# A Graph-Based Solution for Register Coupling in Functional Verification

Zhang Yuxuan, Jiang Guofan, Lu Yinchao, and Gou Pengfei

***Abstract*—Predicting status of registers is required in verification development to accurately emulate behaviors of DUT. However, it is a complicated thing when register couplings exists in DUTs. To unleash verification development, we propose a graph-based solution, with a model of "topology + behavior" for register couplings, to accurate emulate states and behaviors of DUT. This work is inspired by realistic verification requirement in industry-level developments. With searching mechanism, register couplings can be efficiently and accurately processed at runtime. Verification works can be significantly simplified without remarkable resource costs and performance loss. Our experiment and analysis finally suggest tempting benefits of this method in functional verification development.**

***Index Terms*—Chip development, functional verification, EDA.**

## I. Introduction

EDA has dramatically propelled design and verification technology in the past. While the explosion of design complexity, with Moore's Law, continuously challenges EDA technologies. This paper focuses on functional verification and the register features, which belongs to the domain of front-end digital logic development and EDA methodology.

### A. Functional Verification Overview

Functional verification finds flaws and ensures correctness of logic design in an early phase, now has been a big domain in modern chip development. However, along with the growing complexity of design target, verification development tends to be a systematical engineering. It spends much time and costs more resources of work effort in verification than RTL design itself. On one hand, researchers start to enhance simulation tools in performance by either optimize computing technique such as parallelism, or leverage special hardware approach. On the other hand, they enhance and standardize methodologies to enable efficient verification developments.

Driven by continuous increment of industrial requirement, it has achieved preeminent progress of methodology of functional verification in the past decades. In TLM (Transaction-Level Modeling) [1], the use of packaged transactions transcends traditional RTL (Register Transfer Level) behavior modeling, makes verification development efficient. Modern functional verification methodology, fuses concepts from software engineering, provides abstract-level modeling and object-oriented features. It does not only focus on functional checking and coverage analyzing, but also accent a system perspective so that enable very complicated large-scale engineering. OVM (Open Verification Methodology) [2] and UVM [3] (Universal Verification Methodology), provide a sequence-dominated framework, widely applied and examined in industry-level development. Fusion/RTX [4] -IBM's methodology, advanced in its flexibility, specializes in server processors and chips. These methodologies usually support a chain of sequencer-driver-monitor, to organize and handle transaction. And the kernel of these methodologies is the implementation of a reference model, like a mirror, to emulate DUT behaviors.

Registers, such as configuration register of digital design, or operation register of CPU, are generally regarded as the interfaces of software and hardware. Besides, as significant parts in a control flow, registers drive functional behaviors. However, integrated register solution is exerted in verification methodologies only in the recent, such as RGM (Register and Memory Package) [5] and REG (UVM Register Layer Classes) [5]. Registers-Modeling [6] is also implemented in Fusion/RTX as a special component. They are common in bridging functional operation to protocol-level behaviors, to represent register access and organization, and to manipulate registers' mirror as a special reference model.

### B. Register Coupling in Verification

We present this paper for solving "*register coupling*", — A troublesome problem entangles verification development. The typical coupling is dependency between registers (fields) in a design, when one changes another one changes as well. While "*register coupling*" has a general definition in this paper, not only limited in those explicit dependencies exist in logic design, but also involves implicit ones presented by software interface like *broadcast*, *indirect register*. Therefore, we categorize coupling patterns into explicit and implicit, and list several most common ones below as examples.

1) *Value dependency:* The value of one register/field is dynamically calculated by other ones.
2) *Attribute dependency*: The attribute (i.e. read-write, read-only, write-only) of one register/field is dynamically determined by a condition related with the value of other ones.
3) *Indirect register*: A set of registers cannot be directly access through bus, but can only be indirectly access by configuring a pair of index/value registers.
4) *Broadcast*: When writing to a broadcast address, registers in corresponding different groups are concurrently

Manuscript received January 12, 2015; revised June 16, 2015.

Zhang Yuxuan, Jiang Guofan, Lu Yinchao, and Gou Pengfei are with China System and Technology Laboratory, IBM, Shanghai, P.R. China (e-mail: {zyxsh, jianggf, luyinch, goupengf}@cn.ibm.com).

updated.

### C. Our Work

Currently, existing methodologies are deficient in providing a complete approach for model register coupling. Our work, for the first time, proposes a uniform model for various coupling patterns. The main idea is to separate connection and behavior of coupling, and then map to a directed-graph model. It helps to facilitate verification development by substituting most of developers' manual works with automatic searching. And this approach has advantage in solving complicated compound couplings. We present a register mirror subsystem, which enables friendly integration in existing methodologies, to support the reference model to deliver decision. The following sections of this paper are structured as follow: Section II introduces the modeling and mapping of problem; Section III elaborates the implementation in a systematic point of view, to construct a platform, and to integrate in test bench; finally this solution is evaluated in realistic verification development in Section IV, and finally the conclusion is drawn in Section V.

## II. Mapping Coupling to Graph

This section introduces mapping coupling to graph. Then presents the key of this approach, involves the flattened fields organization and runtime search mechanism.

### A. Modeling

To model different coupling patterns uniformly, the thought is to explore their commonness — The connection relationship between register/field pairs. We abstract dependency of register/field as topology feature by detaching them from traditional behavior descriptions. Then, coupling problem equals to a typical directed graph problem [7]. Mapping to a directed-graph, each register/field is projected to a vertex; the dependency of a coupling is represented by a directed edge. The behavior of each coupling is attached on corresponding vertices, represented as expressions. Coupling can be emulated in mirror by automatic search mechanism, synchronized with a register bus operation.

Fig. 1 shows register coupling in a perspective of graph according to coupling patterns and compound of a realistic IP core specification. There are typical patterns presented, and also intricate compound of them. Each vertex represents a specific register/field, and direct edge represents a dependency from one field to another field.

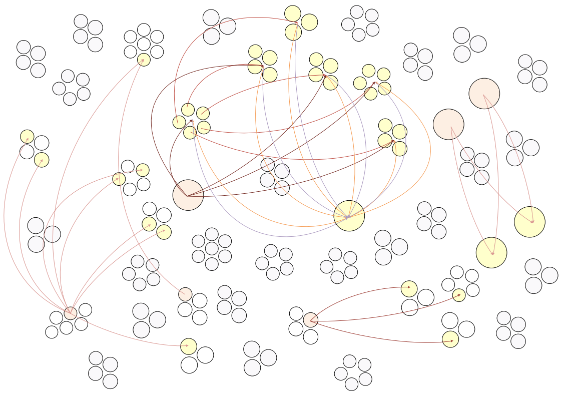


Fig. 1. Mapping register couplings in a graph model.

While functional behaviors of coupling vary with patterns, connection relationship of each coupling, no matter explicit or implicit, renders similar topology attributes in common. Fig. 2 symbolizes typical patterns. Fig. 2(a) and Fig. 2(b) show the graphic representation of the pattern of attribute dependency and a typical value dependency separately; Fig. 2(c) shows a cascade of coupling that a simple compound of value dependency and attribute dependency; Fig. 2(d) shows a broadcast.

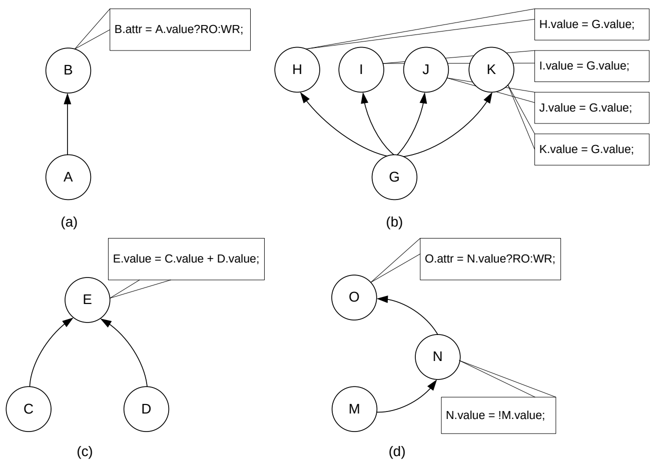


Fig. 2. Graph description of basic coupling patterns.

This model approach has advantage in solving compound of coupling patterns. With this model, complicated relations of compound are not necessarily described by developers, but can be solved by automatic searching. Fig. 3 is an example compound of multi-coupling, in which three typical coupling problems are involved. Additionally, this model is only available when states of coupling are finite, meaningful and predictable.

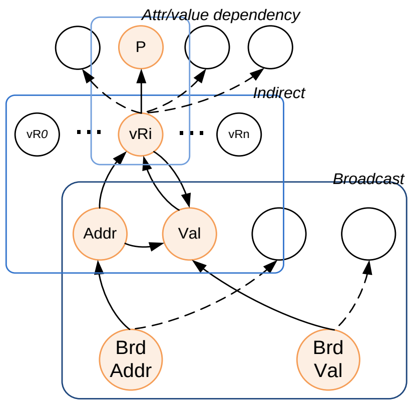


Fig. 3. Compound of multiple coupling patterns.

### B. Flattened Reg-field Structure

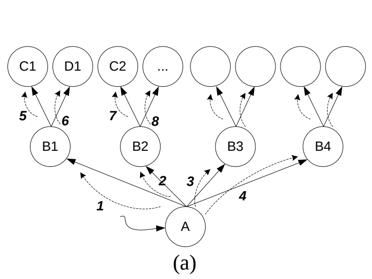
Given that coupling effect between register fields, we model coupling in granularity of register fields. The whole register database is organized in a flattened structure of register fields. Then, couplings are indirectly represented by an adjacency matrix, which is almost sparse. Furthermore, to solve couplings and their compounds, the search program need to travel in either positive direction or opposite direction on this directed-graph. That's the reason that orthogonal list is appropriately applied to store the matrix. In the memory space, vertices and edges are stored discretely. Each edge indicates a connection, that only ‘*1*’s are stored. The time to complete the construct is $O(v*e)$, and the spatial cost is $O(v+e)$, ($v$ indicate the count of vertices, $e$ indicates the count of edges). This structure, finally, plays a role of mirror of verification test bench at simulation time.

### C. Searching Strategy

Searching program is used in our solution to dynamically predict register mirror. The searching program involves two major travelling procedures transmission and evaluation.

The transmission procedure transmits the propagation wave of register update followed by state changes throughout the topology. Considering the feature of coupling problem, BFS (Breadth First Search) [8] is adopted in this approach to search adjacent vertices of a given vertex. Once the value of a field changed, the edges from corresponding vertex are activated, then evaluate and update adjacent vertices. The transmission procedure travels related edges and vertices in order. We also optimize it to reducing redundant computing.

When evaluate and update each field on the transmission route, it is necessary to collect all potentially dependant vertices. This procedure is evaluation which needs to reversely search in the directed graph. DFS (Depth first search) [8] is properly used here to travel dependent vertices reversely. The trace of searching, in reverse edge directions, forms a DFS spanning tree, without cycling. Fig. 4(a) and Fig. 4(b) show transmission and evaluation separately.



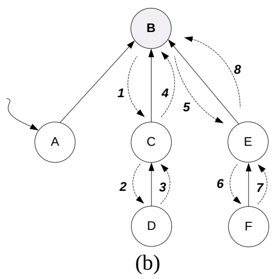


Fig. 4. Examples of coupling searching, (a) transmission, (b) evaluation.

We use a dynamic switch mechanism in this model to emulate simulation — only when a change happens could edges that emit from it be activated. Namely, if the state of a vertex is not actually changed at runtime, the searching will be terminated. It can be available in two aspects, 1) optionally trigger next level transmission to reduce redundant computation; 2) determine edge-connections at runtime to handle potential circling.

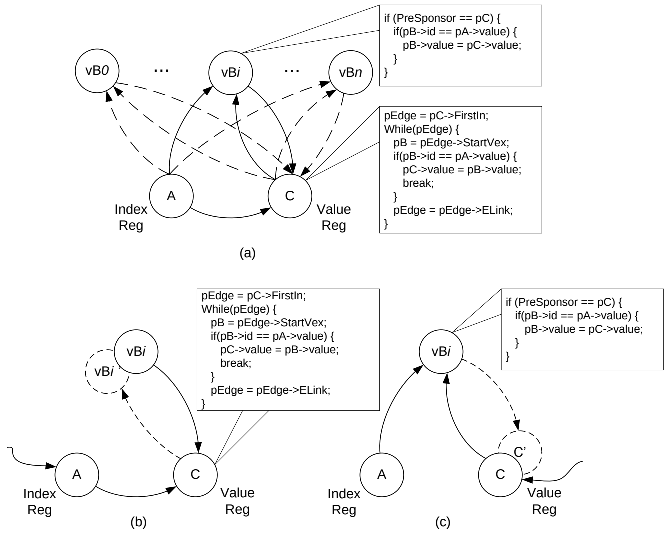
The switch determines edge connection in the correct direction and cut off potential circles in a transmission procedure. It can solve practical problems like indirect registers, which has circles when mapping to graph as shown in Fig. 5. *vB* represents indirect fields vector; *A* and *C* are index and value registers separately. Although there are circle relationship among registers in a static mapping, but these circles is not exist at runtime affected by dynamic switch.

## III. Building Register Subsystem

This section elaborates the implementation of the register subsystem, and presents how it cooperates with verification methodology, and how it practically operates in a test bench.

### A. The Subsystem

We integrate all functional components into a subsystem. These components include a mirror database which record predicted registers’ properties to support reference model and checkers; and an engine that supports search algorithms to handle update of mirror; and there is also a pair of bridges which convert register operations for subsystem with scatter/gather mechanism. Fig. 6 shows a schematic view of the register subsystem.



```
if (PreSponsor == pC) {
  if(pB->id == pA->value) {
    pB->value = pC->value;
  }
}
```

```
pEdge = pC->FirstIn;
While(pEdge) {
  pB = pEdge->StartVex;
  if(pB->id == pA->value) {
    pC->value = pB->value;
    break;
  }
  pEdge = pEdge->ELink;
}
```

```
pEdge = pC->FirstIn;
While(pEdge) {
  pB = pEdge->StartVex;
  if(pB->id == pA->value) {
    pC->value = pB->value;
    break;
  }
  pEdge = pEdge->ELink;
}
```

```
if (PreSponsor == pC) {
  if(pB->id == pA->value) {
    pB->value = pC->value;
  }
}
```

Fig. 5. An example of indirect registers pattern; its solution in graph model and behavior description.

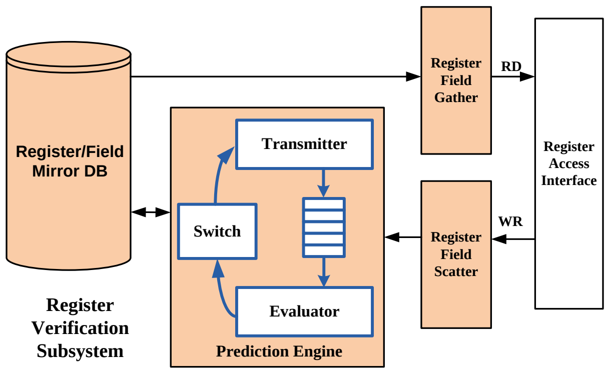


Fig. 6. Schematic view of register subsystem.

The subsystem serves as part of test bench to support reference model as shown in Fig. 7. When a bus transaction is captured by a corresponding monitor, this transaction will be conveyed to register subsystem from a software tunnel. With search engine and mirror database, the subsystem predicts DUT behaviors according to mirror behaviors customized by developers.

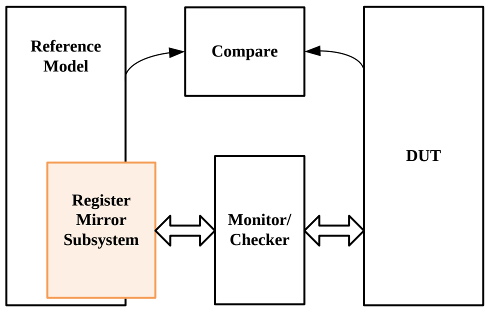


Fig. 7. Integration of register subsystem in a verification environment.

### B. *Access Interface-Scatter/Gather*

As all register fields are flattened in mirror database of register subsystem, and graph vertices associate with register fields rather than registers. We provide a gather/scatter mechanism, for register operation to bridge regular register access to register subsystem. We assume this procedure independent to bus protocols. For each read operation, separated fields are gathered into a register as a whole as shown in Fig. 8; for each write operation, register is segmented into field slices and outputted in order, with ordered fields from event queue trigger vertices' evaluation successively, as shown in Fig. 9.

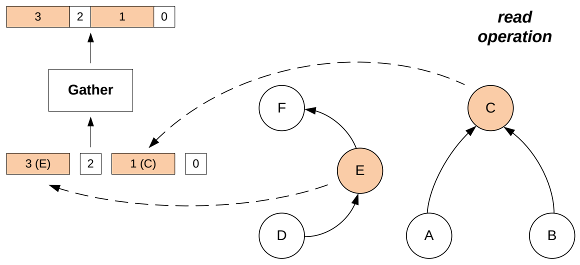


Fig. 8. Gather register fields when reading.

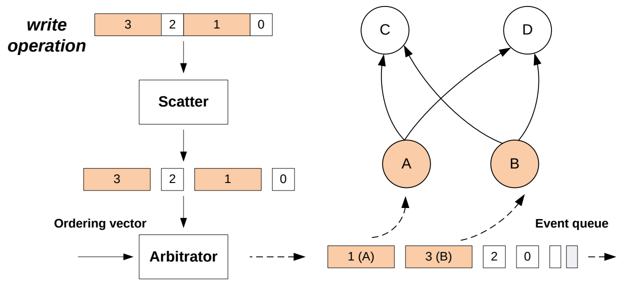


Fig. 9. Scatter register fields when writing.

### C. *Development Flow*

Design specification is usually described by natural language. Developers need to collect and analyze information of design spec by manual or script process. In the development flow, register information in design spec is firstly translated into an intermediate format in which couplings are presented as entries. This intermediate format is loaded and translated to topology structure by verification program at build phase of runtime.

## IV. Analysis and Evaluation

In this section, we estimate the potential benefits of register-subsystem by analyzing practical chip verification projects. We evaluate the performance of the subsystem in verification environment.

### A. *Coupling Complexity*

Register coupling pervasively exists in logic designs, might be more diverse than above patterns, thereby perplexing functional verification. Although it is difficult to evaluate how many workload or codes can be reduced when using this method, but it can be indirectly evidenced by quantify the complexity of register coupling in a project.

We illustrate the complexity of register coupling with three industry-level IP core designs. These three designs are considered representative for either of them has control-intensive and data-intensive features, frequent applied as hardware components in state-of-the-art network devices and data centers. Their detailed information is reported in Table I where HSS is the largest one in the amount of registers and fields.

TABLE I: Register Information of Three IP Core Designs

| **Core** | **Information** | | | |
|---|---|---|---|---|
| | ***virtual register*** | ***virtual register*** | ***Field*** | ***Description*** |
| MC | 130 | 175 | 270 | Core of DDR4 memory controller. |
| PHY | 1093 | 1227 | 5876 | Data-path of DDR memory system, physics layer protocol. |
| HSS | 1110 | 1242 | 3619 | High-speed Serdes, Data communication core, large scale mixed-signal design. |

We quantify the percentage of coupling-involved fields in Fig. 10, implicit and explicit ones in Fig. 11. The analysis is based on register field which is the basic grain of dependency. The statistic shows a varied percentage of coupling-related register/field among the three designs. Most register fields of PHY and HSS relate couplings, and most of them are implicit.

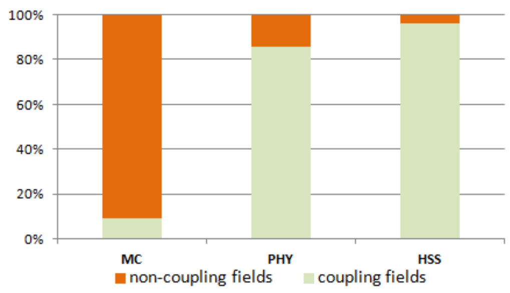


Fig. 10. Count of coupling fields in three developments.

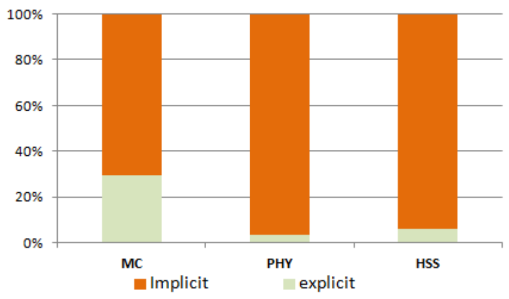


Fig. 11. Percentage of implicit and explicit patterns.

A coupling register, that involves functional state, usually determine export of reference model, or status of functional checker. Based on verification criteria, register mirror should be synchronized in terms of verification. Nevertheless, existing methodology, either UVM or Fusion, lacks a valid mechanism to support coupling. Developers usually need to code conditions for each coupling manually. In a simulation flow, each time when a register operation executes, conditions are triggered to decide whether to update register mirror.

Compound of patterns complicates coupling in width and depth. For instance, a pair of value-dependency registers updated by broadcast. For verification developers, they have to consider the overlap and cascade of effects of multiple couplings. When comes to compound coupling in multi-dimensions, those conditions tend to be more intricate than individual ones, and more difficult for developers to describe its behavior in a test bench. We quantify the complexity of each coupling with three factors by evaluate each register, 1) the depth of a coupling cascade; 2) the number of registers effect as condition vector; and 3) the number of registers to be affected. These factors are illustrated in Fig. 12(a), (b) and (c) report each complex dimension of the three developments.

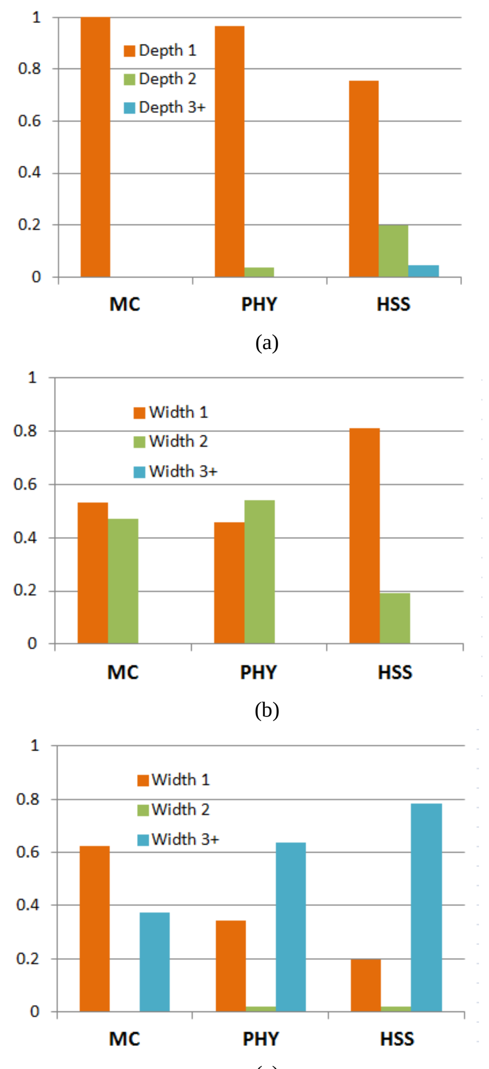


Fig. 12. Indication of development complexity of couplings. (a) cascade Depth; (b) fan-in Width; (c) fan-out Width.

## B. Simulation Costs

We use iProf [9] to emulate an ideal complexity of coupling in a testbench, from sparse connections to intensive connections, to evaluate potential simulation costs. In the experiment, we construct a dummy register subsystem with heavy register couplings (2000 registers with 2000 couplings) in MC testbench. Additional memory cost of register subsystem is 0.7% (48KB for data structure cost) of testbench before run-phase. And this cost will not increase once constructed at build-phase. At run-phase, compared with logic simulation, the cost of CPU and memory of register subsystem is nearly ignorable. Therefore, the cost of verification environment won't result to remarkable performance loss and resource burden.

## V. Related Works

Only in recent decades, mature methodologies substitute direct test in functional verification. And only in recent, specialized approaches were adopted in verification methodologies to manage registers. Both RGM [5] and REG [6] provide mechanism to bridge programming interface to protocol-level transactions, and enable convenient testbench integration. IP-XACT [10] has also been adopted by Accellera Group [11] that to enable automatic generation of codes for register layer classes [12], [13].

However, register coupling is not really well supported in existing methodologies. Although some works used to try to solve behaviors of registers, never successfully establish a uniform model for coupling problem. For example the emulation of indirect register in RGM and REG which is not flexible enough to adapt functional requirements. A special method in IBM Fusion/RTX is used to verify multi-address register which has different access attributes [14], [15]. Work of [16] only solves conflict of register binding for formal verification, and it is similar with indirect registers. In general, the above methods solve specified patterns with specified methods. However, none of them proposed a uniform model to solve this problem, that coupling of configure registers still torments functional verification. Other previous works of registers' verification mainly focus on testbench building, but rarely successful in modeling the relationship of registers, e.g. [17] codifies register descriptions for SoC verification, and works in [18] only optimize structure reusability of register modules but ignore behavior and relation.

Although graph model applied to solve EDA problems, e.g. circuit synthesizing [19] and parallel simulation partitioning [20], but our proposal is fundamentally discrepant with these works. This work focuses on a totally different problem, which only specializes in functional verification rather than circuit and logic designs.

## VI. Conclusion

In this paper, we propose an efficient method to handle register coupling. It explores the commonness of coupling problems to provide a uniform model. It is important in register testing and reference model building. It can significantly reduce the complexity of development overhead of registers and related logic behaviors. It is advantage in aspects of following.

1) Universality. For the first time, the coupling behaviors of registers/fields are modeled as one problem, mapping to a graph model universally.
2) Flexibility. No longer circumscribed with fixed patterns, this solution supports customizable behavior description to facilitate a flexible modeling for explicit and implicit couplings;
3) Scalability. Providing systematical graph-based data structure and algorithms, the modeling method delivers a scalable platform to adapt the variation of verification tasks.

The static analysis has shown a significant value of the graph-based register subsystem in verification development.

In the future, we will consider how to better describe coupling problem with IP-XACT format, and improve further automation. The subsystem will be delivered as a library with application interfaces for mainstream HDL, C/C++ and different verification methodologies in our future work.

## ACKNOWLEDGMENT

Thank colleagues in CDC Dev team of IBM CSTL for helping us in data and experiments, and managers for their continuous support.

## REFERENCES

[1] L. Cai and D. Gajski, "Transaction level modeling: An overview," in *Proc. International Conference on HW/SW Codesign and System Synthesis*, Oct. 2003, pp. 19-24.

[2] *OVM User Guide*, version 2.1.1, Cadence, Mentor Graphics Inc., March 2010.

[3] *Universal Verification Methodology (UVM) 1.1 User's Guide*, Accellera, May 2011.

[4] *Fusion User's Guide*, IBM, 2007.

[5] *Open Verification Methodology Register and Memory Model*, Version 2.4, Cadence Design Systems Inc., August 2010.

[6] *UVM Register Layer Classes*, Cadence Design Systems Inc., June 2012.

[7] J. Bondy and U. Murty, *Graph Theory with Applications*, Elsevier Science Publishing, 1976, ch. 3.

[8] E. Horowitz, S. Sahni, and D. Mehta, *Fundamentals of Data Structures*, Silicon Pr, June 2006, ch. 6.

[9] *Advanced Profiler*, Version 13.20, Cadence Design System Inc., January 2014.

[10] *Standard Structure for Packaging, Integrating, and Reusing IP within Tools Flows*, IEEE Std 1685-2009 for IP-XACT.

[11] Accellera Group. [Online]. Available: http://www.accellera.org

[12] D. Murray, "Verification and automation improvement using IP-XACT," in *Proc. Design and Verification Conference and Exhibition*, 2012.

[13] W. Kruijtzer, Van der Wolf, E. de Kock *et al*., "Industrial IP integration flows based on IP-XACT standards," *Design, Automation and Test in Europe*, pp. 32-37, 2008.

[14] S. Uldrikis, *Register Modeling Class Description and Figtree Utilities*, 2011.

[15] W. Roesner *et al*., *FigTree, API, Semantics, and Behavior, Featuring Scary Usage Cases*, July 12, 2001.

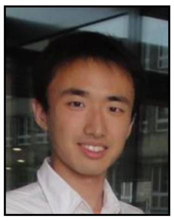
[16] C. Blank, "Formal verification of register binding," presented at Workshop on Advances in Verification, 2000.

[17] N. Kim *et al*., "How to automate millions lines of top-level UVM testbench and handle huge register classes," presented at SoC Design Conference, 2012.

[18] Y. X. Zhang, N. Xu, and Z. G. Liu, "Vertical and horizontal: Towards adaptable register layer for scalable core verification," presented at CDNLive Cadence User Conference, Sept. 2013.

[19] J. Gargers, H. J. Promel, and A. Steger, "Finding Clusters in VLSI Circuits," in *Proc. International Conference on Computer-Aided Design*, Nov. 1990, pp. 520-523.

[20] J. Cong and M. Smith, "A parallel bottom-up clustering algorithm with applications to circuit partitioning in VLSI design," in *Proc. Design Automation Conference*, June 1993, pp. 755-760.



**Zhang Yuxuan** received the BS and MS degrees in computer science from Northwestern Polytechnical University, Xi'An, China, in 2009 and 2012. He joined IBM and worked on chip development since 2012. Now, Yuxuan is a developer of IBM XL C/C++ Compiler, focuses on z/series mainframe. His research interests include computer architecture, high-performance computing and compiler.

**Jiang Guofan** received his BS and MS degrees both in information engineering from Zhejiang Univeristy, Hangzhou, in 2006 and 2008, respectively. From 2008, he worked in IBM China System and Technology Laboratory as an advisory hardware development engineer. Guofan is an expert of functional verification and phisical design. His research interests include transceivers for high-speed wireline communications and high-performance processor microarchitecture.

**Lu Yinchao** received the BS degree in communication engineering and MS degree in microelectronics engineering from the Southeast University, China, in 2009 and 2012, respectively. His diploma thesis focuses on reconfigurable system, low power SoC and cipher processor. In 2012, he joined IBM Microelectronics, Shanghai Chip Design Center, as a verification engineer, focuses efforts on IP and SoC verification methodology. Yinchao is an expert of DDR protocol and High-Speed Serial Link. He has recently moved to Freescale Semiconductor.

**Gou Pengfei** received the BS, MS and PhD degrees from Harbin Institute of Technology. He joined IBM since 2012 as a staff verification engineer, worked on chip functional verification for servers. He worked on IBM XL Compiler backend development. He has recently joined NVIDIA Corporation as a senior GPU Stream Multiprocessors architect. Pengfei is also an expert in system simulator. His research interests ranging from microarchitecture, hardware implementation to software development.