# Masters Thesis Proposal

## Reduction of Redundancy in Directed Random Verification through Checkpointing

Jesse Craig
Department of Computer Science
University of Vermont, Burlington, VT
**jcraig@cs.uvm.edu**

*Abstract*
*The growing complexity of function and gate counts found in modern semiconductor IP designs has made the verification of those designs the dominant cost of their development and implementation. Design teams rarely have the resources to exhaustively verify their designs, leading to a growing number of verification escapes. This thesis will study an optimization for one style of verification, directed random verification, which enables more of a design to be verified using the same amount of simulation resources. The method of optimization seeks to reduce the inherent redundancy found in directed random verification environments by checkpointing user selected, randomly controlled, decision points. Several verification scenarios will be created to explore the optimizations method's effects on both the efficiency of the verification that can be achieved and the challenges with implementing the method.*

**Keywords**
Hardware Simulation, Hardware Verification, Directed Random Verification, Constrained Random Verification, Hardware Verification Languages, System-on-a-chip Verification, Simulation Checkpointing

## 1 Overview

Directed random verification is a popular method of verifying complex digital IP designs. The method relies on the use of one testcase that is composed of many random elements. The random nature of the testcase allows one testcase to test many different parts of a design in many random combinations without requiring a verification engineer to explicitly enumerate each combination. As shown in papers [2] and [7], this method decreases the amount of verification engineering resource needed to verify a design by shifting that work to an automated system. An example testcase that uses this method might contain three iterations of a loop that executes at random one of the functions {A, B, or C} of the design being tested. Each time this testcase is executed, it will test one of the twenty-seven possible sequences of A, B, and C. If the goal of the verification engineer is to exercise all twenty-seven combinations on the design the verification engineer needs only to execute the one testcase repeatedly until all twenty-seven combinations have been tried. Instead of the verification engineer needing to write twenty-seven separate testcases they only write the one testcase and rely on the computer to work until all twenty-seven combinations have been executed at least once. Unfortunately, the random nature of the verification environment often means that significantly more than twenty-seven executions of the testcase will be needed before all twenty-seven combinations have been executed at least once. This is one of the forms of redundant execution that this thesis work seeks to reduce in order to decrease the amount of execution time needed to achieve all the verification engineer's goals.

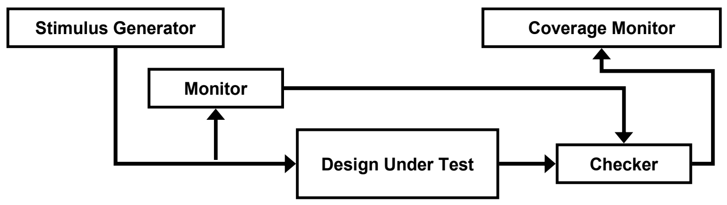

**Figure 1: Basic Directed Random Verification Environment**

Papers [1] and [2] describe a directed random verification environment as generally being composed of five basic elements, a stimulus generator, coverage monitor, design under test, and one or more pairs of monitors and checkers, as illustrated in Figure 1. The stimulus generator is used to exercise the design being verified, also known as the design under test. It controls what actions the design under test will perform and indirectly controls which coverage goals will be achieved. It is the stimulus generator that contains the random elements discussed above. The monitor records the stimulus given to the design under test, converts it into a useful form, and provides it to the checker. The checker acts as a golden verification model. It calculates an expected result using the information provided by the monitor and compares that to the actual result exhibited by the design under test. Any differences between the expected and actual result indicate a potential error in the design under test. The checker also identifies when coverage events have occurred and reports these to the coverage monitor. These events are collected by the coverage monitor and compared against the coverage goals defined by the verification engineer. The coverage monitor is used to track the progress of the verification environment.

To fully understand the inherent redundancy found in directed random verification environments a trivial stimulus generator is shown in Figure 2. This stimulus generator is composed of several 'if' statements that are controlled by random conditionals, and a set of actions {A, B, C, D, E, F, G, H, I, J, K, L, M, and N} that correspond to functions for the design under test to perform. A decision diagram is also shown, describing the complete set of legal executions the stimulus generator can perform. The execution path of the stimulus generator is randomly chosen from the set {A→*E, A→*L, A→*J, A→*M, and A→*N}. The coverage goals chosen for the example verification environment are the execution of the set of actions {E, L, J, M, and N}.

```
stimulus_generator
begin
A;
if( random_boolean() == true ){
    B; C; D; E;
}else{
    F;
    if( random_boolean() == true ){
        G; I;
        if( random_boolean() == true ){
            L;
        }else{
            M;
        }
    }else{
        H;
        if( random_boolean() == true ){
            J;
        }else{
            K; N;
        }
    }
}
end
```

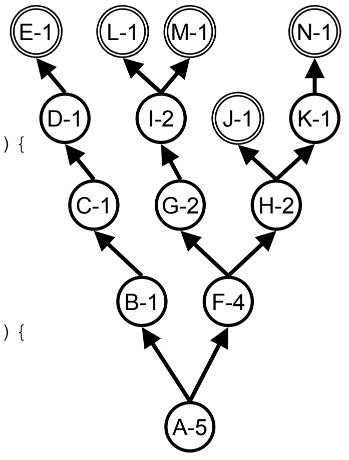

**Figure 2: Example Stimulus Generator and Associated Decision Diagram**

In order for all the coverage goals to be achieved the stimulus generator must be executed at least five times, but probably more. Assuming that the `random_boolean` function is truly random, the statistical odds of executing the path A→*N are 12.5% while the odds of executing the path A→*E are 50%. It can be expected that many executions of the path A→*E will take place before a single execution of the path A→*N ever occurs. Even if the stimulus generator is only executed the minimum five times, each time taking a different execution path, the node labeled A will still be redundantly executed five times and the node F four times. While this example is far too trivial to capture the true quantity of redundancy in directed random verification environments, it does illustrate how that redundancy is created and the general magnitude of the problem.

The aim of this thesis work is to implement a method of reducing this redundancy and examine the method's benefit into the goal of achieving all the coverage goals set forth by the verification engineer in the least amount of time. The method works by allowing the execution of the verification environment to be frozen at pre-determined, randomly controlled, decision points and replicated with each replica executing a unique possible outcome of that decision point. The technique of freezing and restarting the execution of verification environments is referred to as checkpointing. In Figure 2, the four 'if' statements controlled by random conditionals would be examples of randomly controlled decision points. In this stimulus generator, execution could be checkpointed directly after the execution of action F and two replicas of the verification environment could be spawned–one forced to execute the path F→G and the other F→H. This reduces the minimum number of times action F must be executed from four to two, and increases the statistical odds of executing the path A→*N from 12.5% to 25%. This potential reduction in redundant execution allows the coverage goals to be achieved in less time and adds only the overhead needed to checkpoint and restart the verification environment. To understand whether the reduction of redundancy found in this trivial example is indicative of what is to be gained in real-world verification environments; experimentation will be performed comparing the reduction of redundancy method to current methods using directed random verification environments based on commercial designs. Using these created verification environments the true benefit of the method can be better understood. Work will also be done to discover how this method can best be applied during the creation of directed random verification environments.

## 2 Related Work

The concept of using checkpointing during verification is neither new nor novel. In paper [5] it is shown that checkpointing can be used to recover long running verification tests after a system failure. Papers [3] and [4] use checkpointing in combination with specialized hardware-based simulation accelerators to migrate a verification environment between the hardware portion of the simulator and the software side. Hardware accelerated simulation benefits from much higher performance than software simulation but is difficult to use when debugging the cause of a failure. Using periodic checkpointing, a hardware/software co-simulation environment can detect bugs using hardware acceleration and then revert to the last saved checkpoint to re-execute the simulation in software for debugging purposes. While none of these ideas are well suited to solving the problem of the redundancy found in directed random verification environments they do show that the basic technology of checkpointing and restarting simulation environments is feasible.

Other work has been done to increase the performance of directed random verification that does not rely on checkpointing. Because each possible execution path in a directed random verification environment has a statistical chance of being executed it is possible for certain coverage goals to be achievable yet never achieved in a reasonable amount of time. Paper [2] shows that verification engineers often must create specialized directed testcases to fulfill coverage goals not achieved by directed random verification. This increases the amount of verification effort needed from the verification engineer and effectively reduces the efficiencies gained by using directed random verification. In paper [6], work has been done on a system that combines random based verification with formal verification techniques. An undirected random verification environment is used to drive simulation of a design under test. During this simulation if any portion of the design is not exhaustively explored in a given time a formal verification operation is used to discover the missing stimulus. This method overcomes the state-space explosion problem inherent in formal verification by only exploring the state-space to a limited state depth and relying on the random verification environment to cover the majority of the state space.

## 3 Method of Redundancy Reduction

The redundancy inherent in directed random verification is directly related to the randomly controlled decision points found in the stimulus generator. The simplest form of these decisions points is a randomly controlled binary decision such as that shown in the ‘if’ statement below. This decision point has two random outcomes, the execution of statement α or β.

```
if ( random_boolean() == true ) {
      α;
}else{
      β;
}
```

In order for a traditional directed random verification environment to ensure that both statements α and β are executed, the system must execute the ‘if’ statement until both outcomes occur. This execution unfortunately includes all the statements leading up to the ‘if’ statement. Even in an ideal system that ensures statements α and β are both executed after only two executions of the ‘if’ statement, the system still redundantly executes all the statements prior to the ‘if’ statement. By checkpointing the verification environment at the ‘if’ statement and replicating it on two separate simulators, one executing statement α and the other β, this redundancy is avoided.

Ideally, every randomly controlled decision point in the stimulus generator would be checkpointed and its possible outcomes exhaustively explored on separate replicas of the verification environments. This would effectively remove all the explicitly redundant execution from the verification environment. Unfortunately, the number of checkpoints created and outcomes to be explored grows exponentially as execution progresses. This exponential growth quickly exceeds the processing and storage resources available. To take advantage of the method, a verification engineer must select specific decision points to checkpoint based on the engineer’s expert knowledge of the verification environment, the coverage goals, and the design under test. The example code shown below illustrates this point.

```
execute_binary_operation(
      random_opcode(),
      random_operand(),
      random_operand());
```

The statement, `execute_binary_operation`, is designed to test the various operations provided by an arithmetic logic unit being verified. The statement is composed of three decision points controlled by the random values returned from the `random_opcode` and `random_operand` statements. If the verification engineer chose to checkpoint and exhaustively explore the results of either of the decision points controlled by the `random_operand` statement, the verification environment would spend a significant amount of time testing only one of the many operations that the arithmetic logic unit provided. In contrast, if the verification engineer chose the decision point controlled by the `random_opcode` statement, the verification environment would explore each operation equally. The verification engineer must also take into account the processing time necessary to checkpoint and restart the verification environment. If too little execution of the verification environment takes place between checkpoints, the savings gained by the reduction of redundancy will be lost to the overhead needed for checkpointing.

Implementing the reduction of redundancy method in a stimulus generator will be done by making a modification to the basic `random` function found in most high-level verification languages. Each randomly controlled decision point that the verification engineer wishes to checkpoint will be changed from using the traditional `random` function to the new, `fork_random` function. This new random function combines the checkpointing and outcome selection portions of the reduction of redundancy method. The function first checkpoints the verification environment then, when the verification environment is replicated and restarted, the function returns a value to the caller based on which outcome of the decision point that replica has been selected to explore. For the simple binary decision point discussed above, the new `fork_random_boolean` function would first checkpoint the verification environment then create two replicas. When the two replicas are restarted, the `fork_random_boolean` function would return α to the caller in one of the replicas and β to the caller in the other. Another advantage of implementing the redundancy reduction method with the `fork_random` function is that a verification engineer can easily integrate the method into already existing verification environments.

## 4 Experimentation

In order to measure the effectiveness of the reduction of redundancy method, the method will be applied to several directed random verification environments based on commercial designs. To facilitate this experimentation a working verification system will be created implementing the redundancy reduction technique. This system will be used to measure the amount of CPU time needed to reach the coverage goals of the verification environments, both with and without the redundancy reduction technique.

### 4.1 Reduction of Redundancy Verification Environment

The Reduction of Redundancy verification environment will be composed of a single Checkpoint Manager controlling one or more Simulators. The Checkpoint Manager will also utilize a Checkpoint Queue to store and sort checkpoints for execution. This verification environment will be used to measure the effectiveness of the reduction of redundancy technique at optimizing the verification of various IP designs.

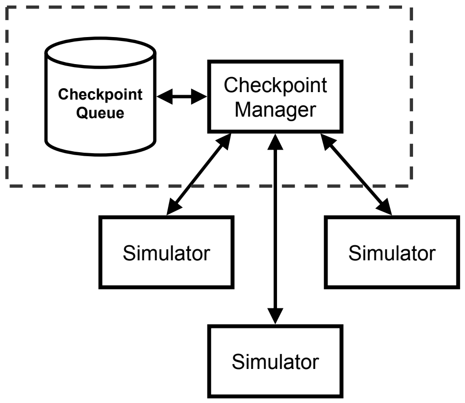

**Figure 3: Reduction of Redundancy Verification Environment**

#### *4.1.1 Simulator*

The Simulator will manage the behavioral level simulation of a directed random verification environment. The simulation is behavioral in the sense that it simulates the logic at a high level of abstraction (like that made possible by modern hardware design languages) as opposed to a low level of abstraction (where the logic design has been implemented using discrete logic gates). The Simulator takes as its input a directed random verification environment: a stimulus generator, a coverage monitor, one or more monitors and checkers, and the design under test, all described at the behavioral level. This directed random verification environment must also provide support for checkpointing and the ability to communicate with the Checkpoint Manager. To enable comparisons to be made between the reduction of redundancy method and the traditional directed random verification method, the Simulator will be able to transition between both modes of execution without any changes being made to the stimulus generators of the directed random verification environments.

### *4.1.2 Checkpoint Manager / Checkpoint Queue*

The Checkpoint Manager acts as a central control agent for a farm composed of one or more Simulators. The manager controls which checkpoint each Simulator executes and which outcome of the associated randomly controlled decision point the simulation will explore. The manager also acts as a central data collection agent, aggregating the performance data from the simulations with the coverage data from the coverage monitors. The Checkpoint Queue is used by the manager to store and prioritize checkpoints waiting for Simulators. This prioritization can be a simple first-in-first-out scheme or a more complex scheme based on specific goals chosen by the verification engineer. As each Simulator performs a checkpoint operation, that checkpoint is given to the manager and stored in the queue along with the outcomes to explore from the randomly controlled decisions point. The now free Simulator is then given a new checkpoint from the queue with its associated outcome and asked to continue the checkpointed simulation.

## 4.2 Directed Random Verification Environments

Using the reduction of redundancy verification environment described in section 4.1, two directed random verification environments will be created to measure the effectiveness of the redundancy reduction method. The environments will be representative of two broad categories of modern IP design: bus protocol IP and microprocessor IP.

### *4.2.1 Bus Protocol Model*

The first verification environment will model a simple logic block bridging two buses. This type of IP is commonly found in system-on-a-chip applications bridging between two on-chip buses or between an on-chip bus and an off-chip communication link. Example IP includes USB host controllers, PCI-X links, and ARM’s APB to AHB Bridge. In this directed random verification environment a stimulus generator controls two bus transactors. These transactors send data packets back-and-forth across the bridge core through both the local and on-chip busses. The communication initiated by each transactor is monitored by the appropriate bus monitor before it enters the bridge core and checked by the corresponding checker when it leaves on the other bus. The monitors and checkers also ensure that bus control packets, such as ACK and NAK packets, are correctly communicated and that protocol rules involving timeouts and corrupt packet detection are obeyed by the bridge core. The checkers will also identify when pre-determined coverage events have occurred and communicate this information to the coverage monitor. These coverage goals will include ensuring that a spectrum of data packets are successfully transmitted in both directions, that corrupt packets are successfully detected and the correct protocol is followed, and that the protocol rules governing timeout events are obeyed.

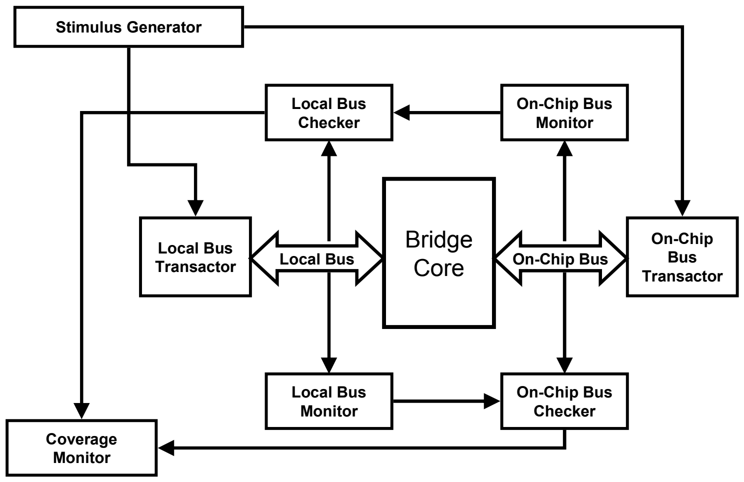

**Figure 4: Bus Protocol Model Verification Environment**

### *4.2.2 Microprocessor Model*

The verification environment based on the microprocessor model will attempt to verify the correct function of a processor for all the operations it provides. The processor will implement standard arithmetic and logic operations including addition, subtraction, shift, and several others. The processor will also implement a simple interrupt controller that detects hardware triggered interrupts and performs a context switch to execute a specified set of instructions. This type of processor is a simplified version of many embedded microprocessors used in consumer electronics and manufacturing control systems. In this environment the stimulus generator will control which instructions, and with what operands, the processor executes. The stimulus generator will also intersperse the triggering of interrupts amidst these instructions. The instruction monitor and data checker will work together to calculate the expected response from the processor and compare that with the actual response the processor exhibits. This verification environment's coverage goals will consist of executing each instruction with a diverse number of operands, executing combinations of instructions, and executing combinations of instructions while triggering interrupts.

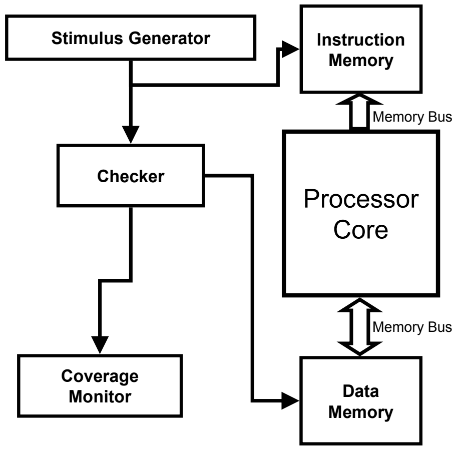

**Figure 5: Processor Model Verification Environment**

## 5 Expected Results

Under ideal circumstances a verification environment will achieve its coverage goals in linear execution time. This exceeds the performance of current directed random verification environments where each coverage goal is achieved in an ever increasing amount of execution time. This decaying performance over time is a result of the redundant execution exceeding the amount of new execution performed. Figure 6 shows the decaying performance of traditional methods and the expected performance that will be achieved using the reduction of redundancy method.

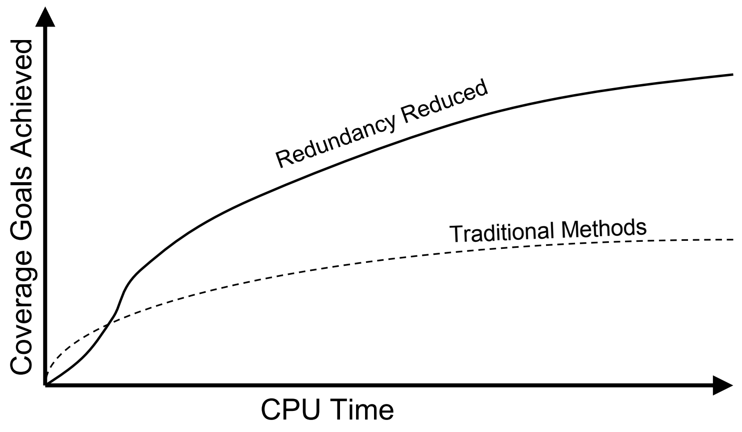

**Figure 6: Expected Results**

It is expected that by reducing the amount of redundant execution, a verification environment will be able to achieve its coverage goals faster. However, because the reduction of redundancy method does not completely remove the redundant execution from the system, the performance is still expected to decay over time. Initially the reduction of redundancy method is even expected to perform worse than traditional methods. Assuming the redundancy reduction environment uses a first-in-first-out Checkpoint queue, the decision diagram will be executed in a breadth-first manner. When this is combined with the fact that most coverage goals are found closer to the leaves of the decision diagram than the root, the redundancy reduction method can be expected to achieve fewer coverage goals early on. A verification environment using the reduction of redundancy methods also executes verification tests slower because of the overhead of storing and recovering checkpoints. While the impact of this checkpointing overhead is a function of how often the verification environment is checkpointed (and therefore controlled by the verification engineer) it is expected to have an effect on performance.

## References

[1] F. Haque, J. Michelson, K. Khan. *The Art of Verification with Vera*. Published by Verification Central, September 2001.

[2] *A Reference Verification Methodology for Vera.* The Synopsys Verification Avenue Technical Bulletin, vol. 4, issue 1, February 2004.

[3] Z. Yang, B. Min. *Si-Emulation: System Verification Using Simulation and Emulation*. Proceedings of the 2000 International Test Conference, October 2000.

[4] D. Beece, R. Damiano, G. Papp, R. Schoen. *The EVE Companion Simulator.* Proceedings of the 1990 European Design Automation Conference, March 1990.

[5] K. Shum. *Fault Tolerant Cluster Computing through Replication*. Proceedings of the 1997 International Conference on Parallel and Distributed Systems, 1997.

[6] A. Aziz, M. Ganai. *Enhancing Simulation with BDDs and ATPG.* Proceedings of the 1999 Design Automation Conference, June 1999.

[7] L. Zhongshu, Y. Xiaolang, W. Jiebing, X. Zhihan. *A Dynamic Random Instruction and Stimulus Generation for functions Verification of Embedded Processors.* Proceedings of the 5th International Conference on ASIC, October 2003.