

Data Security for Web-based CAD

Scott Hauck, Stephen Knol

Department of Electrical and Computer Engineering

Northwestern University

Evanston, IL 60208-3118 USA

{hauck, sknol}@ece.nwu.edu

Abstract

Internet-based computing has significant potential for improving most high-performance computing, including VLSI CAD. In this paper we consider the ramifications of the Internet on electronics design within two models for Web-based CAD. We investigate the security of these systems, and propose methods for protection against threats both from unrelated users, as well as from the CAD tools and tool developers themselves. These techniques provide methods for hiding unnecessary information, thus making reverse-engineering more difficult. Such techniques will be key to the development of future Internet-based CAD applications, since serious CAD users will be unwilling to use any CAD methodology that risks exposing their designs to outsiders. By enabling Web-based CAD, these techniques can improve CAD performance, enable collaborative design, and create a usage-based pricing methodology.

Content Indicators: B.7.2 Design Aids, Algorithms, Security, Web-based CAD, Data Security

1. Introduction

With the development of fabrication processes that can put tens of millions of transistors on a single chip, the design process has gotten so complex that human designers cannot create circuits on their own. VLSI CAD research efforts and software vendors have successfully created tools to handle many of these mapping tasks, making software an integral part of all complex circuit design efforts. However, while such tools can handle much of the design process, making VLSI designers more productive, these tools do not come without a price. Due to the complexity of the tasks undertaken by these VLSI CAD tools, the software can take a significant amount of time to compute. Runtimes of hours or days are common for many tasks, and may grow to weeks with future 100 Million transistor designs, slowing down the development of these circuits. For example, tools for using logic emulators to speed up the mapping process can take a day or more to operate [6], meaning that before simulation acceleration hardware can be used the designer must wait a significant amount of time. For tasks such as design space exploration, where multiple candidate circuit structures are synthesized in order to determine the best approach to adopt, multi-day runtimes are not surprising. These delays increase time-to-market, and reduce circuit quality by restricting the number of alternative implementations that can be considered.

It is possible to speed up CAD algorithms via high-performance computing systems. Parallel Supercomputers and Networks-of-Workstations can be brought to bear on most CAD problems, speeding up the algorithms. Custom

hardware such as logic emulators and simulation accelerators can speed up some algorithms, particularly circuit simulation. However, these hardware resources can be extremely expensive, making them unavailable to a large segment of the CAD user community.

A similar dynamic has occurred in general-purpose computing, leading to the creation of supercomputing centers. Such centers provide supercomputer access to a broad user community by hooking these machines to a network and allowing remote access. We believe a solution to the performance problem for VLSI CAD is similar: develop a Web-based CAD Compute Center, providing both hardware and software for high-performance computing, and make it accessible over the Internet via a World-Wide-Web interface. This will allow the cost of the high-performance computing resources (supercomputers, networks-of-workstations, and simulation accelerators/logic emulators) to be amortized over a large user base, and give even small VLSI design houses access to the computation resources that are becoming a requirement for VLSI design.

Beyond just providing affordable access to high-performance computing resources, a Web-based CAD Compute Center can have several benefits for VLSI design. By providing an Internet interface to a design project's CAD resources, it will enable user groups distributed geographically to use a single set of tools, and collaborate on a complex design project. Collaborative design is becoming much more important because the integration of complete systems onto a single chip requires a large variety of skills, skills that may not all be found at a single site.

The final benefit of a centralized CAD compute center is the ability to implement a per-use pricing policy for CAD software. Current CAD software is sold on a per-license (per-seat) basis, meaning that an organization that ever wishes to use N copies of a CAD tool simultaneously must pay for continuous use of those N tools. Thus, under the current pricing scheme there is no differentiation between a casual user that uses a given capability rarely, and thus is unwilling to pay a significant amount for a tool, and power users that constantly use that tool. Both software companies and users are promoting a usage-based pricing structure, where users are billed based upon the actual value (measured in time executing the algorithm) the software delivers to the user. By creating a centralized CAD compute center such a per-use pricing model can be implemented (and will be required in order to fairly allocate computer time to the center's users), something that is much more difficult to do in today's model of software distributed to each end-user's machine.

While such a CAD compute center has tremendous potential for supporting current and future VLSI design, there are several issues that must be addressed to make such a system a reality. In this paper we consider one of these problems, the development of security protocols for maintaining the privacy of user designs within an Internet-based computing paradigm.

2. Web-based CAD

There have been several research projects that have harnessed the World Wide Web for providing access to VLSI CAD tools [2, 3, 8 – 11, 13]. In general, there are two models for such systems: the Applet model, and the

Compute Center model. Under the Applet model the CAD algorithm is sent to the user's machine as Java Applets (or any other similar technology) and the algorithm executes on the user's machine. Thus, in this model the user must provide the compute resources for performing the CAD task, meaning that there is no way to amortize the cost of high-performance computing resources. Under the Compute Center model the design files are transferred to a centralized, shared compute center, where the CAD algorithms are applied to the design. Thus, the tools operate on the resources of the center.

As with anything that operates over the Internet, data security is an important concern for both Web-based CAD models. The design files for a VLSI chip represent a significant intellectual property asset, and most users will be unwilling to use any computing model that does not protect the secrecy of these files. Note that even within a company Intranet, where CAD resources are provided within a protected network accessible only to that company's computers, data security is still a significant concern since access to design files is usually restricted to only those employees working directly on that project.

The security challenges for a Web-based CAD system can be broken into three categories: *Client Security*, *Transmission Security*, and *Server Security*. *Client Security* encompasses all of the concerns of protecting data on the user's machine from attacks unrelated to the Web-based CAD system. *Transmission Security* deals with the need for ensuring that data transmitted between the user and a Compute Server is not intercepted and decrypted, and includes both information sent to and received from the server. Note that the Applet model does not include these concerns since the data never leaves the user's machine. Finally, *Server Security* represents the concern that the Web-based CAD tools and developers themselves may not be trusted, and thus steps must be taken to minimize the amount of information that can be obtained by the CAD tools.

Client and Transmission Security are faced by most Internet systems, and thus we can use standard solutions to these problems. In order to protect machines connected to the Internet techniques such as firewalls and other security protocols have been developed. Securing transmissions between the compute center and the user can be handled by public-key encryption, such as PGP. All communications from the user to the compute center will be encrypted by the center's public key (ensuring that only the compute center can decrypt the information), and communication back uses the user's public key. Thus, we can handle the Client Security and Transmission Security issues with standard solutions from the Internet community, and therefore they do not pose special problems for Web-based CAD.

Server Security is a problem somewhat unique to Web-based CAD (and, as we will see, the solutions are very application-specific). Most security methods for Web-based systems concentrate on keeping Java Applets and other Web-based tools from gaining access to data they are not intended to view. Thus, restrictions are placed on file read and write operations. However, the problem with Web-based CAD is that the user must reveal parts of their design to the applets or compute center, since the CAD tools need this information to perform their optimizations. However, the users do not want this information divulged to others, since the circuit designs are often very valuable intellectual property, and are kept as trade secrets. Thus, even though the user may give

information to the CAD tools in order to allow them to optimize the design, there must be some guarantees that the entire design is not then revealed to the CAD tool owners, who may not be trusted. For example, the owners of the compute server or CAD applets may be university researchers, and commercial VLSI design projects may be unwilling to use such a system without strong guarantees that the design data will be kept confidential (often requiring guarantees stronger than even nondisclosure or confidentiality agreements).

In the rest of this paper we present new methods for Server Security for Web-based CAD. As we will see, these methods primarily revolve around the idea that only the information necessary to perform a given optimization should be supplied to the CAD tool, while all other information should be hidden or removed. Note that similar techniques can be used for benchmarking, releasing only those features of a circuit necessary to test a given optimization, as well as the sale of cores and other circuit components, where designs sold to customers must be made as resistant to reverse-engineering as possible.

3. Data Security Techniques

The main challenge for Server Security for Web-based CAD is the fact that the CAD user must divulge information to the CAD tool for it to perform its optimization, yet the user needs assurances that this does not let others learn the details of their design. Although intellectual property agreements may help, relying on these agreements is too risky for many potential users. The solution to this dilemma is to make sure that only the information necessary for the tool to perform its optimization is given to the center. For example, comments in a file do nothing to help the CAD algorithms operate, and thus stripping the comments from all files before submitting them to the Web-based CAD algorithms helps limit the amount of information divulged.

While comment stripping is a universal information-hiding transformation for circuits, since this removes information unnecessary for any optimization algorithm, many other transformations are not universal. As we will see, some of the methods for Server Security eliminate information that is unused in some optimization steps, yet is required for others. Thus, as we consider different methods for Server Security, we must realize that not all techniques can be used before each CAD algorithm, and some methods will be only useful in specific circumstances. Also, some CAD optimizations, such as logic synthesis and logic simulation, require a great deal of information about the input circuit, and thus there are only limited possibilities for information hiding.

To apply the information-hiding transformations proposed in this paper it is necessary to develop programs to perform these operations before the information is sent to the Web-based CAD algorithm, as well as reintegration operations to replace the hidden information into the results from the Web-based CAD tools. Note that this program cannot itself be part of the Web-based CAD toolsuite, since it will have access to the raw design files, and can itself be viewed as a security vulnerability. The answer to this problem is to provide simple information hiding algorithms in source code, allowing users to verify that the algorithm does not contain any security vulnerabilities.

In the sections that follow, we will discuss numerous methods for providing Server Security for Web-based CAD.

3.1 Current Techniques

There are techniques that have already been developed specifically for protecting simulatable files, such as processor models and other cores. For example, Cadence provides an encryption method that allows encrypted files to be distributed to end-users with some hope that the design information cannot be recovered. However, this methodology protects the source files from direct observation (similar to Transmission Security), but does not protect the information from the Cadence CAD tools themselves. In fact, methods have already been demonstrated for coaxing the design information out of Cadence's Verilog-XL simulator [4].

LEDA's KRYPTON [12] code scrambler provides a more promising approach. The system hides information by renaming all variable names in a circuit to a hard to read series of "I", "1", "0", and "O" characters, removing formatting and comments, applying constant propagation, and other optimizations. Note that not only do these optimizations make it harder for a human to understand the source code, by eliminating the original variable names from the code this eliminates information from the circuit, since in (well-written) code the variable names carry information about the values they contain. This alteration to variable names should also be a universal transformation, since most or all CAD algorithms should not need this information.

3.2 Netlist Flattening

Hierarchical circuit descriptions, such as those found in hardware description languages, group together common subcircuits to condense the circuit description, and highlight commonalities in the circuit. By replacing calls to subcircuits by the logic itself, and removing hierarchy levels, this information can be removed from the circuit without changing the circuit functionality.

3.3 Functionality Removal

CAD optimizations such as partitioning, placement, and routing do not care about the exact functions computed in a circuit, but instead view the circuit netlist as a graph consisting of only connectivity information, and perhaps node size. Thus, while the CAD tool may take in circuit descriptions containing the exact logic functionality, it is immediately thrown away. When using such a tool we can eliminate the logic functions, or replace them with a default (i.e. all nodes become AND gates), before submitting the design. This removes most information from the data provided to the CAD tool, while still allowing the tool to perform its optimizations.

3.4 Net Direction Alteration

Even after the functionality in a circuit is removed, the direction of signal flow can give hints to circuit behavior, such as bus widths and other structures. In some circuit representations it is possible to simply remove the direction information, turning the circuit into an undirected graph. Other formats may require signal directionality, making removing this information difficult. For example, the XNF format (input to the Xilinx FPGA tools [14]) requires each gate to have a single output, and generally requires each net to have exactly one source. Although we cannot remove net directionality from such a circuit, we can destroy the value of this information by reversing the direction of some nets. Simply reversing the direction of a single net will not work,

since the original source will no longer generate any signals, and the original sink will generate two signals. The solution is to find a path in the circuit such that reversing all of the signals on the path maintains the format requirements. One possibility is to find a directed path from a circuit input to any output, and then reverse these signals. Since along this path every node will have been the source of one signal, and the sink of another, when the signals are reversed each node will still be the source of exactly one signal. Another method is to perform the same process for state bits instead of circuit I/Os, except that a directed path must be found starting from the state bit's output, and leading to an input of that stateholding element (a directed cycle containing the stateholding function).

```

ReverseIOs()
  Foreach circuit output O
    mark all nodes as untouched
    Decide, with 50% probability, to skip this node and continue loop;
    Otherwise, call sub_reverseIOs(output O, net NULL)
    If call returns TRUE
      make O an input
      store information in dictionary to reverse change
  End foreach

Sub_reverseIOs(node, net)
  If node is touched return FALSE
  Mark node as touched
  If node is circuit input
    Make node an output, with net providing signal
    Store information in dictionary to reverse change
    Return TRUE
  Else Foreach input signal I of node, in random order
    Foreach source S of signal I, in random order
      Call sub_reverseIOs(S, I)
      If call returns TRUE
        Make I an output of node
        Make net an input of node
        Store information in dictionary to reverse change
        Return TRUE
    End foreach
  End foreach
  Since no path found, return FALSE

```

Figure 1. Pseudocode for net direction alteration algorithm starting at circuit I/Os.

It is important to realize that removing or changing the net direction can only be done before certain optimizations. The main problem is that by removing net direction, or changing signal directions, critical path information can be lost. Thus, this technique can be applied before area-optimization algorithms for partitioning, placement, and routing, but may not be valid for delay optimizing tools. Slack assignment algorithms, where critical path information is added to the circuit before the main algorithm is run, may be tolerant of this data hiding technique as long as the slacks are computed before nets are reversed.

3.5 Net Removal

Perhaps the most easily identifiable parts of a circuit are the high-fanout signals, particularly clock and reset signals. Once these signals are found the registers are easily determined, giving an idea of the overall circuit structure. However, for some CAD algorithms these high-fanout nets are unnecessary, and in fact their removal may improve the speed of the CAD tool. For example, partitioning and placement algorithms can normally ignore very high fanout nets, since it is impossible to minimize their length. Routing algorithms for FPGAs can also ignore these signals, since they will be carried on special resources, and need not be considered during normal routing. Thus, high fanout nets can be removed before sending a design to such CAD tools, helping to hide the stateholding functions in the circuit.

```
ReverseFlipFlops()
  Foreach circuit flipflop F
    mark all nodes as untouched
    Decide, with 50% probability, to skip this node and continue loop;
    Otherwise
      Let net be the input of flipflop F
      Let node be an arbitrary source of net
      call sub_reverseFlipFlops(node node, net net, flipflop F)
  End foreach

Sub_reverseFlipFlops (node, net, originalFF)
  If node == originalFF
    swap FF's input and output
    Store information in dictionary to reverse change
    Return TRUE
  If node is touched return FALSE
  Mark node as touched
  Foreach input signal I of node, in random order
    Foreach source S of signal I, in random order
      Call sub_reverseFlipFlops (S, I)
      If call returns TRUE;
        Make I an output of node
        Make net an input of node
        Store information in dictionary to reverse change
        Return TRUE;
    End foreach
  End foreach
  Since no path found, return FALSE
```

Figure 2. Pseudocode for net direction alteration algorithm starting at circuit flipflops.

3.6 Circuit Partitioning

One method for speeding up CAD tools is to break down a large design problem into smaller pieces. For example, the placement problem for a large VLSI chip can be broken down to a series of subproblems, where each run of the placement tool places only a fraction of the overall chip area. This technique can also be effective for hiding the details of the design being optimized. Instead of giving a CAD tool the entire circuit to optimize, the circuit is broken down into subproblems and separately optimized. As long as there is a way to break the problem into subproblems without introducing significant inefficiencies this methodology can be applied. Note that it is critical

to ensure that the information needed to reconnect the pieces is hidden, so reintegration information is retained at the user site.

Not only can splitting up a problem hide information by making it hard to reassemble the circuit, it is possible to take this further by adding bogus data to the processing, or optimizing unrelated circuits at the same time. Specifically, even though the CAD tools may only see one part of the circuit at the time, they could collect all input files received within a given time period and try to merge them into a complete circuit. Optimizing two or more unrelated circuits at the same time will make it difficult to decide which elements belong to which circuit. However, details of the circuit domain may leave characteristic features on the circuit, making separation of unrelated subcircuits possible. Bogus data can instead be added to the data stream. Although this will slow down processing because the CAD tools must also optimize the bogus data, this overhead can be kept small. For example, if a circuit is split into 100 subproblems, and 10 bogus subproblems are also added, this means only a 10% overhead, yet significantly complicates reverse engineering. A similar technique is used in chip design, where bogus circuitry is added to otherwise unused portions of the chip, complicating reverse-engineering attempts. In order to generate this bogus data modified portions of the current design, or portions from other designs, can be used. Appropriate techniques for generating random circuits based upon real circuits have been presented elsewhere [7].

3.7 Component Simplification

High fanin nodes and high fanout signals can provide convenient points of reference, aiding reverse engineering. For example, if a repeated component is one of the few circuit structures using a 7-input logic gate, finding all 7-input gates in the circuit helps find all instances of that component. This can also be a landmark for connecting data from different steps in the mapping process. For example, a partitioning algorithm may be given the entire circuit, but with all information except connectivity removed. After partitioning a technology mapper would receive complete information on the logic in each of the subcomponents. With information from both steps, a user could look for characteristic node fanins and net fanouts to determine which inputs to the technology mapping algorithm correspond to which partitions created by the partitioner, allowing for reintegration of the subcomponents. Finally, the grouping of logic into high-fanin gates is the result of design decisions, containing information similar to the hierarchy information described earlier.

The answer is to break all components down to smaller elements, hiding this information. For logic functions this is often easy, since multi-input gates can be converted into trees of 2-input functions. Breaking up high fanout signals can be more difficult. However, there are several approaches. First, buffers can be inserted, splitting the fanout among multiple drivers. Also, the source logic for this net can be duplicated, splitting the fanout amongst the copies. Finally, some algorithms require that all multi-terminal nets be replaced with 2-terminal nets. For example, Spectral methods for logic partitioning often convert multi-terminal nets to two terminal nets via clique expansion or other methods [1]. This can be taken advantage of for Server Security for Web-based CAD by

performing the multi-terminal to two terminal net conversion on the user's machine with trusted software, eliminating this information from the input netlists.

```
SimplifyComponents()  
  Foreach logic node L  
    Let N be the number of inputs to L  
    If N > 2 and L's function ∈ {AND, NAND, OR, NOR, XOR, XNOR}  
      Create a complete binary tree of nodes with N inputs  
      Connect all inputs to L to the bottom of the tree instead  
      Remove L from the circuit  
      Set the function of the nodes in the tree based on L's function  
        AND: all nodes are ANDs  
        OR: all nodes are Ors  
        XOR: all nodes are XORs  
        NAND: all nodes but root are ANDs. Root is NAND  
        NOR: all nodes but root are Ors. Root is NOR  
        XNOR: all nodes but root are XOR. Root is XNOR  
    End foreach
```

Figure 3. Pseudocode for component simplification algorithm.

It might be possible to undo some of this simplification if the logic functions are retained. For example, all fanout-free logic cones of the same function, with perhaps an inversion on the final output, could be recombined. However, if functionality removal is also applied it becomes much less likely that this transformation can be reversed.

```

EncryptIOs()
  Create empty list resynthesize

  /* Process inputs */
  Create lists inputTargets and inputSources,
    both initially containing all circuit inputs
  While inputTargets not empty
    Randomly pick T from inputTargets, S from inputSources
    Create a circuit node with inputs T and S, output X, function XOR
    Add node to resynthesize
    Replace all other logic connections to T with a connection to X
    Add X to inputSources
    Remove T from inputTargets
    Remove S from inputTargets (if present)
  Endwhile

  /* Process outputs */
  Create lists outputTargets and outputSources,
    both initially containing all circuit outputs
  While outputTargets not empty
    Randomly pick T from outputTargets, S from outputSources
    Create circuit node with inputs S and TMP, output T, function XOR
    Add node to resynthesize
    Replace all other logic connections to T with a connection to TMP
    Remove T from outputTargets
    Remove S from outputTargets (if present)
  Endwhile

  /* Find region to resynthesize */
  Create empty list frontier
  Add all nodes adjacent to nodes in resynthesize into frontier
  While frontier not empty
    Remove arbitrary node F from frontier
    Foreach node N connected to F
      If N not in frontier nor resynthesize
        Decide, with 50% probability, to add N to frontier
      End Foreach
    Add F to resynthesize
  Endwhile

  /* Merge encryption circuitry into surrounding logic */
  Foreach maximal connected subcircuit S with nodes only in resynthesize
    Resynthesize S via any logic synthesis technique that will hide the
      XORs, such as 2-level logic minimization.
  End foreach

```

Figure 4. Pseudocode for I/O Encryption algorithm.

3.8 I/O Encryption & Splitting

For the specific case of logic simulation/emulation there are modifications that can be done to the circuit to improve Server Security. For simulation the circuit does not have to have exactly the same behavior as the real circuit, but the user must be able to know from the simulation how the real circuit will behave. For example, we can add an inverter to one of the outputs of the simulation, knowing that whatever the simulation does, the real circuit will produce the opposite value on that output. We can extend this transformation further, radically

changing the interface behavior. One simple method is to take two output signals and replace one of them with the XOR of the two values. Thus, if the circuit used to compute A and B, and now computes A and A XOR B, we can recover the value of B by computing the XOR of the two signals. Similar optimizations can be applied to the circuit inputs, and can be applied multiple times to further encrypt the I/O behavior. Even more complex I/O modifications can be applied, as long as there is a relatively simple method for recovering the circuit I/O behavior given the simulation behavior.

If we just add this extra logic as a wrapper around the original circuit, someone aware of this transformation should be able to recover the original circuit. However, these extra gates can be merged into the surrounding circuitry via logic synthesis techniques, intertwining the encryption circuitry with logic essential to the circuit's functioning.

One optimization to the I/O behavior that should be extremely hard for an attacker to undo is to split the circuit inputs into several separate signals. For example, if an input to the circuit fans out to three destinations, it can be replaced with three separate inputs, each going to only one destination. Thus, after this transformation the CAD tool no longer knows that those three destinations should always receive the same value, yet the user of the circuit can easily ensure that this always happens.

```
SplitIOs()  
  Foreach circuit input I  
    Foreach node N connected to I  
      Replace I's connection to N with a newly generated input  
    End foreach  
  End foreach
```

Figure 5. Pseudocode for I/O Splitting algorithm.

For all of these I/O transformations, the effects can be hidden from the user quite simply. The user specifies the inputs to the original circuit, and a simple software algorithm can generate the corresponding inputs to the transformed circuit. Once the CAD tools create output for the simulation, the results are fed through a similar transformation algorithm on the user's machine, which takes the simulation output and recreates the corresponding output from the original circuit. In this way, I/O encryption and splitting can be used to help improve Server Security, while the user will see no changes in the simulation behavior.

4. Data Security Technique Interactions

As discussed in the previous section, there are numerous ways to provide Server Security for Web-based CAD. These techniques alter the circuit information sent to the CAD tool, minimizing the amount of information that can be compromised if the CAD tools or Compute Server proves to be untrustworthy or malicious. Although these techniques cannot protect all data about a design, since the CAD tools require information about the design being optimized in order to operate, these techniques make it more difficult to reconstruct the complete circuit description, and minimize the vulnerability introduced by Web-based CAD.

As pointed out in the description of some of the security techniques, not all optimizations can be used before some CAD steps. For example, although comment removal and variable renaming can be applied before submitting the design to any CAD tool, since these tools should not depend on these features of the input netlist, I/O encryption and splitting can only be applied in a limited set of circumstances since it alters the behavior of the circuit being manipulated. A list of which security techniques can be applied during different mapping phases can be found in Table 1.

Security Technique	Mapping Step				
	Partitioning	Tech. Map	Place	Route	Simulate/Emulate
Variable Rename, Comment Remove	Yes	Yes	Yes	Yes	Yes
Netlist Flattening	Yes ¹	Yes	Yes	Yes	Yes ¹
Functionality Removal	Yes	No	Yes	Yes	No
Net Direction Alteration	Yes	Yes	Yes ²	Maybe ³	No
Net Removal	Yes	Yes	Yes	Yes ⁴	No
Circuit Partitioning	No ⁵	Yes ²	Yes ²	Yes ²	No
Component Simplification	Yes	Yes	Yes ²	Yes	Yes
I/O Encryption, Splitting	No ⁶	No ⁶	No ⁶	No ⁶	Yes

Table 1. Applicability of security techniques to different mapping steps. We focus on the steps needed to map to FPGA-based systems, our primary focus area. Notes: 1. Some optimizations, which may allow that CAD step to operate faster, can no longer be applied. 2. Some degradation in quality possible. 3. FPGAs with directional routing resources cannot be routed. 4. A separate routing step for the removed signals will need to be performed. 5. While it may be possible to pre-partition a design locally into regions, and then finish the partitioning elsewhere, much of the performance benefits of web-based CAD are lost. 6. If the circuit is destined for simulation only, this technique could be applied.

While some security techniques are simply inappropriate in some circumstances, use of others may imply a tradeoff between mapping quality or performance and data security. For example, net direction alteration changes the directionality of signals in a circuit. Such a transformation can be applied before most mincut based partitioners, since these algorithms do not use information on net directionality. However, such a partitioner might use logic replication [5], the process of duplicating logic nodes to reduce the communication between partitions. Such an optimization does require circuit directionality, since when replicating a logic node the output of that node is always removed from the cutset because the signal is computed in each partition simultaneously, while inputs to the logic node are forced into the cutset unless their sources are also replicated. This means that if a circuit undergoes net direction alteration before partitioning, any replication features in the partitioner must be turned off, adversely affecting the resulting quality.

Some security techniques may impact CAD runtimes instead of quality. For example, techniques such as I/O encryption alter the topology, reducing locality by connecting unrelated portions of the circuit. When this circuit is mapped to a logic emulator, the loss of locality may increase the required communications in the system, slowing

down the emulator's performance. Thus, when deciding which security transformations should be applied to a circuit, the resulting impact on the quality and runtimes of the CAD algorithms must also be considered.

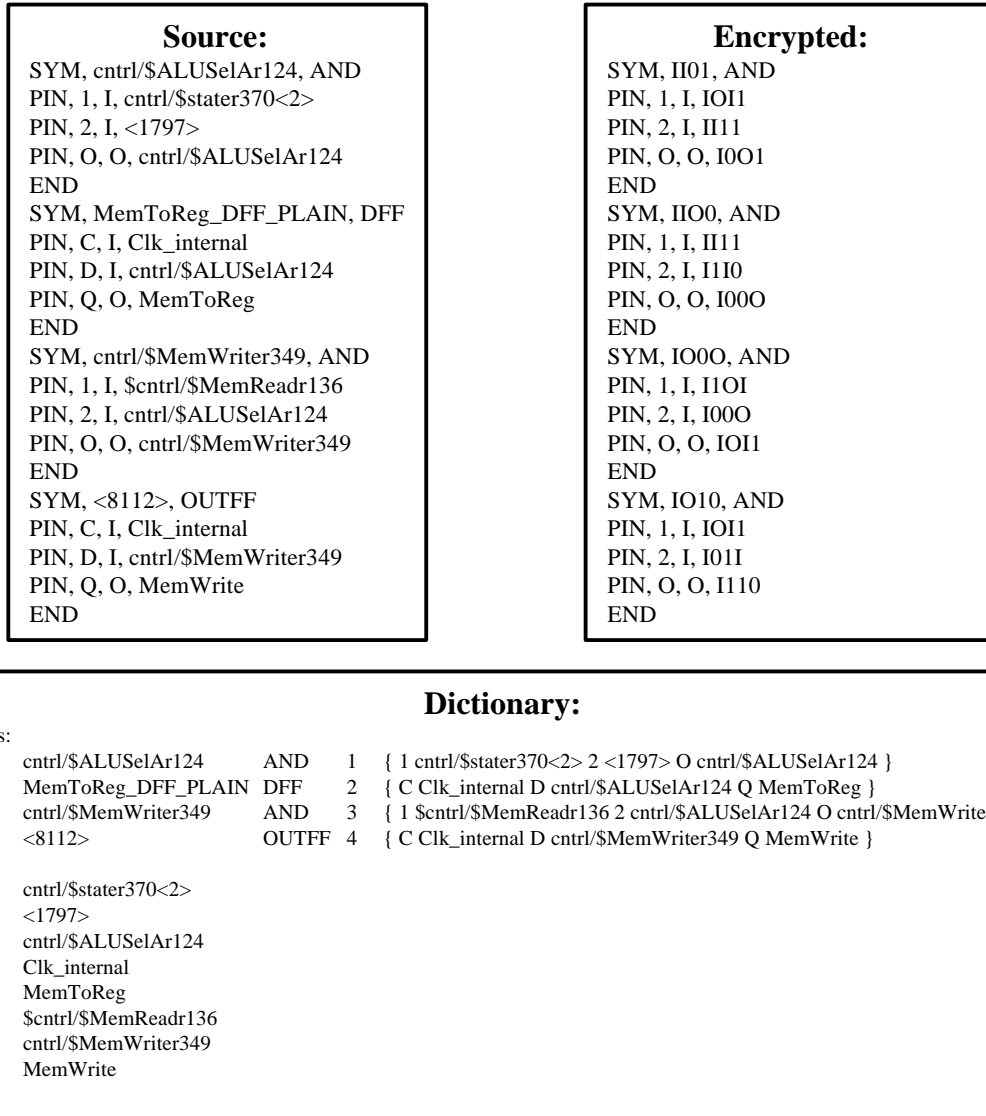


Figure 6. Encryption example, containing excerpts from the source, encrypted, and dictionary files. The source file is altered to create the encrypted version, which in this case retains only the connectivity information. A dictionary file is also created which allows the source to be recreated from the encrypted version. For nodes the dictionary contains the encrypted and original names, function, position in the file, and the position of input and output signals. For nets the dictionary contains only the encrypted and original names. The user retains the dictionary, while the encrypted version is sent to the CAD center.

5. Web-based CAD Data Security Testbed

In order to test the feasibility of our data security techniques we have developed a prototype encryption system. Our program begins with a circuit in XNF format, the format used by the Xilinx FPGA toolsuite. The algorithm

transforms the circuit into an encrypted form. It also creates a dictionary file, which contains the information needed to reverse the encryption steps (Figure 6). Via command line options various encryption techniques can be applied. Our system includes variable renaming, node order alteration, functionality removal, component simplification, and net direction alteration. The resulting file is a valid XNF file, which can be submitted to CAD algorithms that accept XNF.

Results for running our encryption algorithm on some standard benchmark circuits are shown in the tables. Table 2 reports the size of the files considered, along with the time taken to encrypt and decrypt these files. Table 3 reports the results of net direction alteration. Note that while we choose to randomly reverse half of the circuit outputs and flip flops, this tends to result in an overall reversal of about 20% of the elements in the circuit. However, this is more than sufficient to make the direction of the nets in the circuit useless for reverse engineering. Table 4 lists the results of component simplification, indicating the number of nodes simplified, as well as the node count in the resulting file.

Benchmark	# Nodes	Original Size	Encrypted Size	Dictionary Size	Encryption Time	Decryption Time
s38417	25,589	1,793,176	1,998,848	1,819,200	89.4	61.3
s38584	22,451	1,664,681	1,785,856	1,794,048	128.5	59.6
s35932	19,880	1,526,721	1,593,904	1,627,505	133.8	57.6
s15850	11,071	747,686	830,327	841,427	23.9	23.3
s13207	9,445	639,481	729,053	734,204	19.0	20.7
s9234	6,098	405,464	462,345	459,564	12.0	12.5
s5378	3,225	236,735	262,790	283,863	6.6	7.0
s1423	831	57,278	64,738	57,871	1.8	2.1
s208	129	8,691	10,188	9,490	0.4	0.5

Table 2. Results from protecting benchmarks with the prototype data security algorithm. Circuits had their signal names encrypted, their order in the file randomized, the functions removed, large fanout nodes broken into smaller subfunctions, and half of all outputs and flip-flops used as starting points for net reversal. File sizes are in bytes. Times are in seconds on a SPARCstation 5, and encryption time includes the time to create the dictionary files to allow for undoing the encryption. Note that, due to limitations in our current prototype, decryption times do not include the undoing of component simplification.

6. Conclusions

The Internet is revolutionizing most communication and computation systems. It also has the potential to radically change the way future electronic systems are designed by creating a new methodology for VLSI CAD. However, just like every other domain, security concerns with an Internet based computing paradigm may overwhelm the potential benefits.

In this paper we considered two models of Web-based CAD: Applets and Compute Centers. We demonstrated how the security concerns of these systems can be broken down to Client, Transmission, and Server security concerns.

Client security entails the securing of the user’s machine against external intrusion, and can be handled by firewall techniques currently in place. Transmission security encompasses concerns over transporting information to a Compute Center, and can be supported by public key encryption techniques. The remaining problem is Server security, the concern that the CAD tools themselves (or the creators of these tools) are not trustworthy, and thus may divulge information given to these tools.

Benchmark	Portion of components reversed		
	I/Os	Internal Nodes	Nets
s38417	33%	23%	23%
s38584	7%	21%	20%
s35932	19%	45%	45%
s15850	11%	24%	24%
s13207	10%	19%	19%
s9234	13%	13%	13%
s5378	34%	13%	14%
s1423	23%	28%	28%
s208	12%	25%	22%

Table 3. Percentage of circuit structures reversed during net direction alteration. Both I/Os and flip-flops were used as starting points for reversal.

Benchmark	Initial # Nodes	# Decomposed	Final # Nodes
s38417	25,589	997	26,729
s38584	22,451	1,373	24,506
s35932	19,880	0	19,880
s15850	11,071	285	11,496
s13207	9,445	393	10,086
s9234	6,098	219	6,445
s5378	3,225	367	3,654
s1423	831	12	848
s208	129	9	138

Table 4. Results of component simplification. “Initial # Nodes” is the number of nodes in the source file, and “# Decomposed” is the number of nodes that were simplified. “Final # Nodes” is the number of nodes in the encrypted file.

In this paper we proposed numerous new techniques for server security for Web-based CAD, as well as demonstrated how existing techniques can be harnessed for this new domain. Techniques such as variable renaming and formatting and comment removal eliminate the self-documenting code features of hardware description languages, while netlist flattening and component simplification removes grouping information. Further information hiding can be accomplished by understanding what information in a circuit is not needed by a given CAD tool. These techniques include functionality removal, net direction alteration, and net removal.

Circuit partitioning can also serve to hide information by only providing subportions of the circuit, and potentially providing bogus data. Finally, I/O encryption and splitting can alter the interface of circuits under simulation, complicating reverse-engineering attempts.

By combining these techniques together a complete security methodology for Web-based CAD can be provided. This will help create a Web-based CAD methodology that can provide significant benefits for future VLSI design efforts. Applet-based CAD can enable collaborative design between geographically distant designers and create a usage-based pricing methodology for CAD software. Compute Centers can provide these same benefits, as well as providing high-performance computation resources in an affordable manner by amortizing their costs across a large user base.

7. Acknowledgements

This research was supported by a grant from the National Science Foundation and software donations from Xilinx, Inc. We also wish to thank Mohammed Khalid from the University of Toronto for providing example XNF files used in this research.

8. References

- [1] C. J. Alpert, A. B. Kahng, "Recent Directions in Netlist Partitioning: A Survey", *Integration: the VLSI Journal*, Vol. 19, No. 1-2, pp. 1-81, 1995.
- [2] C. Barrett, D. Dill, J. Levitt, *SVC*, <http://agamemnon.stanford.edu/levitt/vc.html>, 1997.
- [3] D. Chakrabarti, P. Joisha, P. Banerjee, *The WADE Project*, <http://cpdcser.ece.nwu.edu:8080/WADE/wade.html>, 1997.
- [4] J. Cooley, "SPY vs. SPY: the VMC Story", *Integrated System Design*, Vol. 8, No. 90, pp. 56-64, December, 1996.
- [5] M. Enos, S. Hauck, M. Sarrafzadeh, "Replication for Logic Bipartitioning", *ICCAD*, pp. 342-349, 1997.
- [6] J. Gateley, "Logic Emulation Aids Design Process", *ASIC & EDA*, July, 1994.
- [7] M. Hutton, J. Rose, D. Corneil, "Generation of Synthetic Sequential Benchmark Circuits", *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 149-155, 1997.
- [8] D. Lidsky, J. M. Rabaey, "Early Power Exploration--a World Wide Web Application", *Design Automation Conference*, June 1996.
- [9] David Lidsky, *PowePlay*, <http://infopad.eecs.berkeley.edu/PowerPlay>, 1997.
- [10] P. Losleben, D. Boning, A. Bhandarkar, A. Gower, H. Huang, E. Jhong, J. Jovanovic, P.-F. Lam, M. McIlrath, W. Moyne, S. Narayan, L. Wang, S. Wang, T. Wu, E. Zhou, *National Research Enterprise*, <http://www-snf.stanford.edu/ComputationalPrototyping/comproto/nre.html>, 1997.
- [11] A. R. Newton, F. Chan, N. Ghazal, M. Horton, H. Hse, A. Hsia, S. Leung, K. Nguyen, M. Shilman, M. Spiller, S. Szollar, J. Young, J. Zhang, *Weld*, <http://www-cad.eecs.berkeley.edu/Respep/Research/weld>, 1997.
- [12] K. O'Brien, S. Maginot "Non-Reversible VHDL Source-Source Encryption", *EURODAC'94 / EURO-VHDL'94*, pp. 480-385, 1994.
- [13] K. Saraswat, J. P. McVittie, W. Abdel-Ati, S. Abdollahi-Alibeik, D. Bang, P. Canupp, J. Han, Z.-K. Hsiau, M. Joshi, P. Kapur, S. Ma, B. Shieh, *Speedie*, <http://speedie.stanford.edu>, 1997.
- [14] *The Programmable Logic Data Book*, San Jose, CA: Xilinx, Inc., 1994.