ELECTRONIC WORKSHOPS IN COMPUTING

Series edited by Professor C.J. van Rijsbergen

**Mary Sheeran, Chalmers Technical University, Sweden, and Satnam Singh,
University of Glasgow, UK (Eds)**

Designing Correct Circuits

Proceedings of the 3rd Workshop on Designing Correct Circuits
(DCC96), Båstad, Sweden, 2-4 September 1996

ISBN 3-540-76102-0

Paper:

A Symbolic Modelling Approach for the Formal Verification of Integrated Mixed- Mode Systems

Stefan Hendricx and Luc Claesen

Published in collaboration with the
British Computer Society

©Copyright in this paper belongs to the author(s)

A Symbolic Modelling Approach for the Formal Verification of Integrated Mixed-Mode Systems

Stefan Hendricx

IMEC vzw
Kapeldreef 75, 3001 Heverlee, Belgium

Luc Claesen

IMEC vzw/Katholieke Universiteit Leuven
Kapeldreef 75, 3001 Heverlee, Belgium

Abstract

In this paper, a symbolic modelling approach is presented for the formal representation and verification of mixed analog/digital systems. The proposed modelling technique can be incorporated in the SFG-Tracing – a pragmatic methodology originally aimed at the formal verification of digital (VLSI) designs. Existing symbolic analysis and reasoning techniques can be employed to analyse the digital subsystems of a mixed analog/digital design. The development of appropriate, symbolic models to express the functional behaviour of the individual analog components, ultimately enables us to exploit a symbolic evaluation or simulation tool to formally verify the overall functional behaviour of a mixed-mode system.

1 Introduction

Even though conventional, simulation-based verification techniques are still the most widely used means for verifying the correctness of a given design or implementation [15], the benefits of formal verification [12] arouse the interest of almost every researcher in the field of verification. As a consequence, a wide variety of approaches has been devised in the past, covering formalisms such as Boolean Logic [7, 9], Higher Order Logic or HOL [1, 18], Boyer-Moore Logic [20, 23, 24] and several others.

Up to now, the majority of these formal hardware verification methods was dedicated to the verification or validation of digital systems [2, 3, 4, 5] and only occasionally, research efforts have been made towards the formal modelling and symbolic verification of mixed analog/digital systems. One attempt to formalise mixed-mode systems is illustrated by Boute in [11] and concerns a functional and declarative language¹ called *FunMath*. Hellestrand likewise discusses a functional (specification) language applicable to mixed analog/digital systems [19]. Although promising, both functional formalisms suffer from a number of disadvantages: first of all, they are not decidable and don't have a mechanised implementation. As a consequence, it is difficult – if not impossible – to mould them into efficient automatic verification tools. Secondly, extensive user-interactions and manipulations, as well as a high level of specialisation and expertise, are required of the user when applying functional languages for verification purposes. On average, hardware designers will therefore be refrained from incorporating such functional specifications in their design process.

This paper presents a more pragmatic approach for the symbolic modelling and formal verification of mixed-mode systems. The proposed modelling technique can be incorporated in the SFG-Tracing verification methodology. Before briefly discussing the SFG-Tracing methodology and its modifications towards the formal verification of mixed-mode systems in section 3, section 2 will in short illustrate the growing tendency of integrating both analog and digital components on a single VLSI chip. In sections 4 and 5, the key concept of our new approach is considered: a symbolic modelling of analog macro building blocks. Future research activities are sketched in section 6. General conclusions are presented in section 7.

¹ According to R. Boute, the ultimate objective of his functional calculus is to enable the formal specification and verification of electrical, as well as mechanical, optical and algorithmic systems.

2 Integrated mixed-mode systems

Even though the data-processing essential to many electronic hardware systems is generally taken care of by digital subsystems, most applications additionally require analog interfaces with their physical environment [21]. In the past, such analog systems for interface purposes were usually implemented as so-called ‘glue’ chips – i.e. integrated (analog) circuits placed peripheral with respect to a central digital chip. During the last few years however, an ever growing tendency can be observed towards the integration of both analog and digital subsystems onto a single piece of silicon. This tendency can be largely attributed to the appealing characteristics – significant reduction of size and power consumption, high system reliability and improved speed performance – of integrated mixed-mode systems.

A wide variety of applications benefits from the characteristics of integrated mixed-mode systems: DSP, GSM, biological implantable circuits and others. The SmartPen™ [14], a novel application developed at Imec in co-operation with SmartPen Inc., also makes use of the previously mentioned advantages. Basically, the SmartPen™ is a new computer input device. Although resembling an ordinary pen, the SmartPen™ is a stand-alone system that communicates the written gestures to a computer via an RF-transmitter. The transmitted, digitised information can be stored and a reconstruction of the written text can be visualised on a computer screen. In order to imitate the appearance of a conventional pen, the circuitry of the SmartPen™ must be kept small in size and power consumption. This can only be accomplished by resorting to integrated mixed analog/digital chips. Figure 1 depicts a schematic representation of the pen and a photograph of the analog/digital data processing chip used inside the SmartPen™.

Figure 1: Schematic representation of the SmartPen™ application (left) and a photograph of the corresponding Analog/Digital data processing ASIC (right).

Practical experience shows us that, similar to the digital design process, the design of integrated mixed-mode circuits tends to be the subject of conceptual design errors. The availability of proper verification tools for this class of designs can therefore increase both the quality of the designs and the productivity of the designer. Existing techniques for the verification of integrated mixed-mode systems however are mostly simulation-based [15], and prove to be inadequate to provide sufficient confidence about the correctness of a design within an acceptable time-frame. Clearly, practical and more efficient verification-tools appear desired in order to successfully guarantee the quality of a mixed-mode design. Building on the success of the SFG-Tracing methodology [13, 17], we devised a formal, symbolic approach which enables us to verify the functional behaviour of integrated mixed analog/digital systems. The following chapter gives a brief overview of the SFG-Tracing methodology and how it can be modified towards mixed analog/digital circuits.

3 The SFG-Tracing methodology

3.1 Verification of digital (VLSI) designs using SFG-Tracing

The SFG-Tracing methodology has been developed for the formal correctness verification of digital (VLSI) designs, specified at the algorithmic level or signal flow graph (SFG) level. The foremost goal of this (formal) verification methodology is to overcome the gap between low level implementations (e.g. gate level or transistor-switch level circuits) and high level specifications. The practical value of the methodology has already been demonstrated by formally proving the correctness of a VLSI design of up to 230.000 transistors [26].

The strength of our verification methodology is based on the exploitation of the algorithmic information inherent to high level SFG specifications. By using this information, SFG-Tracing makes it possible to reduce the global verification problem to a series² of smaller – and thus more manageable – verification sub-problems. In order to achieve this reduction, the original signal flow graph will first be partitioned into a number of subgraphs. Key elements of the graph partitioning are explicit (user-defined) relationships between well-chosen algorithmic SFG-signals – the so-called reference signals – and spatial/temporal signals in the implementation. The reference signals define or delimit particular subgraphs in the signal flow graph, and the correctness of the implementation with respect to the individual graph-partitions can then be demonstrated by means of (switch-level) symbolic evaluation tools [8]. The basic concepts underlying our methodology are illustrated in figure 2; reference signals, delimiting an SFG-subgraph, are related with specific signals in a lower level representation, such as a switch-level or gate-level implementation.

Figure 2: In the SFG-Tracing Methodology, mapping-functions relate (reference) signals in the SFG-specification (left) to signals in a switch-level implementation (right).

As can be seen, the SFG-Tracing methodology does not perform a partitioning of the implementation, only a partitioning of the specification. Consequently, verifying the correctness of a given implementation with respect to one of its sub-graph specifications requires special attention. For instance, the implementation to be verified must first be brought into a suitable state, from which the actual correctness sub-proof can begin. During this initialisation phase, explicit symbolic values will be assigned to the state variables (i.e. circuit memory nodes) associated with the sub-graph specification under consideration. Explicit symbolic values are however not assigned to *all* nodes, because this usually results in unacceptable simulation efforts for real-life applications. A significant reduction of the simulation effort is obtained by simply assigning symbolic “don’t care” values X to the remaining circuit nodes.

Graph-partitioning, system initialisation and symbolic value assignments are only a few of the many aspects of the SFG-Tracing technique. Since a more elaborate and complete discussion on this verification methodology falls outside the scope of this paper, the interested reader is referred to [13], where a detailed account of the theoretical aspects underlying the SFG-Tracing methodology is presented.

² By ensuring that each sub-proof in this series is independent from the other sub-proofs, the separate correctness proofs can be executed in parallel. This can lead to a significant reduction of the execution time of the formal verification process.

In order to further illustrate the SFG-Tracing approach, the formal verification of a simple application – the bcd-recognizer introduced by Dietmeyer and discussed in [25] – is considered below. The algorithmic behaviour of this application is quite straightforward: a digital 4-bit word is applied to the input of the system (one 4-bit word for each algorithmic sample-period) and the recognizer determines whether or not the input represents a binary coded decimal. The (algorithmic) behaviour of the bcd-recognizer can be described by the following (high-level) VHDL description:

```

ENTITY bcd_recognizer IS
PORT(bcd_in(3), bcd_in(2), bcd_in(1), bcd_in(0) : IN bit;
      algorithmic_clock      : IN bit;
      bcd_ok                 : OUT boolean);
END bcd_recognizer;

ARCHITECTURE algorithmic OF bcd_recognizer IS
BEGIN
  Wait until (algorithmic_clock'rising_edge);
  bcd_ok <= ((bcd_in(3), bcd_in(2), bcd_in(1), bcd_in(0)) < "1010");
END algorithmic;

```

The algorithmic behaviour specified above still allows a variety of distinct implementations for the bcd-recognizer. In this paper, a bit-serial version of the bcd-recognizer is preferred, since it is better suited for demonstrating the SFG-Tracing methodology. Figure 3 depicts a synthesized gate-level implementation of a bit-serial bcd-recognizer, using a 2-phase non-overlapping clock. Note that an additional input *reset* has been added to the design in order to ensure a proper initialisation of its memory elements.

Figure 3 : Gate-level implementation of a bit-serial bcd-recognizer

How can the correctness of this implementation be verified by means of the SFG-Tracing? As mentioned before, the first step would comprise of a partitioning of the signal flow graph specification. Due to the simplicity of the example, however, the actual graph partitioning can be omitted in this case. Consequently, we must only define the relationship between the input/output signals in the specification and specific spatial/temporal signals in the implementation. At this point, it is important to realize that a fundamental difference may exist between the timing aspects of the algorithmic level and those of the implementation level. At the algorithmic level, an abstract clock was introduced only to declare the synchronous nature of the system and no constraints were imposed on how (e.g. in a parallel or serial way) the data is presented to the system. At the implementation level however, the 4-bit words to be analysed will be applied at the input *bcd_in* in a bit-by-bit fashion³ and the timing is controlled by a 2-phase non-overlapping clock, which is clearly different from the algorithmic clock. The desired mapping relation between specification and implementation should account for the various timing aspects of the design. Figure 4 presents a graphical illustration of the relationship between reference signals and implementation signals.

³ Most significant bits first

Figure 4 : Illustration of the relationship between reference signals in the specification and spatial/temporal signals in the implementation

The mapping information in figure 4 will be used to guide the symbolic simulation of the given implementation. The symbolic simulation will start with bringing the system into a known initial state, and this by initializing the memory-elements using the *reset* input. Next, the most significant bit – denoted by the symbolic value *in3* – of the 4-bit word to be analysed, is clocked in at the first clock pulse after the initialisation phase. At the subsequent clock-cycles, the following three bits – represented by the symbolic values *in2*, *in1* and *in0* – are to be clocked in. By the end of the fourth clock-cycle, the output *bcd_ok* should behave as specified. In other words, it should be checked that the following symbolic expression⁴ is a tautology :

$$out = \overline{in1} \wedge \overline{in2} \vee \overline{in3}$$

It can be verified that the implementation given in figure 3 does indeed satisfy this specification.

Evidently, the bcd-recognizer is only a simple application and, even though it serves as a good example for explaining the SFG-Tracing methodology, it is too small to demonstrate the numerous facets of the SFG-Tracing in detail. As mentioned before, a more detailed discussion on the subject is given in [13]. The objective of this paper, however, is to show how the SFG-Tracing, originally conceived for verifying digital VLSI designs, can be used for the formal verification of mixed analog/digital systems.

3.2 Extensions towards mixed-mode systems

When dealing with mixed analog/digital systems, the signal flow graph specification can easily be extended towards analog signals and operations. Figure 5 provides an illustration of this extension. This mixed-mode system selects one of four analog inputs and compares it with an analog reference voltage *Vref*. Based upon the result of this comparison, the logic part of the system determines the output *Out*. In the corresponding signal flow graph, analog signals are denoted by dashed lines, whereas full lines indicate digital signals.

Figure 5: Signal-flow graph (left) of a mixed analog/digital implementation (right). Dashed lines correspond to analog signals, full lines denote digital signals.

⁴ This symbolic expression reflects the specification that the symbolic bit-word $in_3 in_2 in_1 in_0 < 1010$

The extended signal flow graphs can again be partitioned into subgraphs and the designer can provide the required mapping functions, relating reference signals in the SFG with signals in the implementation. Thus, the SFG-Tracing methodology can in principle be applied to verify this type of design. Furthermore, the correctness of digital subgraphs can still be verified by means of the implementation-level symbolic evaluator. The problem that remains is how to incorporate the analog components in our verification process. Our approach to solving this problem lies in modelling the functional behaviour of the analog components in a symbolic way. Including these symbolic, functional models in the symbolic evaluations enables us to formally verify the correctness of a mixed-mode system with respect to its analog/digital SFG specification.

4 Analog macro building blocks

Practical experience shows that, on average, over 90% of an integrated mixed-mode design consists of digital MOS technology and that analog contributions are mainly restricted to interface purposes – e.g. conversion of analog sensor-outputs or the analog sensor itself. In order to construct the analog subsystems, designers of integrated mixed-mode systems usually resort to predefined analog library components or so-called ‘macro building blocks’. Without demeaning the general applicability of the proposed verification technique, we have also made use of the paradigm of library-based design and have, up to now, only considered a limited set of predefined analog macro building blocks. Nevertheless, customised analog subsystems – for instance, systems designed from scratch to answer to specific needs – can also be incorporated in the verification process, provided that their functional behaviour is accurately known. Restricting ourselves to a set of predefined components does however offer a main advantage: it enables automation and can hence bring the verification process closer to the designer.

Due to the nature of signal flow graph specifications – imposing uni-directionalism on both analog and digital operations – only predefined analog components for which uni-directional models can be derived, have initially been considered. As such, the following types of predefined analog macro blocks have been included in our current research:

- Analog multiplexers
- Operational amplifiers
- Analog comparators
- Voltage references
- Analog-to-digital converters
- Digital-to-analog converters
- Programmable gain amplifiers

For each type of predefined macro building blocks, several distinct implementations are, in general, available to the designer. The analog component library of ES2 – European Silicon Structures – for example, contains 2 different implementations of digital-to-analog signal converters, 9 comparators and 9 programmable gain amplifiers. Our ultimate goal is to take each individual implementation into account, to derive suitable symbolic models for their functional behaviour and to incorporate them into a symbolic analysis and verification tool. The actual technique used to derive a symbolic model for the functional behaviour of a component is the topic of the next chapter.

5 Symbolic modelling technique

The terminology ‘digital core approach to mixed-signal simulation’ is often used to identify the specific class of simulation-based verification techniques, where digital simulation tools are used to verify mixed-mode designs. For that purpose, analog components have to be modelled in a suitable digital manner. Accordingly, the approach discussed in this paper can be labelled a ‘symbolic core approach to mixed-signal simulation’: analog components are modelled in a symbolic way and a symbolic evaluation or simulation tool [8] will be applied to verify the overall integrated mixed-mode system. This chapter now introduces some aspects of the symbolic modelling of analog components.

5.1. Symbolic representation of analog signals

When the functional behaviour of an analog component is to be modelled in a digital way, a digital representation of the analog signals involved is first required. Encoding the absolute voltage of an analog signal by a digital 8-bit word⁵ or vector, allows us to replace a single analog wire by an 8-bit digital bus. Thus, a signal of 5V (0V) on an analog wire x – modelled by the digital bus $x_7x_6x_5x_4x_3x_2x_1x_0$ – can, for instance, be encoded by 11111111 (00000000). Similarly, the vector 00100000 will correspond to a voltage of 1.255V on the analog wire. Representing analog signals by means of digital vectors is quite common practice in the field of mixed-mode applications and vendors such as ES2 even provide the necessary functional models of their analog macro building blocks for (digital core) simulation purposes.

The above digital representation of analog signals can easily be extended towards a symbolic representation. In addition to the Boolean constants 0 and 1, arbitrary Boolean variables must be allowed in our vector representation. As such, the symbolic vector 00a00000 – the symbol a denotes a Boolean variable – covers both the analog voltages of 0V (00000000) and 1.255V (00100000). Likewise, for Boolean variables x_i ($i = 0..7$), the general symbolic bitvector $x_7x_6x_5x_4x_3x_2x_1x_0$ will cover the complete analog voltage range between 0V and 5V.

5.2 Symbolic models for analog components

The symbolic models that are used to describe the functional behaviour of individual analog components are based upon a symbolic representation of the analog signals involved and not on the physical wires themselves. Figure 6 illustrates this principle for a mixed-signal component (e.g. an analog-to-digital converter) with an analog input in and a digital output Out ; the symbolic model for this component makes use of a symbolic vector $in_7in_6in_5in_4in_3in_2in_1in_0$ instead of the analog wire in .

Figure 6: The symbolic model (left) for a component (right) is based upon the symbolic representation $in_7in_6in_5in_4in_3in_2in_1in_0$ of the actual analog signal in .

An important remark that can be made at this point, is that our symbolic models will **not** account for the detailed dynamic behaviour – frequency response, timing, poles and zeros of the system, etc. – of the individual mixed-mode components. Neither does our approach account for analog design issues such as cross-talk, transmission line effects, electro-magnetic interference and so forth. We are merely interested in the weak observable (functional) input-output behaviour of a given component for steady-state values of input and output at the end of sample intervals, and for that purpose the mixed-mode components can be modelled at a highly abstract level. For our set of analog macro blocks, for instance, this abstraction can be made and proves to be adequate to demonstrate the overall functionality of the mixed-mode systems that have been considered in our research so far.

Evidently, when the complete system behaviour must be verified into detail, the designer must resort to more dedicated tools and modelling techniques (e.g. SPICE) than the method presented in this paper. Our symbolic verification approach can however be used for a preliminary verification of a design. First, the mixed-mode design to be verified is modelled symbolically and the (abstract) functional behaviour is then verified using the proposed verification technique. Once – possibly after a number of iterative (re)design-and-verification steps – it is shown that the system functionally behaves as specified, the design verification can proceed at a lower level of abstraction. The process of successively lowering the level of abstraction is also quite common practice in traditional simulation-based verification techniques.

The following subsections deal with two particular examples of mixed-mode components – an analog multiplexer module and a digital-to-analog converter – and they describe how the functional behaviour of these components can be modelled symbolically.

⁵ Depending on the required accuracy of the model, a different width of the digital bus may be preferred.

5.2.1 Example 1: an Analog Multiplexer Module

An analog multiplexer module serves as the first example to illustrate the symbolic models for analog macro building blocks. At an abstract level, the functional behaviour of this component can be summarised as in table 1.

Input On (digital)	Input In (analog)	Output Out (analog)
0	-	Z
X or Z	-	X
1	X	X
1	in	in

Table 1: Functional input/output behaviour of an analog multiplexer module. The notation X denotes unknown signal values, Z denotes tri-state signals.

Based on this functional behaviour, the symbolic model for the multiplexer module should at least have an input On , an input-vector $In_7In_6In_5In_4In_3In_2In_1In_0$ and an output-vector $Out_7Out_6Out_5Out_4Out_3Out_2Out_1Out_0$. In order to incorporate tri-state signal values, our uni-directional model will additionally be extended with an input Z_i and an output Z_o . The abstract functional behaviour specified in table 1 can be modelled in terms of the following Boolean (symbolic) expressions:

$$Z_o = \overline{On \wedge Z_i}$$

$$Out_j = (In_j \wedge On \wedge \overline{Z_i}) \vee (Z_o \wedge X), \text{ for } j = 0..1$$

Using these expressions, it is possible to construct a complete symbolic model for the analog multiplexer module. The appendix contains a description of the symbolic model, to be used in a specific symbolic evaluation tool.

5.2.2 Example 2: a digital-to-analog converter

The digital-to-analog converter is another common component of most integrated mixed-mode systems. Together with a comparator and a logic unit for successive approximation, this type of converter is often used as an important part of an analog-to-digital converter (see figure 7). Basically, a digital-to-analog converter has a digital bus In and two analog voltages V_{refp} and V_{refn} as inputs, and a single analog output Out . The purpose of the component is to convert the digital signal In into the corresponding (or decoded) analog signal Out , using V_{refp} and V_{refn} as reference voltages for the conversion procedure.

When In is an 8-bit digital bus and the analog signals are modelled by 8-bit symbolic vectors, the functional behaviour of the component can be expressed as follows:

$$Out = V_{refn} + \left(V_{refp} - V_{refn} + 1 \right) \cdot \frac{In}{256}$$

The operations in the above expression are to be interpreted as digital operations on bit vectors. It is possible to derive sets of Boolean expressions to capture the behaviour of the different additive and multiplicative operations of the given expression. However, since most applications use 0V and 5V as their reference voltages, and multiplication's are difficult to represent in terms of Binary Decision Diagrams, the previous expression can be simplified to:

$$Out = In$$

By writing the (bit-level) expressions in the formalism of the previous example, the symbolic models can again be used in the symbolic evaluation and reasoning tool.

⁶ Z_i will be 1 (0) if the analog input is (not) tristate and Z_o is 1 (0) if the analog output signal is (not) tri-state.

5.3 Application example

In this section, a more concrete example of integrated mixed-mode designs is presented, which we want to formally verify using the proposed modelling technique. The example concerns a mixed-mode subsystem of the SmartPen™ application. In figure 7, a simplified version of this system is depicted. A sensor array – strain gauges in wheatstone bridge configurations – provides the analog input voltages (A, B, ..., H) of the system. Alternately, a digital control unit will select one of four amplified voltage differences. The multiplexed voltage is then converted into an 8-bit digital signal, which is subsequently converted into a specific bit-sequential protocol. Using this protocol, an RF-transmitter will finally send the information from the SmartPen™ to an external receiver.

Figure 7: Schematic representation of an integrated mixed-mode system application. The analog-to-digital converter is constructed by means of a comparator, a digital-to-analog converter and successive-approximation (S.A.) logic.

In order to verify this design, it must be demonstrated that its output satisfies the specified transmission protocol. For that purpose, the digital subsystems of the application will first be analysed by means of available symbolic analysis techniques, and proper symbolic models must be derived for the analog parts. By correctly wiring all the symbolic models together and providing the spatial/temporal mappings between specification (e.g. the protocol) and implementation, a symbolic evaluation of the functional behaviour of the overall mixed analog/digital system can be performed.

6 Future Work

The research topic presented in this paper is still a little on the preliminary side, and further research is definitely required to explore the full potential of the proposed verification methodology. One of our objectives in the near future is to compose an extensive library of symbolic, functional models for various analog macro building blocks. Based upon this symbolic library, the feasibility and limitations of the proposed approach – incorporating the symbolic models in the SFG-Tracing methodology – will be investigated by verifying the correctness of a number of integrated mixed-mode systems (for instance, the application shown in section 5.3.) developed in the integrated microsystems design group of imec.

An additional goal of our ongoing research is to allow alternative symbolic representations for modelling the functional behaviour of components. Up to now, only binary decision diagrams or BDDs have been employed to

describe the analog components, and this due to the practical limitations of the symbolic simulation tools used in our experiments. Representing multiplicative operations, as those in the expressions for the digital-to-analog converters by means of BDDs, tends to be quite inefficient – the size of the corresponding BDDs grows exponentially in the number of bits involved in the operations [6]. Based on recent publications dealing with graph-based representations of Boolean expressions, we are inclined to believe that other representations, such as Binary Moment Diagrams (BMDs) [10] and Kronecker Multiplicative BMDs (K*BMDs) [16], might be more appropriate for our verification approach. The goal is to decide which graph-based representation – or combination of several representations – is the most suited for the symbolic modelling of mixed analog/digital circuits.

7 Conclusions

For the formal verification of mixed-mode systems, this paper presented a pragmatic symbolic modelling approach, which can be incorporated in the SFG-Tracing – a methodology originally aimed at the formal verification of digital VLSI designs. In this new approach, the digital parts of a mixed-mode system are analysed using existing symbolic techniques and proper symbolic models – based on a symbolic vector representation of the analog signals involved – are derived to describe the functional behaviour of the analog components. Incorporating these models in symbolic evaluation and reasoning techniques, enables the formal verification of the overall functional behaviour of an integrated mixed-mode system. Although primarily focused on the analog macro building blocks inherent to library-based designs, the modelling technique discussed should be applicable to general, customised analog components.

8 Acknowledgements

The authors hereby wish to thank Randal E. Bryant and his COSMOS team for making the COSMOS system available to perform the research mentioned in this paper. This research was supported by a scholarship from the Flemish Institute for the promotion of Scientific-Technological Research in Industry (IWT).

9 References

1. C. M. Angelo, D. Verkest, L. Claesen, H. De Man: A synopsis on the comparison of HOL and Boyer-Moore for Formal Hardware Verification. In: Correct Hardware Design Methodologies – Proceedings of Advanced Research Workshop, Turin 1991. North-Holland, Amsterdam, 1992, pp. 421-426
2. David P. Appenzeller, Andreas Keuhmann: Formal Verification of a PowerPCTM Microprocessor. In: Proc. IEEE International Conference on Computer Design: VLSI in Computers and Processors, Austin (Texas) 1995. IEEE Computer Society Press, 1995
3. Derek Lee Beatty. A Methodology for Formal Hardware Verification, with Application to Microprocessors. PhD thesis, Carnegie Mellon University – Computer Science Department, Pittsburgh (PA), 1993
4. J. P. Billon and J. C. Madre: Original concepts of PRIAM, an industrial tool for efficient formal verification of combinational circuits. In: Proceedings of the IFIP WG 10.2 International Working Conference on the Fusion of Hardware Design and Verification (ed. G. J. Milne), Glasgow 1988. North-Holland, 1988, pp. 487-501
5. Dominique Borrione, Laurence Pierre and Ashraf Salem: PREVAIL, a proof environment for VHDL descriptions. In: Correct Hardware Design Methodologies – Proceedings of Advanced Research Workshop, Turin 1991. North-Holland, Amsterdam, 1992, pp. 163-186
6. Randal E. Bryant. Graph-based algorithms for Boolean function manipulation. IEEE Transactions on Computers 1986; Vol. C-35, No. 8:677-691
7. Randal E. Bryant. Boolean analysis of MOS circuits. IEEE Transactions on Computer-Aided Design 1987; CAD-6(4):634-649
8. Randal E. Bryant, Derek Beatty, Karl Brace, Kyeongsoon Cho, Thomas Sheffler: COSMOS – a compiled simulator for MOS circuits. In: 24th Design Automation Conference. 1987

9. Randal E. Bryant, Derek L. Beatty, Carl-Johan H. Segers: Formal Hardware Verification by Symbolic Ternary Trajectory Evaluation. In: 28th Design Automation Conference. 1991
10. Randal E. Bryant, Y.-A. Chen: Verification of arithmetic circuits with binary moment diagrams. In: 32nd Design Automation Conference. 1995, pp. 535-541
11. R. T. Boute: Declarative Languages – still a long way to go. In: Computer Hardware Description Languages and their Applications. Elsevier Science Publishers, 1991, pp. 165-192
12. P. Camurati and P. Prinetto. Formal Verification of Hardware Correctness: Introduction and Survey of Current Research. IEEE Computer 1988 (July), 8-19
13. L. Claesen, F. Proesmans, E. Verlind, H. De Man: SFG-Tracing: a Methodology for the Automatic Verification of MOS Transistor Level Implementations from High Level Behavioural Specifications. In: Proceedings ACM-SIGDA International Workshop on Formal Methods in VLSI Design (ed. P. A. Subrahmanyam), 1991
14. L. Claesen, D. Beullens, R. Martens, et al.: SmartPen – An Application of Integrated Microsystems and Embedded Hardware/Software CoDesign. In: IEEE Users Forum of European Design and Test Conference, Paris 1996. IEEE Computer Society, pp. 201-205
15. D. Conner. Mixed Analog-Digital Simulation. EDN 1994 (September); 39-44
16. Rolf Drechsler, B. Becker, S. Ruppertz: K*BMDs: A new Data Structure for Verification. In: IEEE Proceedings of European Design and Test Conference, Paris 1996. IEEE Computer Society, 1996, pp. 2-8
17. M. Genoe, L. Claesen, E. Proesmans, E. Verlind, H. De Man: Illustration of the SFG-Tracing Multi-Level Behavioural Verification Methodology, by the Correctness Proof of a High to Low Level Synthesis Application in CATHEDRAL-II. In: Proceedings of the IEEE ICCD-91 Conference. Cambridge MA, 1991
18. M. J. C. Gordon, T. F. Melham. Introduction to HOL: a theorem proving environment for Higher Order Logic. Cambridge University Press, 1993
19. G. R. Hellestrand: CAD for Systems Engineering: Integrating Algorithms - Electronics - Mechanics; Invited paper. In: 3rd ASEAN Symposium on Microelectronics and Information Technology, Bangkok, July 1994.
20. Warren A. Hunt. Microprocessor design verification. Journal of Automated Reasoning 1989; 5:429-460
21. C. A. Mead. Silicon Compilers and Foundries will usher in user designed VLSI. Electronics 1982; 1 07-111
22. T. Melham. Higher Order Logic and Hardware Verification. Cambridge University Press, 1993
23. Laurence Pierre: Describing and Verifying Synchronous Circuits with the Boyer-Moore Theorem Prover. In: IFIP WG 10.5., Proceedings of the Advanced Research Working Conference on Correct Hardware Design and Verification Methods (eds. P. Camurati & H. Eveking). Frankfurt/Main 1995. Springer-Verlag, 1995, pp. 35-55
24. D. Verkest, J. Vandenbergh, L. Claesen, H. De Man: A description methodology for parameterized modules in the Boyer-Moore Logic. In: Theorem provers in circuit design (eds. V. Stavridou, T. Melham, R. Boute). North-Holland, 1992
25. P. Camurati, et al.: Application Selection: Finite State Machines. In: report ESPRIT CHARME-PDT-1.A-01, July 1990, pp. 6-11
26. M. Genoe, L. Claesen, E. Verlind: Formal Verification of High-Level Synthesis by means of SFG-Tracing. In: Proceedings of the 6th ACM/SIGDA & IEEE Workshop on High Level Synthesis (ed. D. Gajski), Dana Point Resort, California, 1992, pp. 1992

10 Appendix

In this appendix, a description of the symbolic model for the multiplexer module, discussed in section 5.2.2., is listed. The model makes use of the Lgc-file syntax of the symbolic simulator COSMOS [8]. Basically, the Boolean expressions given in section 5.2.2. are formulated using a dual-rail encoding and all operations are expressed in terms of the primitive operations *id*(entity), *and* and *or*. The notation $\langle number \rangle$ is used to denote intermediate or temporary results and $:0 (:1)$ denotes the Boolean constant 0 (1).

```

suffix L,H ;
prim id, and, or ;

leaf multiplexer_module(out0_0, out0_1, out1_0, out1_1, out2_0, out2_1,
                        out3_0, out3_1, out4_0, out4_1, out5_0, out5_1,
                        out6_0, out6_1, out7_0, out7_1, Zo_0, Zo_1;
                        Zi_0, Zi_1, on_0, on_1, in0_0, in0_1, in1_0,
                        in1_1, in2_0, in2_1, in3_0, in3_1, in4_0, in4_1,
                        in5_0, in5_1, in6_0, in6_1, in7_0, in7_1)

{
/* Determining (on and not(Zi)) */
or(\0;on_0,Zi_1)
and(\1;on_1,Zi_0)

/* Determining the output Z: */
/* "tristate" */
id( Zo_1; \0 )
id( Zo_0; \1 )

/* Writing down the expression */
/* "out[i] = in[i]*on+not(on)*X */
/* bit out[0] */
or( \2; in0_0, \0 )
and( \3; in0_1, \1 )
or( \4; \1, :1 )
and( \5; \0, :1 )
or( out0_1 ; \3, \5 )
and( out0_0 ; \2, \4 )

/* bit out[1] */
or( \6; in1_0, \0 )
and( \7; in1_1, \1 )
or( \8; \1, :1 )
and( \9; \0, :1 )
or( out1_1 ; \7, \9 )
and( out1_0 ; \6, \8 )

/* bit out[2] */
or( \10; in2_0, \0 )
and( \11; in2_1, \1 )
or( \12; \1, :1 )
and( \13; \0, :1 )
or( out2_1 ; \11, \13 )
and( out2_0 ; \10, \12 )

/* bit out[3] */
or( \14; in3_0, \0 )
and( \15; in3_1, \1 )
or( \16; \1, :1 )

and( \17; \0, :1 )
or( out3_1 ; \15, \17 )
and( out3_0 ; \14, \16 )

/* bit out[4] */
or( \18; in4_0, \0 )
and( \19; in4_1, \1 )
or( \20; \1, :1 )
and( \21; \0, :1 )
or( out4_1 ; \19, \21 )
and( out4_0 ; \18, \20 )

/* bit out[5] */
or( \22; in5_0, \0 )
and( \23; in5_1, \1 )
or( \24; \1, :1 )
and( \25; \0, :1 )
or( out5_1 ; \23, \25 )
and( out5_0 ; \22, \24 )

/* bit out[6] */
or( \26; in6_0, \0 )
and( \27; in6_1, \1 )
or( \28; \1, :1 )
and( \29; \0, :1 )
or( out6_1 ; \27, \29 )
and( out6_0 ; \26, \28 )

/* bit out[7] */
or( \30; in7_0, \0 )
and( \31; in7_1, \1 )
or( \32; \1, :1 )
and( \33; \0, :1 )
or( out7_1 ; \31, \33 )
and( out7_0 ; \30, \32 )
}

```

The description of the corresponding symbolic model for the second example (section 5.2.2) is not included in this appendix, due to the impractical length of this description.