive circuits modeled at the transistor level. The circuits substrate networks are connected together in the simulation through the common backplane node.

Chapter 4 – Simulation Results

The simulation results from the setups described in Chapter 2 are presented in this chapter. The PRNG simulations are shown for verification between measurement, transistor level simulation, and the improved DMM method. The 8051 microprocessor simulations are presented to verify the use of the DMM in a large and complex circuit which is not easily simulated at the transistor level.

4.1 Pseudo-Random Number Generators

The results for the PRNG test setups are presented in this section. The transistor level simulation subsection compares measurements and transistor level simulations of the PRNGs. This comparison verifies the complete setup of the substrate noise simulation. The digital macro-model subsection compares the transistor level simulation results with simulation results where the PRNGs are replaced with equivalent DMM blocks. This comparison verifies the improved DMM presented in this thesis. Each comparison is performed for the CBL and NCL versions of the PRNGs.

4.1.1 Transistor Level Simulation

The measurements and transistor level simulation results for the CBL PRNGs are shown in Fig. 4.1. The shape of the noise waveforms in simulation are similar to the shape displayed by the measurements. There are two noteable differences between the measurements and the simulations. The first is the difference in waveform shown by the circled region. The second is the size of the voltage peak, pointed to by the arrows. Both of these differences are believed to be connected. There are two local peaks in the circled region of Fig. 4.1(a). The earlier local peak seems to be missing from the circled region of Fig. 4.1(b). This peak is still in Fig. 4.1(b), however it has shifted to the left. The superposition of this peak and the voltage peak coming just before the circled region have formed a single voltage peak which is lower and wider in the simulations. This timing issue is coming from an inability to model the loaded clock generator perfectly in simulation.

The measurement and simulated data are both processed using a fast Fourier transform (FFT) in Matlab. The results of each in the frequency domain are shown in Fig. 4.2. The spectra are similar, with a noise floor peaking around 100 MHz due to package ringing (circled regions). The clock's fundamental frequency and tones are similar, with some higher frequency tones being higher in the measurements.

The measurements and transistor level simulation results for the NCL PRNGs are shown in Fig. 4.3. Both waveforms start out at similar peak voltage values. The measurements shown in Fig. 4.3(a) have more separation between the primary and secondary voltage peaks (circled regions). This difference comes from

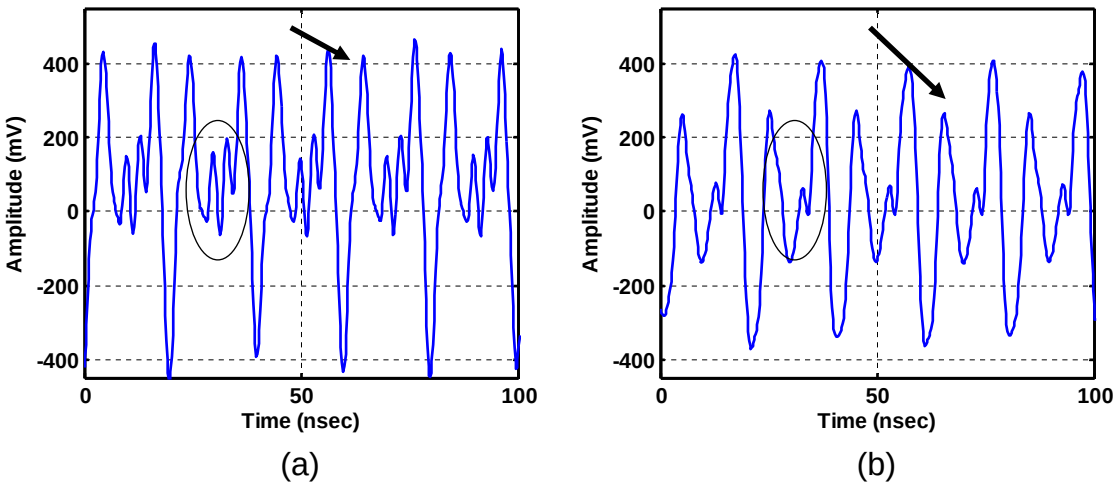

Figure 4.1: Comparison of time domain (a) measurements, and (b) simulations for the CBL PRNGs.

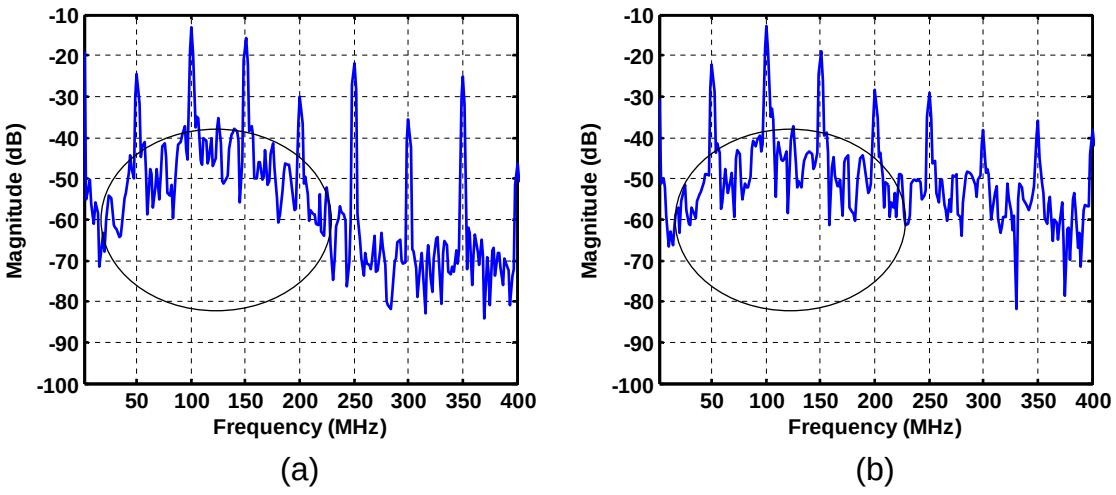

Figure 4.2: Comparison of frequency domain (a) measurements, and (b) simulations for the CBL PRNGs.

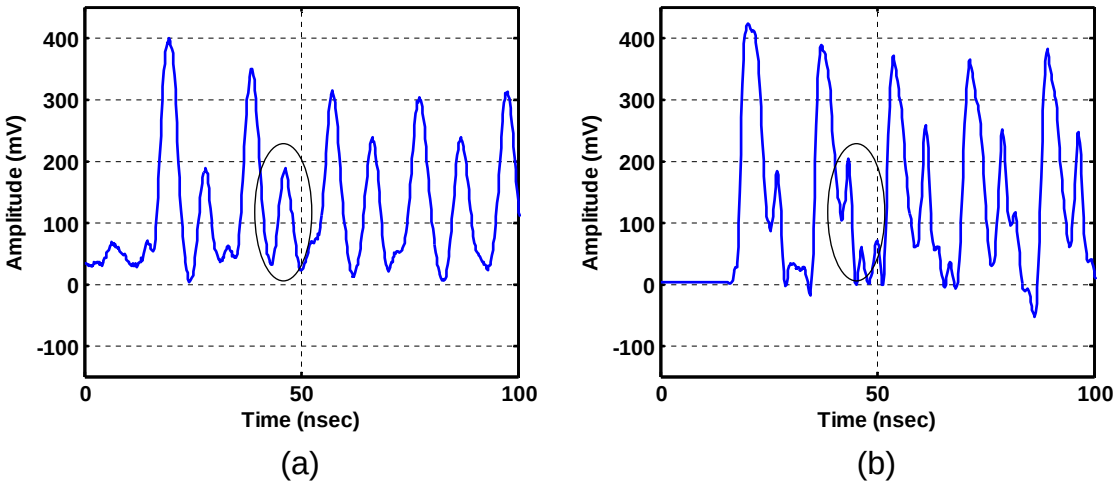

Figure 4.3: Comparison of time domain (a) measurements, and (b) simulations for the NCL PRNGs.

differences in the NCL gates' delays in simulation and measurement. The switching events between gates are occurring faster in the simulation (possibly due to missing parasitics) and are causing the substrate noise injections to happen closer together. Further evidence of this delay difference is noted in the rate at which the voltage peaks occur. Over the same time period, there are nine voltage peaks in the measurement, compared to 10 voltage peaks in the simulation.

An expanded time scale comparison of measurements and transistor level simulation results for the NCL PRNGs is shown in Fig. 4.4. This figure shows another difference between the measurements and simulations. The measurement voltage peaks decay down significantly, while the simulation peaks stay at approximately the same level. The repetitive placement of the PRNG blocks is responsible for this behavior in measurements. The NCL PRNG blocks have no global clock, and only the reset signal keeps them synchronized. When the reset signal first occurs,

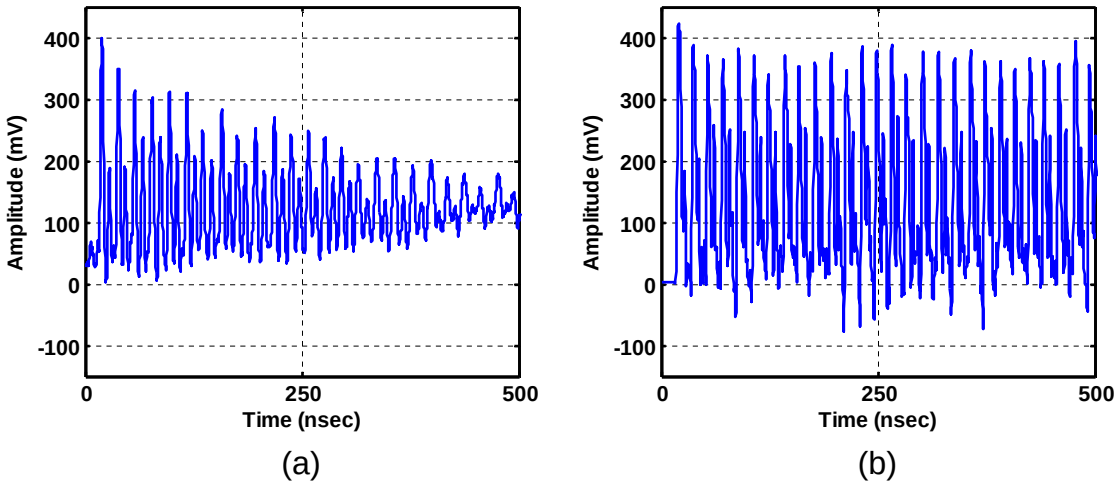

Figure 4.4: Expanded time scale comparison of (a) measurements, and (b) simulations for the NCL PRNGs.

all blocks are injecting noise with the same pattern. However, each block is not identical. Mismatch between the NCL PRNGs cause the blocks to run at different equivalent operating speeds. As time progresses, each block runs at a similar but slightly different frequency, resulting in less addition of switching current peaks. In simulation there is no variation in the NCL PRNG blocks, so the current peaks, and subsequently substrate noise peaks, always add.

To demonstrate this concept, test simulations are performed. A simulation setup with 5 NCL PRNG blocks is constructed using simple supply parasitics and substrate networks. Fig. 4.5 compares two simulations of this setup. In Fig. 4.5(a) the voltage on the backplane node is shown when the Vth of the transistors is held constant. In Fig. 4.5(b) the voltage on the backplane node is shown when the Vth is varied for each NCL PRNG block. The varied Vth simulation results show similar voltage peaks to those in the constant Vth simulation initially. However,

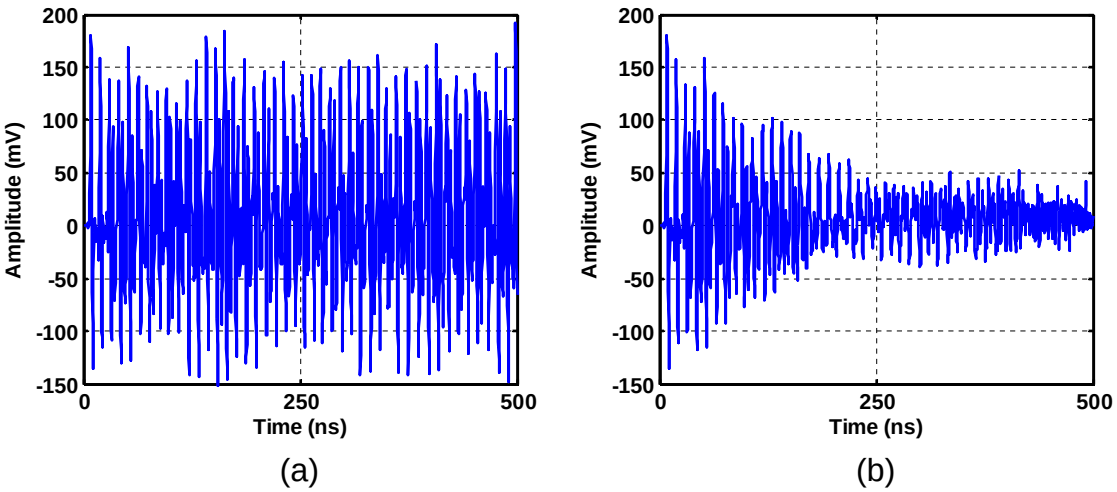

Figure 4.5: Comparison of a test setup with the NCL PRNGs with (a) constant Vth, and (b) varied Vth.

the peaks quickly degrade as the NCL PRNG blocks fall out of synchronization with each other in the varied Vth simulation.

Again, FFTs of the measurement and simulated data are performed in Matlab. The results of each in the frequency domain are shown in Fig. 4.6. As would be expected from the measurement results, there are significant differences in the FFTs. Both FFTs do however display peaks close to the equivalent operating speed of 50 MHz (shown as circled regions) and it's harmonics. The magnitude of the peaks in the FFT of the simulation results is at higher levels as expected from the time domain results.

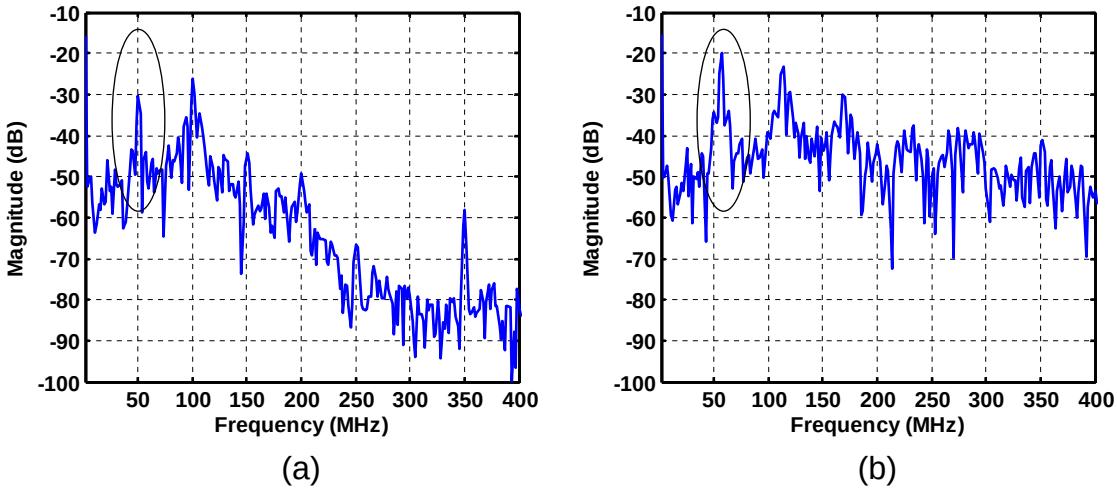

Figure 4.6: Comparison of frequency domain (a) measurements, and (b) simulations for the NCL PRNGs.

4.1.2 Digital Macro-Model

The transistor level and DMM simulation results for the CBL PRNGs are shown in Fig. 4.7. The voltage peaks are slightly higher in the DMM simulations than in the transistor level simulations. The noise shapes and timing compare closely between the simulations.

The transistor level and DMM simulation results for the NCL PRNGs are shown in Fig. 4.8. The DMM simulation's first voltage peak is similar in magnitude to that of the transistor level simulation's first voltage peak. All other voltage peaks are slightly lower in magnitude in the DMM simulation. The difference comes from the delay modeling in the DMM simulations. The NCL block's have no conditional delay paths defined in their HDL descriptions. The generalized delay paths cause the switching activity peaks to occur closer together in the gate level simulation.

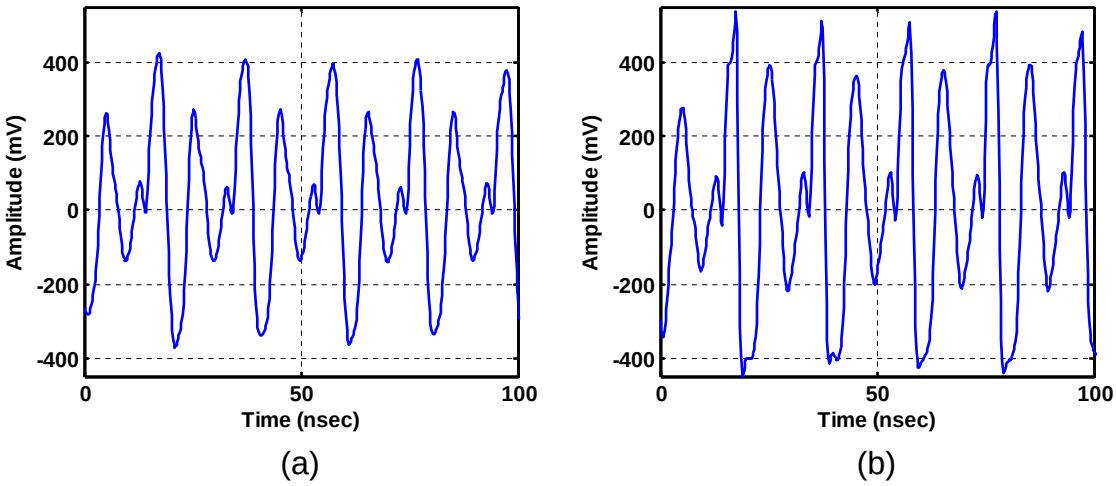

Figure 4.7: Comparison of time domain (a) transistor level simulations, and (b) DMM simulations for the CBL PRNGs.

The difference in the time separation of the voltage peaks is shown by the circled regions in Fig. 4.8. Further evidence of this difference in delay modeling is seen by the rate at which the voltage peaks occur. Over the same time period, the DMM simulation shows 6 sets of the same double peaked waveform, the transistor level simulation shows only 5.

4.2 8051 Microprocessor

The results for the 8051 microprocessor test setups are presented in this section. The equivalent DMM simulations of the PRNGs show good agreement with transistor level simulations. As a larger and more complex simulation, the 8051 microprocessor will verify the improved DMM methodology for a more realistic circuit implementation. Comparisons between the measurements and DMM simulations

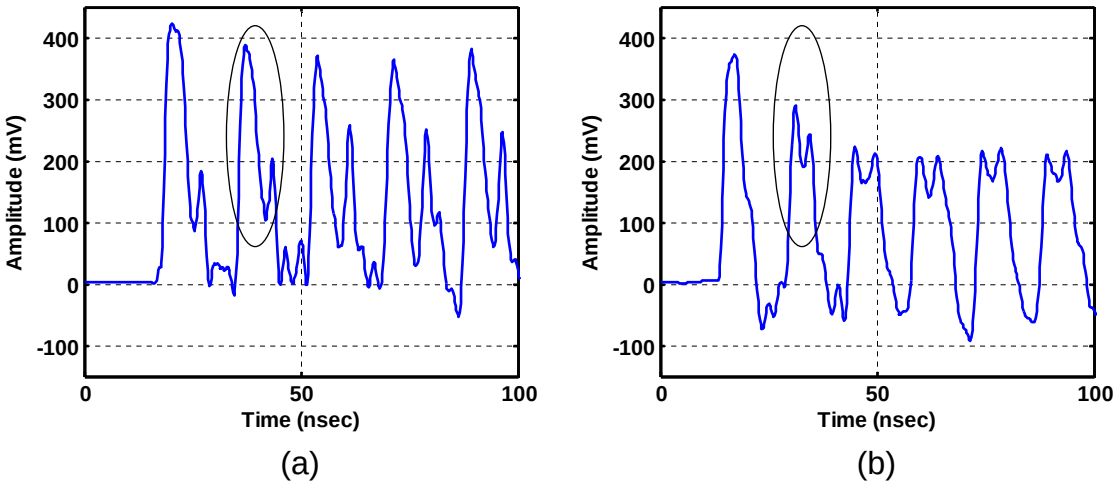

Figure 4.8: Comparison of time domain (a) transistor level simulations, and (b) DMM simulations for the NCL PRNGs.

are presented for both the CBL and NCL cores.

The measured and simulated results for the CBL core are shown in Fig. 4.9. Good agreement between simulations and measurements is seen. The simulations have similar noise waveforms as those seen in measurements. The peak-to-peak values in the simulation are similar but slightly higher for some sections of the waveform than the measurements.

The measured and simulated results for the NCL core are shown in Fig. 4.10. Again, the simulations are in good agreement with the measurements. The noise waveforms and peak-to-peak voltages are similar. The measurements for the NCL core were taken without averaging. Since the transition times can vary significantly over long time intervals with clockless logic, any averaging would corrupt the noise measurements by averaging misaligned voltage peaks. However, without averaging, noise from the measurement setup cannot be removed. This noise is evident in the

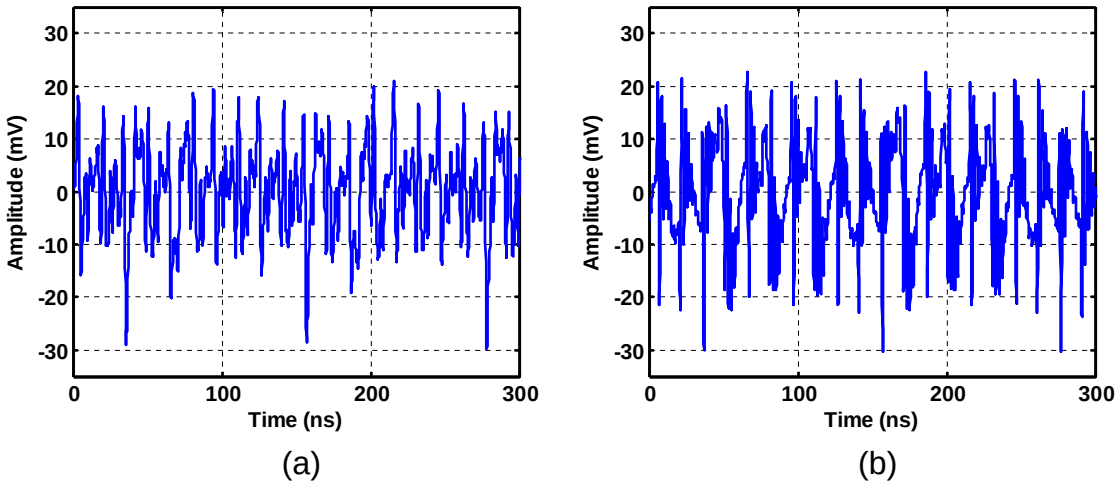

Figure 4.9: Comparison of time domain (a) measurements, and (b) DMM simulations for the CBL 8051 microprocessor core.

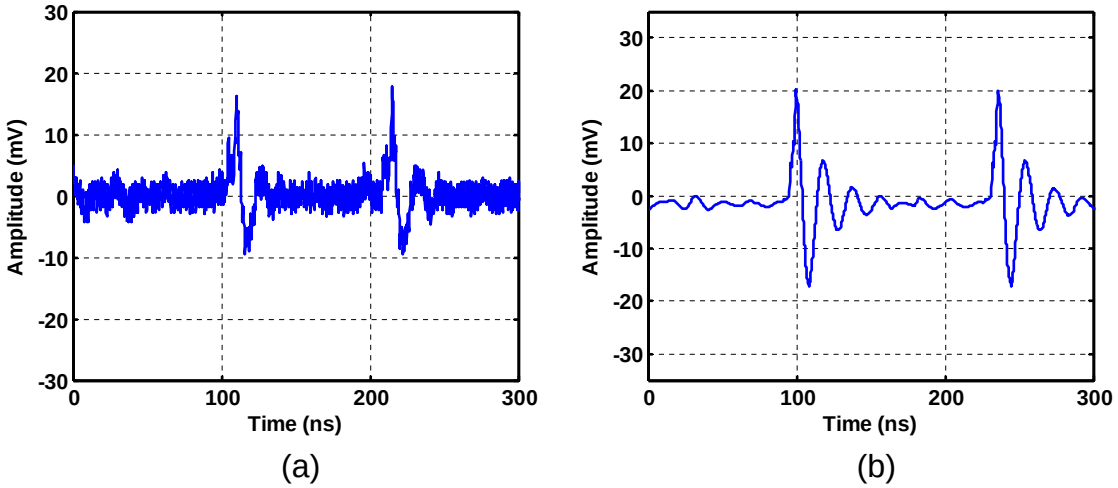

Figure 4.10: Comparison of time domain (a) measurements, and (b) DMM simulations for the NCL 8051 microprocessor core.

measurement results of the NCL core.

Chapter 5 – Critical Aspects of Substrate Noise Simulation

Substrate noise simulations can be very difficult to setup. There are many different parasitics and non-idealities that may have to be included in a simulation for matching with measurements. In this chapter, various aspects of accurate substrate noise simulations will be addressed. The CBL PRNG simulations in Chapter 4 are used as a baseline. By adjusting a single parameter at a time, the effects on the overall accuracy can be carefully examined.

5.1 On-Chip Buffers

The core digital blocks may not be the only circuits injecting noise into the substrate. Clock and data lines are often buffered before coming on-chip or going off-chip. These IO buffers can be very large digital circuits with substantial current switching. In a substrate noise simulation, the inclusion of the noise from these circuits is very important.

On-chip buffers were used in the CBL PRNG test setup to buffer the clock and reset lines coming from the PCB. To analyze the effect of the substrate noise generated by these buffers, the CBL PRNG simulation is performed without including the substrate network for the buffers. A comparison in the time domain between the CBL PRNG transistor level simulations is shown in Figs. 5.1(a) with and (b)

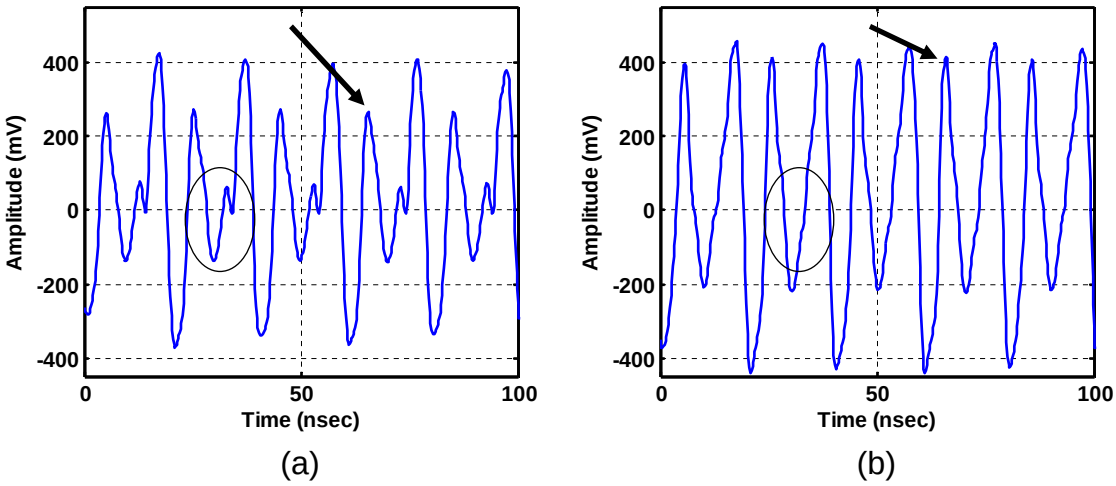

Figure 5.1: Comparison between (a) the complete simulation of the CBL PRNGs, and (b) the same simulation with the IO buffers' substrate network removed.

without the substrate noise from the IO buffers included.

The results without the IO buffers' substrate network in Fig. 5.1(b) show two significant differences from the complete simulation of Fig. 5.1(a). The first difference is the missing smaller peak, shown by the circled regions. The second is the larger secondary peak, pointed to by the arrows. Since the IO buffers' substrate noise is directly proportional to the incoming clock, this example shows the significance of modeling the incoming clock and IO buffers in substrate noise simulations. It also supports the earlier claim that differences between measurement and complete CBL PRNG simulations are due to the clock waveform.

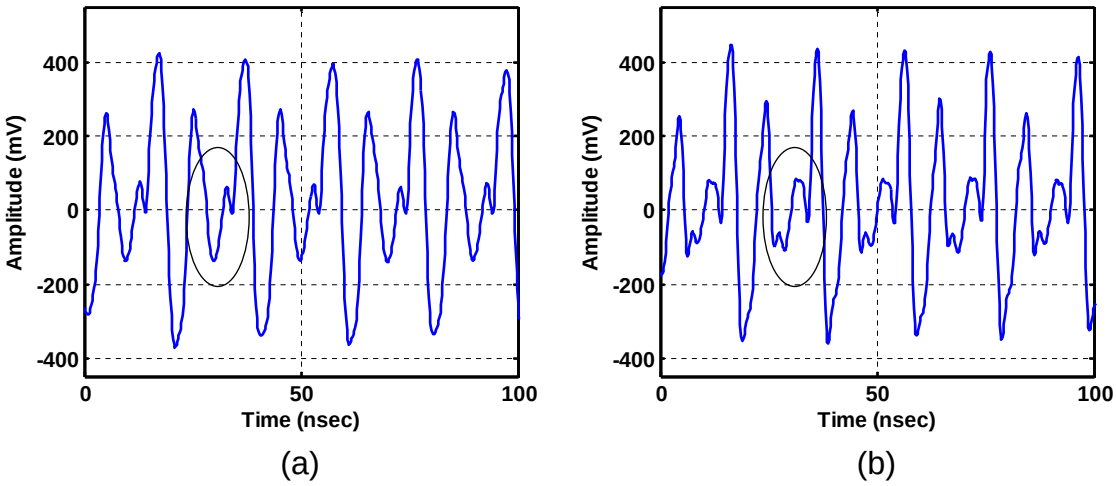

Figure 5.2: Comparison between (a) the complete simulation of the CBL PRNGs, and (b) the same simulation with coupling capacitances between interconnects removed.

5.2 Capacitance Between Interconnects

One of the most common parasitic extractions is the extraction of capacitive coupling between different interconnect layers. Parasitic interconnect capacitances can have significant effects on simulation results. To analyze how much this type of parasitic capacitance changes substrate noise simulations, the CBL PRNG simulation is performed again without the capacitance extracted between interconnect layers. A comparison in the time domain between the the CBL PRNG transistor level simulations is shown in Figs. 5.2(a) with and (b) without the interconnect coupling capacitances.

The noise waveform of the simulation without parasitic interconnect capacitances, Fig. 5.2(b), has one significant difference from the complete simulation in Fig. 5.2(a). The noise shape is changed within the circled regions of Fig. 5.2. The

voltage peaks are slightly sharper in Fig. 5.2(b), but remain at similar levels as the voltage peaks in Fig. 5.2(a). This example shows the capacitance between interconnects affects substrate noise simulations, but only to a limited extend for this test setup and this clock frequency.

5.3 Capacitance to Substrate

Parasitic capacitors are also present between the interconnect layers and the substrate. This coupling can be much weaker, but can add up cumulatively for very large areas. Structures such as bond pads, and large low level metal traces can add up not only in area capacitance, but also in fringing capacitance, to form a path for high frequency noise injection in the substrate.

For the CBL PRNG simulations, the bond pads and probe pads were the only structures large enough to form any significant capacitance to the substrate. To analyze the effects of modeling these structures, a simulation of the CBL PRNGs is performed again without the extraction of the bond pad and probe pad coupling to substrate. Fig. 5.3 shows a comparison in the time domain between the CBL PRNG simulations with and without the pad coupling capacitances.

The results show similar waveforms for both comparison cases. The substrate noise in this setup is neither coupling through, or being dampened by, the bond and probe pad capacitances. Substrate noise injected by other sources is much larger for this test setup and clock frequency.

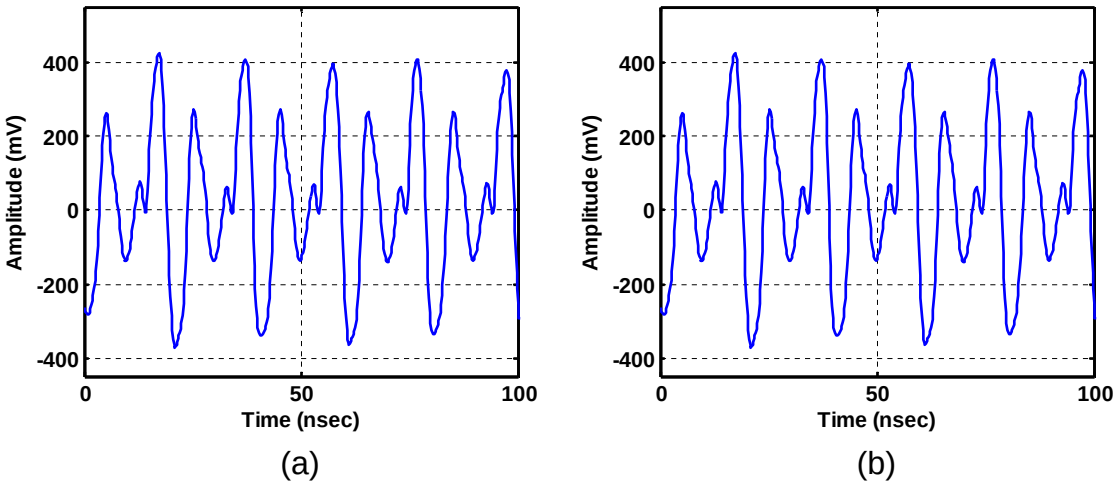

Figure 5.3: Comparison between (a) the complete simulation of the CBL PRNGs, and (b) the same simulation with the pad capacitances to substrate removed.

5.4 High Frequency Substrate Model

As described in Section 2.3, the p-type substrate is modeled as a purely resistive network. This network is found from the use of a particular version of EPIC. By using a different version of EPIC, a higher frequency substrate model can be obtained. For frequencies up to 5GHz, the substrate can be modeled with the structure shown in Fig. 5.4 [22]. The permittivity of the substrate is required for calculating this model. The permittivity of silicon is given as:

$$\epsilon_{Si} = \epsilon_0 \, \epsilon_r = 1.0359 \times 10^{-10} F/m \tag{5.1}$$

To test the effects of using the higher frequency substrate model on simulations, the CBL PRNG simulation is repeated with the substrate network replaced with the higher frequency model. The results of the CBL PRNG transistor level

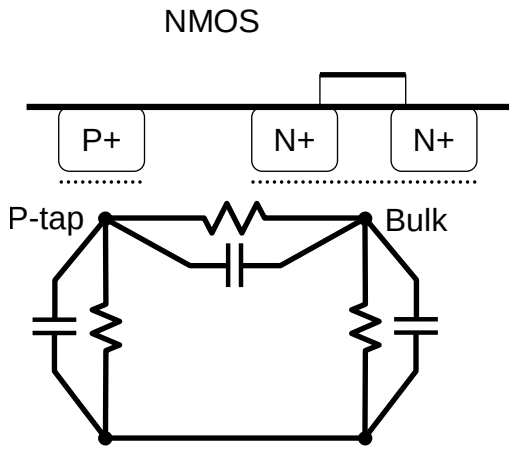

Figure 5.4: The substrate π-network incorporating higher frequency effects.

simulation with the purely resistive network and with the high frequency network are shown in Fig. 5.5.

The results show no noticeable difference with the use of the higher frequency substrate model. For this test setup and clock frequency, the use of a purely resistive substrate model is both sufficient and accurate.

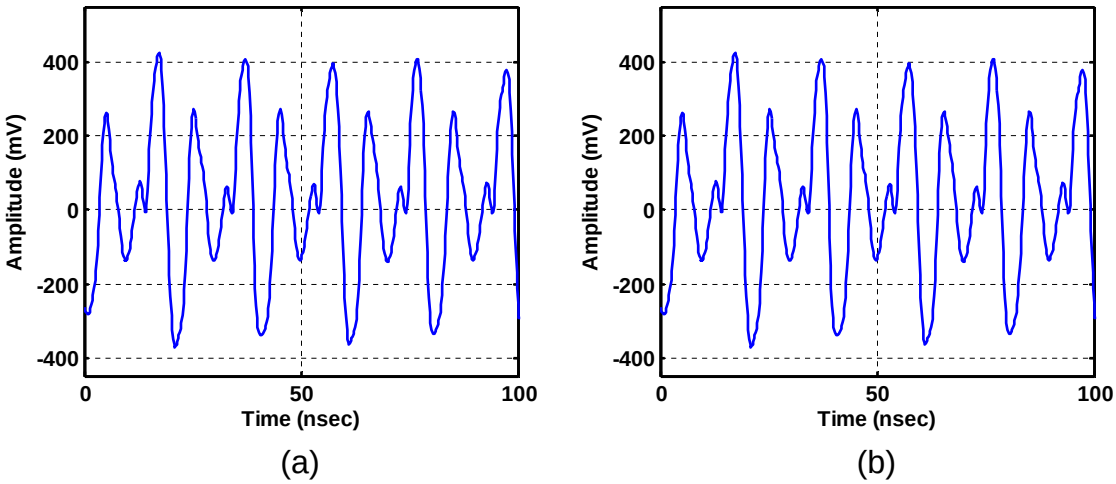

Figure 5.5: Comparison between (a) the complete simulation of the CBL PRNGs, and (b) the same simulation with the high frequency substrate model.

Chapter 6 – Conclusions

The substrate noise simulation methods presented in this thesis have many advantages. This chapter summarizes the advantages and accomplishments of this work. Then a discussion of some lessons learned is presented. Finally, issues and future work with the methods presented are described.

6.1 Summary

In this thesis, the setup of substrate noise simulations is discussed. Several test setups are presented which modeled microchips fabricated in TSMC's 0.25μm logic process. The test setups contained both clocked and unclocked digital logic circuits.

A method for modeling the digital block in a substrate noise simulation is presented. This method is presented not only as a way to overcome some of the issues with a full transistor level simulation, but also as a way to perform pre-layout substrate noise simulations. The method is described in three steps: characterization, digital noise current generation, and substrate noise simulation. The characterization step only needs to be completed once per gate library. The digital current generation needs to be performed once per digital block setup. The final substrate noise simulation is then performed with an equivalent digital model which runs faster and more efficiently in the simulator.

Table 6.1: Execution time of different substrate noise simulations.

	Transistor Level	DMM
CBL PRNG	33h, 52m	14h, 57m
NCL PRNG	49h, 21m	11h, 3m
8051 CBL Core	-	702h, 13m
8051 NCL Core	-	531h, 2m

Table 6.1 shows the execution times of the substrate noise simulations presented in chapter 4 of this thesis. The PRNG simulations show a definitive advantage in execution time with the use of the DMM method. The NCL PRNG simulations show an increase in simulation speed by over 4 times. The CBL PRNG simulations show an increase in simulation speed by over 2 times. The CBL PRNG's DMM simulations are slowed significantly by the need to simulate the switching clock buffers at the transistor level. All PRNG circuits are run for 1μs transient simulations.

The 8051 microprocessor will not simulate at the transistor level due to convergence issues for such a large circuit. The DMMs for the 8051 microprocessor circuits are run for 15μs transient simulations.

The digital macro modeling method presented in this thesis is not only able to simulate large static CMOS blocks, but is general and applicable to different types of digital logic, such as asynchronous NCL. In fact, this is the first method to address noise generation from asynchronous blocks. Validations with measured results demonstrate the accuracy and efficacy of the method.

6.2 Lessons Learned

External parasitics are important for an accurate prediction of substrate noise. The off-chip parasitics are often more important than the on-chip parasitics. The package, bond wires, and PCB parasitics may all include large inductances and capacitances. The series inductances will increase the magnitude of the supply bounce seen on-chip when switching events occur. These inductances, along with parasitic and decoupling capacitances, will also influence the ringing on the supplies after a switching event. All of this activity on the supply lines will couple through the NMOS or PMOS taps to the substrate, and form the majority of the substrate noise. The one exception to off-chip parasitics having more influence on simulation results than on-chip parasitics is the on-chip trace resistances. If critical lines are not made sufficiently wide, but are considerably long, the on-chip series resistance can easily grow larger than any off-chip resistances.

The exact test setup has to be accurately modeled. The input sources used in the lab drastically change the substrate noise waveforms. Knowing which equipment and connections were used, or are going to be used, for a particular substrate noise measurement is essential in getting simulations that match. Many of the manuals for the different test equipment have sections describing the properties of the signals they generate, or the equivalent models that should be used when they are connected. It is important to decide which models and characteristics are critical, and which are unnecessary at the frequencies of interest in the simulation.

Ground connections of other on-chip circuits make a significant difference in

substrate noise. The simulation of the 8051 microprocessor cores produces unmatched results when the other substrate connections on the die are not modeled. In substrate noise simulations it is a common mistake to include only the circuits of interest in a setup. However, when there are multiple circuits on the same die, and their pins are connected, these can provide pathes for noise to exit the substrate. Noise currents injected into the substrate can use these alternative paths to couple off-chip instead of coupling to the sensitive analog nodes. This will manifest as different peaks and ringing frequencies in the simulation than those seen in measurements.

An exact match of clockless circuits in simulation is difficult. There are many modeling parameters in simulation that can not be perfectly matched with actual silicon. Random noise or variations in process, voltage, and temperature can cause differences from simulation. One benefit of simulating clocked circuits is the ability of the clock to 're-align' simulated and measured waveforms. With clockless logic, even small differences in modeling can lead to larger variations over long periods of time through accumulation. This is particularly demonstrated in the differences between the test simulations for the NCL PRNGs, shown in Fig. 4.5.

6.3 Future Work

The parasitic extractions used in this thesis were done with the help of modified Diva extraction rules. These rules are relatively simple to modify. However, they are no longer supported by Cadence or commonly used in industry. To make future

substrate noise simulation setups more up to date and applicable to industry use, the extraction should be done with a more commonly used tool. Assura is the currently supported parasitic extraction method from Cadence. This would be a logical choice for developing adjusted extraction rules to be used for substrate noise simulation. The rules for Assura are very similar to the rules for Diva, so the method of adjustment may be the same. The exact adjustments made to the Diva rules are explained in more depth in Appendix B. These adjustment were done manually to the rules as well, but a method for automatically adjusting the rules would be very convenient for easy setup with new processes.

When using Silencer!, the first step is to locate the substrate contacts. This is either done manually, by the user, or with the use of Silencer!'s built in contact location function. However, the location of substrate contacts is not trivial. With large layouts, a function is needed in order to automate the drawing of each contact. Layouts are also done in a large variety of ways, making the structures defined as substrate contacts sometimes oddly shaped. A tool called the Contact Locator has been written in Skill code to help in the location of substrate contacts (see Appendix B). The tool helps the user by allowing flexible specifications for the contact finding algorithms, but does not find all contacts automatically. There is a need for future work in developing algorithms that will automatically adjust the contact finding functions based on the layout. These adjustments should then be added into Silencer!'s built in contact location function.

The present DMM flow is a two simulation process. The first simulation is done at the gate level to generate the digital switching patterns. The second

simulation is the use of the equivalent DMM model in a transistor level simulation of the substrate noise sensitive circuitry. Each of these simulations is currently done with different tools. The gate level simulation is done with Modelsim. The transistor level simulation is done with Spectre in the Cadence environment. The integration of the gate level simulation to the Cadence environment would be useful in creating a seamless substrate noise simulation tool. There are some advantages to having the DMM flow done with two separate simulations. If concerned with the effects of one constant digital block on different or changing analog blocks, the first simulation needs to be done only once, and the second simulation is then repeated. However, if this type of iterative simulation process is not needed, it is better to integrate the two simulations as one mixed-signal simulation. Future work could include the automated creation of mixed-signal models for the digital logic gates.

There are several issues in the present DMMs which should be addressed in any continuation of this work. The loading of digital gates affects not only the delays of the propagating digital signals, but also the shape of the noise being injected into the substrate. Unfortunately, this substantially complicates the estimation of the substrate noise waveforms generated by each digital gate. With the current version of the DMM, the digital blocks have a simple load and the waveforms generated by that load are used for every instance of that gate, no matter what the actual load is. The most accurate method would be to record the switching transition waveforms not only for every combination of inputs and internal states, but also for every combination of output loads. However, the number of possible loading

combinations becomes very large with multiple outputs and large fan outs. As an example, one of the largest digital blocks characterized is a Theseus Logic block called THFAX0. This block has 6 inputs, 4 outputs, and 9 internal states. With the current method for finding switching waveforms, this leads to 36,864 different combinations. If the load must now be varied, this number of combinations must be recorded for each variation. With 4 outputs for which the loads can be varied, even a small number of maximum fan outs will lead to unacceptable costs in memory and characterization time. Future work could determine a way to account for the changes in the substrate waveforms due to loading without the use of brute force characterization.

The equivalent parasitics of the digital gates is another issue with the present DMM flow which should be addressed in any future work. The parasitic model incorporates a simple resistor in series with a capacitor between the rails. The values for these are found through simulation of each digital gate at different steady states. When the switching information for a large digital block is determined, the equivalent parasitics are calculated as an average of the steady state parasitics for the gate. A time varying model for the changing resistance and capacitance would allow for more accurate parasitics between the supply rails at specific times.

At present, the DMM flow combines the substrate networks generated locally for each gate to form the complete substrate network for the digital block being tested. One problem with this, however, is that the p-tap contact for each individual gate (shown in Fig. 6.1), will overlap with the p-tap contacts from adjacent gates in the layout, as illustrated in Fig. 6.2. This creates differences between the

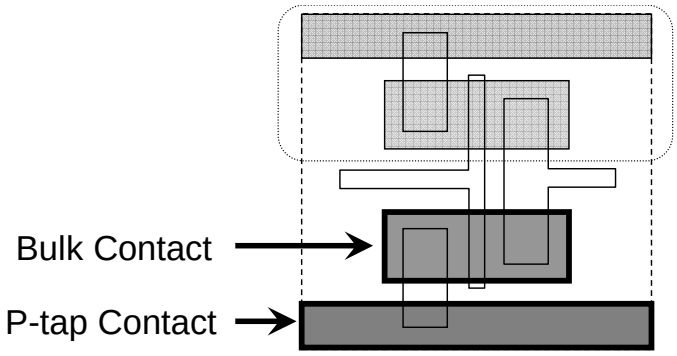

Figure 6.1: Substrate contacts defined for a single gate.

actual substrate network of the overall digital block and the substrate network formed by the combination of the gates' local substrate networks. The combination network overestimates both the perimeter and area of the p-tap contacts, which leads to lower resistances to backplane for the on-chip digital ground node. Future work should include a method to account for this problem, and combine the local substrate networks in a way which yields a complete network equivalent to post-layout substrate extraction of the digital block.

Finally, the simulation benefits of using the DMM can be significantly improved with a modification to the PWL files created by the digital noise current generation step. The PWL files are currently a concatenation of the transition waveforms for each gate. The transition waveforms are formed with data at set time steps. The time steps allow for ease in the concatenation, but cause Spectre to evaluate at each time point when the PWL file is used in simulation. If the PWL files were modified after creation to allow for varying time steps, but retain an equivalent noise current shape, the simulation speed benefits of the DMM could be increased even more.

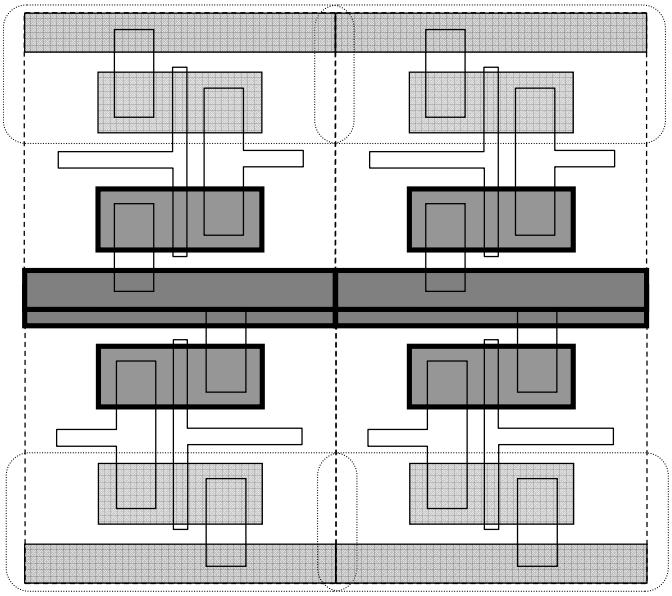

Figure 6.2: Example of defined substrate contacts for individual gates overlapping in the layout.

Bibliography

[1] K. Fant and S. Brandt, "NULL convention logic: A complete and consistent logic for asynchronous digital circuit synthesis," in *Proc. International Conference on Application Specific Systems, Architectures, and Processors*, Aug. 1996, pp. 261–273.

[2] J. Le, C. Hanken, M. Held, M. Hagedorn, K. Mayaram, and T. S. Fiez, "Comparison and impact of substrate noise generated by clocked and clockless digital circuitry," in *Proc. IEEE CICC*, Sept. 2006, pp. 105–108.

[3] *Assura Physical Verification User Guide*, Cadence Design Systems, Inc., Sept. 2006.

[4] P. Birrer, T. S. Fiez, and K. Mayaram, "Silencer!: A tool for substrate noise coupling analysis," in *Proc. IEEE International SOC Conference*, Sept. 2004, pp. 105–108.

[5] S. Mitra, R. A. Rutenbar, L. R. Carley, and D. J. Allstot, "A methodology for rapid estimation of substrate-coupled switching noise," in *Proc. IEEE CICC*, May 1995, pp. 129–132.

[6] E. Charbon, P. Miliozzi, L. P. Carloni, A. Ferrari, and A. Sangiovanni-Vincentelli, "Modeling digital substrate noise injection in mixed-signal IC's," *IEEE Trans. Computer-Aided Design*, vol. 18, pp. 301–310, Mar. 1999.

[7] M. Nagata, T. Morie, and A. Iwata, "Modeling substrate noise generation in CMOS digital integrated circuits," in *Proc. IEEE CICC*, May 2002, pp. 501–504.

[8] M. Badaroglu, G. Van der Plas, P. Wambacq, S. Donnay, G. G. E. Gielen, and H. J. De Man, "SWAN: High-level simulation methodology for digital substrate noise generation," *IEEE Trans. VLSI*, vol. 14, pp. 23–33, Jan. 2006.

[9] H. Habal, "Accurate and efficient simulation of synchronous digital switching noise in systems on a chip," Master's thesis, Oregon State University, July 2004.

[10] M. Held, "A methodology for efficient substrate noise estimation for large scale digital circuits in mixed signal SoC's," Master's thesis, Oregon State University, June 2005.

[11] J. Le, "Comparison and impact of substrate noise due to clocked and clockless circuitry," Master's thesis, Oregon State University, Jan. 2007.

[12] M. van Heijningen, J. Compiet, P. Wambacq, S. Donnay, M. G. E. Engels, and I. Bolsens, "Analysis and experimental verification of digital substrate noise generation for epi-type substrates," *IEEE J. Solid-State Circuits*, vol. 35, pp. 1002–1008, July 2000.

[13] B. Owens, "Simulation, measurement, and suppression of digital noise in mixed-signal integrated circuits," Master's thesis, Oregon State University, Sept. 2003.

[14] R. Schmid, "Measuring board parasitics in high-speed analog design," Texas Instruments, Tech. Rep., Aug. 2003.

[15] D. K. Su, M. J. Loinaz, S. Masui, and B. A. Wooley, "Experimental results and modeling techniques for substrate noise in mixed-signal integrated circuits," *IEEE J. Solid-State Circuits*, vol. 28, pp. 420–430, Apr. 1993.

[16] N. K. Verghese, T. J. Schmerbeck, and D. J. Allstot, *Simulation Techniques and Solutions for Mixed-Signal Coupling in Integrated Circuits.* Kluwer Academic Publishers, 1995.

[17] I. L. Wemple and A. T. Yang, "Integrated circuit substrate coupling models based on Voronoi tessellation," *IEEE Trans. Computer-Aided Design*, vol. 14, pp. 1459–1469, Dec. 1995.

[18] R. Gharpurey and R. G. Meyer, "Modeling and analysis of substrate coupling in integrated circuits," *IEEE J. Solid-State Circuits*, vol. 31, pp. 344–353, Mar. 1996.

[19] J. P. Costa, M. Chou, and L. M. Silveira, "Efficient techniques for accurate modeling and simulation of substrate coupling in mixed-signal IC's," *IEEE Trans. Computer-Aided Design*, vol. 18, pp. 597–607, May 1999.

[20] C. Xu, "Epic: A program for extraction of the resistance and capacitance of substrate with the green's function method," ECE Dept., Oregon State Univ., Tech. Rep., 2002.

[21] H. Habal, K. Mayaram, and T. S. Fiez, "Accurate and efficient simulation of synchronous digital switching noise in systems on a chip," *IEEE Trans. VLSI*, vol. 13, pp. 330–338, Mar. 2005.

[22] C. Xu, T. S. Fiez, and K. Mayaram, "High frequency lumped element models for substrate noise coupling," in *Proc. International Workshop on BMAS*, Oct. 2003, pp. 47–50.

APPENDICES

Appendix A – Using the DMM Method

This appendix describes in more detail the actual coding and implementation of the digital macro-modeling method. This version of the DMM is an adjusted version of the previous DMM presented by Martin Held here at Oregon State University. The code was adjusted significantly from the previous version, with more error checking and gate characterization added.

In this appendix, an overview of the file structure and flow is presented. A hierarchical description of how the DMM is organized is shown, describing where files are located. A functional description is then presented, describing the DMM more from an execution standpoint. The characterization part of the DMM is described in one section, and the digital current generation in another. After this, the steps for creating a new setup are described. Finally, an example of adjusting the flow (specifically the digital current generation) is shown. The DMM can also be used to help model the stimulus to digital blocks instead of the current through them. This section will describe the setup which was used to generate voltage source inputs for simulating the 8051 ram at the transistor level.

Disclaimer: I have tried very hard to make this code as bug free as possible, however I have only had a certain number of cases with which to test it. When setting up anything new with the code, always double check a simple case and be wary of bugs. There is always the possibility that some bug which was not obvious

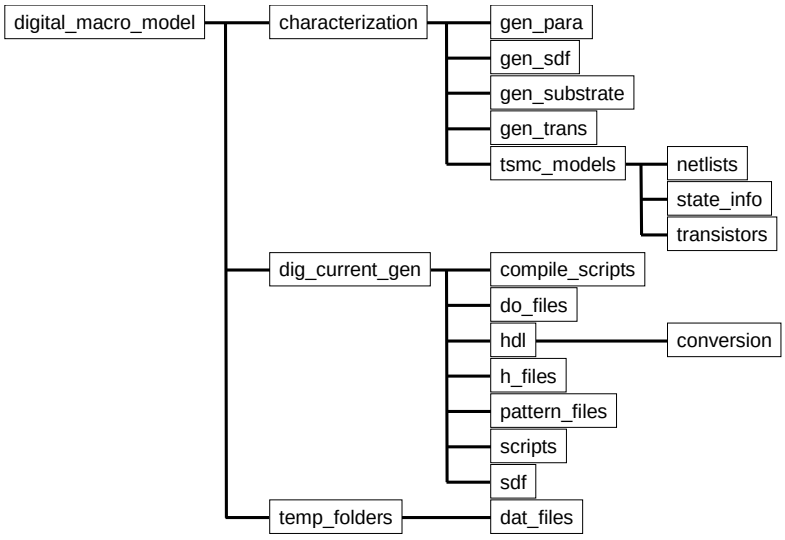

Figure A.1: The folder system for the DMM as a tree graph.

in the setups I checked could cause serious problems for a new setup.

A.1 File System

The file system for the DMM is shown as a tree structure in Fig. A.1. For simplicity only the folders are shown in this figure. All of the files in the DMM use relative paths to allow correct functionality when the complete structure is copied or moved to a new location. All files are also correctly setup to run on either Sun or Linux operating systems. The files used in the DMM are found in the top level digital_macro_model folder. The next level of folders in this top level folder contain the files needed for characterization, digital noise current generation, and the storing of temporary data.

The characterization folder contains all of the scripts and files needed to do

the characterization of the digital libraries. There are five sub folders within this folder:

- *gen_para* - This folder contains the scripts needed in order to characterize the equivalent parasitics of each gate.
- *gen_sdf* - This folder contains not only the scripts for generating the SDF files for each gate, but scripts for determining the switching voltages of each gate.
- *gen_substrate* - This folder contains the scripts needed for the generation of the local substrate networks of each gate.
- *gen_trans* - This folder contains the scripts needed to characterize the transition waveforms of each gate.
- *tsmc_models* - This folder contains the basic model information used in all current DMM setups. This folder also has three sub folders:
  - *netlists*: This folder contains the extracted netlists of all gates.
  - *state_info*: This folder contains the state maps and initial conditions for the gates which have multiple internal logic states.
  - *transistors*: This folder contains the different process and run specific transistor models to be accessed during characterization.

The dig_current_gen folder contains most of the scripts and files needed to run the digital noise current generation part of the DMM. During the digital noise

current generation, files from the characterization steps will be accessed and used accordingly. There are seven sub folders inside of the dig_current_gen folder, as well as the setup run files and temporary run information files. The seven folders are:

- *compile_scripts* - This folder contains the scripts used to correctly compile all the needed blocks for a particular setup.
- *do_files* - This folder contains all the do files needed when simulating different setups in Modelsim.
- *hdl* - This folder contains the hardware description language files for all digital blocks and test benches. This folder also contains one sub folder:
  - *conversion*: This folder contains scripts to convert HDL code from Verilog to VHDL.
- *h_files* - This folder is used to temporarily store all the transition waveforms of the digital block currently being modeled.
- *pattern_files* - This folder contains files for the pattern generator used to load and run the 8051 microprocessor in the lab. The convert2force.cal script converts these pattern files to do files for simulating the 8051 microprocessor in Modelsim.
- *scripts* - This folder contains the various scripts which are used to perform the digital current generation.

- *sdf* - This folder contains the sdf files used in the digital current generation. The sdf for the digital block currently being modeled is saved here as delay_file.sdf. In some DMM setups there are digital blocks used which are not being modeled, but must have correct delay information for the gate level simulation. The sdf files for these blocks are also stored in this folder.

Temp_folders is actually representative of several folders in the file structure. For each digital current generation setup, a new folder must be created to contain all of the output information. All of this output information is re-created and written over each time the digital current generation is run. The raw PWL files are all stored in the dat_files sub folder.

A.2 Characterization

The characterization is run as a series of independent scripts. The scripts are run separately for flexibility reasons. There are cases where you may need to run some sets of scripts multiple times, but not all of them. The characterization scripts are divided into the five following subsections.

A.2.1 Model Information

The model information for the digital libraries is stored in the tsmc_models folder. It is stored in three sub folders: netlists, state_info, and transistors. A header file containing a summary of the gate information is stored in this folder as well, called

lib_cell_header.lib. The format for this file is one gate per line. Each line contains the name of the gate first. The ID number is arbitrarily assigned and listed as the second value on the line. The number of internal logic states follows next. The last two values are the number of outputs and the number of inputs to the gate. Whenever a gate is added to the library, it should be added correctly into this header file.

The netlists folder contains all of the extracted netlists of the gates. The final netlists are labeled as GATENAME.cell.scs. The labels are changed to pins in the layout before extraction. The extraction is performed with parasitic capacitances added. The current extracted netlists were generated with the Diva extraction rules provided by TSMC for the 0.25μm RF process. Even though the logic process was used in the simulations in this thesis, the extraction rules are very similar between the two. The version of the RF rules that Oregon State University has is also newer than the logic process rules. The extraction from Cadence is saved as GATENAME.cell. This file is then trimmed down to a sub-circuit definition, and then saved as the GATENAME.cell.scs by the script convert_netlists.pl. The script also separates out the NMOS bulks from VSS and attaches them to an added pin called BULK. This script checks the entire folder for .cell files which don't have an equivalent .cell.scs file, so the entire folder can be converted at once, or individually as new netlists are created.

The state_info folder contains all the state information for gates in the library which have multiple internal logic states. A gate must have a folder containing it's state information here if it is described as having two or more states in the

lib_cell_header.lib file. The state_info folder contains all the sub folders for the gate information, and a Matlab script called state_map_gen.m for helping to generate state maps for gates. The state map is a file which contains one number (representing the state number) per line. Each line represents an output and previous input combination. The purpose of the state map is to associate arbitrarily defined states to the gate's actual IO levels.

To demonstrate a state map, an example for a flip-flop is shown in Fig. A.2. The state map index number for the flip-flop is composed of the current outputs as the most significant bits, and the previous inputs as the least significant bits. The order of the outputs and inputs is equivalent to the order in the HDL logic libraries and extracted netlists. State 1 is defined for each index where Q is 0 and QN is 1. State 2 is defined for each index where Q is 1 and QN is 0. Since a flip-flop's QN output is defined as the opposite of Q, the other output patterns are not valid, and are defined as state 0. This adds some error checking functionality which is used in the digital noise current generation.

What is saved in the state map file is only the State column from Fig. A.2. The line number in the file is the index number. In this case, and in fact in all of the gates with multiple internal logic states which have been characterized so far, the state is distinct by the outputs, and not the previous inputs. The previous inputs are still used to remain flexible in the future for gates which may have different internal logic states for the same output patterns.

Before the state_map_gen.m script is run, a folder for the gate must already be created. When the script is run, general information like the gate name and

Q	QN	D	CK	State
0	0	0	0	**0**
0	0	0	1	**0**
0	0	1	0	**0**
0	0	1	1	**0**
0	1	0	0	**1**
0	1	0	1	**1**
0	1	1	0	**1**
0	1	1	1	**1**
1	0	0	0	**2**
1	0	0	1	**2**
1	0	1	0	**2**
1	0	1	1	**2**
1	1	0	0	**0**
1	1	0	1	**0**
1	1	1	0	**0**
1	1	1	1	**0**

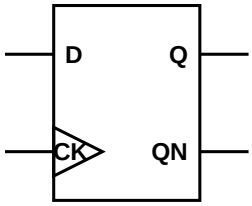

Figure A.2: Example of a state map for a simple D-type flip-flop.

its number of inputs and outputs needs to be entered. At the main menu the state information can then be entered. When state information is added, it is very similar to filling in the table displayed in Fig. A.2. To simplify don't care cases, a wild card character '*' can be used. For the previous flip-flop example, 2 states would need to be added. State 1 and 2 could be added with only the outputs defined, and the previous inputs defined as simply '**'. The 0 states will be defined by default to all unspecified cases. The option to reset or plot the state map is also available. Plotting gives a graphical representation of the states you have filled in so far. Since the index numbers may go rather high with certain gates, displaying the state map as text can be difficult to view. In order to save the state map in the folder of the gate, you must quit and save from the main

menu.

The gate sub folders in the state_info folder contain both the state map for the gate, and the initial conditions for the different states. There is presently no automated way to determine internal logic states. The generate SDF scripts have some checking which will determine if a gate's HDL description has time-varying blocks, but this will only let you know that you need to correct it, not how. Most of the time, after determining a block needed to be characterized for different states, the HDL description was sketched on paper. While looking at the gate description the state information was decided on, as well as the input switching needed to get the gate into those states. By running the generate transition waveforms scripts, all of the switching information to a gate can be setup in a netlist. Adding a writefinal statement to the correct switching simulation can save the final conditions in a .fc file. This file is then named as STATE.fc and placed in the correct gate sub folder. For example, with the previous flip-flop example, not only will there be a state_map.txt file in the gate sub folder, but also two files, 1.fc and 2.fc, which are used as initial conditions in other characterization scripts to set the gate into the correct internal logic state.

The last folder in the tsmc_models directory is the transistors folder. This contains the transistor model files specific to run and process. Each file is named by the run name, and both the bsim3 NMOS and PMOS models are listed. Which model to use is picked as a variable in the characterization script being run.

A.2.2 Generate Substrate

The majority of the scripts used to generate the local substrate networks for each gate are contained in the gen_substrate folder. A file named subsMap.txt needs to first be created by a Silencer! Digital function called the Contact Lister (described in Appendix B.1.2), and then copied into this folder. This file contains the contact geometries for each gate, and is used by the create_gate_subs.pl script.

The names of the gates in the subsMap.txt file are often times not the same as the HDL descriptions. This happens since the Contact Lister will descend to a low enough hierarchy level to find cells which have the correct p-tap and bulk contacts it would expect of a normal gate. Many times the name of this layout cell is different, and is contained within some higher level cell which has the correctly matching name. The create_gate_subs.pl makes an attempt to correctly adjust the name by capitalizing and shortening, but this does not always work. Checking through the gates manually is always recommended. In the worst case, any missing or incorrectly named gates will throw an error when the digital noise current generation is run, and can be tracked down then.

The create_gate_subs.pl script loops through the subsMap.txt file, and creates an EPIC input file for each gate. The file is composed of a header, the contact geometries, and a footer. The header and footer are the same for all gates, and are provided as input files to the script. The header file contains specific profile information, which should be changed if a different process is simulated. After the EPIC input file is created, it is run through the correct version of EPIC. In

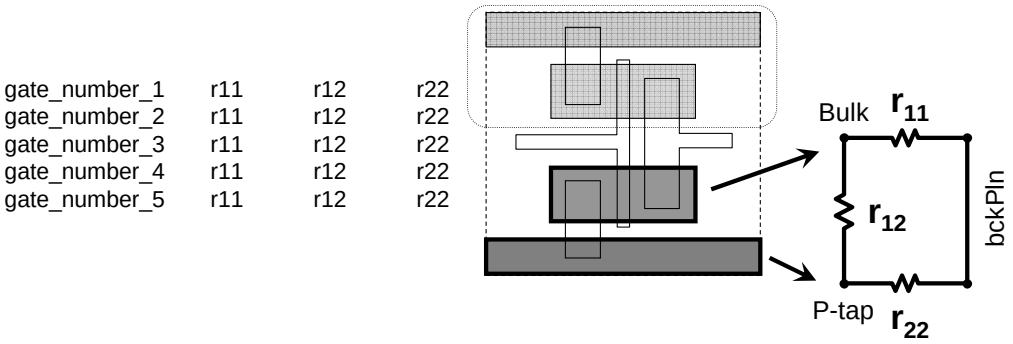

Figure A.3: Representation showing the way the resistive substrate model for each gate is saved.

order for the code to be flexible, it must be able to run on Linux or Sun. EPIC executables compiled in each are contained within the gen_substrate folder, and will be chosen based on which system the user is currently on. The output file from EPIC is then opened, and the resistance values are extracted and saved in relation to the gate name. All local substrate networks are saved in an output file at the end of the script called rmatrices.txt. The format of the rmatrices.txt file is shown in Fig. A.3. This file needs to then be copied to the correct folder inside of gen_trans. When the digital current generation is run, the local substrate information will be looked for one level up from where the transition waveforms are saved.

A.2.3 Generate Transition Wavefoms

The scripts to generate and save the transition waveforms are contained in the gen_trans folder. The top level script is a Matlab script called create_tran.m. The

script displays run information in the command window as it progresses. There are several variables set at the beginning of the script. Those variables are:

- *output_library* - This variable is used to define the location where the output files will be saved. The current format is './CHIP_NAME/RESOLUTION/'.
- *cell_header_file* - This variable points to the cell header file. The file contains general gate information for the standard cell library. The cell header file for all current setups is found in the tsmc_models folder.
- *time_step* - The time step that the simulator saves the transition waveforms in is set by this variable. The combination of this variable and the time_length variable set the number of points in the transition waveforms. For smooth truncation in the digital noise current generation, the number of points should be a power of 2. The units are seconds.
- *time_length* - The length of time that the transition waveforms are recorded for is set by this variable. The combination of this variable and the time_step variable set the number of points in the transition waveforms. For smooth truncation in the digital noise current generation, the number of points should be a power of 2. The units are seconds.
- *output_load* - This variable sets the number of load gates attached to each output when the gates are characterized. This option was not used in the characterization of the current setups, but is available.

- *model_file* - This variable sets the transistor models used in the simulations. The transistor models are found inside the tsmc_models/transistors folder.

- *simulator* - The unix command for calling Spectre is set by this variable. The path to version 5.10 was used previously, however the path to version 6.1 has been used in some cases for faster simulation times. The output format must currently be ASCII because of the Perl script used to extract the data.

- *rising_input* - This variable sets the PWL file used as the input rising stimulus in simulation. This file must be carefully formatted to start rising at the correct time. The start time of the transition waveforms is set in the create_tran_netlist.pl file, and must be set early enough to capture the complete waveform during the input rising.

- *falling_input* - This variable sets the PWL file used as the input falling stimulus in simulation. This file must be carefully formatted to start falling at the correct time. The start time of the transition waveforms is set in the create_tran_netlist.pl file, and must be set early enough to capture the complete waveform during the input falling.

- *VDD* - The value of the VDD rail is set with this variable. The units are volts.

- *VSS* - The value of the VSS rail is set with this variable. The units are volts.

- *cell_vect* - This variable is an array which determines the gates which will be characterized. The array is made up of the the cell IDs, as defined in the

cell_header_file.

The create_tran.m script loops through each gate in the cell_vect array, and characterizes them. In the loop it first calls the Perl script create_tran_netlist.pl to generate Spectre netlists. Several of the variables from the Matlab script are passed down to the Perl script, as well as some new variables set specifically in create_tran_netlist.pl. One of these variables is the load_file. This file contains a gate which acts as an 'average' output loading for this library. The create_tran_netlist.pl script creates the gate specific netlist(s) needed to test the gate for all input transition and internal state possibilities. A netlist is created for each possible internal state of the gate, and each netlist contains multiple altergroups for simulating all input switching patterns. The netlists are saved as files named for the state number (for example tran_gen_s1.scs, tran_gen_s2.scs, tran_gen_s3.scs, etc.).

The netlist(s) are then simulated, and all of the data must be extracted. By looping through the output files, the extract_power.pl script is used to extract and save the data in a file currently called power.out. The file contains the time and current information through the VDD, VSS, and BULK pins. This file is then loaded and organized in Matlab.

After all of the netlist data is loaded in Matlab, it can be saved in the appropriate files. Three files are saved: one each for all the current waveforms through the VDD, VSS, and BULK pins. The data in these files is saved as a large matrix, with the rows being the time index, and the columns being the input transition and internal state index. Fig. A.4 helps to demonstrate this organization by showing the matrix for a hypothetical gate. This gate has only one input, and two

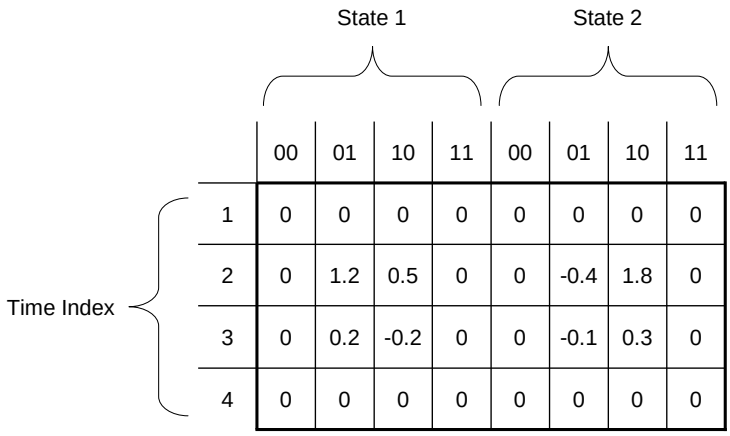

Figure A.4: The matrix format for saved transition data of a hypothetical gate with one input and two internal states.

internal logic states. There are only four time points saved for each waveform in this example. Each column represents a different waveform. The numbering for the columns is the input's initial logic level, followed by the input's final logic level. There are some columns which contain no waveform information for this reason, like 00, which keeps the input constant. For simplicity the gate is characterized sequentially, including non-input transition cases like this. If the gate has more than one input, the order of the inputs used to form the number is based on the order in the HDL and netlist files. The waveforms are also organized by initial internal state, from left to right, as seen by the repeat in the input numbering.

A.2.4 Generate SDF

The scripts for generating the SDF files of gates are contained in the folder gen_sdf. The top level script is a Matlab file called create_sdf.m. This script displays run

information in the command window as it loops through and creates all of the SDF files for the gates specified. There are several variable inputs to this file:

- *output_library* - This variable is used for the location where the output files will be saved. The current format is './CHIP_NAME/'.
- *verilog_file* - This variable sets the first file the script should look in to find an HDL definition of the gates. This file needs to be a Verilog description.
- *verilog_file2* - This variable sets the second file the script should look in to find an HDL definition of the gates. This file needs to be a Verilog description.
- *max_load* - The maximum number of loads connected to each output is defined by this variable. The script will iteratively attach all combinations of loads from 0 to this value on all outputs, and characterize an SDF file for each loading.
- *cell_header_file* - This variable points to the cell header file. The file contains general gate information for the standard cell library. The cell header file for all current setups is found in the tsmc_models folder.
- *model_file* - This variable sets the transistor models used in the simulations. The transistor models are found inside the tsmc_models/transistors folder.
- *simulator* - The unix command for calling Spectre is set by this variable. The path to version 5.10 was used previously, however the path to version 6.1 has been used in some cases for faster simulation times. The output format must currently be ASCII because of the Perl script used to extract the data.

- *rising_input* - This variable sets the PWL file used as the input rising stimulus in simulation.
- *falling_input* - This variable sets the PWL file used as the input falling stimulus in simulation.
- *units* - This variable sets the units of time used in the SDF files. The default is currently 1e-12 (ps). If this value is changed, the timescale variable should also be changed in the make_sdf.pl script for the digital noise current generation.
- *VDD* - The value of the VDD rail is set with this variable. The units are volts.
- *VSS* - The value of the VSS rail is set with this variable. The units are volts.
- *Vsw* - This variable sets the value used to define IO switching. The path delays in a gate are timed from an input crossing this threshold, to an output crossing this threshold. This value can be found with the help of the find_vsw.m script.
- *cell_vect* - This variable is an array which determines the gates which will be characterized. The array is made up of the the cell IDs, as defined in the cell_header_file.

The create_sdf.m script loops through each gate in the cell_vect array, and characterizes them. The script has another nested loop to cycle through all possible

output load combinations in order to develop a library of SDF files to choose from when running the digital current generation. Inside the nested loop, a Perl script, create_sdf_netlist.pl, is called to generate Spectre netlists. Several of the variables from the Matlab script are passed down to the Perl script, as well as some new variables set specifically in create_sdf_netlist.pl. One of these variables is the load_file. This file contains a gate which acts as an 'average' output loading for this library. The created netlists correctly simulate all input transition and internal logic state possibilities for the gate. This information can then be organized to generate all delay paths through the gate. The netlists are saved as files named for the state number (for example sdf_gen_s1.scs, sdf_gen_s2.scs, sdf_gen_s3.scs, etc.).

The netlist(s) are then simulated, and all of the data must be extracted. By looping through the output files, the extract_output.pl script is used to extract and save the data in a file currently called output.out. This file contains the time and voltage information of each gate output. When the data is loaded back into Matlab, it is compared to the Vsw variable, and the output switching time is determined. Since the input switching time is based only on the input switching waveform used, the delay for the paths of this simulation are easily determined. The delay paths for all input switching and internal logic states are all loaded and organized in Matlab. A complete SDF file for each gate is then created from this data.

The delay definitions for cells with internal states is complex in the HDL descriptions. This makes matching them to the SDF delay definitions difficult. For this reason, the SDF files for gates with internal logic states contain only general

delay paths, no conditional delay paths.

The complete SDF file for the gates without internal logic states is then trimmed down to correctly match the HDL description with the convert_sdf.pl script. This script is run twice, once for verilog_file and once for verilog_file2, to find where the gate is described. The script finds the defined conditional delay paths in the HDL description, and adjusts the complete SDF file to match.

There are some instances where the digital block which needed to be simulated had gates with very large output loads. The simulation of all possible output load combinations, as done in create_sdf.m, was not feasible in these cases. An alternative characterization script was created for these cases, create_sdf2.m. This Matlab script has all of the same variable inputs as create_sdf.m with the removal of max_load and cell_vect, but the addition of:

- *component_load_file* - This variable points to a file created in the digital current generation containing all un-characterized gate and load combinations. This file is created when make_sdf.pl fails. The current default is the component_load.txt file.

The create_sdf2.m script works very similar to the create_sdf.m script, however it loops through the component_load_file and determines the appropriate gate and loading to simulate. The component_load_file is created from the digital current generation, and so becomes something of a backwards step in this case. The characterization of the gates for SDF files is done after the digital current generation is run and fails once. This generates the list of gates and loads which must be

simulated, and then the create_sdf2.m file characterizes what is needed.

One missing piece of information from the SDF files for flip-flops is the timingcheck values. A work around for this is the use of the add_timingcheck.pl script. This script is contained in the output_library. It uses the file ff_list.txt to determine which gates are flip-flops and need to be checked to make sure their SDF files contain timingcheck information. The timingcheck information is either copied from a pre-existing SDF file, or determined manually, and placed in a file named timingcheck.txt inside of the gate's folder in the output_library. The add_timingcheck.pl script checks all flip-flops defined in the output_library and adds the timingcheck information to any SDF files missing it. The timingcheck information is defined by default in the HDL description of the flip-flops, however it is defined too small in most cases. This causes errors when simulating with the correct delays from the new SDF files.

In the create_sdf.m and create_sdf2.m scripts, the Vsw variable must be defined to determine the delays from input to output. This switching voltage varies from gate to gate and input to input, but the complexity of defining every possibility makes this unfeasible. The find_vsw.m script is provided to help find an average Vsw for a standard cell library. This is a Matlab script, and has the following variables as inputs:

- *cell_header_file* - This variable points to the cell header file. The file contains general gate information for the standard cell library. The cell header file for all current setups is found in the tsmc_models folder.

- *model_file* - This variable sets the transistor models used in the simulations. The transistor models are found inside the tsmc_models/transistors folder.

- *simulator* - The unix command for calling Spectre is set by this variable. The path to version 5.10 was used previously, however the path to version 6.1 has been used in some cases for faster simulation times. The output format must currently be ASCII because of the Perl script used to extract the data.

- *VDD* - The value of the VDD rail is set with this variable. The units are volts.

- *VSS* - The value of the VSS rail is set with this variable. The units are volts.

- *cell_folder_loc* - This variable sets the location of the gate netlists. This is set here and passed down to other scripts. The current default for this is in the tsmc_models/netlists folder.

- *Rail_th* - This variable sets the percentage of the overall supply voltage that the tied together input/output must be away from the VDD and VSS rails before being considered a valid switching voltage. If the tied together input/output is at a voltage level equivalent to either rail, this combination does not cause switching.

- *v_mid* - This variable gives an initial guess to the switching voltage. Unfortunately, the DC simulation may not correctly settle to a switching voltage. This initial condition helps the simulator find the switching stable state, in-

stead of the stable state(s) where the output stays at one of the rails. The units are volts.

- *cell_vect* - This variable is an array which determines the gates which will be characterized. The array is made up of the the cell IDs, as defined in the cell_header_file.

The find_vsw.m script loops through all the gates specified in the cell_vect array. It then calls the Perl script get_iolist.pl to find all of the inputs and outputs for the gate. Nested loops are used to go through all combinations of inputs and outputs. The Perl script create_vsw_netlist is called in the loop to generate netlists for each of these IO combinations. The netlists created contain DC simulations. Each simulation connects a specified input to a specified output through a 1 Ω resistor. The other inputs of the gate are then varied systematically in different altergroups. DC simulations are used instead of transient for simplicity and faster execution.

After the netlist is created, it is simulated, and the data extracted with the extract_vsw.pl script. This script pulls out the dc value of the input/output combination that is connected together. This value is passed back to Matlab and tested. If it is determined that it is far enough away from either supply rail, it is saved as a switching voltage. As find_vsw.m is being run, it is constantly saving switching information to a log file named vsw_summary.log. Average switching information for each gate and the overall run are displayed in both the Matlab command window and in more detail in this log file.

A.2.5 Generate Parasitics

The scripts which characterize gates for their equivalent parasitics are contained in the gen_para folder. The top level script is a Matlab file named create_para.m. This script displays run information in the command window. It contains several variables for specifying inputs:

- *output_library* - This variable is used for the location where the output files will be saved. The current format for this is './CHIP_NAME/'.
- *output_load* - This variable sets the number of load gates attached to each output when the gates are characterized. This option was not used in the characterization of the equivalent parasitics, but is available.
- *model_file* - This variable sets the transistor models used in the simulations. The transistor models are found inside the tsmc_models/transistors folder.
- *simulator* - The unix command for calling Spectre is set by this variable. The path to version 5.10 was used previously, however the path to version 6.1 has been used in some cases for faster simulation times. The output format must currently be ASCII because of the Perl script used to extract the data.
- *cell_header_file* - This variable points to the cell header file. The file contains general gate information for the standard cell library. The cell header file for all current setups is found in the tsmc_models folder.

- *VDD* - The value of the VDD rail is set with this variable. The units are volts.
- *VSS* - The value of the VSS rail is set with this variable. The units are volts.
- *Vstep* - This variable sets the voltage step which is applied to the VSS rail during the simulation. The units are volts.
- *Tstep* - This variable sets the time over which the voltage step happens on the VSS rail during the simulation. The units are seconds.
- *Tsettling* - This variable sets the time allowed for settling when the transient is first started. This settling occurs before the voltage step is applied to the VSS rail. The units are seconds.
- *cell_vect* - This variable is an array which determines the gates which will be characterized. The array is made up of the the cell IDs, as defined in the cell_header_file.

The create_para.m script loops through the gates specified in the cell_vect map. Inside the loop it calls the Perl script create_para_netlist.pl to generate the netlist of each gate. Several of the variables defined in create_para.m are passed down to this script, as well as some locally defined variables. One of these variables is the load_file. This file contains a gate which acts as an 'average' output loading for this library. The script creates netlists which simulate all possible steady state input and internal state combinations. The simulation applies a step voltage to the VSS rail of each combination, and saves the current through this rail. The

netlists are saved as files named for the state number (for example para_gen_s1.scs, para_gen_s2.scs, para_gen_s3.scs, etc.).

Once the netlist(s) are created, they are then simulated, and the data extracted with the extract_para.pl script. This script actually opens and runs through the transient data twice, once to find the final current value, then once again to find the time when the current is at 80% of the final value. It then returns these values to Matlab.

All of the data to determine the equivalent parasitics is now available. The current equation for a capacitor can be rearranged to the form of equation A.1. To find this capacitance the final current and voltage step characteristics (Vstep over Tstep) are simply used. The current response of a series resistor and capacitor to a voltage step can be determined by equation A.2. Re-arranging this equation yields equation A.3, which is used to find the equivalent series resistance between the supply rails. The time value is the time to get the current to 80% of the final current, and the C is the previously determined equivalent capacitance.

$$C = \frac{I}{dv/dt} \tag{A.1}$$

$$I = I_{final}\ (1 - e^{\frac{R}{C}t}) \tag{A.2}$$

$$R = \frac{t}{C \times ln(0.2)} \tag{A.3}$$

A.3 Digital Noise Current Generation

The top level digital noise current generation script is called generate_idats. An executable file is created for different setups named generate_BLOCKNAME. As an example the NCL core of the 8051 is run from the command line by typing 'generate_ncl8051'. This runs the executable file which has all options for that particular setup, and then passes them to the generate_idats script. The option arguments for generate_idats are described as comments at the beginning of the code.

The generate_idats flow is described in Fig. A.5. The generate_idats script is a shell script which runs 11 other sets of programs/scripts to perform the complete noise current generation. After each script is completed, generate_idats will check for errors and halt if it finds the script failed.

The generate_idats program is a c-shell script. It starts with some initialization. This includes setting the location of awk depending on the operating system, setting the offset variable to the value in the offset file of the current transition library, and making a copy of the modeled digital block's HDL description. The HDL copy is renamed with 'adjusted_' prepended to the original name (for example ncl8051.vhd becomes adjusted_ncl8051.vhd). This copy can now be modified and the original preserved.

The first script called by generate_idats is get_instances.pl (the current version is get_instances4.pl). This Perl script steps through the modeled digital block's HDL description and compiles information about the components and instances.

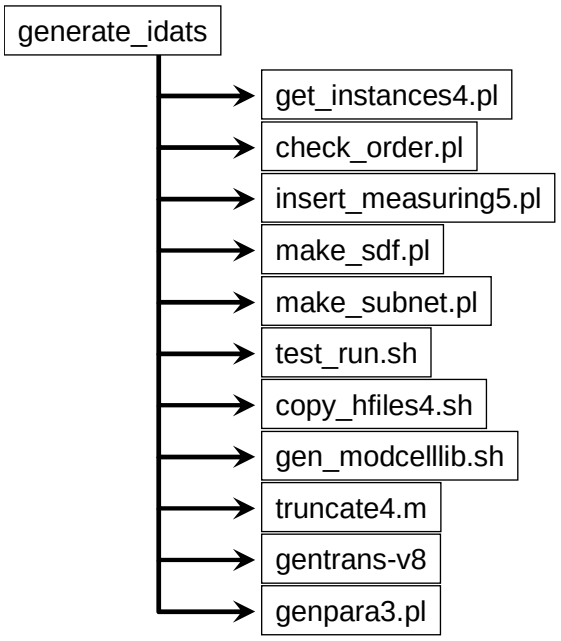

Figure A.5: The program flow of the digital noise current generation.

It saves this information to various files in the run directory for later read in and use.

The next script called by generate_idats is check_order.pl. This Perl script is used specifically for error checking, it is not actually needed for DMM generation. The current DMM method relies on using input logic transitions to help find the transition waveforms that gates generate. If the order of the inputs in the modeled digital block's HDL description doesn't match the order used when the transition waveforms were characterized, the modeling will be incorrect. This script compares the order of inputs found for each gate during get_instances.pl with the netlists used for transition waveform characterization.

The insert_measuring.pl (the current version is insert_measuring5.pl) script is

then called by generate_idats. This Perl script uses the instance information found by the get_instances.pl script to add logic transition recording code into the modeled digital block's HDL description. The code sets up a loop which checks the inputs of gates at time intervals during the gate level simulation. If the logic levels change between iterations of the loop, then a transition is found, and the information is stored in a file. The output information is also stored after a transition is found, for gates which have internal logic states.

The next script called by generate_idats is make_sdf.pl. This Perl script combines the SDF information of each gate in the modeled digital block. The instance information found by the get_instances.pl script is used to determine the gates and output loads. If the gate SDF information is missing, the program keeps track, and continues. At the end of the script the user is given the chance to continue or halt execution of generate_idats if there was missing SDF information. If a gate's delay information is not specified, the default delays in the HDL description of the gate will be used. A file is also created (called component_load.txt currently) to keep track of any missing gate and load information. This file can be used by the create_sdf2.m characterization script to generate specifically the missing gate SDF files. The SDF file created by this script only represents the modeled digital block. If there are other digital logic blocks which must have correct delay information back annotated in during the Modelsim simulation, these must be created prior to the running of this generate_idats setup.

The make_subnet.pl script is then called by generate_idats. This Perl script combines the local substrate information for all gates in the modeled digital block

to form the complete substrate network. The instance information found by the get_instances.pl script is used along with the rmatrices.txt (found in the transition waveform library) to create the r matrix which represents the substrate of the modeled digital block. The r matrix is saved in the temporary data folder for this setup (for example digital_macro_model/temp_ncl8051).

The next thing which generate_idats does is to setup and simulate the adjusted HDL code in Modelsim. The shell script test_run.sh is used to do this. This script cleans out the old work directory, and creates a new one. It uses a special compile script to compile only the needed HDL blocks into the new work directory. This compile script is specific for the setup being run, and must be created before generate_idats is run. A do file is constructed and then used, along with the SDF file, when Modelsim is run on the testbench for this setup. As the simulation is running, the input transition and output state information are being saved to files in the temporary data folder for this setup (digital_macro_model/temp_ncl8051).

The input transition and output state files must have the simulation step resolution added as the first line to them. This simulation step resolution is the rate at which the input transitions were checked for in the gate level simulation. The transition waveforms for the gates of this modeled digital block are then copied locally to the h_files folder by the copy_hfiles.sh script (current version copy_hfiles4.sh). The gen_modcelllib.sh script then adds the length of these transition waveforms to the instance information file previously created by get_instances.pl.

The next script called by generate_idats is truncate.m (the current version is truncate4.m). This Matlab script truncates the length of the locally copied

transition waveforms. These waveforms are reduced to a length which preserves a specified percentage of the total energy. The truncate script cycles through all gates and each type of transition waveform (those through the VDD, VSS, and bulk nodes of the gate). The new length waveforms are then saved in the h_files folder, and the length of each is added to the instance information file to be used later.

The gentrans program (current version gentrans-v8) is called by generate_idats next. This C program uses the input transition and output state information to concatenate together the appropriate transition waveforms and form the noise currents of the gates. After loading the needed data, the gentrans program searches through and finds the input transition events for each gate. The gates are looped through, and the transition waveforms for each input transition event are strung together at the correct times to form the noise currents through the VDD, VSS, and bulk nodes. These noise currents are saved as PWL files in the temporary data folder for this setup (for example digital_macro_model/temp_ncl8051/dat_files). The gentrans program is compiled and can be remade by typing 'make' in the scripts folder. This call uses the Makefile, which can be adjusted for different versions or compile options.

The last script called by generate_idats is genpara.pl (the current version is genpara3.pl). This Perl script creates the equivalent parasitics between the supply rails of the modeled digital block. The script uses the input transition and output state files created during the Modelsim gate level simulation to determine the time varying equivalent parasitics. This version of genpara.pl saves only the time

averaged parasitics for each gate in the temporary data folder for this setup (for example digital_macro_model/temp_ncl8051).

A.4 Making a New Setup

This section will describe the steps to setting up a new digital block for the DMM method. The setup process can vary in complexity depending on how much the new digital block uses old gate characterization information. This section will attempt to be general enough to describe a setup done from scratch.

A.4.1 HDL Simulation

The HDL description of the digital block is the starting point of the DMM setup. The HDL must be a gate level description, with all gates being those found in the standard cell library. Before any gate characterization or model development, the gate level simulation should first be correctly setup.

A testbench must be developed to correctly simulate the digital block. Do files can be used to add the proper stimulus for simulation, however there must be one level of HDL code inside which the digital block is run. The do files which are used for the simulation are created as two files (these are combined again later in the digital noise current generation). The first file is wave_BLOCKNAME.do (for example wave_ncl8051.do). This file usually contains the add wave commands or any other setup commands. The second file is test_BLOCKNAME.do (for example

test_ncl8051.do). This file usually contains the force and run commands used to run the gate level simulation.

Modelsim can compile or run either VHDL or Verilog descriptions, however, the modeled digital block must be described in VHDL for the DMM. The transition sensing code added in the insert_measuring.pl script is specifically VHDL code. If the digital block is described as Verilog, it can be converted to VHDL through the use of the convertV2VHDL.pl script (found in the hdl/conversion folder). This script was designed for a specific format of Verilog code, always try to compile and verify the functionality of the output code.

A shell script to compile the HDL code needed for the gate level simulation should be created as compile_BLOCKNAME.script (ex: compile_ncl8051.script). This script should compile all Verilog and VHDL code needed in the simulation of the testbench. This script will later be used by the digital noise current generation. For that, the digital block's name will need to be prepended with 'adjusted_' and an extra file compiled for the transition sensing code. This extra file can be compiled with the addition of the following line:

vcom -93 hdl/txt_util.vhd

A.4.2 Library Characterization

After the gate level simulation is correctly running, the gates used in the digital block should be characterized. To begin this, a list of the gates used in the digital block should be created.

The characterization step can be approached from two different ways. One is the characterization of specific gates in a digital block. The other is the characterization of all the gates in a specific standard cell library. The former will be described in this section. However, with a complete list of the gates in a standard cell library, the later's steps are not significantly different.

First, the gates' netlists must be extracted from layout information. These extractions should be performed to include parasitic coupling capacitances. The netlists should be saved in the tsmc_models/netlists folder (or another process folder than tsmc_models). The format for the netlist names is GATENAME.cell. The convert_netlists.pl script can be run from this folder to adjust the netlist to the correct sub circuit format. The correct transistor model for these netlists should also be obtained, and placed in the tsmc_models/transistors folder (or another process folder than tsmc_models).

A file named lib_cell_header.lib needs to be created/appended to store gate information. The file is located in the tsmc_models folder (or a different named process folder). The format of lines in this file are:

gate_name ID_number states outputs inputs

The gate name is the same as the name used in the netlist and HDL description. The ID number is an arbitrarily assigned unique number for the gates in the library. The states are the number of internal logic states that the gate has. The number of states may not yet be known, this can be left as 1 by default. The outputs and inputs are the number of each that the gate has.

With the netlists and lib_cell_header.lib file created, the characterization of the gates used in the digital block can now begin. Many of the details for this characterization were previously described in Section A.2.

The internal logic states of the gates which have them should first be characterized. After which, the generation of local substrates should be done for all gates used in this digital block. The EPIC input header needs to be correct for the particular process the digital block is fabricated in.

Next, the transition waveforms for all gates need to be created. The resolution on these transition waveforms is up to the user. Time steps of 100ps provided a reasonable trade-off between accuracy and simulation time for the results presented in this thesis. The time length of the transition waveforms was chosen to give a number of data points which is a power of 2. The length of the transition waveforms being a power of 2 is recommended for accurate truncation during the digital noise current generation.

The generation of the SDF files for each gate in the digital block should be preformed next. The create_sdf2.m script was used for characterization being performed on gates of an already known digital block. If a standard cell library is being characterized, the create_sdf.m script is recommended, with an appropriate value selected for the maximum load variable. The characterization of the gates for the switching threshold may also need to be performed for more accurate SDF files at this point. This switching threshold is used in either of the SDF generation scripts.

The last characterization which needs to be performed is the generation of the

equivalent rail parasitics. The Vstep, Tstep, and Tsettling variables should be correctly setup to find the parasitics of most gates. The values can be adjusted for more or less resolution if the gates are significantly different than those currently tested.

A.4.3 Generate Script

Now that the gates are all correctly characterized, the setup of the digital noise current generation can be done. With the setup of the HDL simulation, many things are already in place. The compile script and do files should already be created, and placed in the correct folders.

A temporary data folder for the setup should be created in the top level folder (for example temp_ncl8051). All final data from the digital noise current generation will be placed in this folder.

Next, a file needs to be created to execute the generate_idats script. It is possible to simply call this script every time you would like to run the digital noise current generation. However, with 12 inputs, it is easier to save this command in a file and make it executable. All inputs to generate_idats are described in the comments at the start of the code. The name of the block should be consistent throughout the setup file names.

Lastly, the test_run.sh script must be adjusted. Adjusting scripts in the DMM in order to add a new setup is not the most desirable solution. This step should be adjusted in future versions of the code. When Modelsim is executed in this script,

it is done with a case statement. A new case should be added for any new block name. Modelsim should be simulated at 1ps resolution. The SDF file(s) should be back annotated to the correct block(s) and the typical option used.

A.5 A Different Application

This section describes a slightly different application of the DMM method. Till now the method has been shown to model digital logic gates for substrate noise simulation. One main part of this modeling is the digital noise current generation, which models the currents through a gate based on the gates input transitions and previous internal states. By stopping this process early, and only finding the input transitions to a gate, another application is utilized.

The 8051 microprocessor contains a shared RAM block. This block is not composed of digital logic gates from the standard cell library, and cannot be modeled with the DMM. Therefore, the transistor level netlist of this block must be used in the final substrate noise simulation. However, the RAM is an internal block in the 8051 microprocessor, and is not directly connected to the chip pins. The issue is therefore how to simulate the RAM at the transistor level, without having to simulate the rest of the 8051 microprocessor at the transistor level.

The c-shell script generate_vdats was created to generate the stimulus needed to run the RAM at the same time as the rest of the substrate noise simulation. The script takes 11 inputs, and an executable options file is created similar to those made for generate_idats (for example generate_ncl8051_ramnoise).

The generate_vdats script runs similar to generate_idats. It first uses the scripts get_instances.pl and insert_measuring.pl to adjust the HDL code. In this case however, the RAM is surrounded by an extra wrapper, so that the only block it identifies as a gate is the RAM. The transition sensing code added is then modified slightly by the change2binary.pl script. The gate level simulation is then performed in Modelsim with the test_run.sh script, with all of the input transition and output state information of the RAM block saved in files.

Finally, generate_vdats uses the genVtrans.pl script. This is a Perl script which uses the input transition information, and changes it into PWL files usable by voltage sources for stimulus in a Spectre simulation.

Appendix B – Silencer! Adjustments

This appendix will describe the adjustments made to Silencer!, a substrate noise simulation tool, for use in this research. Not only is additional code written, but some previous coding has been adjusted or replaced in order to allow for more functionality in substrate noise simulations. The first section will describe the Silencer! Digital add-on. The functions in this add-on were developed to be used specifically on digital blocks in the Cadence environment. The Silencer! Extracted add-on will be described in the next section. This add-on enables the use of extracted views inside the Silencer! flow. Lastly, various issues with the execution of Silencer! and the adjustments made will be described.

One thing should be noted about the way in which Silencer! was used in these adjustments. Silencer! currently has two layers used for defining connections to the substrate, SC_Inj and SC_Sen. The two layers were implemented as a way to distinguish between an aggressor and victim in the substrate noise simulation. This distinction is not however used in EPIC, and rarely in the Silencer! code itself. For simplicity reasons I used only SC_Inj in the majority of the add-on and adjustment code.

B.1 Silencer! Digital

Silencer! Digital is an add-on to Silencer!, previously started by Martin Held. The coding has been cleaned and adjusted, but still maintains many of the original functions. The package is a collection of Skill files which are loaded along with Silencer! at the start of the Cadence environment. The functions were written specifically for digital cells, and enable the DMM to work correctly throughout its flow.

The file system for Silencer! Digital is shown in Fig. B.1. The top level folder is named Silencer_Digital. The loadSilDig.il file contains the Skill code for the load function. This function is called in the .cdsinit.user file, and loads all of the Silencer! Digital functions at startup. This function also reloads itself, making it useful to call after making changes to any part of Silencer! Digital. The sdFinder.il and sdLister.il files contain the Contact Finder and Contact Lister respectively. These tools will be described in more detail in the following subsections. The functions for placing the completed DMM into a schematic are contained in the sdPlaceDMM.il file. This tool is also described more in one of the following subsections. The sdFunctions.il file contains many simple functions. These are general functions used throughout Silencer! Digital. The sdSubCont.il file contains the contact location functions. These functions are used in the Contact Finder and Lister, and help to determine different types of substrate contacts.

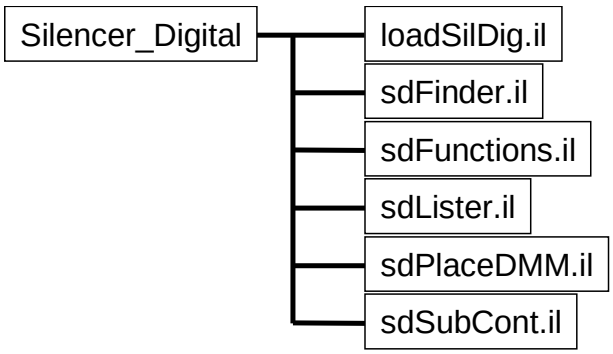

Figure B.1: The file system for Silencer! Digital as a tree graph.

B.1.1 Contact Finder

The Contact Finder is a tool developed to assist in the location of substrate contacts in a layout. Silencer! currently has an automated method for finding substrate contacts which uses polygons of the SC_Inj and SC_Sen layers. This automated method however does not allow for the flexibility needed to find substrate contacts from a variety of different layout styles. The Contact Finder tool enables the user to customize the types of contacts created, and how they are found, in an easy to use GUI. Finding substrate contacts prior to using Silencer!'s built in 'LOCATE Substrate Ports' function is also required in the use of the Silencer! Extracted add-on.

The Contact Finder is accessed in the layout editor, from the Silencer! menu, with 'Use Substrate Contact Finder'. The GUI for the Contact Finder is shown in Fig. B.2. The different controls on the GUI are:

- *Discard contacts smaller than* - This option allows the user to specify a minimum size for the contact areas being considered. In some large layouts

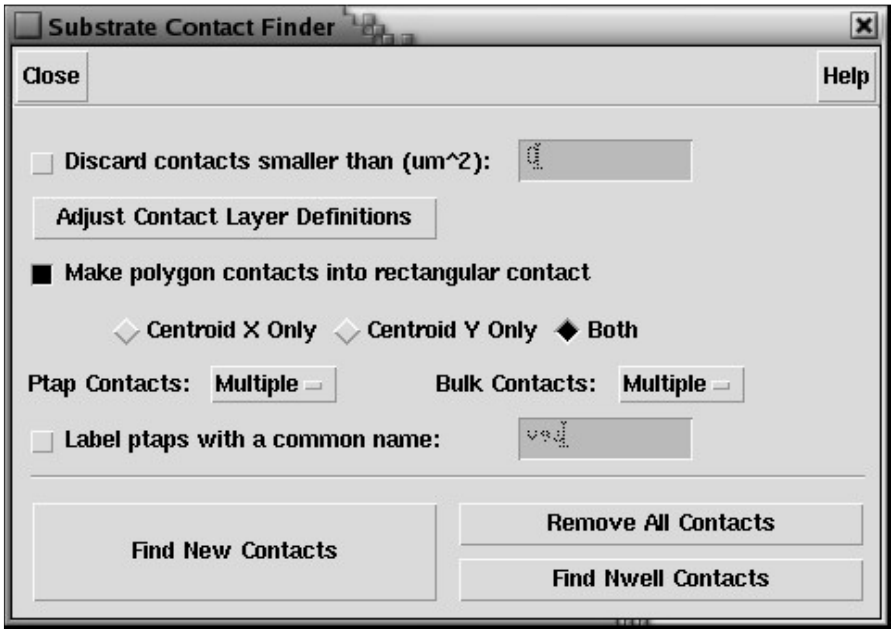

Figure B.2: GUI of the Contact Finder.

it may be advantageous to neglect very small substrate contacts, and consider only the large ones. If the button is pushed, the minimum area can be entered into the text box.

- *Adjust Contact Layer Definitions* - This button opens a GUI used for entering the substrate contact layer definitions. The definitions are loaded and saved to a file named contfinder_param.txt in the Silencer folder.

- *Make polygon contacts into rectangular contact* - This option specifies the way in which polygon contacts are treated. The version of EPIC currently integrated into Silencer! does not accept contacts which are polygons. If this option is set, it converts the polygon contacts into rectangular contacts. A

common centroid method is used to find the equivalent rectangular contact. The radio buttons below this option further specify the way the conversion is performed. 'Centroid X Only' sets the minimum and maximum Y values in the rectangle to be the same as the polygon, then uses the centroid to set the X values. 'Centroid Y Only' sets the minimum and maximum X values in the rectangle to be the same as the polygon, then uses the centroid to set the Y values. 'Both' tries to preserve the ratio of width in X and Y directions and uses the common centroid of each. Area is preserved in all conversions.

- *Ptap Contacts* - This cyclic field specifies how the Ptap Contacts are created. 'None', 'One', or 'Multiple' ptap contacts can be created. When 'One' is selected, all ptap contacts found in the layout are combined to a single equivalent contact. For this option 'Make polygon contacts into rectangular contact' is automatically selected, since combining to an equivalent polygon would be too complex.

- *Bulk Contacts* - This cyclic field specifies how the Bulk Contacts are created. 'None', 'One', or 'Multiple' bulk contacts can be created. When 'One' is selected, all bulk contacts found in the layout are combined to a single equivalent contact. For this option 'Make polygon contacts into rectangular contact' is automatically selected, since combining to an equivalent polygon would be too complex.

- *Label ptaps with a common name* - This option allows the Ptap Contacts to be labeled to a common name. A small label is placed inside the contact

layer. If the button is pushed, the name can be entered in the text box. This option is only useful for the old method of Silencer!, which uses contact labels, unlike Silencer! Extracted, which uses connectivity information.

- *Find New Contacts* - This button searches for and creates new Ptap and Bulk Contacts in the current layout view.

- *Remove All Contacts* - This button removes all shapes in the SC_Inj layer from the current layout view.

- *Find Nwell Contacts* - This button searches for and creates new Nwell Contacts in the current layout view.

One important control in the Contact Finder GUI is the 'Adjust Contact Layer Definitions' button. Clicking this will open the GUI shown in Fig. B.3. This allows for the adjustment of the contact layer definitions used in the Contact Finder. The Contact Finder defines three types of substrate contact: Ptap Contact, Bulk Contact, and Nwell Contact. These definitions are simply names which match up to the controls in the Contact Finder GUI. The user is free to use the definitions for some other type of contact than the name. Up to 5 layers can be used to form the boolean logic definition of the contact formed. The priority is critical in the contact definitions, where the boolean logic between layers will be performed from top to bottom. The Bulk Contact definition for example will be defined in Fig. B.3 as ((((DIFF and NIMP) or NDIFF) and not NWELL) and not PSUB2). If less that the 5 layers are to be used, the cyclic field should be changed to 'End'

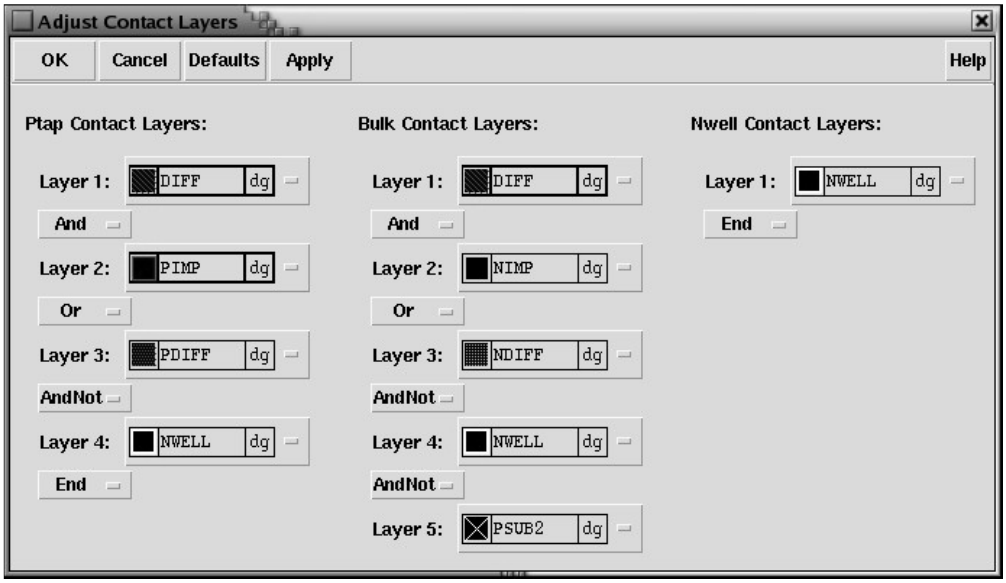

Figure B.3: GUI for adjusting the substrate contact layer definitions.

after the final layer. Clicking 'OK' or 'Apply' saves the settings to a file named contfinder_param.txt inside the Silencer folder.

After all controls are set as desired, and the contact layers are correctly defined, the user can affect the layout by using the 'Find New Contacts', 'Remove All Contacts', and 'Find Nwell Contacts' buttons. When the 'Find New Contacts' button is clicked new Ptap and Bulk Contacts are created. The 'Find Nwell Contacts' button creates new Nwell Contacts. Each button should only be pushed once before clearing out old contacts. Overlapping contacts will cause problems when sent to EPIC. For this reason the 'Remove All Contacts' button is provided to remove all shapes in the SC_Inj layer from the layout. The button is separate and not included as part of the create contact buttons to allow flexibility to the user. This flexibility comes with the added responsibility to keep track of overlapping

contacts. Only a warning message is displayed if the functions find an overlap in the SC_Inj layer.

B.1.2 Contact Lister

The Contact Lister is a tool developed for compiling substrate contact information for the characterization portion of the DMM method. The generate substrate portion of the characterization requires a file with all of the substrate contact information for each gate. This tool runs some of the same substrate contact locating functions found in the Contact Finder on multiple layouts, and gathers the geometrical data into an carefully formatted output.

The Contact Lister is accessed in the layout editor from the Silencer! menu with 'Use Substrate Contact Lister'. The GUI for the Contact Lister is shown in Fig. B.4. The different controls on the GUI are:

- *Location of cells to list* - This option is used to determine which cells the lister should search through for contact information. 'This cell's hierarchy' sets the lister to look at all layouts of cells within the current layout and hierarchically below. 'This cell's library' sets the lister to look at all layouts of cells within the same library as the current layout. The cell's hierarchy is the recommended method.

- *Output filename* - This option sets the name and location of the output file containing the substrate contact geometries. The file location is in reference to where icfb was run.

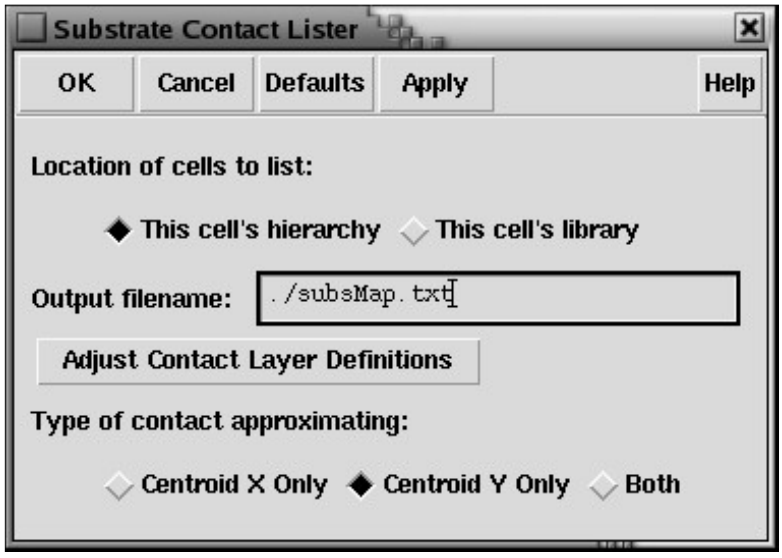

Figure B.4: GUI of the Contact Lister.

- *Adjust Contact Layer Definitions* -This button opens a GUI used for entering the substrate contact layer definitions. The definitions are loaded and saved to a file named contfinder_param.txt in the Silencer folder. This is described in more detail in the previous section.

- *Type of contact approximating* - These radio buttons specify the method used to convert polygon contacts into rectangular contacts. 'Centroid X Only' sets the minimum and maximum Y values in the rectangle to be the same as the polygon, then uses the centroid to set the X values. 'Centroid Y Only' sets the minimum and maximum X values in the rectangle to be the same as the polygon, then uses the centroid to set the Y values. 'Both' tries to preserve the ratio of width in X and Y directions and uses the common centroid of each. Area is preserved in all conversions.

The output of the Contact Lister is a file with the substrate contacts for each gate found. The local substrate networks used in the DMM method contain only 2 contacts, one for the gate's ptap contact, one for the gate's bulk contact. For this reason, all contacts found must be condensed down to a single rectangular contact of either type. There are therefore less options in the Contact Lister than the Contact Finder.

The Contact Lister searches through the cell structure specified and looks for layout views. It opens each layout and attempts to create substrate contacts. If the layout contains a ptap and a bulk contact, the geometry information is saved into the output file. Some cells which are not gates from the standard cell library may end up getting contact information saved in this file. The file should be visually inspected after creation, and before submitting to the substrate generation scripts.

B.1.3 DMM Placing Tool

After a DMM equivalent of a digital block has been created (as described in Appendix A) the model must be added into a schematic view for simulation in the final substrate noise simulation. The model can be very large for digital blocks with thousands of gates. This tool enables the user to automatically place the complete DMM into a schematic view.

The DMM Placing Tool is accessed in the layout editor from the Silencer! menu with 'Place Digital Macro Model'. The GUI for the DMM Placing Tool is shown in Fig. B.5. The different controls on the GUI are:

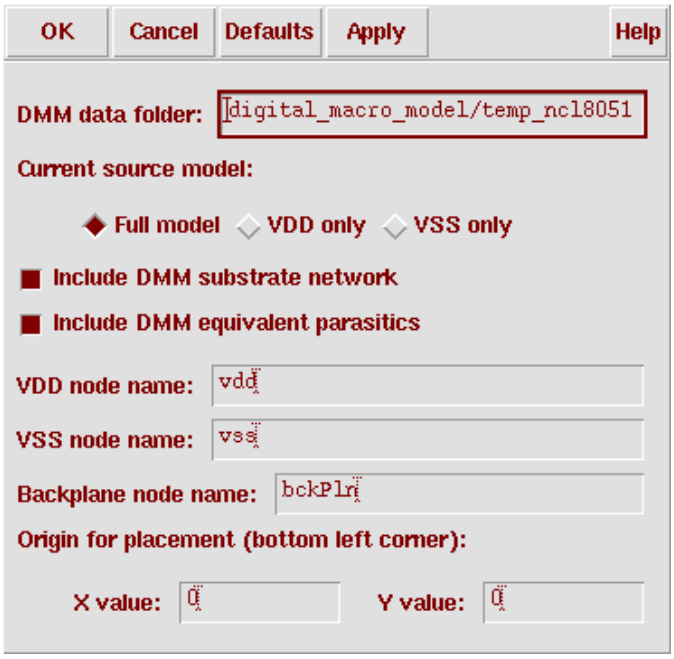

Figure B.5: GUI of the DMM Placing Tool.

- *DMM data folder* - This field sets the folder name and location where the data for the DMM is found.

- *Current source model* - This option sets the current sources used for the DMM. If 'Full model' is selected, all three current sources are used for each gate, one each for VDD, VSS, and bulk currents. If 'VDD only' is selected, only the VDD current is used for each gate, connected between the VDD and VSS nodes. If 'VSS only' is selected, only the VSS current is used for each gate, connected between the VDD and VSS nodes.

- *Include DMM substrate network* - This option enables the placement of the resistor matrix which represents the substrate network of the complete digital

block. This resistor matrix was created during the DMM generation, and is currently stored as a file named 'rmatrix.txt' in the DMM data folder. This option is automatically unselected if the current source model is set to either 'VDD only' or 'VSS only'.

- *Include DMM equivalent parasitics* - This option enables the placement of the equivalent gate parasitics. The equivalent parasitics were created during the DMM generation, and are currently stored as a file named 'res_cap.dat' in the DMM data folder.

- *VDD node name* - This option sets the node name of the DMM's VDD terminal in the schematic.

- *VSS node name* - This option sets the node name of the DMM's VSS terminal in the schematic.

- *Backplane node name* - This option sets the node name of the backplane connection to the substrate network. This option is automatically disabled if the 'Include DMM substrate network' is not set.

- *Origin for placement* - These values set the origin that the tool uses for reference to begin placement. The model is placed above and to the right of this point.

Clicking the 'Apply' or 'OK' buttons places the specified DMM into the schematic at the point designated by the origin. The placing tool allows for the flexibility to add different parts or versions of the DMM into the schematic. Adding pins and

creating a symbol then allows for a clean placement in a substrate noise simulation setup.

B.2 Silencer! Extracted

Silencer! Extracted is an add-on to Silencer! which enables the use of the extracted view in substrate network extraction. Silencer! currently requires a schematic which is properly labeled in relation to the layout to correctly connect the generated substrate network. This add-on allows a layout to be extracted, and the generated connectivity information used directly to connect the substrate network. The inclusion of parasitic capacitances in substrate noise simulations is also made more convenient with this add-on.

A flow chart which shows how to use Silencer! Extracted to generate substrate network information is shown in Fig. B.6. The layout view is the starting point of the flow. Using the Contact Finder places the SC_Inj layer defined substrate contacts into the layout. The substrate contacts must be found previous to Diva extraction, since the extraction rules must connect the SC_Inj contacts to the correct nodes in the extracted netlist. For this reason we cannot use the built in 'Select New Injector Region' function in Silencer!, since it does not locate substrate contacts until after the 'LOCATE Substrate Ports' step. A special set of Diva extraction rules are then used on the layout view containing substrate contacts. This generates an extracted view for the layout with substrate contacts, which now has the correct connectivity. Parasitic interconnect capacitances can also be

added to the extracted view during this step. Next, 'LOCATE Substrate Ports' is selected from the Silencer! menu. This function is somewhat misnamed. Although it does enable the location of substrate ports under areas previously defined by 'Select New Injector Region', this functionality is not used in this flow. The other thing which this function does is to create the EPIC input file, and copy the view to a new cell with _snc appended to the name. This sets up the use of the 'CALCULATE Substrate Newtwork' function. This starts EPIC (or whichever substrate extraction tool is currently being used) in the background. After this is completed, the substrate network is added back into the extracted view. The option to place the substrate network automatically must be selected from the Models & Options menu the same way it was previously done for the schematic. A option from the pull down menu will also add a previously generated substrate network into the extracted view, the same way it is added into the schematic view with the current Silencer! code.

In order to use the extracted view in a substrate noise simulation, a few steps must be taken before the Diva extraction is performed. Any IO pins for the layout must be placed in order be correctly netlisted during the extraction. If the substrate network is going to be connected to something else through the common backplane node, a pin should also be place named 'bckPln'. This pin should touch no layers, but will be correctly connected when the substrate network is added back into the extracted view. Making a symbol for the final extracted view with substrate network then allows for the simulation of the block in a schematic view.

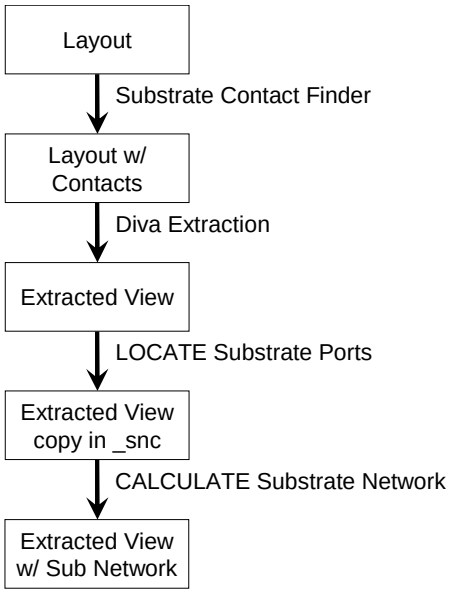

Figure B.6: Flow chart describing the steps to use Silencer! Extracted.

B.2.1 Diva Extraction Rules

The standard Diva extraction rules for a particular process must be adjusted to facilitate the use of the Silencer! Extracted add-on. The adjustments to the file depend on the way the original rules were written, however some general requirements can be listed.

1. The substrate must be divided correctly into local areas. The substrate is usually defined as a layer named 'sub' and initially set to the complete background. Every connection to the substrate is connected to the same node. For the purposes of Silencer! Extracted, the substrate should not be an equipotential like this. The connections to the substrate should be isolated in the netlist, and then connected through the substrate network.

To perform this isolation the 'sub' layer is broken up to be defined directly below ptap contacts or transistor bulk regions.

2. The Silencer! specific layers must be added to the rules. The layers must be recognized as layers at the beginning of the rules. The layers also need to be saved as interconnects at the end of the file. If the layers are not saved, they will not be represented in the extracted view, which is required for Silencer! Extracted. Net layers of the special Silencer! layers were added to the layer map specifically for this purpose.

3. The substrate contacts must be connected to the correct nodes. Vias must be constructed to connect the SC_Inj layer, that the substrate contacts are made of, to the local substrate areas. Since the bulks of the transistors and ptap contacts are usually connected to the substrate layer, this will also connect the substrate contacts.

4. Capacitive coupling should be added between SC_Inj and other layers. Most Diva extraction rules include parasitic capacitive coupling between standard interconnect layers. By adding similar code between the SC_Inj layer and other interconnects, substrate contacts can be created for interconnect coupling to substrate. If a large area of metal 1 is present in the layout, this coupling to the substrate may be significant. By adding a substrate contact directly below the metal, it will be capacitively coupled to the substrate network. Capacitive coupling defined between the SC_Inj and the Nwell layer will enable the simulation of coupling through the nwell in a similar manner.

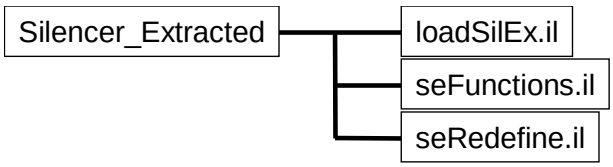

Figure B.7: The file system for Silencer! Extracted as a tree graph.

There are two rules files currently adjusted, one for TSMC's 0.25μm RF process, and one for TSMC's 0.35μm process. Each set of rules has been modified to be used with the Silencer! Extracted add-on. Both the 0.25μm and 0.35μm rules require the 'PARASITIC_C' and 'parasitics' switches, respectively, to enable capacitive coupling to the SC_Inj layer. The 0.25μm rules were also written with the intent to allow for normal extraction, or extraction for use in Silencer! Extracted. A switch named 'Sub_Injector_Bulk_Connect' must be set in the 0.25μm rules when using in conjunction with Silencer! Extracted.

B.2.2 Skill Coding

The file system for Silencer! Extracted is shown in Fig. B.7. The top level folder is named Silencer_Extracted. In a similar manner to loadSilDig.il, loadSilEx.il loads all the Skill files containing Silencer! Extracted functions. This load function is called in the .cdsinit.user file as well. The seFunctions.il file contains all of the newly created functions used as the Silencer! Extracted add-on. The sdRedefine.il file contains functions which were previously defined in Silencer!, but are redefined for the Silencer! Extracted add-on. The loadSilEx.il file must therefore be loaded after the original Silencer! Skill in the .cdsinit.user file.

Some of the most significant changes in the previously defined functions came from the treatment of contact labeling, and the placement of the completed substrate networks in extracted views. The substrate contacts previously used labels placed inside the contact layer for net naming. Silencer! Extracted needed the functions redefined in order to check the connectivity of the substrate contact layers. These net names are then used again when the substrate network is added back in to the extracted view. An extracted view will not simulate if the last extracted time stamp is different than the last modified time stamp. The substrate network placement now includes code to modify both time stamps for the extracted view. The new functions still allow for the previous Silencer! flow to be performed on the layout view.

B.3 Various Adjustments

There are few other adjustments made to the Silencer! tool which did not fall into any of the previous sections. This section briefly describes some of the issues with the execution of Silencer!, and the various adjustments done to try and account for them.

Negative resistors can appear in a substrate network due to round off errors in the calculations of a substrate network extraction program. Tighter extraction options will reduce the number of these negative resistors found in a substrate network, but also slow down the execution time. The negative resistors are usually supposed to be very large resistors. The removal of the negative resistance values

from the network is desirable over having negative resistors in a simulation. For this reason, the epic2spectre.rb script has been adjusted to replace all negative resistors with a '-' value. This is interpreted by the substrate network placements scripts as a place holder, and no actual resistor is placed. After the placement it will be obvious how many negative resistors were created by the number of holes in the triangle shaped substrate network.

The minimum panel size is one of the EPIC settings adjusted in the 'Models & Options' menu. If this option is set too large, very large substrate networks can take much longer to simulate as the contacts are being divided into an excessive number of panels. If this option is set too high, EPIC will say it is re-adjusting the panel size to account for the smallest contact dimension, but really still fails. To help find this minimum panel size, the Perl script min_panelsize.pl was placed in the Silencer/Files folder. This script searches through the subsPorts.txt file and finds the minimum dimension in the contact geometries. The minimum panel size should then be set to this value or lower. Unfortunately the subsPorts.txt file is not created until after the EPIC input file has been created in Silencer!. For this reason, the 'LOCATE Substrate Ports' function may need to be run twice, once to determine the minimum panel size, then again after the minimum panel size has been adjusted in 'Models & Options'.

With the use of the Silencer! Extracted add-on, many contacts can now have the same name/node connection. When this happens, the network can be simplified by the parallel resistor combinations. For some configurations, the network reduction is significant (contacts connected to vss is a common example). A con-

dense_SCnetlist.pl Perl script was written to help reduce the substrate network while Silencer! is running. The script reduces the substrate network directly after the SCnetlist.txt has been created, and is called in the snc_model.rb script. A special version of this script is used for macro model 1. Not only is the network reduced for parallel contacts, but dummy resistors are added in for any missing branches. These dummy resistors are needed to correctly place the substrate network back in the Cadence environment.