Modeling and Analysis Technique for the Formal Verification of System-on-Chip Address Maps

Extended Abstract

Niels Mook
NXP
Eindhoven, The Netherlands
niels.mook@nxp.com

Erwin de Kock
NXP
Eindhoven, The Netherlands
erwin.de.kock@nxp.com

Bas Arts
NXP
Eindhoven, The Netherlands
bas.arts@nxp.com

Soham Chakraborty
Delft University of Technology
Delft, The Netherlands
s.s.chakraborty@tudelft.nl

Arie van Deursen
Delft University of Technology
Delft, The Netherlands
arie.vandeursen@tudelft.nl

***Abstract*—This paper proposes a modeling and analysis technique to verify SoC address maps. The approach involves (i) modeling the specification and implementation address map using a unified graph model, and (ii) analysis of equivalence in terms of address maps between two such models. Using a state-of-the-art mid-size SoC design, we demonstrate the proposed solution is able to analyze and verify address maps of complex SoC designs and to identify the causes of discrepancies.**

I. Introduction

Specification and implementation of System-on-Chip (SoC) address maps are typically manual tasks. To discover mistakes, functional verification is performed in iterative, time-consuming cycles. Industry trends [1] show that on average, verification takes up at least 50% of median project time. With decreasing time-to-market for SoC development, techniques that speed up verification are needed.

To address this need, this work proposes a unified graph-based data model and analysis technique to perform a static, formal verification of SoC address map implementations against their specifications. A specification describes the intended (global) address map as seen by all initiators in the SoC, while an implementation describes the realized address map implemented by the composition of the IP blocks in the SoC. The implementations are described in IP-XACT [2] format and the specifications are described in a spreadsheet format.

IP-XACT can describe hierarchical SoC integrations by abstracting reusable IP blocks into component objects and connecting them in design objects. An interconnection between a target and initiator interface maps the target memory elements to the initiator address space. Furthermore, it can define hierarchical interconnections between two initiator or target bus interfaces, resulting in a hierarchical structure of address maps.

Universal Verification Methodology (UVM) [3] is a popular verification technique at the SoC integration stage that reduces verification time. The difference with UVM is that our proposed technique verifies static address maps in an explicit, formal, and structural manner as opposed to the implicit, simulation-based, functional manner of UVM. This enables early discovery of errors in static address maps, thereby potentially reducing the number of verification cycles required.

II. Address Map Graphs

This work proposes a directed graph model to represent both the address map specifications and implementations of a design, called the **address map graph** (AMG). It may be disconnected and cyclic. Each **node** represents a memory element with a base address and address range. Each **directed edge** represents an address map between two memory elements. An edge has a source node, target node, and possible offset at which addresses are mapped to the source node. A set of consecutive edges forms a path, starting in a root node and ending in a leaf node, as shown along the top of Fig. 1. The dashed lines under each node and edge represent address axes, while the bars represent the mapped addresses. Edge offsets are indicated by an arrow. Offsets and constraining address space boundaries may result inaccessible addresses mapped outside of the source address space, we call **clipping**. Each path has a resulting interval of root node addresses mapped to an equally sized interval in the leaf node. These two address intervals we call the domain D and codomain C. Together, they define the **bitmapping** of a path. We have developed a recursive calculation to define bitmappings for acyclic paths. Furthermore, given a bijective map between root and leaf nodes of two graphs, we define **graph bitmapping equivalence** (GBE) to describe if their combined set of bitmappings are equal. In addition, we define partial equivalence to hold if the domain and codomain of a bitmapping are respectively subsets of the domain and codomain of another bitmapping. This typically means that only part of the specified bitmapping is implemented.

Our solution consists of a modular flow as outlined in Fig. 2. It accepts an IP-XACT design description and an address map specification spreadsheet, and processes them both into

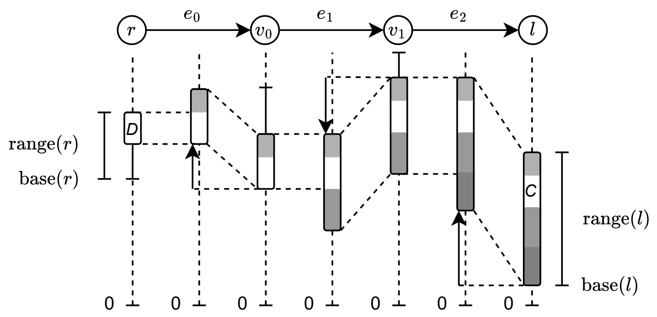

Fig. 1. A 3-edge path, clippings indicated in gray

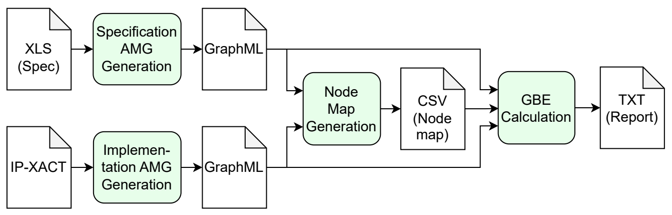

Fig. 2. Approach overview with our innovation indicated in green

our graph model. To compute the SoC address maps, we recursively traverse the SoC component hierarchy. In general, all components and their elements relevant to the address map are converted into nodes, and their memory constructs into edges. Both models are then further analyzed to determine GBE and generate a GBE report containing all equivalences.

In contrast to our model, existing literature [4] represents nodes as IP-XACT bus interface elements, and edges as their connections through various mapping structures. However, its recursive construction of the model over hierarchical IP-XACT components has inspired our general approach to construct our model. Its model is used for the purpose of visualization and editing of the address maps in Kactus2 [5] as opposed to our purpose of verification.

III. Experimental Evaluation

To evaluate the effectiveness of our approach, we demonstrate it with the IP-XACT description and address maps specification spreadsheet of a state-of-the-art, medium-size SoC design. The design has passed all simulation-based verification steps of its design flow before application of our solution flow.

We have implemented the solution flow as outlined in Fig. 2, where each program is a **generator** that is executable by the IP-XACT design environment. We have processed the address maps specified by the spreadsheet and implemented by the IP-XACT design description into a specification AMG H and implementation AMG G, respectively. Table I summarizes the properties of both graphs. The large number of nodes and edges of the implementation AMG relative to the specification AMG indicates the capability to capture the complexity of the design. Application of the rest of our solution flow showed GBE did not hold and that equivalence of bitmappings were distributed according to Table II. Around 9% of bitmappings were non-equivalent. Inspection of reported faulty bitmappings

TABLE I
Properties of evaluated AMG graphs

Evaluation Property	Specification AMG H	Implementation AMG G
Number of nodes	120	2018
- Number of root nodes	5	658
- Number of leaf nodes	115	650
Number of edges	134	6151
Automatically mapped	73	316
Manually mapped	18	31

TABLE II
Evaluation GBE report summary

Bitmapping Category	Result
Number of equiv. bitmappings	66
Number of partial equiv. bitmappings	90
Number of non-equiv. bitmappings	15
- Caused by implementation	8
- Caused by specification	7

facilitated the identification of the following causes of the non-equivalences: incomplete specification, incomplete IP-XACT design description, gaps in merged bitmappings, inconsistent codomain offsets, and use of address handling not described in IP-XACT. These results indicate the capability to identify inconsistencies in a verified design.

IV. Discussion and Future Work

The results demonstrate our solution's ability to identify hard-to-find inconsistencies between implemented and specified address maps for complex SoC designs. Compared to UVM, our solution requires minimal configuration, and provides nearly automatic and immediate formal verification of static address maps. Further research can be conducted to evaluate the effectiveness of our solution in enhancing or complementing existing methods like UVM, such as by reducing the number of verification cycles, generating register models from AMG models as input for simulation, or how found address map inconsistencies can guide the definition of cover groups. Furthermore, our approach can be extended to handle runtime-dependent IP-XACT properties as introduced by IEEE 1685-2022. This could involve the introduction of symbolic expression in proposed definitions, such that GBE involves resolution of symbolic equations.

References

[1] H. Foster, "2022 Wilson Research Group IC/ASIC functional verification trends," Siemens Digital Industries Software, Tech. Rep. Nov. 2023.

[2] IEEE, "IEEE standard for IP-XACT, standard structure for packaging, integrating, and reusing IP within tool flows," *IEEE Std 1685-2022 (Revision of IEEE Std 1685-2014)*, pp. 1–750, Feb. 2023.

[3] IEEE, "IEEE standard for universal verification methodology language reference manual," *IEEE Std 1800.2-2020 (Revision of IEEE Std 1800.2-2017)*, pp. 1–458, Sep. 2020.

[4] E. Pekkarinen, M. Teuho, and T. D. Hamalainen, "Analysis and visualization of product memory layout in IP-XACT," in *2017 Euromicro Conference on Digital System Design (DSD)*, Aug. 2017, pp. 155–162.

[5] A. Kamppi, L. Matilainen, J.-M. Maatta, E. Salminen, T. D. Hamalainen, and M. Hannikainen, "Kactus2: Environment for embedded product development using IP-XACT and MCAPI," in *2011 14th Euromicro Conference on Digital System Design*, Aug. 2011, pp. 262–265.