Automation of Processor Verification Using Recurrent Neural Networks

Martin Fajcik, Pavel Smrz
Department of Computer Graphics and Multimedia
Brno University of Technology, Czech Republic
{ifajcik, smrz}@fit.vutbr.cz

Marcela Zachariasova
Department of Computer Systems
Brno University of Technology, Czech Republic
zachariasova@fit.vutbr.cz

Abstract—When considering simulation-based verification of processors, the current trend is to generate stimuli using pseudo-random generators (PRGs), apply them to the processor inputs and monitor the achieved coverage of its functionality in order to determine verification completeness. Stimuli can have different forms, for example, they can be represented by bit vectors applied to the input ports of the processor or by programs that are loaded directly into the program memory. In this paper, we propose a new technique dynamically altering constraints for PRG via recurrent neural network, which receives a coverage feedback from the simulation of design under verification. For the demonstration purposes we used processors provided by Codasip as their coverage state space is reasonably big and differs for various kinds of processors. Nevertheless, techniques presented in this paper are widely applicable. The results of experiments show that not only the coverage closure is achieved much sooner, but we are able to isolate a small set of stimuli with high coverage that can be used for running regression tests.

Index Terms—Functional Verification, Automation of Verification, Neural network, Recurrent Neural Network, Hopfield Network, UVM, Coverage-Driven Verification, Optimization Problem, Combinatorial Optimization

I. INTRODUCTION

With a raising demand on application-specific processors in today’s market it becomes a necessity to come up with new improved approaches that tackle the challenging task of their complex verification. The current trend is to automate some of the routines of simulation-based verification, for example, generation of stimuli or targeting corner cases in coverage.

Pseudo-random stimuli generation is well defined in the SystemVerilog language standard [1], in the Universal Verification Methodology (UVM) [2] and even in a new standard for portable stimuli (currently in its final revision state). Various pseudo-random stimuli generators (PRGs) can be utilized for that purpose, either those inbuilt in RTL simulators or external ones (e.g. in C++) connected through direct programming interface. Pseudo-randomness is achieved through constraints which are able to restrict the process of generation to gainful scenarios. This approach was devised to strike the balance between direct and random simulations. Instead of writing detailed verification patterns manually, the user provides a set of constraints and PRG generates a huge amount of stimuli in the order of seconds.

The completeness of stimuli generation can be measured by various metrics commonly known as coverage (we will consider RTL coverage metrics in this paper such as branch, statement, expression or functional coverage). We say that the processor reached coverage closure when the value of every monitored coverage metric is high enough to proclaim it as verified. Such feedback-controlled method based on coverage analysis is called Coverage-Driven Verification (CDV). A problem of this approach comes from redundancy inbuilt in randomness, because if the coverage feedback is not properly propagated to PRG and reflected by suitable constraints, the same redundant stimuli can be applied to Design Under Verification (DUV) without any coverage increase. Even intuitively we can observe that if the distribution of generated stimuli would reasonably change in every consequent stimuli generation (i.e. lower probability for already generated values), we would accelerate the verification processes and reach corner cases and coverage closure faster.

From the practical point of view, this paper targets two industry-wide problems:

1) Speeding up the verification process by doing optimization of PRG constraints in such a way that coverage closure of DUV is achieved as fast as possible.
2) Seeking the smallest set of stimuli which reaches coverage closure for DUV. Such stimuli set is then ideal for regression testing.

To resolve these problems, a new method for optimization of PRG constraints based on recurrent neural network (RNN) is proposed. This method has 3 significant contributions. Firstly, it automates CDV – verification runs are repeated until either the coverage closure is reached or terminating conditions are met. Secondly, the proposed non-invasive solution can easily work with all verification environments that support PRG and coverage analysis. Thirdly, values of some RNN parameters and methods for their further tuning are suggested based on extensive experiments.

II. RELATED WORK

Combinatorial optimization techniques already appeared in works related to automation of verification processes. Kitchen

This paper has been supported by Brno University of Technology (FIT-S-17-3994) and by the EC Horizon 2020 project MegaM@Rt2 ECSEL-JU, No. 7337494.
and Kuehlmann in [3] introduced a PRG using hybrid constraint solver based on Markov-chain Monte Carlo methods, which dynamically alters PRG distribution. However, these distributions were not biased from the coverage analysis like this work and further mentioned works are. In the past, two general approaches based on constraint analysis have been proposed: feedback-based CDV (FBCDV) and CDV by construction (CDVBC). FBCDV relies on a feedback from coverage analysis to be present during simulation and modifies the constraints to the PRG. In contrast, CDVBC approach relies on a generated external model of DUV which is used to generate stimuli designed to accurately hit the coverage tasks. CDVBC based approaches are commonly based on transformation of coverage tasks into Boolean logic (e.g., conjunctive normal form) and harvesting the power of SAT solvers [4] [5]. Although it is guaranteed that this systematic approach hits every coverage task, and using incremental SAT solvers it can even hit multiple tasks at once, it has been proven that the SAT problem is NP-complete and thus probably (if P!=NP) can never be optimal. There are also feedback-based CDV approaches utilizing SAT solvers, e.g. in [6] the DUV is reduced based on a current program slice (to reduce its complexity). PRG constraints are then treated as constraint satisfaction problem and solved with a word-level SAT solver. The next category covers not fully automated approaches. For instance in [7], the tool based on feedback-adjusted Markov Models called StressTest was proposed. The tool is capable of on-the-fly optimization of constraints, but needs an assistance of engineer to provide a template describing interface protocols of the system. Finally, there are FBCDV approaches based on genetic algorithms (GA). Authors of [8] applied GA for automated generation of stimuli based on source code of specific software application. Naturally, such approach neglected all the details concerning the processor hardware irrelevant to the verified application. The most similar approach to this paper is proposed in [9]. The work introduced capability of on-the-fly constraint optimization as well as seeking of an optimal stimuli set. GA was integrated directly into the UVM veriﬁcation environment with proﬁtable values of GA parameters. The work presented in this paper builds on the GA approach (co-authored by M. Zachariasova). The GA optimizer has been replaced with a simple but powerful RNN. The proposed approach can be implemented with any veriﬁcation environment that utilizes coverage analysis and PRG. Various neural network models with different network dynamics are proposed and compared to each other in Section IV. We also provide a short comparison to the GA approach from [9] in order to ﬁnd out which kind of optimizer is more suitable for the CDV task.

III. PROPOSED APPROACH

A. Integration of combinatorial optimization and veriﬁcation environment

Every optimization technique needs a method to evaluate quality of its solutions. This method is usually referred to as an objective function (OF) (also known as the cost or the fitness function). Since we are proposing a feedback-based CDV technique, the OF value needs to be determined based on the coverage analysis data. That is the reason why the proposed technique needs to be placed in a closed-loop architecture as shown in Fig. 1.

![Fig. 1. Integration of RNN into the closed-loop solution.](image)

Each loop in the automated veriﬁcation life-cycle is composed of following steps:

1. The optimizer (in our case RNN) generates changes to PRG constraints.
2. A stimuli set is generated via PRG.
3. The controller program starts a new simulation (or series of simulations) using the newly generated stimuli set.
4. At the end of simulation, most of HDL simulators are capable of exporting the database containing information collected during the simulation.
5. A subset of the collected data (in our case various types of coverage metrics) is used to evaluate OF.

Please note that this approach is non-invasive to the veriﬁcation environment.

The PRG constraints deﬁne syntax and semantics of valid stimuli set. They can take many forms ranging from deﬁning forbidden port combinations, valid instruction combinations, probabilities of generated values, to restrictions of already used variables. The constraints can also reduce generated stimuli to a subset valid for various portable stimuli scenarios. In our experimental evaluation we focus on veriﬁcation of processors. Therefore, the optimizer is changing probability constraints on the instruction set level, the instruction level and the sub-instruction level (all these are closely explained in Fig. III-C2). The generated set of stimuli is in fact represented by a set of programs compiled into binaries. For the veriﬁcation purposes we have used the UVM environment and all the scripts controlling RTL simulation, PRG and RNN were written in Python 2.7.

B. Hopﬁeld model

Today, neural networks (NN) play an important role in the ﬁeld of artiﬁcial intelligence, where they become an universal approach for modeling cognitive processes based on human brain image. They are currently a popular concept for the purpose of classiﬁcation and regression. In [10], J.J. Hopﬁeld has shown that neural networks are also capable of dealing with combinatorial optimization problems when, using his previously proposed RNN called Hopﬁeld network (HN), he was able to solve traveling salesman problem. In this paper,
we have been inspired by his idea and we propose a similar
network models capable of dealing with automation of CDV.

HN work with autonomous units called neurons. We will
label each neuron from the network with a number from
arithmetic progression \( m_{k+1} = m_k + 1 \) starting from
the number 1. A neuron \( i \) can be defined by a triplet: input
vector \( \mathbf{x} = [x_1, \ldots, x_n]^T \) (also called feature vector), output state \( v_i \),
and threshold \( \theta_i \). Each neuron combination has a connection
parameter called connection weight. We will denote the value
of such weight from the neuron \( i \) to the neuron \( j \) as \( w_{ij} \). For
HN of size \( n \) weights between neurons can be defined by the
weight matrix (1).

\[
W = [w_1, \ldots, w_n] = \begin{bmatrix}
  w_{11} & \cdots & w_{1n} \\
  \vdots & \ddots & \vdots \\
  w_{n1} & \cdots & w_{nn}
\end{bmatrix}
\]  

(1)

The output of the neuron \( i \) is given by the composition (4) of
linear basis function \( \xi \) (2) with the sigmoid activation function
(sometimes called the logistic function) \( \Psi \) (3).

\[
\xi(\mathbf{x}, W, \theta_i) = \mathbf{x}W^\top - \theta_i = \sum_{k=1}^{n} w_{ki}x_k - \theta_i
\]  

(2)

\[
\Psi(z) = \frac{1}{1 + e^{-z}}
\]  

(3)

\[
v_i = \Psi(\xi(\mathbf{x}, W, \theta_i))
\]  

(4)

The schema in Fig. 2 wraps up previous definitions. From the
output \( v_i \) of each neuron \( i \) we can define a network output
vector (or a state vector) \( \mathbf{v} = [v_1, \ldots, v_n]^T \).

In addition, the network constructed from such neurons
needs to satisfy several restrictions to meet the criteria of
the HN. Let \( \mathcal{N} \) denote a set of NN neurons, then we define
following constraints:

1) There must be a defined connection weight between each
   of two neurons (HN is a fully connected network).

2) Diagonal values of \( W \) are equal to 0, so the value of
   the connection to the neuron itself is not used during the
   network evolution (see the further text for explaining the
   evolution process).

\[
\forall i \in \mathcal{N} : w_{ii} = 0
\]  

(5)

3) The network weights are symmetric.

\[
\forall i, j \in \mathcal{N} : w_{ij} = w_{ji}
\]  

(6)

Since HN is a RNN, its state changes over time during its
evolution. To be able to determine, when the network is
in which state, we define the model time as follows: let a
variable \( t; t \in \mathbb{N} \land t > 0 \) be a time variable, then \( v_i^{(t)} \) is a
parametrized notation of the neuron outputs and \( \mathbf{v}^{(t)} \) is a
network state which will be used to denote their value in
certain evolution time \( t \). Equation (4) that uses model time
can now be rewritten as \( v_i^{(t+1)} = \Psi(\xi(\mathbf{v}^{(t)}, \mathbf{W}, \theta_i)) \). Next,
we define a network trajectory in time \( t \) to be a sequence
of states \( \mathbf{v}^{(1)}, \mathbf{v}^{(2)}, \ldots, \mathbf{v}^{(t)} \). There are two approaches used
for the neuron activation policy. In the synchronous policy,
all the units are activated simultaneously in each time step.
In the asynchronous policy, only one unit at each time step
is activated. The unit being activated can be chosen either
randomly or sequentially. In the following text, we will assume
that we are working with the random asynchronous policy.
In order to solve optimization problems, the OF which is
in context of neural networks commonly called the energy
function \( (E) \), needs to be defined. In [10], J.J. Hopfield defined
a monotonically decreasing energy function, which can be
used for simple problems like TSP, but does not seem to be
applicable in this case (we will define our own in the further
text).

C. Proposed solutions

It turns out that the mapping of the CDV problem to the
RNN optimization model is quite straightforward. The state
space of the processor is reachable via input stimuli with
probabilities defined as constraints for PRG. Since probability
\( P(A) \) of an event \( A \) has a domain of \([0, 1]\), we can represent
each of these probabilities with a neuron using a sigmoid
activation. In [11], an approach for designing custom energy
function \( E \) and tips for tuning \( \mathbf{W} \) and \( \mathbf{\theta} \) are presented.
As a result, we defined energy function as (7). Importance
of each coverage metric \( c \in C \) is expressed via relative
weight coefficients \( a_c \). Function \( \text{Coverage}_c(\mathbf{v}) \) reflects the
total coverage reached so far with stimuli generated from the
network state \( \mathbf{v}^{(t)} \) together with all previously accepted NN
states of evolution trajectory \( \mathbf{v}^{(1)}, \mathbf{v}^{(2)}, \ldots, \mathbf{v}^{(t-1)} \).

\[
E(\mathbf{v}^{(t)}) = \sum_{c \in C} a_c P(\text{Coverage}_c(\mathbf{v}^{(t)}))
\]  

(7)

It can be seen that if the network evolution with trajectory
\( \mathbf{v}^{(1)}, \mathbf{v}^{(2)}, \ldots \) starts in any state and only neurons leading to
the energy difference \( \Delta E \geq 0; \Delta E = E(\mathbf{v}^{(t)}) - E(\mathbf{v}^{(t+1)}) \)
are activated, the optimal solution state \( \mathbf{v}^* \) or a near optimal
local minimum state will be reached eventually. Note that
our objective is to reward new constraint settings which
have not been used yet and lead verification process to so far unexplored processor states, instead of seeking the best initial configuration (as we would if we would not consider previously collected coverage data). Finally, $P$ represents a penalty function. In fact, it just transforms coverage values to fit the minimization problem expectations. Since the maximum coverage for each metric is 100%, we define the penalty function as (8).

$$P(x) = (100 - x)^2$$

(8)

To clarify, we have been using various coverage metrics in the evaluation, e.g. functional (defined by covergroups and coverpoints in SystemVerilog), statement, branch, expression and FSM coverage (defined automatically by the simulation tool). For weight coefficients $a_o$, the value 1 has been assigned to functional coverage (since it is the most important in terms of functional verification) and the value 0.0001 to other coverage metrics.

The next task is to select model parameters $W, \theta$. These had to be chosen in such a way that reachability of any network state would be possible and probability of reaching these states should not be too small. Ideally, during the NN evolution the output value of each neuron should be able to move in both directions, closer to 0 or closer to 1, but in general, all output values should have an average probability value 0.5. This is the reason for defining a balanced weights model, which must satisfy the property (9).

$$\forall i \in N(v_i^{(t)} = 0.5) \implies \forall i \in N(v_i^{(t+1)} = 0.5)$$

(9)

Finally, the last modification is the sigmoid function (3) which needs to be parametrized with the parameter $\lambda$ in order to adjust the function steepness as shown in (10). Intuitively, the slope needs to fit the network in such a way that wide range of values between $[0, 1]$ can be selected. The value of this parameter was chosen experimentally, see Section IV.

$$\Psi(z)_\lambda = \frac{1}{1 + e^{-z\lambda}}$$

(10)

To finalize this section, the following two balanced weights models are proposed.

1) Bipolar model: The idea behind this concept is to split non-diagonal weights in $W$ into two equal groups, assigning one group with the value 1 and other group with $-1$ in such a way that each neuron would have a half of the connections with the weight 1, and the second half with the weight $-1$ as is shown in Algorithm 1. In case of the odd number of non-zero weighted connections, weights of one group need to be adjusted in order to satisfy the balanced weights model constraint as is presented in Algorithm 2. It can be shown (by mathematical induction) that the weight matrix of this model cannot be symmetric.

2) Acyclic graph model: This model is designed specifically for the processor verification since it uses PRG generating programs. The NN topology comes from directed acyclic graph corresponding to the model of the instruction set architecture (ISA). ISA is usually designed on multiple levels. Sub-instruction level describes elements, from which instructions are composed, for instance, instruction operation codes, registers or immediate operands. Instruction elements are then put together from these sub-instructions. In addition, sometimes it is more beneficial to put elements or other sets together into a set and describe more complex elements on the instruction set level (e.g. all 2-operand instructions contained in 2_operands set have the same syntax, they just differ with opcodes). Our PRG constraint model corresponds to ISA model. Using such constraint model, we are able to set probability constraint for each member of the set, so each neuron represents a unique instance of the pair (set,member), where member is either another set or an element. In this model, we have connected only those neuron instances in which the member of one instance was the set of another instance or both instances have the same set. For demonstration, see an example of such graph in Fig. 3. PRG using such constraint model generates instruction or sub-instruction with a probability of the connection to his parent with respect to probabilities the parent has with its grandparent and so on (analogically to a Bayesian network). The construction of $W$ matrix for this model is shown in Algorithm 3.

D. Model usage

These models are able to solve industry-wide problems presented in Section I. Algorithm 4 can optimize PRG constraints on-the-fly and speed up the verification process, see results in
Algorithm 3: Acyclic graph model weights

Data: Number of neurons \( n \), Unique instance vector \( e \)
Result: Weight matrix \( W \) of size \( n \times n \)

\[
\begin{align*}
&\text{for } i \leftarrow 0 \text{ to } n \text{ do} \\
&\quad n_{\text{connections}} \leftarrow 0; \\
&\quad \text{sum} \leftarrow 0; \\
&\quad \text{for } j \leftarrow 0 \text{ to } n \text{ do} \\
&\quad \quad \text{if } i = j \text{ then} \\
&\quad \quad \quad n_{\text{connections}} \leftarrow n_{\text{connections}} + 1; \\
&\quad \quad \text{else if } e_i,\text{member} = e_j,\text{set} \text{ or } e_i,\text{set} = e_j,\text{member} \text{ or } e_i,\text{set} = e_j,\text{set} \text{ then} \\
&\quad \quad \quad w_{i,j} \leftarrow \text{sign}; \\
&\quad \quad \quad \text{sum} \leftarrow \text{sum} + \text{sign}; \\
&\quad \quad \quad \text{sign} \leftarrow -\text{sign}; \\
&\quad \quad \quad n_{\text{connections}} \leftarrow n_{\text{connections}} + 1; \\
&\quad \quad \text{end} \\
&\quad \text{end} \\
&\quad \text{if } n_{\text{connections}} - 1 \text{ is odd then} \\
&\quad \quad \text{oddModifier} \leftarrow \frac{n_{\text{connections}} - 2}{n_{\text{connections}}}; \\
&\quad \quad \text{else} \\
&\quad \quad \quad \text{oddModifier} \leftarrow 1; \\
&\quad \text{end} \\
&\text{MODIFYWEIGHTS(oddModifier, sum, W)} \\
&\text{return } W
\end{align*}
\]

Section IV. Algorithm 5 seeks the smallest processor program set that can be used as a regression test suite.

IV. RESULTS

All experiments were realized with Intel Core i7 3610QM processor with 8GB RAM and 64-bit Debian 8.7. The objects of verification were two processors provided by Codasip, each of markedly different complexity. Codasip uRISC (area 16k gates in 55LP, frequency 400MHz) is a 32-bit RISC micro-architecture with 4-stage pipeline, used mainly for demonstration and tutorial purposes. The NN controlling PRG for uRISC verification contained 41 neurons. Codix Cobalt (area 24k gates in 55LP, frequency 500MHz) is a high-performance production 32-bit RISC micro-architecture with 5-stage pipeline [12]. The NN controlling PRG for Cobalt verification contained 1020 neurons. All the generated programs were loaded directly into the memory of the processor, each program with an approximate length of 100 instructions. In one of the experiments we tried a range of random distribution models to initialize the initial NN state, such as triangular distribution and various uniform distributions with interval restrictions and it turns out that the uniform distribution with values drawn from the interval \([0.4,0.6]\) seems to give the best results. After several experiments, the sigmoid steepness parameter \( \lambda \) has been set to 0.9. Finally, we have experimented with various values of the parameter defining the maximal length of the epoch and the value 20 seems to be an ideal trade-
Fig. 4. Comparison of the proposed models and the GA during the Codasip urISC processor verification. The optimal stimuli set was found in 12.05 hours, the final set was able to reach 100% coverage in 8.94 minutes and contained 70 assembler programs. In contrast, default approach generated 390 programs in the shown interval.

Fig. 5. Comparison of the proposed models during the Codix Cobalt processor verification. After 63.73 hours of the optimal stimuli set optimization the final set was able to reach 87.6% coverage in 55.39 minutes and it contained 326 programs. The default approach generated 1000 programs during the measured interval.

A large speed up of the verification process and have been able to find fairly small set of regression tests. However, in some initial time interval of the verification process, state-of-the-art approach (in our experiments referred to as ‘default’) seems to be more efficient than the proposed approaches. In the future work we would like to focus on this part of the proposed technique in order to develop further speed-ups and to define an interconnection of our work to a new portable stimuli standard after it will be publicly available.

REFERENCES

[1] “IEEE Standard for SystemVerilog—Unified Hardware Design, Specification, and Verification Language,” IEEE Std 1800-2012 (Revision of IEEE Std 1800-2009), 2013.
[2] “Universal Verification Methodology (UVM) 1.2 Users Guide,” Accelleras Systems Initiative, 2015.
[3] N. Kitchen and A. Kuehlmann, “Stimulus generation for constrained random simulation,” in 2007 IEEE/ACM International Conference on Computer-Aided Design, Nov 2007, pp. 258–265.
[4] H.-H. Yeh and C. Y. Huang, “Automatic Constraint Generation for guided random simulation,” in 2010 15th Asia and South Pacific Design Automation Conference (ASP-DAC), Jan 2010, pp. 613–618.
[5] A.-C. Cheng, C.-C. J. Yen, C. G. Val, S. Bayless, A. J. Hu, I. H.-R. Jiang, and J.-Y. Jou, “Efficient Coverage-Driven Stimulus Generation Using Simultaneous SAT Solving, with Application to SystemVerilog,” ACM Trans. Des. Autom. Electron. Syst., vol. 20, no. 1, pp. 7:1–7:23, Nov. 2014. [Online]. Available: http://doi.acm.org/10.1145/26531400
[6] Y. Guo, W. Qu, T. Li, and S. Li, “Coverage Driven Test Generation Framework for RTL Functional Verification,” in 2007 10th IEEE International Conference on Computer-Aided Design and Computer Graphics, Oct 2007, pp. 321–326.
[7] I. Wagner, V. Bertacco, and T. Austin, “Microprocessor Verification via Feedback-Adjusted Markov Models,” IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 26, no. 6, pp. 1126–1138, June 2007.
[8] O. Goloubeva, M. S. Reorda, and M. Violante, “Automatic generation of validation stimuli for application-specific processors,” in Proceedings Design, Automation and Test in Europe Conference and Exhibition, vol. 1, Feb 2004, pp. 188–193 Vol.1.
[9] M. Simkova and Z. Kotasek, “Automation and Optimization of Coverage-driven Verification,” in 2015 Euro incurco Conference on Digital System Design, Aug 2015, pp. 87–94.
[10] J. Hopfield, “Neural Networks and Physical Systems with Emergent Collective Computational Abilities,” Proc. Nat. Acad. Sci., vol. 79, pp. 2554–2558, 1982.
[11] M. G. Lagoudakis, “Neural Networks and Optimization Problems - A Case Study: The Minimum Cost Spare Allocation Problem,” 1997.
[12] Codasip Ltd. (2017) Codasip Cobalt Processor Specification. [Online]. Available: https://www.codasip.com/codix-cores/#cobalt