Formal Verification based on Assume and Guarantee Approach - A Case Study

Subir K. Roy, Hiroaki Iwashita and Tsuneo Nakata
Fujitsu Laboratories Ltd.,
1-1 Kamikodanaka 4-chome, Nakahara-ku, Kawasaki 211-8588, Japan.
{skroy,hiroaki,nakata}@flab.fujitsu.co.jp

Abstract— In this paper, we present a verification case study of the Audio Output Interface (AOF) subsystem based on a compositional verification approach known as the *Assume-Guarantee* approach. We illustrate, how designers can leverage their detailed knowledge of a design to partition it at the module level, and thereby, enable the *Assume-Guarantee* approach to overcome intrinsic limitations of a formal verification tool to successfully verify large designs.

1 Introduction

Higher device integration has spurred the growth of system on chip (SOC) designs. SOC designs being inherently large, their formal verification needs to be carried out using a divide and conquer approach to overcome the problems of state and memory explosion present in symbolic model checking. In literature, such approaches are referred to as *compositional verification* . The methods proposed in [4,5,6,7] are based on specifications described as properties in temporal logics such as CTL, or its subset ACTL, while, those in [8,9] are based on *refinement checking*, where both, the specification and the implemention are described using the same language, and the implementation description is analyzed for correctness with respect to the specification description. New languages have been proposed to support refinement checking. High level design methodologies in the industrial context are tied to *Verilog* and *VHDL*, because of the commercial CAD tool support available for them. Therefore, approaches based on refinement checking cannot be directly used in industrial designs.

In this paper, we present a verification case study based on the *Assume-Guarantee* approach, using a state of art symbolic model checking tool, BINGO [1,2,3]. BINGO supports property verification for *Verilog* and *VHDL* design descriptions. The paper is organized as follows. In section 2, we briefly review the *Assume-Guarantee* approach. In section 3, we present key facts based on our analysis of the AOF architecture with respect to the specified property. In section 4, we apply *Assume-Guarantee* approach to verify AOF. In section 5, we present verification results, conclusions and suggest possible areas of investigation.

2 Assume-Guarantee Reasoning

In Fig. 1, we have a system composed of two finite state modules P and Q, which communicate with each other over wires S, and also with their external environment (dashed vertical lines).

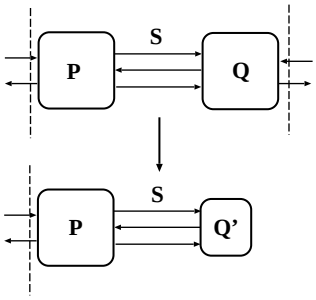

Fig. 1 Compositional Minimization

Let, $P \parallel Q$ be the parallel composition of P and Q. Let, φ_P, φ_Q be the module specifications, or properties of P and Q, respectively. In general, a component/module of any system is optimized and implemented to function only in the environment of that system. Therefore, P will not satisfy φ_P in any arbitrary environment.

As the reachable state space of P, in isolation, can be much larger than the state space of P composed with Q, any unexpected input that it receives, will cause P to traverse an unintended path in its state space, consequently, resulting in an incorrect behaviour. This is known as the *Environment Problem* [4,6,7]. In *Assume-Guarantee Reasoning*, the *Environment Problem* is deferred by expressing any assumptions process P makes about process Q in a temporal formula ψ_Q. These *assumptions* can be used when checking that P satisfies its specification φ_P. These assumptions must, of course, be discharged relative to Q, whence they are known as *guarantees*. To do this, we use the following inference rule [7]. If $P \parallel \psi_Q \models \varphi_P$, and $Q \models \psi_Q$ then, $P \parallel Q \models \varphi_P$. By $P \parallel \psi_Q \models \varphi_P$, we imply that $P \parallel \grave{Q} \models \varphi_P$, for all $\grave{Q}$ such that $\grave{Q} \models \varphi_Q$.

By properly formulating the set of *assume* and *guarantee properties*, it is possible to demonstrate the correctness of the entire system without constructing the global state graph. The decoupling of the module interconnec-

tion necessitates recasting an original property in terms of the properties of the decoupled modules. Essentially, this amounts to a decomposition of the original property. It is here, that we need to employ a detailed analysis of the design. Decomposition renders the original verification problem into smaller verification problems involving only the individual decoupled modules.

3 Description of Audio Output Interface

AOF is a multi-media interface sub-system in a SOC, known as the *Multi Media Extension Chip (MM-ASIC).* MM-ASIC extends the multi-media capability of an existing processor. The original description of AOF has 774 latch elements. AOF has 5 levels of module hierarchy as shown in Fig. 2. These modules can be classified under three major functional blocks related to multiple clock processing (root module - AO_CLK), 16 bit audio data processing (root module - AO_CH) and interrupt request processing (root module - AO_IRQ).

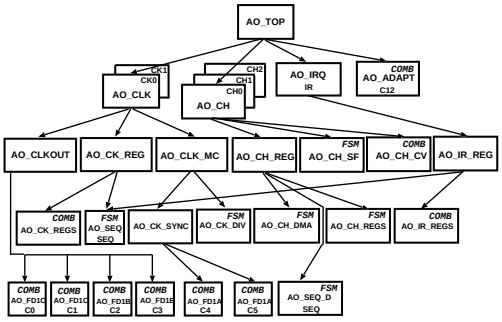

Fig. 2 Module Hierarchy of AOF

The designers specified an important *Data Starvation Property* for verification.

Property 1 *As long as $DACK$ (data acknowledge signal) is received properly in response to $DREQ$ (data request signal), UDR (data starvation signal) should never be asserted.*

3.1 Signal Flow Analysis for Data Starvation Property

AO_CH_SF asserts UDR. Primary signals $DACK$ and $DREQ$ play a crucial role in module AO_CH_DMA. AO_CK_DIV, the third important module generates event timing signals to test for the data starvation condition. The relevent interaction between the three modules with respect to the property is shown in Fig. 3 and involves internal signals $BCLKD$, $LRCLKU$, $LRCLKD$, GAT and $STANDBY$.

AOF being a multiple clock design, we modified it into a single clock design, by generating internally each of its clock from a single reference clock, $GLOBAL\_CLK$. For example, in the verification model IO_CLK is derived

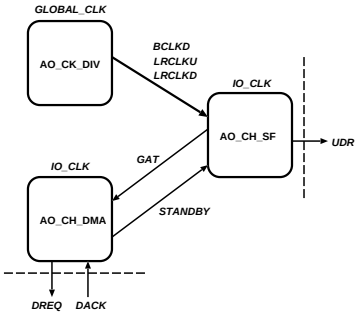

Fig. 3 Module Interaction for Property 1

from $GLOBAL\_CLK$ by dividing it by 16. It is important to note that, only AO_CK_DIV is clocked by $GLOBAL\_CLK$, while AO_CH_DMA and AO_CH_SF are clocked by IO_CLK. The *Assume-Guarantee* approach can exploit this to significantly reduce the computational cost of verification.

4 Assume-Guarantee applied to AOF

We present, below, two key observations derived from our detailed analysis of the state transition graphs (STGs) of the three modules. *Observation 1* relates to the behaviour of AO_CK_DIV, while *Observation 2* states a local property for AO_CH_DMA.

Observation 1 *$LRCLKU$ and $LRCLKD$ remain high for a single IO_CLK period and remain low for 127 IO_CLK periods.*

The STG of AO_CH_DMA (Fig. 5 in Appendix) has directed cycles of transition edges for processing and generating signals $DACK$, $DREQ$ and $STANDBY$ respectively. Each cycle has transition edges where $DREQ$ or $STANDBY$ are generated. The shortest and longest directed cycles (excluding self loops) involve 3 and 5 arcs, respectively. As, AO_CH_DMA is clocked by IO_CLK in the implementation model, we can state the following observation.

Observation 2 *A single traversal of any directed cycle in the STG of AO_CH_DMA, under conditions imposed by the Data Starvation Property, will require atmost 5 IO_CLK clock cycles.*

From the STG of AO_CH_SF (Fig. 6), we observe that in each of the directed cycles of transition edges which process $STANDBY$ and generate UDR and GAT, there is atleast one transition arc controlled by either $LRCLKU$, or $LRCLKD$. Thus, a complete traversal of any of these directed cycles will require atleast 128 IO_CLK cycles by *Observation 1.* Therefore, with $LRCLKU$ and $LRCLKD$ as the event timing signals, $STANDBY$ is asserted by AO_CH_DMA before the transition edges in the STG of AO_CH_SF, which process $STANDBY$ and

generate UDR, are reached during a traversal of these directed cycles. This condition ensures that UDR is never asserted. Therefore, $LRCLKU$ and $LRCLKD$ bring about the synchronization of transitions in AO_CH_SF and AO_CH_DMA to ensure correct behaviour with respect to the data starvation property. We use these facts to specify an *assumption* on $STANDBY$ for AO_CH_SF as given below.

4.1 Assumption on $STANDBY$

With $DACK$ asserted, if $STANDBY$ is asserted within 5 IO_CLK clock cycles of GAT being asserted, then UDR is never asserted.

The above *assumption* presumes other *assumptions* on signals $BCLKD$, $LRCLKU$ and $LRCLKD$. These *assumptions* can be easily derived from the timing diagram of these signals with respect to IO_CLK. When the assumptions are treated as *guarantees* for AO_CK_DIV we need to recast them with respect to $GLOBAL\_CLK$.

4.2 Guarantee on GAT

From the STG of AO_CH_SF we observed that, if $STANDBY$ is asserted, eventually, GAT would be asserted. This is a *liveness property* to ensure that a new data request is issued by AO_CH_DMA. It serves as the *guarantee* on GAT for AO_CH_SF.

4.3 Verification of AO_CH_SF

The *assumptions* on signals $BCLKD$, $LRCLKU$ and $LRCLKD$ are reduced to a collection of simple Boolean propositions for each IO_CLK cycle. This collection being repetitive, can be easily specified as a path set expression in BINGO [1,2] which internally creates an equivalent FSM. For the *assumption* on $STANDBY$, we wrote an *Assumption FSM* module, whose STG is shown in Fig. 4.

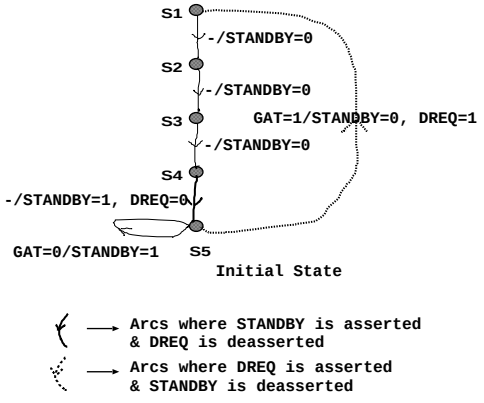

Fig. 4 State Transition Graph of ASUM_FSM

Two verification scripts corresponding to each of the *guarantees* were run. The verification results are summarized in Table 1. The second column in Table 1 gives the signals with respect to which the *guarantees* in the corresponding modules are verified. The fifth and sixth columns refer, respectively, to the number of latch bits and inputs, after using the *ReduceFSM* command in BINGO. This command removes all latches and inputs that are not related to the signals involved in the property.

4.4 Verification of AO_CH_DMA

We assume an appropriate behaviour for the environmental signal $DACK$, in response to the output signal $DREQ$ and specify this as an input constraint in the BINGO script. For GAT, a liveness property was specified as a *guarantee* on AO_CH_SF. With respect to AO_CH_DMA, non-determinizing GAT is sufficient to verify the guarantee on $STANDBY$ imposed by AO_CH_SF. Table 1 gives the verification results.

4.5 Verification of AO_CK_DIV

Guarantees on signals $BCLKD$, $LRCLKU$ and $LRCLKD$ respectively, of AO_CK_DIV, were written as seperate path set expressions in BINGO and verified individually. As these signals are correlated a seperate path set expression (PSE) was written and verified seperately. Table 1 gives the verification results. The last row in Table 1 gives the results for the same property verified using only abstraction, and not decomposition (indicated by the entry in the second column).

5 Summary and Conclusions

From Table 1, it is clear that computational cost of individual module verification based on *Assume-Guarantee* approach is cheaper, as compared to the model abstraction approach. This approach needs detailed design information for its success and therefore, not easy to automate. However, for instances where it is applicable, it offers the largest possible benefits in terms of reduced computational complexity. The success of this approach has been reported only with respect to ACTL properties. For a given set of interacting components, it may not always be possible to decompose the given global property into sub-properties belonging to ACTL. A good direction to investigate would be to find the rules by which a non-verifiable design instance can be transformed into a verifiable instance by modifying the behaviour of individual modules and by changing the structure of interaction of these modules.

References

1. H. Iwashita, T. Nakata, and F. Hirose, "CTL Model Checking Based on Forward State Traversal", in *Proc. ICCAD-96*, pp. 82-87, 1996.
2. H. Iwashita and T. Nakata, "Forward Model Checking Techniques Oriented to Buggy Designs", in *Proc. ICCAD-97*, pp. 400-404, 1997.
3. K. L. McMillan, "Symbolic Model Checking", *Kluwer Academic Publishers*, 1993.

Module	Guar.	FFs	Inps	FFs Reduce FSM	Inps Reduce FSM	Imgs	CPU Time sec	Mem. Alloc. MB	Peak BDD Nodes
AO_CH_SF	GAT	20	4	25	3	7	15	58	2464
	UDR	20	4	25	3	45	17	58	2608
AO_CH_DMA	DREQ	14	3	12	3	10	8	57	1175
	STANDBY	14	3	12	3	10	8	57	1171
AO_CK_DIV	BCLKD	32	5	28	4	615	75	58	17409
	LRCLKU	32	5	28	4	1084	175	62	50236
	LRCLKD	32	5	28	4	1258	182	61	40581
	COMBINE	32	5	28	4	1620	255	63	40591
AOF	Abstr	53	17	51	8	1610	557	88.3	272010

Table 1: Verification Data using Assume-Guarantee and Abstraction

4. S. Berezin, S. Campos, and E. M. Clarke, "Compositional Reasoning in Model Checking", *Technical Report - CMU-CS-98-106, School of Computer Science, Carnegie Mellon University*, 1998.
5. M. Kaufmann, A. Martin, and C. Pixley, "Design Constraints in Symbolic Model Checking", *Proc. CAV-98*, pp. 477-487, 1998.
6. E. M. Clarke, D. E. Long, K. L. McMillan, "A Language for Compositional Specification and Verification of Finite State Hardware Controllers", *Proceedings of the IEEE*, Vol. 79, pp.1283-1292, Sept. 1991.
7. E. M. Clarke, D. E. Long, K. L. McMillan, "Compositional Model Checking", *Proceedings of the Fourth Annual Symposium on Logic in Computer Science*, pp.353-362, June 1989.
8. K. L. McMillan, "A compositional rule for hardware design refinement", *Computer Aided Verification (CAV97)*, O. Grumberg (Ed.), Haifa, Israel, pp. 24-35, 1997.
9. T.A.Henzinger, S. Qadeer, and S.K.Rajamani, "You assume, We guarantee : Methodology and Case Studies" *CAV98: Computer Aided Verification*, Lecture Notes in Computer Science, Springer-Verlag, pp. 440-451, 1998.

Appendix

The STGs of *AO_CH_DMA* and *AO_CH_SF* are shown below.

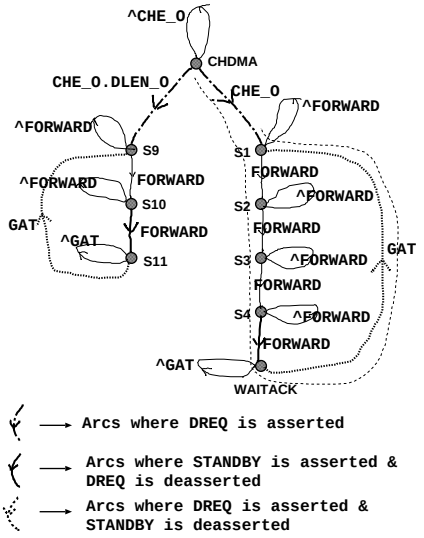

Fig. 5 State Transition Graph of AO_CH_DMA

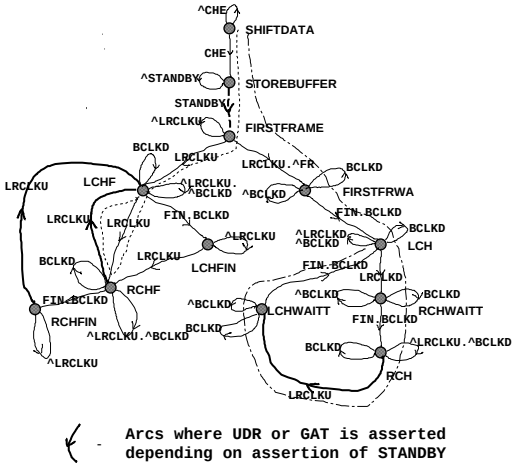

Fig. 6 State Transition Graph of AO_CH_SF