# Using Computational Game Theory To Guide Verification and Security in Hardware Designs

Andrew M. Smith*†, Jackson R. Mayo‡, Vivian Kammler§, Robert C. Armstrong*, and Yevgeniy Vorobeychik¶
*Digital and Quantum Information Systems, Sandia National Laboratories, Livermore, California 94551–0969
Email: amsmit@sandia.gov
†Department of Computer Science, University of California, Davis, CA 95616–8562
‡Scalable Modeling and Analysis Systems, Sandia National Laboratories, Livermore, California 94551–0969
§Embedded Systems Analysis, Sandia National Laboratories, Albuquerque, NM 87185
¶Department of Computer Science, Vanderbilt University, Nashville, TN 37235

***Abstract*—Verifying that hardware design implementations adhere to specifications is a time intensive and sometimes intractable problem due to the massive size of the system's state space. Formal methods techniques can be used to prove certain tractable specification properties; however, they are expensive, and often require subject matter experts to develop and solve. Nonetheless, hardware verification is a critical process to ensure security and safety properties are met, and encapsulates problems associated with trust and reliability. For complex designs where coverage of the entire state space is unattainable, prioritizing regions most vulnerable to security or reliability threats would allow efficient allocation of valuable verification resources. Stackelberg security games model interactions between a defender, whose goal is to assign resources to protect a set of targets, and an attacker, who aims to inflict maximum damage on the targets after first observing the defender's strategy. In equilibrium, the defender has an optimal security deployment strategy, given the attacker's best response. We apply this Stackelberg security framework to synthesized hardware implementations using the design's network structure and logic to inform defender valuations and verification costs. The defender's strategy in equilibrium is thus interpreted as a prioritization of the allocation of verification resources in the presence of an adversary. We demonstrate this technique on several open-source synthesized hardware designs.**

## I. Introduction

Validation and verification are crucial processes in hardware design, and directly affect the confidence associated with a properly functioning, sufficiently reliable and secure system. Hardware validation involves ensuring design properties and specifications are accurate for the actual intent of the design; this process has few objective measures for accuracy or completeness, and relies heavily on effective communication between design teams. Verification of a hardware design typically involves using a mathematical model to prove that properties in a validated specification hold in the design's implementation. Since a complete and correct specification ideally includes properties of how a system should *and should not* behave in desired operating environments, a completely verified system is also secure and trusted, with respect to the design's intent[1]. Verifying that a design implementation complies with a validated specification has an exact, objective measure of completeness for each specification property: an exhaustive search of the system's state space, a nontrivial and extremely costly (sometimes intractable) process. Even with current state space pruning techniques used in formal verification tools, expertise in model checkers/theorem provers and specific design functionality is often required for guiding complete verification proofs, if possible, and results in up to $80\%$ of the design process [1]. This results in potentially using valuable resources exploring portions of the design that have little or no effect on critical elements of the system. The ability to prioritize particularly vulnerable regions of a design for further, more costly verification methods will direct efforts towards critical safety and security requirements and increase confidence in overall design fidelity

Game theory has been used for securing physical assets, such as power infrastructure, strategic resources, and airports. In such settings, Stackelberg game formulations are typically used, with the assumption that an attacker can observe security decisions put in place by the defender. These applications require deployment of costly physical resources, and strategies are often framed in the form of money allocation (for sensors, guard salaries, etc.). Verification as a security strategy in hardware design, however, is more time-sensitive due to strict deployment schedules. The cost of verifying a hardware design is a function of how complex (i.e., state space size of the system) and how critical (i.e., the number of safety and security properties must be verified) the design is. Furthermore, designers' valuation of a portion of the circuit ultimately reduces to the logic implementation and how likely failures are to spread to other, more critical portions of the circuit.

In this work, we develop a game-theoretic model for prioritizing components of a hardware netlist implementation to guide costly verification efforts. To achieve this, we make 3 main contributions:

1) we inform defender's utility via abstraction of hardware netlists into Boolean networks, using functional influence as a measure of expected failure propagation;
2) we extend Stackelberg security games (SSGs) to hardware implementations (hardware SSGs), interpreting defender equilibrium strategies as a prioritization of

[1]Concluding that a design is completely secure and trusted is a strong statement, but is prefaced with an equally strong, idealistic assumption: that all inputs into the verification process are flawless. We make this statement to draw ties between verification, security, and trust.

verification effort;
3) we apply our game model to synthesized open-source hardware designs popular in the trust community.

The rest of this paper is laid out as follows. In Section II, we discuss previous related research. Section III describes in detail our extension of Stackelberg security games to verification prioritization in hardware designs, including formulation of player utilities. Section IV applies our game theoretic model to open-source hardware design. Finally, Section V further discusses the implications of our model and future work.

## II. Previous Work

Security games have been used to guide protection strategies in physical and cyber security [2]. In both settings, Stackelberg games (also known as leader-follower or defender-attacker games) are commonly used to model the fact that the attacker can often observe realizations of a defense strategy over time before attacking. Existence of equilibria in Stackelberg games, and their equivalence with Nash equilibria (under certain reasonable constraints) have been well studied [3]. Interdependencies between defenders and their targets have also been considered in Stackelberg security games. However, these models often assume a defender only controls one target [4]. Synthesized hardware designs are complex logic networks, that are usually analyzed by one or many verification engineers. Game theoretic models of such large, complex systems often require computational methods to gain insight into equilibrium bounds. Vorobeychik, et al. developed a computational model for solving and estimating single defender security games on networks [5]. Smith, et al. extend this approach to approximating average-case equilibrium in general multi-defender, multi-target settings [6]. Lou, et al. prove formal equilibrium bounds of average-case equilibria (in certain settings) and provide a more in-depth computational analysis [7]. We specialize the framework for approximating average-case Stackelberg equilibria to a hardware design setting, and interpret defender strategies to be applicable to verification.

Few game theoretic models have been applied to hardware design settings. Games applied to trust in hardware design abstract away the design itself in lieu of understanding the interactions of categories of Trojans (attacker actions) and their corresponding detection methods (defender actions) [8], [9]. Security games in deployed FPGAs have also been studied, where an attacker attempts to gain access to a defender's bitstream for reverse engineering [10]. However, each player's utility depends only on an abstract notion of design value. To our knowledge, our model is the first to consider hardware design structure and functionality in a game theoretic setting.

Previous work in guiding formal verification of hardware circuits over the past couple decades has overcome several hurdles associated with design complexity. Relevant works include leveraging interdependencies between complex verification tasks, informative interaction between various proof methods, and guidance from automated test generation methods. Jones, et al. develop a verification framework for large scale designs, allowing (among other significant organizational tasks) decomposition of verification tasks and reusability of artifacts between proofs [11]. Bhadra, et al. provide an overview of hybrid techniques for guiding verification, combining the strength of mathematical provability of formal methods and the scalability of simulation, testing, and other more informal methods [12]. Recently, automated use of state space coverage from past simulations is used to guide simulation-based verification (sometimes referred to as *informal verification*) via machine learning techniques [13]. Such methods can also work as hybrid techniques for assisting formal methods. Our proposed prioritization metric would be of use upstream from any of the previously discussed methods, providing an efficient ordering of critical design components to cover.

## III. Model

For our problem, we are interested in finding regions in a complex hardware design (represented as a netlist) that are the most vulnerable to attack. Since potential targets in synthesized netlists represent logic gates, the primary property of interest is a gate's propensity to propagate a failure or attack to the output of the system itself, or to some predetermined high value signal. We apply the general Stackelberg security game (SSG) model on interdependent targets of Lou, et. al [6], [7] to scenarios specific to security in hardware designs. It is important to note that we define *security* to mean protection against any fault in the system (with respect to the design's specification), from inadvertent design flaws to malicious attacks.

### A. Defining the Game

In SSGs, there are typically 2 players: a defender, who aims to protect a set of resources, and an attacker that aims to corrupt a target that maximizes their expected utility. The players move sequentially, with the defender leading, and the attacker observing the defender's (possibly mixed) strategy before responding. Such games are representative of security scenarios in hardware development, since we can assume the attacker has the ability to run the same set of vulnerability analysis tools on a particular implementation as the defender.

For our model, there are 2 players: an attacker, $A$, and a defender (or hardware designer/verification engineer), $D$. The defender has a continuous action space $\mathbf{q} = \langle q_1, q_2, \cdots, q_{|T|} \rangle$, where $q_t \in [0, 1]$, for all sequential logic nodes in the netlist, $t \in T$, which represents the level of effort given to verifying node $t$. Each sequential node has a cost to verify, $c_t$, $\forall t \in T$, relative to the number of inputs, and an inherent value, $v_t$, $\forall t \in T$, that the defender loses if an attack is successful on node $t$. The defender's utility function, $u$, selects a strategy, $\mathbf{q}$, that minimizes total loss, considering both loss from attack and cost to verify (with respect to certain equilibrium constraints, given in Section III-D). The attacker's action space is defined as $p \in \{1, \cdots, |T|\}$, representing a choice of one of the defender's targets to attack. The attacker's utility function, $v$, maximizes total loss for the defender. Note that the attacker can be modeled to represent other forms of attack (i.e.,

a random failure, avoiding detection, etc.), however, in the interest of prioritizing critical regions, we wish to find defender strategies that reflect protection against maximum damage.

In the rest of this section, we discuss (i) the computation of defender node values and defense costs, (ii) utility calculation for the defender and the attacker, and (iii) a mixed integer linear program (MILP) based equilibrium solution for solving SSGs in hardware design scenarios.

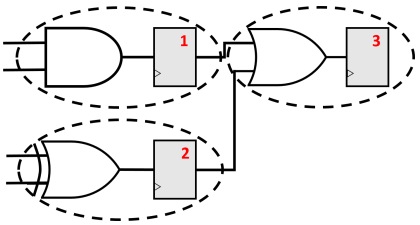
### B. Computing Defender Values

The defender's value of a target in a network combines the inherent, independent value of a target, as well as how a successful attack on that target propagates to other targets. To capture this in our hardware setting, we perform a static analysis of a Boolean network constructed from the netlist implementation of a hardware design to represent the defender's targets. To make this transformation, we select each output wire of each flip-flop to be a Boolean node. We then construct the sequential fan-in of each output wire to be the input to each node, terminating once another flip-flop is reached. Connections between nodes are maintained as edges, such that if a flip-flop output node $i$ is an input to a node $j$, there is a directed edge $(i, j)$. This process results in a Boolean network $G = (T, E)$, where $T$ is the set of nodes or targets, and $E$ is the set of directed edges between nodes. The Boolean function (transfer function) of each node is represented as a truth table defined by the combinational logic gates contained within each Boolean node, which can be obtained from the target technology's specification or through independent simulation. Figure 1 illustrates the conversion process.

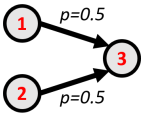
Computing defender values based on this Boolean network relies on two well-studied concepts, *influence*[2] and *independent cascade contagion*.

*Influence* — The probability that changing the value of bit $i$ of an input vector $y$ changes the output of the transfer function $f_j(y)$ for Boolean node $j$. Formally, $\text{Inf}_i(f_j) = \Pr[f_j(y) \neq f_j(y^{(i)})]$, where $y^{(i)}$ indicates an input vector $y$ with the $i$th bit flipped. Relatively low influence inputs are more likely to get overlooked in standard coverage tests, and therefore can serve as points of failure or malicious insertion [14], [15]. High influence inputs are of more interest in critical applications, however, due to their propensity to spread failure throughout the system. Influence can be calculated exactly by enumerating the truth table, or via harmonic analysis [16]. Influence calculation for a node scales exponentially with respect to the number of inputs. Sequential logic segments with particularly large fan-in, such as counters, can cause this calculation to become a bottleneck (or even intractable). Partitioning and parallel, independent computation of nodes, and estimation via truth table row sampling or Fourier coefficient set size reduction [17] can be used in these cases.

*Independent cascade contagion* — A process for determining contagion spread, typically used in disease spread or information propagation in social network contexts. More recently this has been used as a method for calculating failure spread in interdependent systems [5]–[7]. The independent cascade contagion process takes as input a graph $G = (T, E)$ with targets $T$, and a set of edges $E$, where $(i, i') \in E$ indicates that a successful attack on $i$ may affect $i'$ with some probability $p_{ii'}$. The process starts at some target $i \in T$ and spreads to each to each outgoing node $i'$ with probability $p_{ii'}$, which then spreads to $i'$'s neighbors, et cetera, affecting each node only once. With enough samples of this process, an expected affliction value can be obtained for each node given various starting points.




(a) A sample netlist schematic, partitioned into sequential regions.




(b) Boolean network of the sample netlist above, with corresponding cascade probabilities, $p$; nodes indexed by flip-flop IDs.

Fig. 1: Illustrates a simple transformation from a netlist to a Boolean network. Cascade probabilities, $p$, are determined by the influence (control value) of each input.

Once a Boolean network is created, we calculate the *influence* of each input on each node, assuming independently and identically distributed values (i.e., we assume each row in a truth table is equally likely). For each edge $(i, j)$ in the Boolean network, we set $p_{ij}$, the cascade probability, equal to $\text{Inf}_i(f_j)$, the input influence of the input corresponding to node $i$ on node $j$. Using inherent valuations of each node combined with cascade probabilities on each edge, an *independent cascade contagion* process is applied to the network to determine the valuations of each node as a result of potential failure propagation. Final defender valuations are ultimately determined by running this process $K$ times for each node, and is expected to converge quickly in tree-like network structures [5]. As a result of this iterative process, defender valuations now inherently include network structure and functional dependence. We will now describe how the defender's (and attacker's) utility is computed using these target valuations.

### C. Player Utilities

The defender's utility at a particular target $t$ depends on three variables: (i) $q_t$, the level of verification effort applied to $t$, (ii) $v_t$, the initial valuation of $t$, and (iii) $\rho_{t'}(t)$, for some $t' \in T$, the probability that a successful attack on $t$ spreads to

[2]The influence of an individual input on a transfer function is also known as control value (CV).

$t'$. We model the defender's utility function as a *loss* function, so that successfully attacked nodes result in negative utility. Formally, if a node $t$ is successfully attacked,

$$U_t = \mathbb{E}[\sum_{t' \in T} (-v_{t'})\rho_{t'}(t)].$$

While interesting utility functions can result from initially uniform valuations (as in the previous work in [5]–[7]), hardware design experts may have preliminary knowledge that, regardless of network structure, some nodes are more critical than others. For instance, nodes within a controlling state machine may be an expected target for attack or a known source of harmful design flaws. Such nodes may include output signals that are susceptible to data leakage (system security) or denial of service (system availability). If this information is available, higher prior values can be given to these targets.

Since we are aiming to prioritize the most critical components with respect to failure or attack, we wish to model an attacker that maximizes loss for the defender. As such, we assume the attacker's utility is $V_t = -U_t$, for a target node $t$.

### D. Computing Stackelberg Equilibrium in Hardware SSGs

Given the above SSG model, our solution method aims to find equilibrium strategies such that neither the defender nor the attacker wish to deviate from their strategy. This assumes the defender knows the attacker's utility function (to maximize damage), and the attacker observes the probability of complete verification at each of the defender's targets prior to selecting a node to attack. Such an equilibrium is known as the Stackelberg equilibrium. We adopt the average-case Stackelberg equilibrium (ASE) proposed in Smith, et al. [6]. ASE demonstrates the equilibrium properties stated above, and enforces that if the attacker is ever indifferent over which target node to attack, then one is selected uniformly at random. We suggest using ASE to include uncertainty over attacker actions for equally attractive targets, and the potential to include multiple (potentially noncooperative) hardware design teams in future work.

To compute ASE for hardware SSGs, we use a specialized version of the mixed-integer linear program (MILP) formulation developed in Smith, et al. [6]. While the original formulation includes multiple defenders and arbitrary security configurations, we model one centralized hardware designer/verification engineer and one security configuration (verification effort).

The objective function, shown in Equation 1, ultimately maximizes the defender's utility, considering the verification costs, $c$ for each node. Solutions of this maximization function must adhere to equilibrium constraints 2-8.

Constraints 4- 7 enforce that the defender's utility is being maximized with respect to the attacker's best strategy after observing verification efforts. Individually, constraint 4 forces the attacker to attack at least one node. Constraints 6 creates a variable, $s_t$, indicating the gap between the optimal attacker value $v$ and the value of attacking target node $t$. If $s_t = 0$, $t$ is an optimal node to attack. Constraints 7 together with constraints 5 ensure that the attacker chooses to attack a target $t$ if and only if it is the maximum utility target to attack given the defender's strategy $q$. Finally, constraint 8 computes the defender's utility, $u$, given defender strategy $\mathbf{q}$ and attacker strategy $\mathbf{a}$. Note that this is a nonlinear constraint, but can be linearized, resulting in a more computationally efficient mixed-integer linear program. Sorting the resulting $\mathbf{q}^*$ from this optimization from highest to lowest results in a prioritization of components for which we can allocate verification resources.

$$\max_{a,\mathbf{q},s,u,v} u - \sum_{t \in T} c_t q_t \tag{1}$$

s.t.

$$0 \leq q_t \leq 1 \qquad \forall\ t \in T \tag{2}$$

$$a_t \in \{0,1\} \qquad \forall\ t \in T \tag{3}$$

$$\sum_{t \in T} a_t \geq 1 \tag{4}$$

$$0 \leq v - (1 - q_t)V_t \leq (1 - a_t)M \qquad \forall\ t \in T \tag{5}$$

$$s_t = v - (1 - q_t)V_t \qquad \forall\ t \in T \tag{6}$$

$$a_t + Ms_t \geq 1 \qquad \forall\ t \in T \tag{7}$$

$$u = \frac{1}{\sum_{t \in T} a_t} \left( \sum_{t \in T} a_t (1 - q_t) U_t \right), \tag{8}$$

$$\tag{9}$$

where $M$ is some sufficiently large number, and $a_t = 1$ if node $t$ is attacked.

### E. Interpreting equilibrium strategies

We interpret the defender's strategy in equilibrium, $\mathbf{q}^*$, as the distribution of verification effort that should be given to each node in the Boolean network representation of the synthesized hardware design. A high $q_t^*$ value indicates a high ratio of criticality to cost of verification at node $t$. While the exact value of $q_t^*$ for each node does not translate into specific verification tasks, the relative values between targets suggest a prioritization of verification resources. In other words, if verification efforts are distributed as prescribed by $\mathbf{q}^*$, the *confidence* that target $t$ fully adheres to a validated specification (and is therefore trusted and secure) is proportional to $q_t^*$. If $q_t^* = 1$, this suggests that it would be beneficial for the defender to be absolutely confident that $t$ functions according to specification. This does not necessarily mean that targets with $q_t^* = 0$ should be neglected, however. Certain elements of safety and security properties present in requirements are not found in the individual logic gates of the netlist. Prioritization results can be better represented by assigning more insightful initial node valuations derived from the design specification (as discussed in Section III-C).

In our model, we assume the confidence ($q_t$) increases *linearly* with respect to the total cost paid to verify the target (i.e., total cost paid at target $t$ is $c_t * q_t$) . As covered in Mayo, et al., certain node inputs with relatively low complexity may result

in higher confidence than others, causing the cost-confidence relationship to scale relative to input complexity [18]. We leave this analysis for future applications where the cost-confidence relationship can be better estimated given the specific formal verification tools used in practice and designs of interest.

## IV. Empirical Analysis

For this case study, we apply our hardware SSG model to an open-source universal asynchronous receiver/transmitter (UART) design popular in the hardware trust community[3]. Typically, this design suite is used for Trojan detection [14], [15], however, it also serves as a good testbench for comparing the results of various synthesized implementations, especially since the reference design serves as an ideal target specification. For our analysis, we select inserted Trojans that could also represent unintentionally harmful design flaws (i.e., a portion of the logic that does not comply with the written specification). We chose the reference design (UART_REF), as well as versions that include three different Trojans (or "design flaws" in our case): (i) corruption of received data when a certain input is received (UART_T400), (ii) errantly enabling the "receiver ready" signal (UART_T800), and (iii) blocking of the "transmission signal" (UART_T900). Each design is synthesized to a Verilog netlist of FPGA primitives using identical processes and target technology, and converted to a Boolean network via the process described in Section III. To reemphasize, our empirical analysis should be used to prioritize verification efforts, not as a malicious behavior detection tool. Identification of known design flaws are presented here as validation that the rankings are reasonable.

We focus our analysis on varying the base cost of verification with respect to the number of inputs at each node, $C_{Base}$. $C_{Base}$ indirectly represents the sophistication of formal methods tools available for verification. A low base cost means it is inexpensive to formally verify a design, which may indicate advanced heuristics, the availability of large computing resources, or a large verification team, allowing more of the system to be verified. Higher base costs correspond to having less sophisticated tools or dedicated verification engineers available, resulting in the ability to verify only extremely critical regions. In other words, $C_{base}$ indicates the amount of total time necessary to verify some node per input. For a given node, $t$, with $K$ inputs, we set the cost of verification, $c_t = C_{Base} * K$. Figure 2 shows a comparison of the total percentage of nodes covered in each variation of the UART design, varying the base cost. A node $t$ is said to be "covered" (at least partially) by verification if $q_t^* > 0$. Here we see a logarithmic curve, where the gains of using hardware SSGs for prioritization decrease exponentially when $C_{Base} < 0.1$. Less than this value, verification is inexpensive enough to cover a fairly large portion of the circuit. In the next sections, we further compare the verification strategies of each UART implementation.

[3]Available at http://www.trust-hub.org

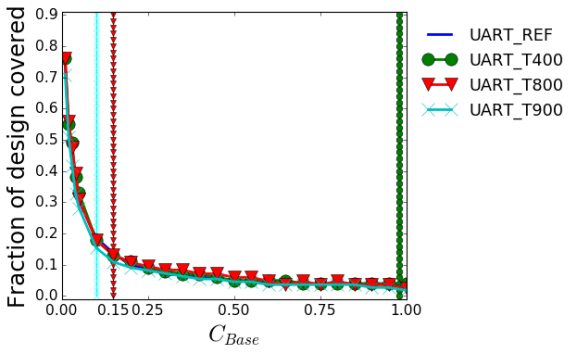



Fig. 2: Comparison of total verification coverage for several implementations of Trust-Hub UART, varying the cost per node input of verification. Vertical lines indicate maximum cost to cover the design flaw in the indicated implementation.

### A. Prioritization of verification strategies

Figure 3 displays the verification prioritization resulting from ASE in hardware SSGs run on the various UART implementations, sampling from $C_{Base}$ values. Perhaps not surprisingly, the prioritization signatures of each design is relatively similar (with few exceptions, particularly in the middle of each spectrum). While we know UART_REF to be the "correct" specification, the deviations in spectrum alone are not enough to tell us if formal verification of the prioritized strategies will reveal the design flaws that exist in the remaining 3 implementations. To investigate deeper, we will discuss the details of each individual implementation.

*1) UART_T400:* The design flaw present in the UART_T400 gives an incorrect data payload in the receiver portion of the UART. The specific register associated with the data affected is named "iRECEIVER/recDatH" in the synthesized netlist, and is indicated by a dashed red line in the prioritized strategy plot in Figure 3 (top, right). This particular register is incredibly crucial to the functionality of the UART, and appears in similar places in the verification ordering of the other implementations as well.

*2) UART_T800:* This implementation contains a complex injection of malicious logic that consisting of a 19-bit combinational trigger, resulting in the "receiver ready" (rec_readyH) signal going high against specification. While the trigger is relatively large, the payload is a primary output to the system, and encapsulated by the sequential nodes in the receiver's state machine. The sequential node representing the least significant bit in the receiver's state machine ("iRECEIVER/state[0]") is indicated in Figure 3 (bottom, left), and is the lowest priority of the other 3 bits contained in the state. While this node is further back in the priority ordering than the node of interest in the UART_T400 implementation, it is covered once the $C_{Base} \leq 0.15$. Referring to Figure 2, this remains in the top 12% of all nodes in the design.

*3) UART_T900:* UART_T900 differs from previous implementations in that the injected logic is a state machine not present in the specification. This state machine blocks

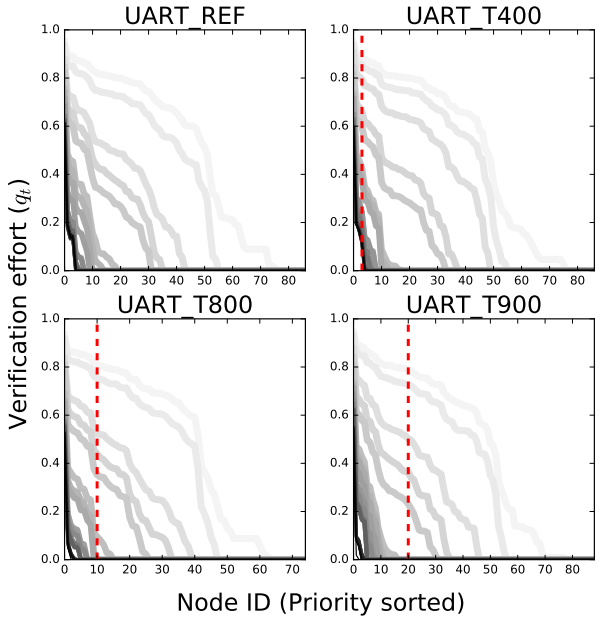



Fig. 3: Priority sorted strategy spectrum $q_t^*$ for each node $t$, varying $C_{Base}$ from $0.01 - 1.0$; darker lines indicate higher base costs. Nodes with low IDs represent high priority verification targets. Dashed red lines indicate the node ID with a known design flaw.

transmission when particular rare states are reached. Referring to Figure 3 (bottom, right), the dashed red line indicates the payload of the injected state machine (named "iXMIT/DataSend_ena", which corresponds to the trigger of the transmission denial of service). Though the reference design did not contain this logic, the effect of an attack on this implementation of the system causes "iXMIT/DataSend_ena" to become covered when $C_{Base} \leq 0.1$. Figure 2 shows that this node is in the top $20\%$ of the design's components.

## V. Discussion and Future Work

In this work, we apply a recently developed interdependent security game model to the guidance of hardware design verification, which we call hardware SSGs. We abstract hardware netlists into Boolean networks, as seen in previous hardware modeling work, and apply influence metrics from complexity theory to inform defender valuations of sequential portions of a design. Using strategies obtained from computing equilibrium in hardware SSGs, we interpret sorted defender strategies as a prioritization of verification efforts. We then apply hardware SSGs to variants of an open source UART design, showing that prioritized verification efforts are likely to capture design flaws known to be present in the specified implementations.

For future work, we are interested in applying this methodology to larger scale systems, where the benefits of prioritization are more likely to be leveraged. This involves using designer insight to assign proper valuations to nodes a priori, as well as the use of formal verification tools to better validate prioritization strategies and apply more realistic cost-to-confidence relationships. As designs scale up, independent design teams are responsible for verification tasks of subcomponents; to account for such scenarios, we would like to apply hardware SSGs in the multidefender setting.

## Acknowledgments

Sandia National Laboratories is a multi-mission laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energys National Nuclear Security Administration under contract DE-AC04-94AL85000.

## References

[1] R. Drechsler *et al.*, *Advanced formal verification*. Springer, 2004, vol. 122.

[2] A. Sinha, T. H. Nguyen, D. Kar, M. Brown, M. Tambe, and A. X. Jiang, "From physical security to cybersecurity," *Journal of Cybersecurity*, p. tyv007, 2015.

[3] D. Korzhyk, Z. Yin, C. Kiekintveld, V. Conitzer, and M. Tambe, "Stackelberg vs. nash in security games: An extended investigation of interchangeability, equivalence, and uniqueness." *J. Artif. Intell. Res.(JAIR)*, vol. 41, pp. 297–327, 2011.

[4] A. Laszka, M. Felegyhazi, and L. Buttyán, "A survey of interdependent security games," *CrySyS*, vol. 2, 2012.

[5] Y. Vorobeychik and J. Letchford, "Securing interdependent assets," *Autonomous Agents and Multi-Agent Systems*, vol. 29, no. 2, pp. 305–333, 2015.

[6] A. Smith, Y. Vorobeychik, and J. Letchford, "Multidefender security games on networks," *ACM SIGMETRICS Performance Evaluation Review*, vol. 41, no. 4, pp. 4–7, 2014.

[7] J. Lou, A. M. Smith, and Y. Vorobeychik, "Multidefender security games," *IEEE Intelligent Systems: Special Issue on Artificial Intelligence and Economics*, 2016.

[8] J. Graf, "Trust games: How game theory can guide the development of hardware trojan detection methods," in *Hardware Oriented Security and Trust (HOST), 2016 IEEE International Symposium on*. IEEE, 2016, pp. 91–96.

[9] C. A. Kamhoua, M. Rodriguez, and K. A. Kwiat, "Testing for hardware trojans: A game-theoretic approach," in *International Conference on Decision and Game Theory for Security*. Springer, 2014, pp. 360–369.

[10] J. Graf and P. Athanas, "How threats drive the development of secure reconfigurable devices," in *2008 IEEE National Aerospace and Electronics Conference*. IEEE, 2008, pp. 239–245.

[11] R. B. Jones, C.-J. H. Seger, M. D. Aagaard, and T. Melham, "Practical formal verification in microprocessor design," *IEEE design & test of computers*, vol. 18, no. 4, 2001.

[12] J. Bhadra, M. S. Abadir, L.-C. Wang, and S. Ray, "A survey of hybrid techniques for functional verification," *IEEE Design & Test of Computers*, vol. 24, no. 2, pp. 0112–122, 2007.

[13] C. Ioannides and K. I. Eder, "Coverage-directed test generation automated by machine learning–a review," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 17, no. 1, p. 7, 2012.

[14] A. Waksman, M. Suozzo, and S. Sethumadhavan, "FANCI: identification of stealthy malicious logic using boolean functional analysis," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. ACM, 2013, pp. 697–708.

[15] J. Zhang, F. Yuan, L. Wei, Y. Liu, and Q. Xu, "Veritrust: verification for hardware trust," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 7, pp. 1148–1161, 2015.

[16] C. Seshadhri, A. M. Smith, Y. Vorobeychik, J. R. Mayo, and R. C. Armstrong, "Characterizing short-term stability for boolean networks over any distribution of transfer functions," *Phys. Rev. E*, vol. 94, p. 012301, Jul 2016.

[17] N. Linial, Y. Mansour, and N. Nisan, "Constant depth circuits, fourier transform, and learnability," *Journal of the ACM (JACM)*, vol. 40, no. 3, pp. 607–620, 1993.

[18] J. Mayo and R. Armstrong, "Tradeoffs in targeted fuzzing of cyber systems by defenders and attackers," in *Proceedings of the Seventh Annual Workshop on Cyber Security and Information Intelligence Research*. ACM, 2011, p. 37.