Regular Article

A New Approach and Tool in Verifying Asynchronous Circuits

Tin T. Nguyen, Khoi-Nguyen Le-Huu, Thang H. Bui, Anh-Vu Dinh-Duc

University of Information Technology, Vietnam.

Correspondence: Anh-Vu Dinh-Duc, anhvuiut.edu.vn
Manuscript communication: received 30 August 2012, accepted 21 March 2013

Abstract—Research in asynchronous circuit approach has been carried out recently when asynchronous circuits are presented more widely in electronic systems. As they are more important in human life, their correctness should be considered carefully. Although there are some EDA tools for design and synthesis of asynchronous circuits, they are lack of methods for verifying the correctness of the produced circuits. In this work, we are about to propose a verification method and apply it in making a new version of the PAiD tool that can enable engineers to design, synthesize and verify asynchronous circuits. Experiments in verifying circuits have been also provided in this work.

Keywords—EDA tool, asynchronous circuits, formal verification.

1 INTRODUCTION

Asynchronous circuits have been proposed for decades to introduce circuits of unsynchronized components communicated by handshake protocols [1–4]. They are ideal for contemporary systems such as system-on-chip and network-on-chip as they get rid of the clock distribution problem of synchronous circuits that can lead to many critical issues such as clock skew, jitter, noise, and high power consumption. However, as the asynchronous circuit technology becomes more interesting recently both in academia and industry, some intriguing issues could be raised up both in research and in practice, including description languages, synthesis, verification and simulation.

Researches in high-level description languages have been carried out for a few decades. This approach allows designers to focus on the behaviors of the circuits instead of their actual physical implementation. Many such languages have been proposed, including CSP [5], CHP [6] and ADL (Asynchronous Description Language) [7]. The last one is created especially for asynchronous circuits to simplify the design stage. Thus, it encourages engineers to use asynchronous circuit methodology in hardware design.

Unfortunately, EDA tools for the development process of asynchronous circuits are still outnumbered by those for synchronous circuits. One of the EDA tools, named TAST [8], is developed at TIMA Lab, France for designing many asynchronous systems. Another EDA tool named PAiD [9] is developed at the Computer Engineering laboratory, Ho Chi Minh City University of Technology for research and teaching at the university. The kernel of the tool is a synthesis process that transforms the circuits described in ADL into logic-gate netlists by utilizing multiple transformation stages to ensure that the circuits can be optimized, verified and simulated.

One of the intermediate languages used in the transformation stages is a combination of Petri net and Data Flow Graph (PN-DFG) [10]. This approach takes the advantages of Petri net in representing the control flows and the advantages of DFG in representing the data flows in an asynchronous environment. Researches recently have shown that engineers can easily represent [10], simulate [11], synthesize, place&route [12] asynchronous circuits at high level of abstraction.

When asynchronous systems are used more widely in human life, their correctness should be considered carefully. A circuit can be proved to be correct using mathematical approach such as theorem proving – a formal verification approach that focuses on concrete mathematic foundations. Interestingly, model checking, another formal verification approach, is a more preferable way to verify the circuit when it takes advantage of the diligence of computers and thus, it is able to automate verifying process. Researches on model checking, especially on hardware design, have been carried out for a while [13–19]. In general, studies on model checking are to reduce the impact of the “state space explosion” problem to the field. Research recently [20] shows that, the use of BDD (Binary Decision Diagram) can increase the number of explored states in the memory dramatically [21]. A famous open-source model checking tool based mostly on BDD is NuSMV [22]. More recently, a new approach that uses random walks and an abstraction heuristic guidance has been proposed [23]. As it is based on NuSMV, it takes the advantages of the symbolic approach in dealing with complex systems.

In this paper, a method for verifying asynchronous circuits will be proposed. It is combined into the PAiD tool to extend the power of the tool in making reliable circuits. It is a model checking method based on NuSMV tool using PN-DFG intermediate model to represent circuits and symbolic technique to reduce the
computational complexity.

The rest of this paper is constructed as follows: Section 2 represents related works and background knowledge, including asynchronous circuit description and representation, PN-DFG, model checking and the transformation from PN-DFG for model checking. The new architecture of PAiD tool is introduced in Section 3. Section 4 represents the developed experimentation. The last section is for discussion and the future work.

2 Technical Background

2.1 Related Works

Research on verifying asynchronous circuits has been focused recently [15, 16, 18, 24–29] and covered several approaches, including verification method analysis [18, 24, 25, 29], low-level design analysis [16], and high-/intermediate-level analysis. In verification method analysis for asynchronous circuits, the application of model checking [18, 24, 25] is used more widely than theorem proving [29], probably because model checking is advanced in verifying circuits represented as state-transition systems [18].

In [27, 30, 31], the authors have proposed research directions in verifying asynchronous circuit designs and defined transformation rules to represent PN-DFG in NuSMV tool to verify. This work makes another step in that direction by studying more about the representation and verification using NuSMV and the application of it into a design and synthesis tool.

In a similar research direction, [28] transformed the input models in CSP-like language CHP to PN-DFG and then to a model checking tool to be verified. It differs from our approach when we focus on PN-DFG only and transform into NuSMV. We also analyze the effectiveness of different representation and encoding approaches.

In symbolic approach, related works in [18, 32] encoded the system-under-check and properties directly into BDDs before checking them. They work with Petri nets while we use PN-DFG.

In reducing the impact of the state space explosion problem, another work [26] takes advantages of the abstraction refinement [33] in checking asynchronous designs. Of course, it still faces the same problem as in [33] that the refinement-chain process may waste our time [34]. Actually, its contribution in partitioning the system into components is considered in our approach as “system-as-component” [31].

Nevertheless, research on EDA tools for designing and synthesizing hardware, especially asynchronous circuits, has been carried out for decades [35]. For asynchronous circuits, an EDA tool named TAST [8] has been developed at TIMA Lab, France. Our tool named PAiD is influent by TAST as it is a design environment and synthesis tool. Some synthesis techniques and the application of formal verification into the PAiD tool make it different from TAST.

2.2 Asynchronous Description Language (ADL)

ADL is implemented as a part of the integrated development and design environment PAiD. As an extension of CHP, ADL is the language based on concurrent processes communicating by the mean of exchanging information via communication channels. It allows us to specify the circuit at a high-level abstraction. Regardless of the underlying implementations (communication protocols, structures, ...), designers can easily describe both the structure and behavior of the circuit. There are some other structures and constraints have been provided for facilitating the simulation, verification and synthesis process.

A high-level design of an asynchronous multiplexer and its behavior described in ADL language are illustrated in Figure 1 and Figure 2, respectively.

![Figure 1. An asynchronous multiplexer](image1)

```
Module Buffer_1_Bit ( in input: bit, out output: bit )
Variable internal : bit;
input » internal ;
output « internal
End Module

Module Mux2_1 (in inp1, inp2: bit; in sel: bit; out outp: bit)
Variable x: bit;
sel » x ;
if (x = 0) then
inp1 » y ; outp « y
elseif (x = 1) then
inp2 » y ; outp « y
end if
End Module

Main (in inp1, inp2: bit; in sel: bit; out outp: bit)
Channel inter1, inter2: bit;
Buf_1_Bit(inp1, inter1) ||
Buf_1_Bit(inp2, inter2) ||
Mux2_1(inter1, inter2, sel, outp)
End Main
```

![Figure 2. ADL description of multiplexer in Figure 1](image2)

```
Module Buffer_1_Bit_1 ( in in: bit, out out: bit )
Variable internal : bit;
in input » internal ;
end output « internal
End Module

Module Buffer_1_Bit_2 ( in in: bit, out out: bit )
Variable internal : bit;
in input » internal ;
end output « internal
End Module

Module Mux2_1 (in inp1, inp2: bit; in sel: bit; out outp: bit)
Variable x: bit;
inp1 » y ;
if (x = 0) then
inp2 » y ; outp « y
elseif (x = 1) then
inp2 » y ; outp « y
end if
End Module

Main (in inp1, inp2: bit; in sel: bit; out outp: bit)
Channel inter1, inter2: bit;
Buf_1_Bit_1(inp1, inter1) ||
Buf_1_Bit_2(inp2, inter2) ||
Mux2_1_1(inter1, inter2, sel, outp)
End Main
```

2.3 Petri Net + Data Flow Graph (PN-DFG)

The emergence of PN-DFG model is an important turning point in modeling asynchronous circuits as
it overcomes obstacles in representing some concepts which cannot be dealt in original Petri Net.

**Definition 2.1** A PN-DFG is a 3-tuple model \( PD = (P, T, L) \) where \( P \) is a finite set of places, \( T \) is a finite set of transitions in the original Petri Net and \( L \) is a set of mappings that associate each place or transition with a DFG.

The important difference between PN-DFG and a general Petri Net is the existence of DFG which takes a key role in modeling asynchronous circuits. Moreover, various responsibilities of DFG are mentioned depending on where it is attached. When attached to place, the DFG describes operations that will be executed when the place holds token. For a transition, the DFG is a guard for firing the transition.

![PN-DFG of Buf_1_Bit module](image)

**Figure 3.** PN-DFG of Buf_1_Bit module

It is imaginable that an asynchronous circuit is usually constructed from a number of smaller components. In this approach, each component can be represented efficiently by an appropriate PN-DFG model [10]. Figure 3 represents the appropriate PN-DFG of module Buf_1.Bit (Figure 2).

### 2.4 Model Checking

Model checking is an automated technique that, given a finite-state model of a system and a formal property, systematically checks whether this property holds (for a given state) in that model [17]. Besides, in this technique, systems and properties are modeled as transition systems and temporal logic expressions, respectively.

**Definition 2.2 (Transition system)** A transition system is a 4-tuple \( M = (S, S_0, T, F) \), where \( S \) is the finite set of states, \( S_0 \subseteq S \) is the set of initial states, \( T \subseteq S \times S \) is the set of transitions, and \( F \subseteq S \) is the set of final states.

Temporal logic is the logic of time which is a combination of logical operators such as: \( \land, \lor, \neg \) and modal operators such as \( \bigcirc \) (Next), \( \Diamond \) (Eventually), \( \Box \) (Globally), \( U \) (Until), \( \forall \) (All) and \( \exists \) (Exists). The first four modal operators are state operators, while the others are path operators (or path quantifiers). There are two commonly used temporal logic systems: LTL (Linear Temporal Logic) and CTL (Computational Tree Logic). In LTL, there is no path operators (\( \forall, \exists \)). Contrary to LTL, in CTL, a path operator should be followed by a state operator. For instance, the requirement “This is a deadlock-free system” can be presented in a CTL expression as \((\forall \Box \neg \text{deadlock})\).

### 2.5 PN-DFG to NuSMV

It is imaginable that a system is a big module that may consist of a number of smaller components or just a module consists of everything. Therefore, there are two main representation approaches in NuSMV: (1) system-as-a-whole, where all components are merged into one without any boundary among them, (2) system-as-components, where all components are represented separated and connections among them are represented explicitly. Of course, there is maybe a hybrid approach that takes advantages from the above two. In representing asynchronous systems, which naturally are combined of asynchronous components, the second approach is seemly the reasonable one [31].

**Theorem 2.3** There is a corresponding NuSMV model for each PN-DFG system.

**Proof:** Given a PN-DFG system, one can easily follow PN-DFG to NuSMV transformation rules [31] to construct a NuSMV model.

Figure 4 represents the transformed NuSMV model of the system in Figure 3.

### 3 PAiD Architecture

This section will provide an overview of PAiD – a tool for design and synthesis of asynchronous circuits. A formal verification tool is added, increasing the power of PAiD.

#### 3.1 Overview of PAiD tool

The key objective of PAiD tool is to facilitate the design of asynchronous circuits. It takes the specification of circuit written either in CHP language or in ADL language as the input and produces desired circuit at gate-level as the output. The general design flow of PAiD can be summarized as following steps:

1) Expression of the asynchronous circuit in ADL
2) PN-DFG representation of the circuit specification
3) Synthesis of circuit
4) Optimization and generation of gate netlist

In addition to the design flow, the PAiD tool is constructed as a shell application relating to a set of tools or commands. These commands can be invoked either from command line mode (CLI) or graphic mode (GUI) (Figure 5).

#### 3.2 The new generation of PAiD tool

Regarding to the lack of checking whether the targeted circuit matches the specification, a new version of PAiD tool is introduced with a method for verification. Its new architecture is represented in Figure 6, in which a verification module is attached.
Figure 4. A partial NuSMV description of Buf_1_Bit Module

Figure 5. PAiD tool in GUI mode

Figure 7 illustrates the proposed verification flow. Actually, in order to be independent of lower level implementation, the PN-DFG model generated by original PAiD tool is described at high level abstraction. In our verification module, we already examined and then chose 4-phase handshaking protocol to expand the

PN-DFG model. Other protocols can be also selected for different implementations. We applied the PN-DFG to SMV transformation rules described earlier to automatically generate appropriate SMV description file afterward. Finally, this SMV file will be used by NuSMV model checking tool for running verifying process.

4 Experimentation

This section will demonstrate their feasibility in circuit verification by applying to some case studies.

4.1 Case Studies

4.1.1 Asynchronous Arbiter: Arbiter [36] is a common electronic device in digital system that controls access
to a shared resource among different client processes. A block diagram of an asynchronous arbiter is depicted in Figure 8.

![Figure 8. Architecture of the asynchronous arbiter](image)

Two clients compete to access a common resource. Each client requests the arbiter to access the resource via its corresponding channel (C1 or C2). The arbiter must determine which request is served first. The channel C then shows the number of selected client to the resource.

A verifying specification “If there is a pending request in channel c1, there should be an execution path so that c presents 1 eventually (Arbiter_P)” is expressed in form of CTL as follows:

\[
AG(c1\text{-}request \rightarrow EF(c = 1))
\]  

4.1.2 Asynchronous Selector: This benchmark represents a selector that consists of one input and two outputs. It will deterministically select one output (S1 or S2) to transfer input data (E) depending on the value read from specific channel (C). The corresponding architecture is depicted in Figure 9.

![Figure 9. Architecture of the asynchronous selector](image)

A verifying specification “If channel C chooses output S1 by sending value 0, the value of channel E should be transferred to S1 eventually (Selector_P)” is expressed in NuSMV as follows:

\[
AG((C = 0 & E = 1) \rightarrow EF(S1 = 1))
\]  

4.1.3 Asynchronous Multiplexer: The asynchronous multiplexer works similarly to the asynchronous selector described above. Depending on the value read from a specific channel, the multiplexer will enable data from appropriate input source to be transferred to output. A two-input multiplexer has been shown in Figure 1.

A verifying specification “If the value read from channel sel is 0, the data of channel inter1 must be transferred to channel output (Multiplexer_P)” is expressed in NuSMV as follows:

\[
AG((sel = 0 & inter1 = 1) \rightarrow EF(output = 1))
\]  

4.1.4 Distributed Mutual Exclusion: The Distributed Mutual Exclusion (DME) is a well-known mutual exclusion problem in which some cyclic processes, called “masters”, share a common resource. The resource is controlled by servers. Each master has its own server. This benchmark analyses one simple solution of DME problem, called “The Reflecting Privilege” [37]. Each server will have a flag indicates whether it has privilege to access common resource or not. There is maximum one privilege in a system. A master can access to common resource if its private server holds the privilege. Otherwise, its server will request privilege to the next server. The privilege will move in an opposite direction to the request.

A specification “Only one master has right to access common resource (DME_P)” is represented in NuSMV as follow (using 2 DME cells):

\[
AG((master0.cs + master1.cs) <= 1)
\]  

where state cs of each component master is defined as toint(master.state = criticalsection).

4.1.5 Asynchronous Pipelined Finite Impulse Response Filter: An asynchronous pipelined finite impulse response filter (FIR filter) is characterized by the following equation:

\[
y(n) = h(n) \times x(n) = \sum_{k=0}^{N-1} (h(k) \times x(n - k))
\]  

It is also called N-tap FIR filter. The architecture of a 3-tap asynchronous pipelined FIR filter is shown in Figure 10. In each level, there are one buffer (L component), one Asynchronous Pipelined Multiplier (APM component) and one Adder component.

A verifying specification “The value of a buffer stays unchanged while it has not been read by the next buffer yet (FIR_P)” is expressed in NuSMV as follows:

\[
AG(x = 1 \rightarrow AF(A[L0 = 1 U L1 = 1]))
\]  

4.2 Experimentation Setup

The chosen environment is an Intel Core i5 2500 Processor PC with 3.4 GB of RAM that runs Fedora 16 Verne. NuSMV 2.5.4 is used as model checking tool. In current experimentation, NuSMV is set to its running default options.
4.3 Experimentation Results

The results derived from the benchmarks include transforming results and verifying results. Transforming results demonstrate whether the new PAiD tool automatically generates proper SMV descriptions of asynchronous circuits from PN-DFG models. Those results are shown in Table I. In this table, some statistic information of the corresponding high-level 4-phase asynchronous circuits has also been represented. It takes the system-under-test in form of PN-DFG, an intermediate-level representation of asynchronous circuits in PAiD tool, and transforms it to be verified in NuSMV, a famous open-source symbolic model checking tool. Although the current approach is only applied at an abstract level during the synthesis process, it shows that the verification can be used in many other internal synthesis levels. Therefore, other correctness concerned at different levels can be verified similarly. For example, at the high level of abstraction, designers may focus on the abstract behaviors of the circuits, but in the low level, they may consider more on the temporal-related correctness of the handshaking protocol. Of course, it should be studied more in the future.

Table I

<table>
<thead>
<tr>
<th>Benchmark specification</th>
<th>Transforming successful?</th>
<th>Statistic information</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Num. of places</td>
</tr>
<tr>
<td>Asynchronous Arbiter</td>
<td>Yes</td>
<td>29</td>
</tr>
<tr>
<td>Asynchronous Selector</td>
<td>Yes</td>
<td>35</td>
</tr>
<tr>
<td>Asynchronous Multiplexer</td>
<td>Yes</td>
<td>41</td>
</tr>
<tr>
<td>Distributed Mutual Exclusion</td>
<td></td>
<td>86</td>
</tr>
<tr>
<td></td>
<td>129</td>
<td>138</td>
</tr>
<tr>
<td></td>
<td>172</td>
<td>184</td>
</tr>
<tr>
<td></td>
<td>258</td>
<td>2/6</td>
</tr>
<tr>
<td></td>
<td>344</td>
<td>368</td>
</tr>
<tr>
<td>Asynchronous Pipelined FIR Filter</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2-tap</td>
<td>Yes</td>
<td>87</td>
</tr>
<tr>
<td>3-tap</td>
<td>Yes</td>
<td>130</td>
</tr>
</tbody>
</table>

Table II

<table>
<thead>
<tr>
<th>N</th>
<th>Property</th>
<th>Running time (s)</th>
<th>Num. of BDD nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Asynchronous Arbiter</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Arbiter_P</td>
<td>0.02</td>
<td>20,234</td>
</tr>
<tr>
<td></td>
<td>Asynchronous Selector</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Selector_P</td>
<td>0.07</td>
<td>140,211</td>
</tr>
<tr>
<td></td>
<td>Asynchronous Multiplexer</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Multiplexer_P</td>
<td>0.17</td>
<td>277,285</td>
</tr>
<tr>
<td></td>
<td>Distributed Mutual Exclusion</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>DME_P</td>
<td>0.22</td>
<td>614,474</td>
</tr>
<tr>
<td></td>
<td>DME_P</td>
<td>0.52</td>
<td>1,189,097</td>
</tr>
<tr>
<td></td>
<td>DME_P</td>
<td>2.05</td>
<td>1,449,572</td>
</tr>
<tr>
<td></td>
<td>DME_P</td>
<td>49.82</td>
<td>7,588,691</td>
</tr>
<tr>
<td></td>
<td>DME_P</td>
<td>Time out</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Asynchronous Pipelined FIR Filter</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>FIR_P</td>
<td>1692.43</td>
<td>21,081,300</td>
</tr>
<tr>
<td></td>
<td>FIR_P</td>
<td>Time out</td>
<td></td>
</tr>
</tbody>
</table>

4.3.1 Experimentation Results

The number of BDD nodes allocated indicates the complexity of checking process. The specific numbers of running times and BDD nodes indicate that corresponding properties are verified as true. Furthermore, it is easy to figure out from the table that when the number of BDD nodes increases, the corresponding running time relatively increases too.

Table II shows that NuSMV model checking can easily check for small models such as asynchronous arbiter, asynchronous selector, asynchronous multiplexer and also some of DME benchmarks. For the 6-cell DME, more than seven million BDD nodes are created but NuSMV model checking took less than one minute to examine the model in its verifying process.

When the problem becomes more complicated such as DME 8 cells or 3-tap asynchronous pipelined FIR filter, NuSMV is running out of expecting time, which is set to 4 hours. They are marked as ‘Time out’ in the table. Hence, the corresponding properties are not verified.

Taking a deep consideration on that table, the progress of running time is not linear to that of BDD nodes allocated. Furthermore, the running time has been grown exponentially. Hence, it can lead to the time-out situation. Clearly, the running time of NuSMV must be studied furthermore in order to verify asynchronous circuits efficiently. This is left for the future works.

5 Conclusion

In this work, a method for verification of asynchronous circuits has been represented. It takes the system-under-test in form of PN-DFG, an intermediate-level representation of asynchronous circuits in PAiD tool, and transforms it to be verified in NuSMV, a famous open-source symbolic model checking tool. Although the current approach is only applied at an abstract level during the synthesis process, it shows that the verification can be used in many other internal synthesis levels. Therefore, other correctness concerned at different levels can be verified similarly. For example, at the high level of abstraction, designers may focus on the abstract behaviors of the circuits, but in the low level, they may consider more on the temporal-related correctness of the handshaking protocol. Of course, it should be studied more in the future.

The PAiD tool for designing and synthesizing asynchronous circuits has also been improved by employing a verification module. The new architecture allows us to continuously extend the tool to utilize more verification techniques in the future. The tool then can be used more widely in academia and industry as it empowers engineers to describe the behaviors of the desired circuits, verify their correctness and synthesize them into logic gates.

However, the research in this field should be extended in many directions such as applying abstraction techniques in verification to reduce the impact of the ‘state-space explosion’ or to guide the search-for-faulty
in model checking techniques. Some EDA related research areas such as multi-level circuit simulation and synthesis optimization should be studied more to help circuit designers make quality products. Other future works may include studies in describing circuits in abstract-level that is similar to the formal specification research area, and component-based circuit design approach that employs both formal specification and component-based automation research areas.

Acknowledgments

This work was supported under Project 113/HD- SKHCN granted by the Department of Science and Technology of Ho Chi Minh City.

References


Tin T. Nguyen received the B.E. (2011) degree in computer engineering from Vietnam National University-HCMC University of Technology. He is an Assistant Lecturer, Faculty of Computer Science and Engineering, HCMC University of Technology. His current interests include embedded systems and chip design, especially asynchronous circuits.

Khoi-Nguyen Le-Huu is an Assistant Lecturer at the University of Information Technology (UIT). He formerly worked as a research assistant at the University of Technology (HCMUT), where he received the B.E. (2012) degree from the Honor program in computer engineering. His research interests include digital design, FPGA-based design, image processing and asynchronous (“clock-less”) circuits design.

Thang H. Bui received the B.E. (1997) degree in computer engineering from Vietnam National University-HCMC University of Technology, M.E. (2001) degree in computer engineering from Asian Institute of Technology, Thailand, and PhD (2010) degree in computer science from New South Wales University, Australia. He is a Lecturer, Faculty of Computer Science and Engineering, Vietnam National University-HCMC University of Technology. His current interests include information systems, formal methods, especially model checking.

Anh-Vu Dinh-Duc is Professor and Vice-Rector at the University of Information Technology (UIT), where he leads the Embedded Systems Group. Previously, he was Professor of Computer Engineering at the University of Technology (HCMUT). His research interests include design automation of embedded systems, hardware/software verification, VLSI CAD, and reconfigurable architectures. Prof. Anh-Vu Dinh-Duc received the Master and Ph.D. degrees in Microelectronics from the Institut National Polytechnique de Grenoble (INPG), France, in 1998 and in 2003, respectively. Prof. Anh-Vu Dinh-Duc currently serves as a program/organizing committee member of several ACM and IEEE conferences. He has been a member of the IEEE since 1997.