A Static Verification Approach for Architectural Integration of Mixed Signal Integrated Circuits

R. Mukhopadhyay\textsuperscript{a,*}, A. Komuravelli\textsuperscript{a}, P. Dasgupta\textsuperscript{a}
S. K. Panda\textsuperscript{a}, S. Mukhopadhyay\textsuperscript{b}

\textsuperscript{a}Dept. of Computer Science and Engg., Indian Institute of Technology,
Kharagpur, India
\textsuperscript{b}Dept. of Electrical Engg., Indian Institute of Technology, Kharagpur, India

Abstract

In this paper we present a static method for verifying the proper integration of analog and mixed signal macro blocks into an integrated circuit. We consider the problem in a setting where there is no golden reference for verifying the validity of the interconnections between the blocks. The proposed verification methodology relies on an abstract modeling of the functional behavior of the blocks and a set of consistency criteria defined over the composition of these abstract models. A new formalism called Mode Sequence Chart (MSeqC) has been presented for capturing the behavior of the blocks at a level of abstraction that is suitable for interconnection verification. We present rules to compose the MSeqCs of each block in an integrated design and present three criteria that indicate possible interconnection faults. We present a tool called AMS-IV (AMS-interconnection verification) that takes the design netlist as input, the MSeqC model of each design block as reference, and tests the three criteria.

Key words: Static Verification, Design Integration, Mixed Signal Circuit, formal model.

1 Introduction

As designers attempt to integrate multiple pre-designed and pre-verified design blocks into an integrated circuit, the task of verifying that the integration has been done correctly

* Corresponding Author.
Email address: rajdeepm@cse.iitkgp.ernet.in (R. Mukhopadhyay).

Preprint submitted to Elsevier
has become a major challenge. The problem becomes significantly more complex when the component blocks are mixed-signal in nature, since mixed-signal simulation is prohibitively complex as compared to pure digital simulation.

In this paper we study the verification problem associated with integrating several complex mixed-signal circuit blocks into an integrated circuit. We have been studying this problem in the context of integrating power management units (PMUs) for portable devices, such as cell phones, PDAs and laptops. A typical PMU contains several linear drop out regulators (LDOs), a few buck regulators, a battery charger and several other blocks for bias generation, UVLO, etc, and digital control. Typically the manufacturers of such PMUs have existing design IPs for these components, and the main challenge is to integrate a combination of these components (as required by the customer) into an integrated PMU within very low time budget. The whole design fails if the components are not interconnected correctly, where interconnection (in the analog sense) also refers to proper polarity, drive strength, etc.

We require a functional specification as a reference in this verification problem, even if the goal is only to verify that the interconnections have been made correctly. This is due to the fact that there is a manual point of entry for the interconnections. There are broadly two practices for integrating multiple mixed-signal circuit blocks into an integrated chip. The most common form is to make the interconnections graphically using a schematic editor (such as Cadence Virtuoso Schematic Editor [26]). The other approach attempts to enter the interconnections textually, either directly using text formats (using HDLS such as Verilog[24], VerilogA [25] etc.) or indirectly using scripting languages. In both approaches the point of entry is human and therefore prone to similar types of errors. Therefore integration verification is not achievable without functional verification of the integrated design.

Formal verification methods for AMS designs based on equivalence checking [2, 15, 23], model checking [14, 10, 3, 16], theorem-proving [8, 12, 11] etc. do not address the problem of verifying integration of large integrated AMS designs. On the other hand run-time verification methods such as [4, 19, 17] require simulation of the integrated AMS design. Typically the integrated designs that we refer to are so large that adequate functional verification through mixed-signal simulation of the integrated netlist is infeasible in practice, as also observed by the author of [22]. At the same time the full functionality of the component blocks is typically not required for verifying the correctness of the integration. For example, it is possible to replace the component blocks with lightweight behavioral models and ramp up the simulation speed of the integrated design. In a related work we adopted this approach using Verilog A [25] models with reasonable success. Based on our experience, we have the following observations:

(1) Developing Verilog A models for complex blocks like buck regulators is non-trivial. It is possibly too much of an effort to develop executable behavioral models for all legacy designs of the components.

(2) Verilog A models are not suitable for static (formal) analysis. We believe that most interconnection errors can be detected statically against a well defined functional specification.
(3) The industry seems to be moving towards platform based design methodologies where third party design IPs are integrated at the physical level (post layout). It is unlikely that behavioral models for such components will be made available.

In this paper we present a new formalism, called mode sequence charts (MSeqC), for modeling the component blocks of an integrated circuit at a very high level of abstraction. There exists a number of formalisms to specify components of a system at various levels of abstractions. In [6] an automata based language for specifying software components have been proposed. An extension of hybrid automata [1] to model interaction among various hybrid systems is proposed in [18]. StateCharts [13], which is another formalism to model complex systems is based on extension of state machine models. Message Sequence Chart [20] is a graphical and textual language for specifying interaction among various communicating system components. Although our model is not the first model that attempts to capture the interaction among various interacting components (which are mixed signal circuits in our case) mode sequence chart captures a circuit specification exactly at the level of abstraction that is appropriate for carrying out our interconnection verification method. A mode sequence chart captures the macro modes at which a mixed signal block works, and the admissible transitions between these modes. Each state of the chart is annotated with a set of assumptions that are made about the environment when the component is in that mode, and a set of properties that the component guarantees while operating in that mode. There are several advantages of using this formalism for the verification problem addressed in this paper.

(1) Mode sequence chart is a very simple formalism as compared to existing formalisms for hybrid systems (such as hybrid automata [1], hybrid petri net [5]). It abstracts out the dynamics of a circuit at a particular mode of operation as a set of linear constraints over the interface signals (voltage, current for analog pins and logic level for digital pins).
(2) Mode sequence charts are amenable for static (formal) analysis in integration verification.
(3) Mode sequence charts can typically be developed from the specs of the block that it models. It is therefore quite useful for modeling third party design IPs for which adequate functional details are unavailable.

Each global state of the integrated circuit is a composition of compatible states of the mode sequence charts of the components. Interconnection errors manifest themselves in terms of incompatibility of local states corresponding to a desired global state, and in terms of unreachability of desired global states. The main contributions of this paper are as follows:

(1) We introduce mode sequence charts as a high level formalism to be used in interconnection verification.
(2) We present the composition rules for integrating the mode sequence charts of the components based on the interconnections entered by the designer in the schematic.
(3) We present three different formal criteria for verifying the correctness of the interconnections. Interconnection errors are typically manifested in terms of violations of these criteria.
We have developed a tool called Analog and Mixed Signal Interconnection Verifier (AMS-IV) Tool that essentially checks the three criteria and produce three corresponding reports. These reports are useful to determine whether the circuit has any interconnection fault.

In the next section we illustrate our basic idea of interconnection verification with a toy example. Section 3 formally presents mode sequence chart and Section 4 describes the interconnection model. In Section 5 we propose three interconnection fault detection criteria and explain them with examples. Section 6 presents the algorithm and implementation of the AMS-IV tool briefly. In Section 7 we present a case study on a voltage mode buck regulator circuit. Section 8 concludes the paper with a few possible topics for future research.

2 Basic Idea

This section illustrates the basic idea of our method of interconnection verification with a toy example.

Example 1 Consider a switched capacitor integrator of Figure 1.

![Switched Capacitor Integrator Circuit](image)

Fig. 1. A switched capacitor integrator circuit [21]

The input voltage \( V_{in} \) is fed to the operational amplifier through the input capacitor \( C_{in} \). The circuit works in two phases. In the first phase the \( S_{in:sig} \) and \( S_{out:gnd} \) switches are closed, and \( S_{reset}, S_{in:gnd}, \) and \( S_{out:sig} \) are kept open, thus charging \( C_{in} \). This is called the sampling phase. In the next phase, called the update phase, \( S_{in:gnd} \) and \( S_{out:sig} \) are closed and \( S_{reset}, S_{in:sig} \), and \( S_{out:gnd} \) are kept open, thereby charging \( C_{sum} \). In this phase the charge accumulated on \( C_{in} \) in the sample phase is passed onto \( C_{sum} \) thus changing the output voltage. It can be shown that for an ideal switched capacitor integrator

\[
V_{out:i} = V_{out:i-1} + V_{in:i}(C_{in}/C_{sum})
\]
where $V_{out}$ is the output voltage at the $i^{th}$ update phase.

A correct operation of the circuit requires correct inputs from the environment. If the switches are not closed and opened in the proper order, the switched capacitor integrator cannot operate properly. A mode sequence chart of the integrator circuit captures this requirement through admissible state transitions. Figure 3 shows a switched capacitor integrator circuit connected with a hypothetical controller. The mode sequence chart of both the modules are also shown. The boxes represent the states of the circuits. Figure 2 shows the basic structure of such a box, that we use throughout the paper to represent state or mode of a circuit. We call such a box the mode template.

<table>
<thead>
<tr>
<th>Name of the mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assume Constraints</td>
</tr>
<tr>
<td>1. $A_1$</td>
</tr>
<tr>
<td>2. $A_2$</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>n. $A_n$</td>
</tr>
<tr>
<td>Guarantee Constraints</td>
</tr>
<tr>
<td>1. $G_1$</td>
</tr>
<tr>
<td>2. $G_2$</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>n. $G_n$</td>
</tr>
</tbody>
</table>

Fig. 2. The Mode Template

There are three sections of a mode template. The top section is used to mention the name of the mode. The second and third sections are used to mention the list of assume and guarantee constraints of the mode. The constraints are presented in the form of a list. Each item of the list need to be ANDed to obtain the overall constraint. For example, in Figure 2 the overall assume constraint is $A_1 \& \& A_2 \& \& A_n$ and the overall guarantee constraint is $G_1 \& \& G_2 \& \& G_n$. Note that an item viz. $A_1$ can contain logical ORs.

In Figure 3, the directed edges represent admissible state transitions. The constraints mentioned in each state are the assumptions and guarantees over the environment at that state. For example, the assumption of the integrator circuit on its interface pins at OFF state is that $S_{\text{reset}}$ is high. Similarly at the OFF0 state the guarantee of the controller on its $sd$ pin is given as $sd$ i.e. $sd$ is high. Only assumptions of integrator circuit and guarantees of the controller are mentioned in the figure. Observe that in the mode sequence chart of the integrator circuit there is no edge between the OFF state (the initial reset condition of the circuit) and the UPDATE state. This correctly models the specification that the integrator circuit should not go to the update phase from the initial reset condition.

The connections shown in the figure are correct. If by mistake, pin $a$ is connected with pin $S_{\text{in: gnd}}$, pin $b$ with pin $S_{\text{in: sig}}$, pin $c$ with pin $S_{\text{out: sig}}$ and pin $d$ with pin $S_{\text{out: gnd}}$ the integrator circuit will fail to operate properly. This is because in presence of these wrong connections when the controller makes a state transition from its initial state to sampling state, $a$ and $c$ become high and $b$ and $d$ remain low. This in turn makes $S_{\text{in: gnd}}$ and $S_{\text{out: sig}}$ high and $S_{\text{in: sig}}$ and $S_{\text{out: gnd}}$ low. When $S_{\text{in: gnd}}$ and $S_{\text{out: sig}}$ are high the integrator circuit should be
Fig. 3. A switched capacitor integrator with a digital controller in its update state. But, it cannot move to the update state from its initial reset state as this transition is prohibited being an illegal transition. Similarly, it can be verified that the integrator circuit cannot transit to update state because of these wrong connections. Thus the integrator circuit can neither transit to SAMPLE or UPDATE state because of the mentioned wrong connections. In other words, both the SAMPLE and UPDATE states of the integrator are unreachable due to the wrong connection. □

3 Mode Sequence Chart (MSeqC)

In the absence of any reference for the correct integration of a set of pre-designed and pre-verified blocks into an integrated circuit, verification of the integration can be done in two broad ways, namely:

(1) Verify the functionality of the integrated design against the specification of the integrated circuit
(2) Check for inherent inconsistencies in the integrated design

The first task is expensive and often the verification coverage is low for large integrated circuits. Moreover, the specification of the functionality of the integrated circuit does not define the correct interconnection between the blocks, rather the correct interconnection leads to the satisfaction of the specification.

On the other hand, interconnection errors between blocks in mixed-signal designs almost always lead to inherent inconsistencies in the integrated design, which can be detected early without knowing the functionality of the integrated design. The beauty of this approach
lies in choosing a level of abstraction for specifying the functionality of the blocks. We shall show that many, if not most, of the interconnection errors manifest themselves as inherent inconsistencies if we use simple models of the blocks which capture the macro modes in which these blocks function.

Analog and mixed-signal circuits typically function in a set of operating modes. For example a voltage amplifier typically works in two distinct operating modes: \textit{OFF} and \textit{LINEAR}, where \textit{OFF} represents the cutoff state of the amplifier and \textit{LINEAR} represents its normal amplifying mode of operation. In each of these operating modes there are various assumptions and guarantees on the voltage and current values of its interface pins.

It is quite easy for a designer to specify the operating modes of a design block and to enumerate the assumptions and guarantees in each operating mode. For example, design specifications of a voltage amplifier routinely contain information like input common mode voltage range $V_{ICMR}$, input voltage range $V_I$ etc., which specify assumptions about the signals at the non-inverting and inverting pins. These assumptions are constraints in terms of the voltage at the inverting and non-inverting pins, which are to be guaranteed by the circuits interfacing with the amplifier to drive the amplifier into its linear mode of operation. Similarly, there are assumptions on input bias current, output load current of the amplifier etc.

Mode Sequence Charts (MSeqC) are simple finite state models of design blocks, where the states capture the macro modes of operation of a block and a state is annotated with the set of assumptions and guarantees that are expected when that block operates in that mode. We will show that many interconnection errors manifest themselves in the form of the following types of inconsistencies in the integrated design, when the blocks are modeled as MSeqCs. The integrated design model is obtained by a parallel composition of the MSeqCs of the blocks.

1. A reachable state of one MSeqC becomes unreachable in the integrated design model.
2. In a reachable state of the integrated design model, the assumptions required by one MSeqC is not guaranteed by the others.
3. The set of reachable states of the integrated design model does not cover all the specified reachable states (given separately). Also, we consider the case where the set of reachable states of the integrated design model contains one or more undesirable states (also given separately).

Our experience shows that the three criteria stated above cover most of the typical interconnection errors at the architectural level of large mixed signal integrated circuits. The formal definitions follow.

The specification $\mathcal{S}$ of a block $M$ is represented by a \textit{mode sequence chart}, which is a tuple $\mathcal{J} = < S, s, \tau, \mathcal{Z}, \mathcal{A}, \mathcal{L}, \mathcal{P}_A, \mathcal{P}_G >$ where

- $S$: set of discrete modes/states of the circuit block.
- $s$: initial state.
• \( \tau \subseteq S \times S \): transition relation between states.
• \( Z \): set of interface signals of the block.
• \( A \): set of attributes, namely, \{TYPE, NATURE, PRIM_SIGNAL, SHAPE\}
• \( \mathcal{L} : S \times Z \times A \rightarrow \text{value} \) where \( \text{value} \) is defined as in Table 1
• \( \mathcal{P}_A \): set of mode/state specific assumptions.
• \( \mathcal{P}_G \): set of mode specific guarantees.

Table 1
Attribute functions and possible values

<table>
<thead>
<tr>
<th>Attribute Functions</th>
<th>Range of Values</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TYPE</td>
<td>input, output, inout</td>
<td>It is the polarity of the pins.</td>
</tr>
<tr>
<td>NATURE</td>
<td>analog, digital</td>
<td>It is used to label the nature of the signal at the interface pin.</td>
</tr>
<tr>
<td>PRIM_SIGNAL</td>
<td>voltage, current</td>
<td>This is used to define the primary signal at a pin. For example, voltage is the primary signal at the output pin of a voltage amplifier, current is the primary signal at the output pin of a current bias block.</td>
</tr>
<tr>
<td>SHAPE</td>
<td>sinusoidal, ramp, square, sawtooth, steady, random</td>
<td>This is used to describe the signal shape of the primary signal at a pin. Random refers to no specific signal shape.</td>
</tr>
</tbody>
</table>

\( Z \) consists of two types of signals:

• \( Z_A \): set of analog/electrical signals.
• \( Z_B \): set of Boolean signals.

\( \mathcal{L} \) is a labelling function that allows us to specify some of the important attributes of the interface pins at every state. This is important, as the attributes of a pin may vary from state to state. For example, a controllable signal generator which produces a sawtooth wave in one state might produce a sinusoidal signal in another mode of operation. Table 1 shows the list of attribute functions and their meaning.

For a Boolean signal \( p \in Z_B \) voltage is the only important signal. \( p \) represents logic high at \( p \) and \( \neg p \) denotes logic low. In the following section we use \( q \) to denote electrical/analog signals, and \( p \) to denote Boolean signals.

The syntax of the properties in \( \mathcal{P}_A \) and \( \mathcal{P}_G \) are defined by the symbol PROP in the following grammar. The symbols && and || are used to denote the usual logical AND and OR opera-
tors respectively.

- **PROP** → PROP | B | PROP & PROP | PROP || PROP
- **CONS** → TERM REL TERM | CONS || CONS
- **B** → B || B | ¬B | p where p ∈ Z
- **TERM** → TERM OP TERM | v(q) | i(q) | f(q) | CONSTANT where CONSTANT ∈ R
- **REL** → > | ≥ | < | ≤ |
- **OP** → + | - | * | /

The meaning of the analog terms are as follows.

- \( v(q) \) : The voltage at \( q \) with respect to some reference node/ground
- \( i(q) \) : The current through \( q \) out of the module.
- \( f(q) \) : Frequency of the primary signal at pin \( q \). This can be used to specify a range of values e.g. \( 10K < f(q) < 1M \) or a constant value \( f(q) = 10K \) (for example a sinusoidal signal has constant frequency).

### 3.1 Example of Modeling Specification with Mode Sequence Chart

An essential module of any portable power management unit (PMU) is a battery charger. Fig 4 shows the block diagram of a battery charger, connected to a battery. A battery charger is a mixed signal circuit consisting of a digital controller and an analog driver circuit. The digital controller, which determines the mode of operation of the analog circuit, is a state machine whose state transitions are triggered by events from the analog part, that are defined over the voltage \( V_{\text{battery}} \) and current \( I_{\text{charger}} \). Depending on the state, termed \( \text{fsm\_state} \), of the digital controller the charging voltage \( V_{\text{battery}} \) and the charging current \( I_{\text{charger}} \) are set by the battery charger.

![Fig. 4. Block Diagram of a Battery Charger](image)

Fig 5 shows a typical charging profile of a battery charger. The charging profile consists of several distinct regions each of which corresponds to a distinct state of the battery charger. These are \( \text{OFF}, \text{precharge}, \text{constant current}, \text{constant voltage}, \text{maintenance} \) and \( \text{ldo mode} \).

A brief description of these modes are given below.
• **OFF Mode:** If the external power supply is not within its specified range, disabled (i.e. the enable signal is not asserted) it enters this mode of operation.

• **Precharge Mode:** In this mode, a constant current (say, 50 mA) is used to charge up the battery. The battery voltage increases with time in this state.

• **Constant Current Mode:** As the battery voltage crosses a predefined threshold (say, 3 V) the charger enters constant current mode, where *fullrate* current i.e. the maximum rated charging current is used to charge up the battery. Battery voltage increases and then saturates to a voltage level, called the termination voltage (say, 4.2 V for a 4.2 V Li-ion battery).

• **Constant Voltage Mode:** As the battery voltage reaches the termination voltage the charger enters this mode of operation. In this mode, the output voltage is kept constant (say, at around 4.2 V). The charging current falls in this state until it reaches a predefined level, called the end of charge condition.

• **Maintenance Mode:** In this mode of operation, charging is stopped, till the battery voltage drops to a specified level, called the *restart voltage*. The charger enters constant voltage mode again, to charge the battery to its termination voltage. This cycle continues unless the external power supply is removed and/or the battery is detached.

• **LDO Mode:** Battery charger enters this mode of operation when no battery is detected. In this mode, the charger circuit works like a linear regulator circuit, regulating the output voltage at some specified level (say 4.2 V).
Figure 6 is the pin diagram of the analog driver of the battery charger circuit. The analog driver BATT_ANA communicates with the digital part of the circuit with its interface pins. Usually in an industry standard battery charger circuit number of such pins is much more as there are some other modes of operation namely, Bad Battery Mode, USB Mode etc. and several programmable features are available. For example, the termination voltage, restart voltage level, precharge current, fullrate current etc. are programmable.

Figure 7 shows pictorial representation of the mode sequence chart for BATT_ANA. It has 6 states and 18 transitions, RESET being the initial state.

![Mode Sequence Chart](chart.png)

**Fig. 7. Mode Sequence Chart for the Analog part of the Battery Charger**

For two states viz. CONST_CURRENT and CONST_VOLTAGE, assumptions and guarantees are provided in a tabular form in Table 2 along with some of the attributes of the interface pins at these states.

### 4 Interconnection Modeling

In the previous section we have discussed how we model the design blocks. We also need a model to capture the interconnection between the design modules. In Figure 8 three blocks are shown to be connected.

To model the fact that pin a, pin b and pin c of M1, M2 and M3 are connected together we use the following three constraints.

\[
\begin{align*}
v(a) &= v(b) = v(c) \\
i(a) + i(b) + i(c) &= 0 \\
f(a) &= f(b) = f(c)
\end{align*}
\]
Table 2
Assume and Guarantee of CONST CURRENT and CONST VOLTAGE states

<table>
<thead>
<tr>
<th>STATE</th>
<th>ASSUME CONSTRAINT</th>
<th>GUARANTEE CONSTRAINT</th>
<th>ATTRIBUTE</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONST CURRENT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1. 4&lt;v(CHG\text{ }IN)&lt;6</td>
<td>1. i(out)=1</td>
<td>CHG\text{ }IN\text{ }TYPE = input</td>
<td></td>
</tr>
<tr>
<td>2. i(ibias1)=800n</td>
<td>2. \neg cv</td>
<td>en.NATURE = digital</td>
<td></td>
</tr>
<tr>
<td>3. i(ibias2)=500n</td>
<td>3. \neg eoc</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4. v(vref)=1.2</td>
<td>4. \neg ov</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5. en</td>
<td>5. 3&lt;v(ana\text{ }vdd)&lt;4.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6. \neg en.prechg</td>
<td>6. 3&lt;v(dig\text{ }vdd)&lt;4.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7. \neg en.do</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8. 3&lt;v(vbatt)&lt;4.2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9. 0&lt;i(ana\text{ }vdd)&lt;1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10. -3&lt;i(dig\text{ }vdd)&lt;3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11. 3&lt;v(out)&lt;4.2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CONST VOLTAGE</td>
<td></td>
<td></td>
<td>same as above</td>
</tr>
<tr>
<td>1. 4&lt;v(CHG\text{ }IN)&lt;6</td>
<td>1. v(out)=4.2</td>
<td>out.PRIM\text{ }SIGNAL = voltage</td>
<td></td>
</tr>
<tr>
<td>2. i(ibias1)=800n</td>
<td>2. cv</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3. i(ibias2)=500n</td>
<td>3. \neg eoc</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4. v(vref)=1.3</td>
<td>4. \neg ov</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5. en</td>
<td>5. v(ana\text{ }vdd)=4.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6. \neg en.prechg</td>
<td>6. v(dig\text{ }vdd)=4.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7. \neg en.do</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8. v(vbatt)=4.2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9. 0&lt;i(ana\text{ }vdd)&lt;1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10. -3&lt;i(dig\text{ }vdd)&lt;3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11. 0&lt;i(out)&lt;1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Fig. 8. Interconnection Model

where $v(a), i(a), f(a)$ are the voltage at pin $a$, current through pin $a$ out of the module and
frequency at pin $a$.

The first two equations are direct application of Kirchoff’s voltage law (KVL) and Kirchoff’s current law (KCL). The current directions shown in Figure 8 are as per the semantics (see section 3). Equation 4 models the fact the frequency of the signal of the connected pins must be same.

Thus, apart from the assume, guarantee constraints at each of the design blocks, we also have constraints due to the interconnections. We designate these constraints due to interconnection by $I$.

5 Interconnection Fault Detection Criterion and Problem Formulation

While connecting several modules to build up a larger integrated system a wrong connection can cause

- **Functional Error**: Incorrect functional behavior of the overall integrated system. This can happen due to any of the following reasons.
  - **Local Fault**: At least one module of the system is not able to behave properly i.e.
    - **Scenario I**: either the module is not able to exhibit one of its desired behavior and/or
    - **Scenario II**: forced to some unspecified state causing it to behave incorrectly.
  - **Global Fault**: Even if there is no local fault the overall behavior of the system can deviate from the correct system level behavior.
- **Performance Error**: Degradation of performance of the overall system with respect to the performance specification of the integrated system e.g. more power consumption, more noisy behavior etc.

A wrong connection can cause any or a combination of the above scenarios. Detection of any such faulty scenario indicates presence of one or more number of wrong interconnections in the integrated circuit.

In the following section we illustrate how functional errors can be detected in our framework where every module comes with a mode sequence chart.

5.1 Detection of Scenario I

The states of a mode sequence chart represent the operation modes of a circuit. Therefore, failure to reach any state of a module means that the circuit cannot exhibit the functionality associated with that state. In other words, when one of the states of an mode sequence chart of a module becomes unreachable, we can say that due to the presence of some wrong interconnection(s) the module can never enter that operating mode (modeled as a state of
Interconnection Fault Detection Criterion I: Detection of at least one unreachable state of mode sequence chart of any module $M$ in an integrated system suggests presence of at least one interconnection fault.

The following example illustrates one scenario where due to interchange of two nets module $M_2$ cannot exhibit its desired behavior at state $OP_3$ (see Figure 10) i.e. state $OP_3$ is unreachable. The connections shown in Figure 9 are correct. Both module $M_1$ and $M_2$ have certain assumptions and guarantees at its different states. $COP_0$ and $OP_0$ are the initial states of $M_1$ and $M_2$. Only those guarantees of module $M_1$ and assumptions of $M_2$ which would be sufficient to explain are mentioned in Figure 9 and Figure 10.

It may be observed that for the given connections of Figure 9 states $COP_0$ and $OP_0$, $COP_1$ and $OP_1$, $COP_2$ and $OP_2$, $COP_3$ and $OP_3$ are compatible with each other. By compatible we mean that both constraints and attributes of the two states are compatible as defined below.

For a system comprised of $N$ modules $M_1, M_2,...,M_N$ a constraint compatible tuple is defined as below. Let $S_{M_i}$ be the set of states of the $M_i$ module.

**Definition 1 Constraint Compatible Tuple:** A system state $\alpha = (\alpha_1, \alpha_2,...\alpha_i,...\alpha_N)$ is said to be constraint compatible if and only if the following formula is satisfiable.

$$\psi_1 = \bigwedge_{i=1}^{N} P_A(\alpha_i) \bigwedge_{i=1}^{N} P_G(\alpha_i) \bigwedge I$$

where $\alpha_i \in S_{M_i}$ and $I$ is the interconnection constraint.

For example, at $COP2$ the guarantee at output pin $a$, $(V(a) = 0.3)$ and assumption at $OP2$ at the connected input pin $x$, $((V(x) > 0) \& \& (V(x) < 0.7))$ are satisfiable. Similarly, guarantees at pins $b$ and $c$ are satisfiable with the corresponding connected input pins $y$ and $z$ respectively. It can be seen that at $COP2$ the guarantee at output pin $a$, $(V(a) = 0.3)$ is not satisfiable with the assumption at $OP3$ at the connected input pin $x$, $((V(x) > 0.8) \& \& (V(x) < 3.0))$. Hence, $COP2$ and $OP3$ are not constraint compatible. Note that for this example $N = 2$.

**Definition 2 Attribute Compatible:** A system state $\alpha = (\alpha_1, \alpha_2,...\alpha_i,...\alpha_N)$ where $\alpha_i \in S_{M_i}$ is said to be attribute compatible if and only if all the attributes of all the connected pins of the $N$ modules match according to the following rule. Consider two connected pins $p_a$ and $p_b$. 

14
• Type attribute is not matched if \(\text{TYPE}(p_a) = \text{TYPE}(p_b) = \text{output}\). In all other combinations it matches.

• Nature attribute is matched if \(\text{NATURE}(p_a) = \text{NATURE}(p_b)\).

• Shape attribute is matched if \(\text{SHAPE}(p_a) = \text{SHAPE}(p_b)\) or one of the shapes is random.

• \text{Prim-Signal} attribute is matched if \(\text{PRIM-SIGNAL}(p_a) = \text{PRIM-SIGNAL}(p_b)\).

Definition 3 Consistent System State: A system state \(\alpha = (\alpha_1, \alpha_2, ..., \alpha_i, ..., \alpha_N)\) where \(\alpha_i \in S_{M_i}\) is said to be consistent if and only if it is both constraint compatible and attribute compatible. Otherwise, it is called an inconsistent system state.

Thus, \((\text{COP}_0, \text{OP}_0);(\text{COP}_1, \text{OP}_1);(\text{COP}_2, \text{OP}_2);(\text{COP}_3, \text{OP}_3)\) are consistent system states. Tuples like \((\text{COP}_0, \text{OP}_1);(\text{COP}_3, \text{OP}_2)\) etc. are inconsistent system states. Note that \((\text{COP}_0, \text{OP}_0), (\text{COP}_1, \text{OP}_2)\) etc. represent a system or global state.

Definition 4 Set of Consistent System States: It is the set of all consistent system states, represented by \(S_{\text{consistent}}\). \(S_{\text{consistent}} \subseteq \times_{i=1}^{N} S_{M_i}\)

A consistent system state represents a global state in which the system can possibly stay. But every consistent global state is not necessarily a valid state of the system. The following example illustrates this.

Consider Figure 10. Pin \(a\) and \(b\) of module \(M_1\) are connected with pin \(y\) and \(x\) of \(M_2\) respectively. Due to the wrong connection the consistent system tuples are \((\text{COP}_0, \text{OP}_0), (\text{COP}_1, \text{OP}_2), (\text{COP}_2, \text{OP}_1)\) and \((\text{COP}_3, \text{OP}_3)\). When \(M_1\) transits from \(\text{COP}_0\) to \(\text{COP}_1\), module \(M_2\) transits from \(\text{OP}_0\) to \(\text{OP}_2\). Although \(M_1\) can make a transition from \(\text{COP}_1\) to \(\text{COP}_3\) (with an intent to drive \(M_2\) to \(\text{OP}_3\)), \(M_2\) cannot make a transition from \(\text{OP}_2\) to \(\text{OP}_3\). This is because mode sequence chart of \(M_2\) does not allow this transition. It can be verified that no such combined transitions exist in \(M_1\) and \(M_2\) that can drive \(M_2\) to
OP3. Thus OP3 is unreachable making (COP3, OP3), which is a consistent system state, an unreachable system state (under the given interconnection). Figure 11 shows the product graph of the two underlying graphs of the mode sequence charts of M1 and M2. Note that there is no path from the initial reachable system state (COP0, OP0) to (COP3, OP3), making the latter an unreachable system state.

**Definition 5 Consistent Transition:** A transition from one consistent system state $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_i, \ldots, \alpha_N)$ to another consistent system state $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_i, \ldots, \sigma_N)$ is consistent iff for all $i$, either $\alpha_i = \sigma_i$ or $(\alpha_i, \sigma_i) \in \tau_i$, where $\alpha_i \in S_{M_i}$, $\sigma_i \in S_{M_i}$ and $\tau_i$ is the transition relation of $M_i$.

$\sigma$ is the successor system state of $\alpha$.

**Definition 6 Set of Reachable System States ($S_{\text{reachable}}$):**

(1) $s = (s_1, \ldots, s_N)$, is a reachable initial system state if and only if it is a consistent state
(2) All reachable states from the initial reachable system state are reachable system states.

**Definition 7 Reachable Local State:** A state $\sigma_i$ of module $M_i$ is said to be reachable if and only if there exists at least one reachable system state $\sigma = (\sigma_1, \ldots, \sigma_i, \ldots, \sigma_N)$.

In the example of Figure 10, $S_{\text{reachable}} = \{(COP0,OP0), (COP2,OP1), (COP1,OP2)\}$ with $(COP0,OP0)$ the initial reachable system state. Both COP3 and OP3 are unreachable local states of modules M1 and M2 respectively.

### 5.2 Detection of Scenario II

Every circuit is designed with some assumptions on the environment. A voltage regulator circuit’s regulation degrades as the load current exceeds its specified (i.e. assumed) value, a switched capacitor integrator circuit expects that its switches be closed in a proper sequence, a differential amplifier cannot operate as per its specification if the tail bias current is different from its assumed value. A circuit fails to function in an expected fashion if its assumptions over the environment is violated. This gives us the second interconnection fault detection criterion.

**Interconnection Fault Detection Criterion II:** The assume constraints on the interface pins of every module in an integrated system must always be implied by the guarantee constraints of the other connected modules (i.e. the environment) at its interface pins. Mathematically, the following formula has to be valid.

$$\psi_2 = (G_{\text{env}} \implies A_{M_i}) \land I$$  \hspace{1cm} (6)

where $G_{\text{env}}$ represents the guarantee of the environment that interacts with module $M_i$ and $A_{M_i}$ is the union of assumptions on its interface pins at each state. $I$ is the interconnection model.

Following example illustrates this fact.

In Figure, 12 M1 and M2 are two connected modules. The connections between V1 and IN-, and V2 and IN+ are wrong connections. Note that, again some of the guarantees of M1 and assumptions of M2 are mentioned as it is sufficient to explain the objective. It can be verified that all the states are reachable, the reachable system states being (T0,S0) and (T1,S1). But the connection between V1 and IN− can drive M2 to some unspecified region. This is because $V(V1) \in \{0, (1 − 3)\}$, i.e. it can be either 0V or take any value between 1 to 3V in its two states, whereas $V(IN−) \in \{0, 1.2\}$ i.e. it assumes either 0V (in S0 state) or 1.2V (in S1 state). Hence, there is a possibility that module M1 drives M2 with a voltage at IN− which is not expected by M2. In other words, the assumptions of M2 at pin IN- is not implied by the guarantees of M1, which forms M2’s environment in this case.
5.3 Detection of Global Fault

It may happen that none of the above two scenarios of local fault takes place but still one or more of the system behaviors deviate from its expected behavior. Consider the following example of a programmable regulator.

Example 2 Depending on the value of Select[1:0] the regulator circuit operates at four different voltages. The operation states of the circuit can be described with Table 3. When Select[1:0] = 00 and d0 = 1 the Switch block connects the v0 pin with the vfb pin. The circuit output settles at 1.5V, such that v0 is at 1.2V making the error at the input of the error amplifier 0. Voltage at the other outputs of the Resistance Feedback network is less than 1.2V. Similarly, when Select[1:0] = 01. d1 = 1. This makes s1 = 1 (for the correct connections shown in Figure 13), and the Switch block connects the v1 pin with the vfb pin. The output voltage this time settles at 1.8V, such that v1 is at 1.2V making the error at the input of the error amplifier 0. v0,v2 and v3 are at different voltage levels other than 1.2V.

Now, if by mistake d0 is connected with s1 and d1 is connected with s0, the circuit behavior will change. At Select[1:0] = 00, the output voltage settles to 1.8V instead of 1.5V, and at Select[1:0]=01, the output voltage settles to 1.5V, instead of 1.8V. So, we can say that due to the wrong connection, the overall system behavior has changed. □

But note that this wrong connection cannot be detected with any of the above two criteria. The four rows of table 3 describe the four possible reachable system states. These are the specified system states of the system. In general, due to interconnection faults many unwanted system states can become reachable and desired system states become unreachable. Thus, as part of global property if $S_{specified}$ and $S_{unwanted}$ represent the set of specified (desired set of reachable states) and the set of unwanted system states respectively we can state the third
Fig. 13. Schematic of a programmable regulator

<table>
<thead>
<tr>
<th>Select[1:0]</th>
<th>d0</th>
<th>d1</th>
<th>d2</th>
<th>d3</th>
<th>s0</th>
<th>s1</th>
<th>s2</th>
<th>s3</th>
<th>v0 (V)</th>
<th>v1 (V)</th>
<th>v2 (V)</th>
<th>v3 (V)</th>
<th>Vout (V)</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1.2</td>
<td>1.2−</td>
<td>1.2−</td>
<td>1.2−</td>
<td>1.5</td>
</tr>
<tr>
<td>01</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1.2+</td>
<td>1.2</td>
<td>1.2−</td>
<td>1.2−</td>
<td>1.8</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1.2+</td>
<td>1.2+</td>
<td>1.2</td>
<td>1.2−</td>
<td>2.0</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1.2+</td>
<td>1.2+</td>
<td>1.2+</td>
<td>1.2</td>
<td>2.5</td>
</tr>
</tbody>
</table>

interconnection fault detection criterion as follows.

**Interconnection Fault Detection Criterion III:** The final set of determined reachable system states $S_{reachable}$ must include all of the specified system states $S_{specified}$ and none of the unwanted system states $S_{unwanted}$ i.e.

$$\psi_3 = (S_{specified} \subseteq S_{reachable}) \land (S_{unwanted} \not\subseteq S_{reachable}) \quad (7)$$

must hold true. The problem formulation follows immediately from the above discussion.

### 5.4 Problem Formulation

*Given N modules $M_1, M_2...M_N$, their specifications $S_1, S_2, ...S_N$ and a design schematic $T$ (in spectre or spice netlist form) where each specification $S_i$ is a mode sequence chart, verify truth of all three interconnection fault detection criteria.*
6 AMS-IV Tool: Algorithm and Its Implementation:

In this section we present the algorithm and discuss some of the implementation issues. Algorithm 6.1 is the pseudo code of the algorithm. The algorithm essentially checks the three criteria discussed in previous section. It starts with the initialisation of the internal data structure.

Algorithm 6.1 [Interconnection Verification Algorithm]

Input:
1. A netlist of the design $T$, comprised of $N$ modules $M_1, \ldots, M_N$.
2. A model file $M$ with descriptions of mode sequence chart for each of the $N$ modules.
3. Specified Set of reachable system states $S_{specified}$ and unwanted system states $S_{unwanted}$.

Output:
1. Set of reachable system states $S_{reachable}$
2. Local states at which $\psi_2$ could not be satisfied.
3. Set of unreachable system states specified in $S_{specified}$.

Step 0: //Reads information from netlist $T$ and the Model file $M$ into internal data structure
Initialize($T,M$);

Step 1: // Start of Criterion I Checking
formConsistentSystemState($G_{arr}$,$index$,$Tuple$,$tuple_index$);

Step 2: formGraph($S_{consistent}$,$G_{product}$);

Step 3: // Form $S_{reachable}$
foreach $s \in S_{initial}$
$S_{reachable} = traverse(s,G_{product})$;

Step 4: // Start of Criterion II Checking
foreach local reachable state $\sigma_i$
if(checkImplication($\sigma_i$))
$\sigma_i$ does not violate criterion II
else
$\sigma_i$ violates criterion II

Step 5: // Start of Criterion III Checking
check_criterion_III($S_{reachable}$,$S_{specified}$);

The next step is to check Criterion I. formConsistentSystemState() is a recursive function that generates sub-tuples and checks both attribute compatibility and constraint compatibility of it using attribute_compatible() function and constraint_compatible() function respectively. Only if both are passed, another state from the next module is introduced to continue checking, otherwise that sub-tuple is discarded for any further checking. Note that a sub-
tuple of length equal to the number of modules in the integrated system is a possible system state. If such a sub-tuple passes both the compatibility checks it is pushed into $S_{consistent}$, the set of consistent system states. Algorithm 6.2 is the description of the function $formConsistentSystemState()$.

Algorithm 6.2 $[formConsistentSystemState(G_{arr}, index, Tuple, tuple\_index)]$

```plaintext
if(index == number of modules)
begin
  if(!attribute\_compatible(Tuple))
    print "ATTRIBUTE CHECK FAILED";
  else
    begin
      if(constraint\_compatible(Tuple))
        push Tuple to $S_{consistent}$
      else
        print "CONSTRAINT CHECK FAILED";
    end
  end
else
begin
  for(each state $S_i$ of G_{arr}[index])
  begin
    Tuple[index] = $S_i$;
    if(!attribute\_compatible(Tuple))
      print "ATTRIBUTE CHECK FAILED FOR THE SUB TUPLE";
    else
      begin
        if(constraint\_compatible(Tuple))
          formConsistentSystemState(G_{arr}, index + 1, Tuple, tuple\_index + 1)
        else
          print "CONSTRAINT CHECK FAILED FOR THE SUB TUPLE";
      end
  end
end
return;
```

attribute\_compatible() checks according to the attribute compatible rules as given in Definition 2. Algorithm 6.3 describes the constraint\_compatible() function.
Algorithm 6.3 \([\text{constraint\_compatible}(\tau)]\)

**Step 0:** \(\mathcal{B} = \mathcal{P}_A(\sigma_1) \land \mathcal{P}_G(\sigma_2) \land ... \land \mathcal{P}_A(\sigma_N) \land \mathcal{P}_G(\sigma_N);\) // construct Boolean Expression

**Step 1:** // Check whether \(\mathcal{B}\) is satisfiable
while(assignment = get\_next\_assignment(\(\mathcal{B}\)))
    if(check\_feasibility(assignment, \(\mathcal{I}\)))
        return true;

**Step 2:** return false;

In step 0 we form a Boolean expression \(\mathcal{B}\) from the guarantee and assume constraints of each of the state appearing in the tuple \(\tau\). We replace each TERM (see Section 3) of a constraint with a Boolean variable. For example, an assume constraint like \((V(a) + V(b) < 2) & & (en))\) is converted to \((c1 & & c2)\) where both \(c1\) and \(c2\) are Boolean variables. In step 1 \textit{get\_next\_assignment()} is used to find out an assignment that satisfies \(\mathcal{B}\). Internally it uses \textit{zChaff}\ [27]. Then this assignment is validated under the given interconnection constraint \(\mathcal{I}\) using \textit{check\_feasibility()}, which uses GLPK \[9\] internally. Whenever we find one valid assignment we return true indicating that \(\tau\) is constraint compatible.

After obtaining the set of consistent system states \(S_{\text{consistent}}\) (see Definition 4) in Algorithm 6.1 we form the product graph following the definition of \textit{consistent transition} (see Definition 5). Finally we traverse the product graph \(G_{\text{product}}\) from each of the initial consistent system states to form the final set of reachable system states \(S_{\text{reachable}}\). After checking Criterion I, the second criterion is detected using \textit{check\_implication()}\. Algorithm 6.4 presents the steps of this function.

Algorithm 6.4 \([\text{check\_implication}(\sigma_i, S_{\text{reachable}})]\) // \(\sigma_i \in S_{M_j}\)

**Step 0:** \(\mathcal{A} = \text{false};\)

**Step 1:** foreach module \(M_K\) connected to \(M_j\)
begin
    foreach \(\tau \in S_{\text{reachable}}\) containing \(\sigma_i\)
    begin
        \(\mathcal{X} = \text{simplify } \mathcal{P}_A(\sigma_l) \text{ where } \sigma_l \text{ appears in } \tau \sigma_l \in S_{M_k},\) by removing constraints over all those pins which are not connected with \(M_j\).
        \(\mathcal{A} = \mathcal{A} \lor \mathcal{X};\) // Forming the assume constraints
    end
end

**Step 2:** \(\mathcal{G} = \mathcal{P}_G(\sigma_i);\) // Guarantee constraint of \(\sigma_i\)
Step 3: $B = \neg (G \implies A)$;

Step 4: while($assignment = get\_next\_assignment(B)$)
  
  if(check\_feasibility($assignment, I$))
    
    return false;

Step 5: return true;

As can be seen from the steps of the algorithm we do not check criterion II i.e. $\psi_2$ directly as given in Equation 6. This is because checking directly $\psi_2$ is an expensive operation in most practical situations due to the presence of a large number of OR operations, hence increasing the number of possibilities to check. We circumvent this problem by dividing the problem into validity checks of smaller implications for all the local reachable states, computed after Criterion I phase.

Let us assume the set of reachable system states, obtained after criterion I phase for the system shown in Figure 14 be $S_{\text{reachable}} = \{(A_1, B_1, C_1), (A_2, B_1, C_1), (A_2, B_3, C_2)\}$.

![Fig. 14. Three connected Modules](image)

Now checking validity of criterion II for module M1 i.e. validity of $F = (G_{\text{env}} \implies A_{M_1}) \land I$ where $G_{\text{env}}$ is the guarantee of the environment interacting with M1 is equivalent to check validity of the following implications:

1. $G(B_1) \implies \mathcal{A}(A_1) \lor \mathcal{A}(A_2)$
2. $G(B_3) \implies \mathcal{A}(A_2)$

Note that, in the above implications we have only chosen guarantee of the M2 module as it is the only module connected directly with M1. Also, since $B_1$ appears with $A_1$ and $A_2$ in two reachable system states, guarantee of $B_1$ needs to imply either assumption of $A_1$ or $A_2$. Since, $A_1$ does not appear with $B_3$ in any reachable system state there is no need to include $A_1$ as guarantee of $B_3$ will anyway fail to imply assumption of $A_1$. Such state wise checking of criterion II and using the reachable system state information obtained after criterion I enables quick termination of criterion II.

After checking the first two criteria, the third criteria is checked. Three outputs are provided to the user as an indication of any possible interconnection fault. They are $S_{\text{reachable}}$, local reachable states at which criterion II fails and which of the specified system states were unreachable, and which of the undesired states became reachable.
7 Case Study

Voltage mode buck converter is a well known voltage regulator circuit [7]. Figure 15 shows the schematic of the buck regulator, we take up as our case study. Our main objective is to demonstrate the efficiency of our method to detect (in terms of failure of any of the three criteria) presence of interconnection faults and how detail modeling of the component blocks can help to increase such fault detection.

In general, an interconnection mistake can happen due to leaving one or more nets hanging, shorting with some other net or by connecting at wrong points. Our tool AMS-IV flags warning messages for hanging nets.

Each component of the buck regulator is modeled using mode sequence chart. Detail modeling of these component blocks increases the fault coverage of the tool. In Section 7.1 we have...
discussed with an example how a simpler (less detail) model of a component can affect the fault coverage. The results shown in Table 4 are carried out with a very detail model of each of the component. The total state space, calculated by taking cartesian product of the set of states of the underlying graphs of each of the block’s mode sequence charts is 28 million. It should be borne in mind that depending on specification number of states in each MSeqC can change, and that changes the total state space and hence the run time for the tool.

Table 4 is a list of some of the interconnection faults and which of the criteria failed to indicate presence of the faults. User time is reported in the last column of the table. The first row corresponds to the scenario where no fault has been injected.

In another experiment we injected faults of different classes and observed the fault coverage for each such fault-class. There are typically 5 classes of faults in a circuit like BUCK regulator. They are

1. **Power Class**: In this class all the power nets belong e.g. the nets connected with power supply and ground pins.
2. **Bias Class**: Nets connected with the pins, which are used for bias supply (reference voltage, bias voltage, bias current) belong to this category.
3. **Control Signal Class**: These are the digital nets used for controlling purpose. For example, enable, clock signals belong to this class.
4. **Digital Signal Class**: These are the other digital nets. In Figure 15 FS1, FS2 connected with the FSEL block are two such nets.
5. **Analog Signal Class**: These are the analog nets other than the nets that belong to power and bias classes. In Figure 15 such pins are PWM of Ramp clk block, VOUT of the EA (error amplifier) block etc.

We injected single faults by interchanging two nets of same class at a time. For example, we connected the PWM of Ramp clk block to the VIN N of the COMP block and connected the VOUT pin of the EA block with the VIN P of the COMP block to inject a fault of the Analog Signal class. We injected all such possible single faults of each class and observed the fault coverage for each class. Table 5 reports the fault coverage results we obtained. It shows an overall fault coverage of 92%.

### 7.1 Writing Specifications

The success of the method depends on correct and detail specification of the components. Incomplete specifications like less number of modes of operations and/or relaxed constraints can make the method miss interconnection faults. For example, the deadband block (**DBand** of Figure 15) is mainly a delay block that produces two clock signals namely CLK HV and CLK LV with some delay with respect to the input clock signal i.e. D CL. Figure 16 describes the behavior of the deadband block.

Figure 17 shows one possible MSeqC for it. Since the outputs O1 and O2 can be either in
Table 4
Fault Detection with AMS-IV Tool

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Manifestation</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO FAULT</td>
<td>Passed with 49 reachable system states.</td>
<td>840</td>
</tr>
<tr>
<td>a97 and VError of Compensator block are swapped</td>
<td>Failed. One reachable system state only</td>
<td>792</td>
</tr>
<tr>
<td>VREF1p2 and VREF2p4 of Ramp,clk are swapped</td>
<td>Failed. One reachable system state.</td>
<td>11</td>
</tr>
<tr>
<td>Gate HV and Gate LV are swapped</td>
<td>Failed. 33 reachable states, with no state starting with DB3</td>
<td>742</td>
</tr>
<tr>
<td>SOFT and a97 are swapped</td>
<td>Failed. One reachable state only.</td>
<td>780</td>
</tr>
<tr>
<td>A_VDD of DP block shorted to ground</td>
<td>Failed. One reachable system state</td>
<td>508</td>
</tr>
<tr>
<td>PWM and VError of COMP are interchanged</td>
<td>Failed. One reachable system state</td>
<td>7</td>
</tr>
<tr>
<td>VError shorted to A_GND</td>
<td>Failed. No reachable system state</td>
<td>6</td>
</tr>
<tr>
<td>Latch output is disconnected from DBand.</td>
<td>Passed</td>
<td>1147</td>
</tr>
</tbody>
</table>

00, 10 and 11 state (see Figure 16), and the input IN can be either 1 or 0 and hence no constraint on IN has been mentioned.

Since the model does not have any constraint on the input pin IN, such a specification cannot
Table 5
Fault Coverage with AMS-IV Tool

<table>
<thead>
<tr>
<th>Fault Class</th>
<th>No. of Faults</th>
<th>Fault Coverage (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power</td>
<td>6</td>
<td>100</td>
</tr>
<tr>
<td>Bias</td>
<td>153</td>
<td>100</td>
</tr>
<tr>
<td>Control</td>
<td>3</td>
<td>100</td>
</tr>
<tr>
<td>Digital</td>
<td>120</td>
<td>83</td>
</tr>
<tr>
<td>Analog</td>
<td>3</td>
<td>100</td>
</tr>
</tbody>
</table>

Fig. 16. Timing Diagram of the Deadband block

Fig. 17. Simple Mode Sequence Chart of the Deadband block

capture any wrong connection at this pin by checking constraints. Also note that it is not a wrong description, it is just a less detailed model that does not capture all useful information.

The same block can be specified using a more detail MSeqC as shown in Figure 18.

Unlike the simple model this mode captures two important information. One is the sequence among the different operation modes and the other one is constraint on the input pin IN.
With this specification we observed criterion III failing (see the last row of Table 4) when the comparator output was connected to the Deadband block’s IN pin and the output of Latch was disconnected from IN. One of the specified system state became unreachable due to this wrong connection. With the simple MSeqC of Figure 17 no such criterion fails. Also note that criterion II also fails in this case indicating potential wrong connections in the design.

8 Conclusion and Future Work

We have presented a static method for interconnection verification of integrated AMS circuits. The method is particularly useful for large designs where simulation is expensive as it helps to detect interconnection faults before going for the more comprehensive simulation based verification method. The method presented in this paper helps to detect interconnection faults without indicating the fault locations, except for scenarios where attribute checking fails. Our future work is directed towards identification of the faults. Any large design is usually built hierarchically. We intend to extend our tool to accept such hierarchical specification of components and then verify interconnections at the various levels of design hierarchy. Extraction of the abstract models of the design blocks from the corresponding netlists is another challenging future work.

References


