WATERMARKING FPGA BITSTREAM FOR IP PROTECTION

A Thesis
Presented to
The Academic Faculty

by

Pratik M. Marolia

In Partial Fulfillment
of the Requirements for the Degree
Master of Science in the
School of Electrical and Computer Engineering

Georgia Institute of Technology
August 2008

WATERMARKING FPGA BITSTREAM FOR IP PROTECTION

Approved by:

Dr. Hsien-Hsin S. Lee, Advisor
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Sung Kyu Lim
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Sudhakar Yalamanchili
School of Electrical and Computer Engineering
Georgia Institute of Technology

Date Approved: May 12, 2008

To my Parents...

ACKNOWLEDGEMENTS

I thank my parents for their support and motivation. They have backed my decisions and have always encouraged me to pursue my interests. I would also like to thank my sister for always being there for me. My family has played an important role in helping me achieve my goals.

I would also like to convey my hearty thanks to my advisor Dr. Hsien-Hsin Lee. He introduced me to a variety of different research subjects and lead me through this Thesis. Dr. Lee understood my interests and motivated me to realize my full potential. I also thank Dr. Sung Kyu Lim and Dr. Sudhakar Yalamanchili for agreeing to be on my thesis committee and providing feedback.

Lastly, I thank all my friends at Microprocessor Architecture Research Society (MARS) for providing the constructive environment. They were always there to help me, when I needed them.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

In this thesis, we address the problem of digital intellectual property (IP) protection for the field programmable gate array (FPGA) designs. Substantial time and effort is required to the design complex circuits; thus, it makes sense to re-use these designs. An IP developer can sell his design to the companies and collect royalty. However, he needs to protect his work from security breach and piracy.

The legal means of IP protection such as patents and license agreements are a deterrent to illegal IP circulation, but they are insufficient to detect an IP protection breach. Watermarking provides a means to identify the owner of a design. Firstly, we propose a watermarking technique that modifies the routing of an FPGA design to make it a function of the signature text. This watermarking technique is a type of constraint-based watermarking technique where we add a signature-based term to the routing cost function. Secondly, we need a method to verify the existence of the watermark in the design. To address this we propose a digital signature generation technique. This technique uses the switch state (ON/OFF) of certain switches on the routing to uniquely identify a design.

Our results show less than 10% speed overhead for a minimum channel width routing. Increasing the channel width by unit length, we could watermark the design with a zero speed overhead. The increase in the wire length is negative for majority of the circuits. Our watermarking technique can be integrated into the current routing algorithm since it does not require an additional step for embedding the watermark. The overall design effort for routing a watermarked design is equivalent to that of routing a non-watermarked design.

CHAPTER I

INTRODUCTION

The integrated circuit fabrication technology has improved substantially. It is now possible to put billions of transistors on a single integrated chip. However, the design and the testing methods are becoming increasingly complex which is driving up the design cycle time and the cost of the final product. As a result of this, the re-usable design paradigm has gained popularity. The company or organization can create and sell their design macros to generate a revenue.

The evolution in functionalities and the size of FPGA makes it possible to create highly complex designs using FPGA. The FPGAs offer an advantage of a lower cost and a faster time to market compared to an application specific integrated circuit (ASIC) design. The system designer is not required to be an expert in FPGA design or synthesis toolchain to utilize the re-usable FPGA design for building complex systems. Use of re-usable FPGA design cores have given rise to a new industry model that guarantees lower cost, faster design time alongwith flexibility associated with an FPGA. However, for the re-usable design industry to flourish, it is necessary to address the need for various levels of security to safeguard the designers' right to IP. In the absence of the security checks an adversary could legitimately sell a copy of an original design as his own creation without being detected.

Non-watermarked digital IP is synonymous with that of an artist's painting that is robbed by a conman before it was signed. The thief can sell the painting as his own creation and make profit. In the absence of the artist's signature on the painting there is no simple way of identifying the real painter. In the digital IP domain, the problem is aggravated due to the fact that digital IP's are replicated and embedded

into products that are readily available in open. The licensing agreements are used to establish trust between the two organizations, but there is no mechanism to detect the breach of trust. On one hand, we want to allow easy exchange of the designs and also want to protect it on the other hand. The IP authors needs to be assured about safety of their design from illegal re-distribution. This originates the need for intellectual property protection (IPP) for digital designs.

FPGA manufacturers have attempted to address the digital design security issue by encrypting the FPGA bitstream. The encryption key is stored on the FPGA in an SRAM array which is powered by a battery. The battery is used to retain the stored key when the FPGA is powered down. The bitstream is stored in encrypted format on the external ROM and decrypted every time the bitstream is loaded onto the FPGA. The need for an external battery to power up the SRAM cells that store the signature bits, can render the device useless when the battery dies. It might be possible to extract the encryption key from the FPGA by using some destructive means. Even if we accept that the encryption may prevent IP theft from assembled systems, however it is not sufficient to detect illegal re-distribution of design by a licensed customer. There still exists a need to provide an higher level of protection for digital IP security.

Watermarking technique provides a technique to detect the security breach. It inserts the author's signature in the design in such a way to inhibit any tampering of the unique mark. For a good watermark, the probability of finding a false positive should be very low. Additionally, the watermark should be transparent, not changing the design function and having minimum area and timing overhead. In [5], the authors surveyed the existing watermarking schemes and outlined the properties for a good watermark. The authors in [11] proposed to store the encoded signature bits in the unused bit positions of the FPGA LUTs. The idea of constraint-based watermarking technique for VLSI design was proposed by Kahng *et* al. [3] They have modeled

watermarking as an optimization problem. The solution of this problem over a set of chosen constraints would give a unique solution. Yet another timing-constraint-based watermarking technique [4] proposed to selectively re-route certain nets to satisfy the specified timing constraints. The timing constraints are correlated with the signature bits. In [12], the authors proposed to use the unique characteristics of a cluster based FPGA architecture to embed a mark into the FPGA bitstream. In [15], the authors proposed inserting watermark in the form of additional bends in the metal layer routing for VLSI designs. All the proposed techniques have an area or an timing overhead. Although it is a well accepted fact that it is not possible to provide security at zero overhead, our goal is to minimize the overheads without compromising the security.

This thesis proposes a watermarking method that modifies the routing of an FPGA design to bear some correspondence with the signature bitstream. Mapping an HDL design on an FPGA involves placing the logical blocks and then routing the interconnections between them. The non-critical nets in the design are routed through a new path that is a function of the signature bits. After modifying the routing, the design signature is generated with unique features of the watermarked design. The switch coordinates of certain switches that lie over the modified routes are captured in a reference file. This file is used during the signature verification process to reverse engineer the signature bits from the design. The reference file is protected by the author or a trusted organization. This watermarking technique has a low timing overhead since it does not penalize the nets with critical delay. It has negligible area overhead since the number of used LUT and design placement is not modified during the watermarking process. The signature extraction idea is unique from the previous methods because this is the first known attempt to use the switch topology of an FPGA to identify the design. The number of programmable switches on an FPGA is very large; hence, for any medium complexity design it is possible to generate a fairly

large digital signature.

In this work, we evaluate the strength and overhead of the watermarking technique using a prototype developed using the Versatile Place and Route (VPR) tool from the University of Toronto[16]. We evaluate the results for segmented FPGA architecture made up of single look-up table (LUT) logic blocks. We use a set of 20 MCNC benchmark circuits for our evaluations. The effect of watermark on circuit speed, channel width, and design area are presented.

The rest of this thesis is organized as follows. Chapter 2 describes the FPGA architecture and the design synthesis process. Chapter 3 discusses the VPR routing algorithm. Our watermarking and fingerprinting technique is described in Chapter 4. Chapter 5 describes the implementation details and Chapter 6 presents the results. We conclude in Chapter 7 with a brief summary of our contributions.

CHAPTER II

FIELD PROGRAMMABLE GATE ARRAY

Field programmable gate array (FPGA) is a re-programmable device. It enables flexible, reconfigurable digital circuit implementations at the hardware level. It is built up of a large number of programmable functional blocks and wire segments that can selectively route the connections between the logic blocks. Although one can implement an algorithm using general purpose microprocessors, FPGA can provide better performance for certain applications (e.g. realtime) with limited flexibility. It also requires special skill set for the design methodology and toolchain to deliver an effective design.

ASIC is another design option to satisfying the need of specific, repetitive algorithms. ASIC is a preferred solution to implement certain complex circuits with very strict time, power, and area budgets. The ASIC design process, beginning from the design specification to the semiconductor bring up takes fairly long time and also the non-recurring fabrication cost is high. The investment for an ASIC can only be justified for large volume requirements. FPGA based solutions are less flexible than general purpose microprocessors yet less rigid than ASIC's. They can address the needs of time critical applications where a software based solution is not acceptable, and the cost of an ASIC is prohibitive. Figure 1 shows a comparison between FPGA, ASIC, and processor with respect to performance and design time. FPGAs are also commonly used for prototype development, or proof of concept.

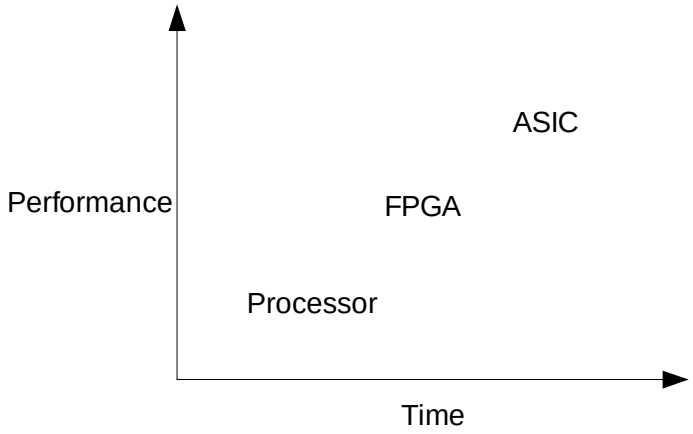


Figure 1: Comparison of different integrated chips (ICs)

2.1 FPGA Architecture

An island-type FPGA architecture is shown in Figure 2. It is made up of logic blocks arranged in a 2-dimensional array with wire segments running between them in horizontal as well as vertical directions. The connection block (C-block) is made up of programmable switches that can make connections between the logic block inputs and outputs, and the wire segments running adjacent to it. The switch block (S-block) provides interconnections between the horizontal and vertical wire segments. A group of wires running in a horizontal direction is called a horizontal channel; similarly, a group of wires running in vertical direction is a vertical channel. The horizontal and vertical channels are made up of segments of different lengths. A segment of length one spans across one logic block; whereas, a length two segment spans across two logic blocks. Long wires span across the entire length of the FPGA, and they are used to route clock signals or long nets.

2.1.1 Logic Block

The logic block as the name suggests, implements a logic function. It is implemented using a LUT, which is a multiplexer with the input lines connected to programmable SRAM cells, as shown in Figure 3a. An *n-input* LUT can implement any n- variable

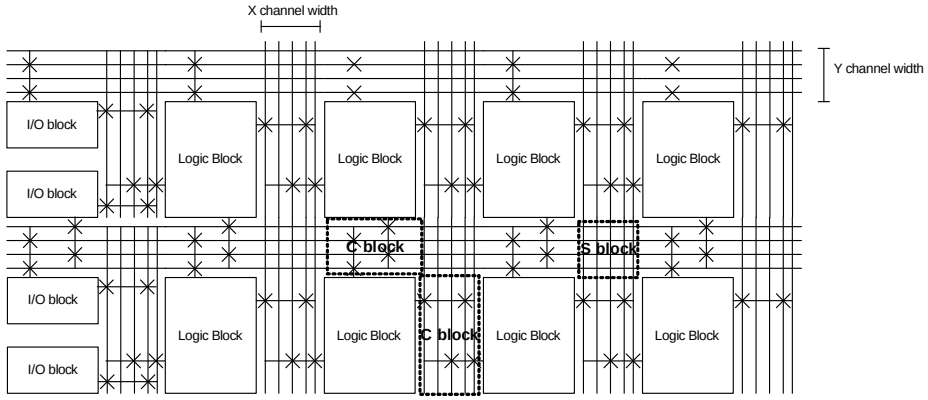
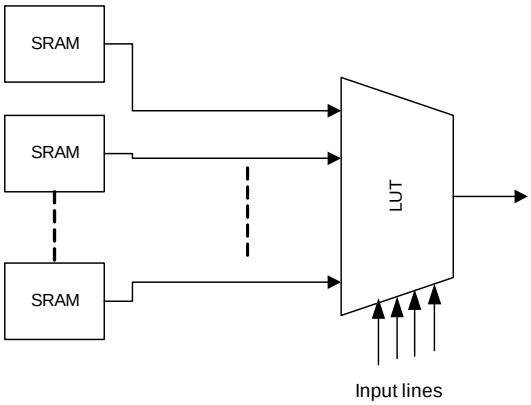


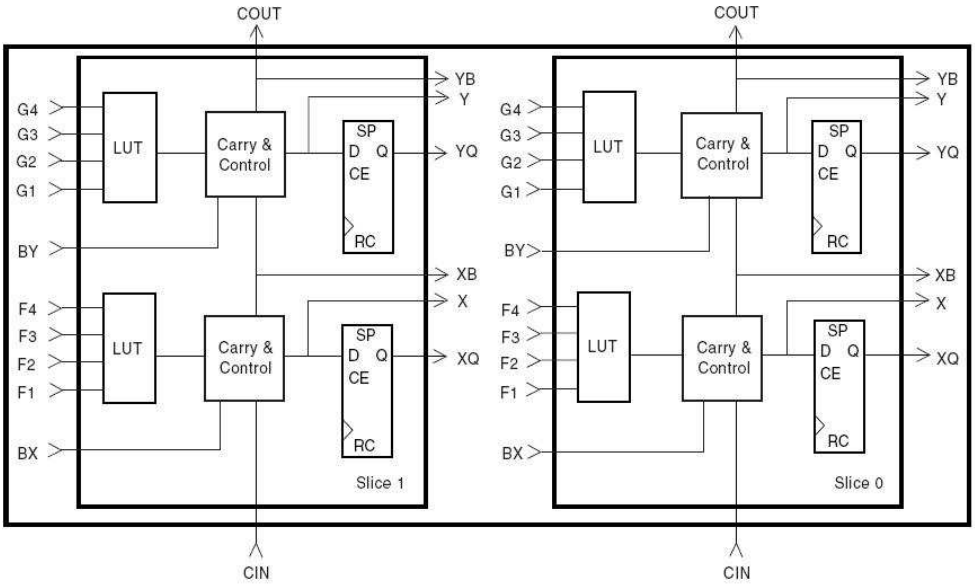
Figure 2: Island type FPGA architecture

logic function. The LUT requires 2^n SRAM bits to store the output value for 2^n input combinations. A logic block may also include a flip-flop to implement a sequential circuit. A combination of a LUT and a flip-flop is called a basic logic element (BLE). A number of BLEs can be grouped together in a logic block. This type of architecture is called a cluster-based logic block.

The exact implementation of a logic block varies across the FPGA families and vendors. The architecture has evolved to provide more functionalities such as a full-adder carry chain, multiple LUTs with internal interconnection network to implement complex logical functions within a single logic block. For example, Figure 3b shows the logic block for Virtex E FPGA from Xilinx [1]. This logic block is made up of four identical 4-input LUTs organized in two slices. A slice is a logical division, intermediate between a logic block and a basic logic element. A slice also contains the logic that combines the function generators to provide five or six input functions. Each LUT can also be used as a 16×1-bit synchronous RAM. The dedicated carry logic provides fast arithmetic carry propagation for the high speed adders. With these multitude of options within a single logic block, FPGA manufacturers are trying to make the FPGAs more flexible and fast. Virtex E was launched in 1999; the subsequent versions of FPGAs launched after Virtex E pack more logic into a logic block and have a higher density interconnection network. Some of them also integrate



(a) Look up table



(b) One logic block of Virtex E

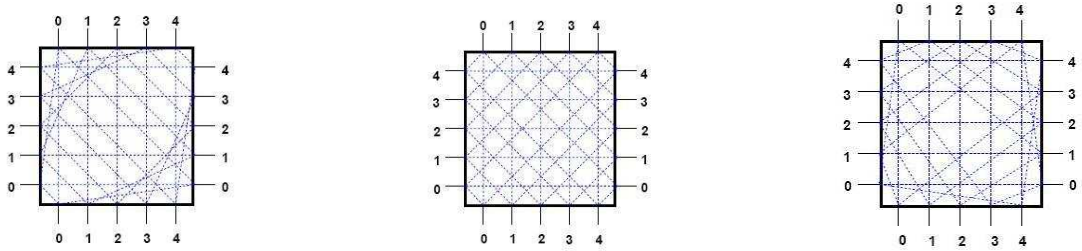
Figure 3: Logic block

a microprocessor core.

2.1.2 Switch Block

The switch block (S-block) is placed at the intersection of vertical and horizontal channels. It can be programmed to route nets from source to destination channel. An S-block can be viewed as a rectangular box with W pins on all the sides. The flexibility of an S-block is defined as F_s, which specifies the number of wires to which a source wire can be connected to. For most of the architectures, the typical F_s value is three.

Figure 4 depicts three common switch block topologies. Each topology allows an incoming track to be connected to three outgoing tracks. A disjoint switch block is symmetric, it will connect an incoming track i to the outgoing track i. This limits the routing flexibility since it restricts a net to the same track on all the channels. The universal switch block provides a greater flexibility. Foe example, as shown in Figure 4b, an incoming track 1 from west can connect to the track 1 in south and east directions and the track 3 in the north direction. The Wilton switch block is similar to the universal topology except that each diagonal connection has been rotated by one wiring segment. This allows wire to change tracks at the S-block, and hence, provides more flexibility for routing compared to the previous two topologies.

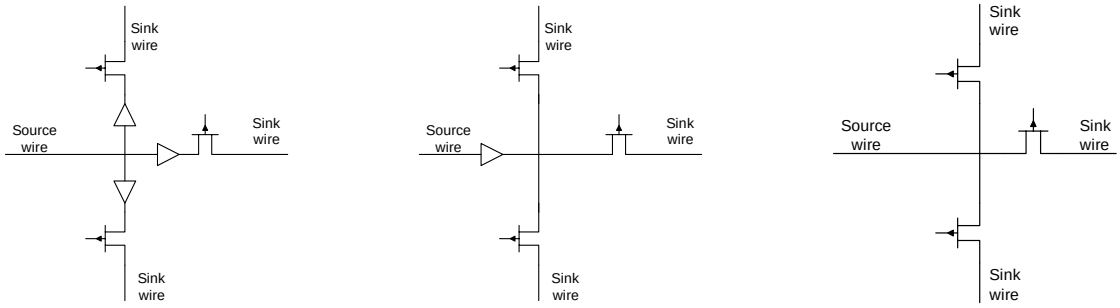


(a) Disjoint (b) Universal (c) Wilton

Figure 4: Switch block topologies

The switch connections are implemented by either using a pass transistor or a buffer. Figure 5 shows a connection within an S-block junction [17], with an F_s of

three. For the no buffer sharing configuration, each tri-state buffer switch in an S-block is implemented as a separate buffer plus a pass transistor. The use of a buffer at each fanout increases the driving capacity of the wire, reducing the propagation delay. On the other hand, the buffer sharing configuration shown in Figure 5b, gives significant area savings. The buffer sharing configuration requires only one tri-state buffer compared to three tri-state buffers needed in the former configuration. The capacitive load seen at the fanout is high; hence, the propagation delay for wires driven by these switches is high. Figure 5c shows a switch junction made with only pass-transistors. This configuration occupies least area, but causes loss of signal integrity. It is not recommended to use pass transistor switches for routing long paths. An FPGA may include different types of switch junctions to optimize the design for speed and area.



(a) No buffer sharing (b) Buffer sharing (c) Pass transistor switch

Figure 5: Switch block junction designs

2.1.3 Input/Output Block

The input/output block (IOB) provides a buffered input and output interface for making connections with the external signals, through the package pins. These blocks are arranged around the periphery of an FPGA. Figure 6 shows the Virtex E IOB [1]. The output signal can be driven either directly to the pad to give an asynchronous output, or through the flip-flop to provide a synchronous output. Similarly, the input signal can be synchronized with a clock, if necessary. A pull-up, a pull-down, and a

weak-keeper circuit is also available at each pad. The IOBs are packed together into banks to form blocks of size almost equivalent to the logic block size.

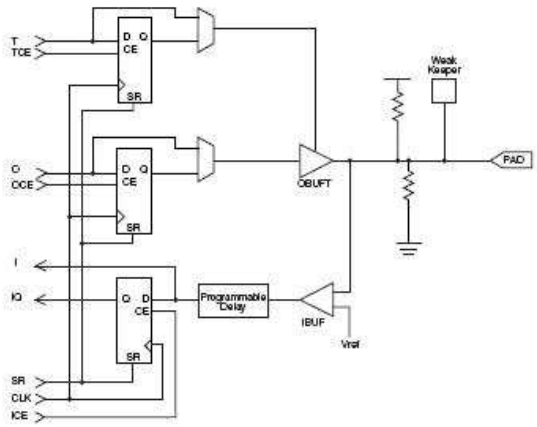


Figure 6: Virtex E Input/Output block

This section introduced the core components of an FPGA, which will help to understand the following sections. In addition to the basic blocks described above, the commercial FPGAs do have certain dedicated features for the logic implementation and the interconnection, but they are not essential for our study.

2.2 FPGA CAD Flow

Having understood the FPGA architecture, we can imagine that programming an FPGA would require the specification of thousands of bits for the state of each LUT and the programmable interconnection. The process of generating these configuration bits is achieved by the use of Computer-Aided Design (CAD) tools. There is no common standard for the format of the configuration bits, often times each FPGA manufacturer offers its own CAD tool to generate a bitstream for their FPGA products.

Figure 7 shows one FPGA CAD flow [17]. The CAD tools convert a high level description of the design specified in a hardware description language (HDL) to the programmable hardware specific format. First, the programmer describes the design

in an HDL, e.g., VHDL or Verilog. Next, the design is synthesized into a technology independent netlist, which is a digital representation of the design. The netlist is optimized to remove redundant logic gates and mapped to the LUTs. At this stage, the CAD tool requires the knowledge about the architecture of the FPGA to be programmed.

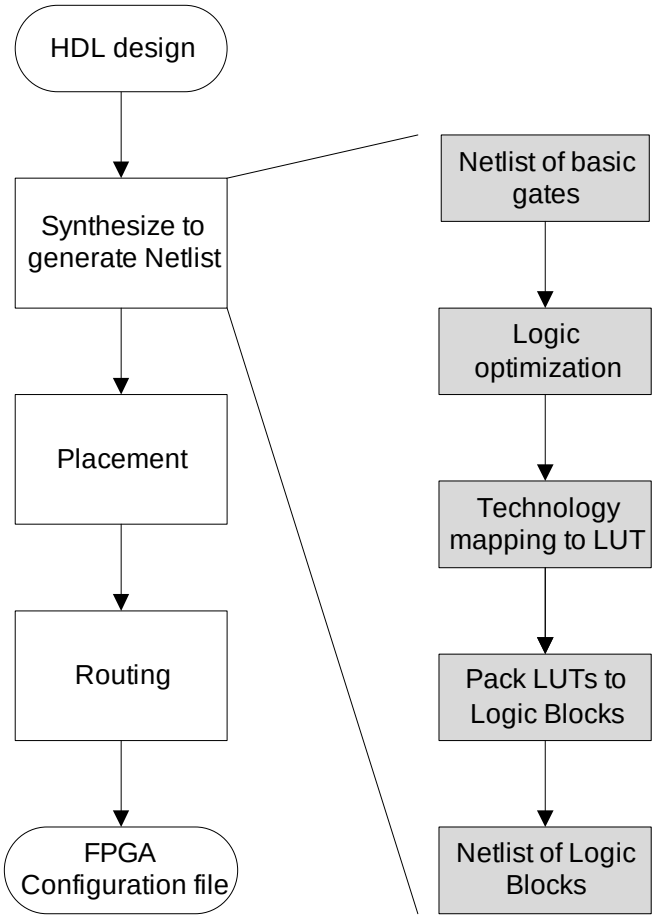


Figure 7: FPGA CAD flow

For a cluster-based FPGA, several LUTs are packed together to fit into a single logic block. The grouping of the LUTs is subject to constraints placed by the number of distinct input and clock signals that can be routed to the logic block. The primary goal of packing the LUTs together in a logic block is to place the connected LUTs close together, this will reduce the routing through the global interconnection network. To minimize the total logic blocks used for implementing a design, the packing stage also

attempts to pack the logic blocks to its maximum capacity.

The next stage of the CAD tool selects the precise position for each logic block within an FPGA. The objective of placement can be one of the following:

1. Routability-driven placement: to balance the routing channel congestion across the FPGA

2. Wirelength-driven placement: to minimize the total routing wire length

3. Timing-driven placement: to maximize the circuit speed

During the routing stage, the connections are made between the input and output pins of the logic blocks, and the IOBs. The goal is to find a path between the respective source and the sink pins, with an objective of either minimizing the congestion or minimizing the delay. The finite routing lines, the limitations placed by the switch block topology, F_s, and F_c adds to the routing complexity.

The process of routing is sometimes divided in two steps: the global routing and the detailed routing. The global router does not require knowledge of the detailed interconnection architecture. This router defines a coarse route for each path. The global router balances the use of routing channels to prevent congestion. The detailed router will assign the specific wire segments and the switch connections to complete the path through the channels chosen by the global router. The next section will give a detailed description of the routing algorithm implemented by the VPR tool.

Finally, the CAD tool will generate the configuration file. This file describes the placed and routed design in a format compatible with the target FPGA.

CHAPTER III

ROUTING

3.1 Routing-Resource Graph

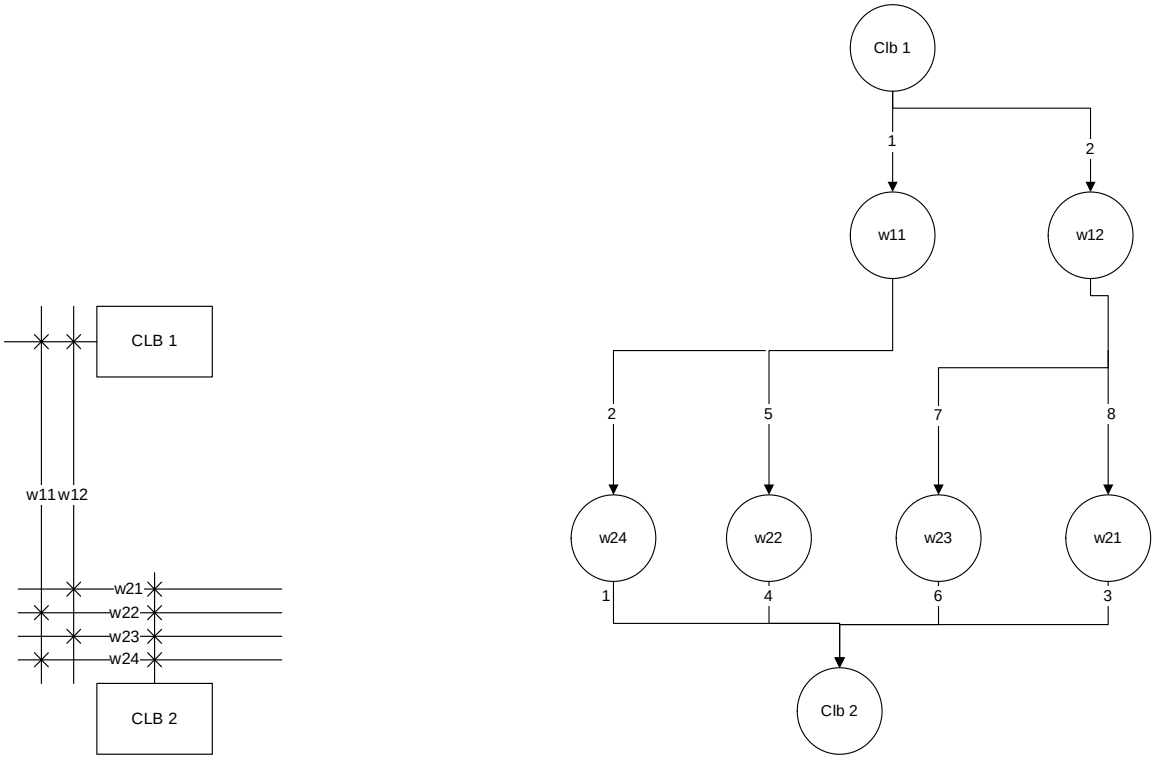
The routing-resource graph (RRG) represents all the routing resources of an FPGA in terms of nodes and edges. This will transform the problem of routing a net to a graph search problem. Each wire segment and logic block pin is represented as a node, while a potential switch connection is represented as an edge on the RRG. The graphical representation is exhaustive, it contains all the connectivity information necessary for the routing.

Figure 8 shows an example of generating an RRG from the FPGA blocks. Consider the partial block of an FPGA shown in the Figure 8a. The first step is to identify the nodes. There are eight nodes, two for CLBs and six for wire segments. The CLB1 pin can be connected to two wire segments, we represent this on RRG by a directed edge from CLB1 pin to nodes w11 and w12. The wire segment w11 can be connected to w24 and w22. This is represented by an edge from w11 to w22 and w24. Similarly, we can represent all the routing resources of an FPGA using this graphical notation. As shown in Figure 8b, each edge has an associated number that signifies the cost of using the edge. We will define the cost function and describe its use, in the graph search algorithm described in Section 3.2.2.

3.2 VPR Routing Algorithm

3.2.1 Background

The routing algorithm finds a path to realize all the nets over the finite routing resources of an FPGA. The optimization goal is to find the shortest path from the source to the sink nodes without congesting the available resources. An iterative



(a) Actual resources available

(b) Routing-Resource Graph

Figure 8: Example of mapping a Routing-Resource Graph

search process is used to solve this complex problem.

Most of the present day routing methods are a variant of the Maze router [6]. The Maze router uses Dijkstra algorithm [8] to find the shortest path between the net source and the sink nodes in a routing-resource graph. Consider a minimal length path from Node A to Node B. Then for any Node C that lies on path AB, the path A to C is also the shortest path. Therefore, to find the shortest path AB, Dijkstra's algorithm finds the minimal length path from Node A to intermediate nodes (in the order of increasing length) until it hits Node B. Figure 9 outlines the Dijkstra algorithm.

An FPGA router has two goals, one is to reduce resource conflicts and the second is to minimize the overall delay. Using Dijkstra algorithm alone to find the shortest

```
Src= source node
Snk= sink node
RR= graph
RT= current routing tree , after execution it will hold the shortest path
dist[n]= distance of node n from Src
prev_node[n]= the node from which the edge has been projected on node   n

Initialization:
for each node n
        dist[n]=infinite
        prev_node[n]=null
prev_node[Src]=0;
Algorithm:
while RR is not empty{
temp= return minimum distance node from RR
If temp is equal to Snk
        add temp to RT and end

for each neighbor  v of temp{
        if dist[v] is greater than dist[temp] + (dist from temp to v)
                assign dist[v]= dist[temp] + (dist from temp to v)
                prev_node[v]=temp
}
}
```

Figure 9: Pseudo code for Dijkstra's Algorithm

path for each source-sink pair will lead to many unroutable nets, because of resource overutilization. A common solution to this is to repeatedly rip up the unroutable nets and re-route them until a valid route is found for all the nets. The order in which the unroutable nets are ripped and re-routed is based on some selection criteria. However, a disadvantage of this approach is that the final result depends on the order in which the nets are selected for re-routing.

The global routing technique for ICs proposed by Nair [14], rips and re-routes all the nets after each routing iteration. This makes the results independent of the order of selection of the nets for rip up and re-route. At the end of a routing iteration, Nair's router assigns a cost to each node which is proportional to the resource overutilization. This will force some nets to divert out from the congested regions. The global routing iteration is repeated, until all the nets negotiate to find a valid routing through the available resources.

Pathfinder [13] is an FPGA specific routing technique derived from Nair's IC

router. It provides a new cost function that accounts for the congestion as well as the critical path delay. Similar to Nair's algorithm, Pathfinder initially allows the nets to share all the routing resources. After each routing iteration a congestion penalty is added to the overused resource nodes. In addition to this, the Pathfinder performs a timing analysis at the end of each iteration and updates the path criticality values for each source-sink path. The congestion penalty and the path criticality terms are included in the cost function to achieve an optimum routing. In sum, Pathfinder's cost function attempts to satisfy both the goals of routing, which is minimizing the delay and balancing the resource utilization.

3.2.2 Pathfinder Negotiated Congestion/Delay Router

The primary contribution of the Pathfinder is the addition of a delay sensitive term to the cost function. The cost of using a node n when routing a signal from source Src_i to sink Snk_{ij} is given by Equation 1a.

$$C_n = Slack_{ij} d_n + (1 - Slack_{ij}) c_n \tag{1a}$$

$$where, c_n = (b_n + h_n) p_n \tag{1b}$$

$$Slack_{ij} = \frac{D_{ij}}{D_{max}} \tag{1c}$$

The term c_n is the congestion cost for a Node n. At the start of the routing, c_n is equal to the base cost, b_n. For a congested routing resource, the present cost p_n gives the number of other resources using the Node n. The historic congestion, h_n is related to the history of congestion at the Node n, during the previous routing iterations. The history term helps to solve the second order congestion problem, which was detailed in [13]. The term $Slack_{ij}$ known as the slack ratio, indicates how small is the delay of the path from Node i to Node j compared to the critical path delay. The slack ratio is calculated as the ratio of path delay, D_{ij} to the critical path delay, D_{max}.

For the critical path, Slack_{ij} has a value of 1. This implies that the cost function in Equation 1a now consists of only the delay term while it ignores the congestion cost term. A critical net is allowed to take the minimum delay path, and it is oblivious to the resource congestion. On the other hand, as the slack ratio begins to increase there will be more freedom to route these nets. The cost function will then increase the congestion cost trying to force the net through the non-congested regions.

Pathfinder router uses the above cost function alongwith breadth-first search to route all the nets, at each routing iteration. After the end of a routing iteration, the router will update the congestion cost h_n for all the nodes. It then rips up and re-routes each net until it finds a realizable routing solution for the circuit.

3.2.3 VPR Router

VPR is an open source place and route tool developed by University of Toronto [16]. The tool takes a technology mapped netlist of the circuit and the FPGA architecture definition as inputs, and it generates the placement and routing files as outputs. It would help to learn about the VPR routing algorithm to adopt it for our watermarking technique.

VPR router was built upon Pathfinder algorithm discussed in Section 3.2.2. VPR provides two options for routing: the routability-driven and the timing-driven. The routability-driven router balances the congestion throughout the FPGA chip, however, it does not optimize the circuit speed. On the other hand, the timing-driven router finds the minimum delay path for the critical nets and also balances the resource usage, at the same time. The primary objective of using an FPGA implementation of a design is to make it fast. So in this study, we are going to concentrate on the VPR timing-driven router. The pseudocode for VPR router is shown in Figure 10.

RT[i] = array of nodes n on the current routing of net i
T_{el}[n] = Elmore delay upto node n on the current routing
R_{up}[n]= Upstream resistance of the path, looking up from node n
TotalCost[n]= Total estimated cost of using node n for wave expansion, it uses A* cost function
PathCost[n]= Path cost from source to node n
SPathCost[n] & STotalCost[n] stores the current minimum cost value from source to node n
Crit[i][j]= Slack ratio for path from source node i to sink node j

VPR Timing Driven Router:

```
While(routing not realizable due to resource overuse){
For(each net i){
        DynamicallyUpdateBaseCost(i)
        RipUp RT[i]
        Update p(n) for all the ripped nodes
        Add the source of net[i] to RT[i]
        for(each sink j of net[i], in order of decreasing Crit[i][j]){
                Queue = Add all nodes n currently in RT[i]
                Call function WaveExpansion()
                Backtrace from sink j to source i adding all new nodes to
                RT[i] & updating p(n) for each node
        }
        Calculate Elmore delay for net RT[i]
        Set STotalCost and SPathCost to infinity
}
Update historic congestion h(n) for each node
Determine the new values of Crit[i][j] based on current routing
}
```

WaveExpansion:

```
\\This is similar to A* algorithm with cost function derived from Pathfinder
while(sink j not found) {
m= return n such that it has lowest TotalCost[n] in the Queue
if(TotalCost[m] < STotalCost[m] && PathCost[m] < SPathCost[m]){
        //This is a lower cost path from source to node m
        STotalCost[m]=TotalCost[m]
        SPathCost[m]=PathCost[m]
        for(each neighboring node q of node m){
                Calculate Rup [q]
                PathCost[q] = PathCost[m] + Cost[q]
                TotalCost[q] = PathCost[q] + α ExpectedCost[q]
        }
}
}//stops when wavefront hits the sink j
```

Figure 10: VPR Timing-driven Router: Pseudocode

VPR borrows the following ideas from Pathfinder algorithm: it rips up and re-routes every net, until no congested routing resources exists and it gradually increases the congestion cost after each iteration. However, there are certain enhancements [17] to the Pathfinder that we shall discuss in detail here.

For routing a single net with k sinks, the wave expansion algorithm shown in Figure 10 is invoked k times, starting with the most critical sink. Consider the connection to Sink j from Net i. The cost of adding a Node n to the net's routing tree is:

$$Cost(n) = Crit[i][j].delay_{Elmore}(n, topology) + [1 - Crit[i][j]].Cong(n). \quad (2)$$

The criticality of Sink j is defined by the equation:

$$Crit[i][j] = max([MaxCrit - \frac{slack[i][j]}{D_{max}}]^{\eta}). \quad (3)$$

In Equation 3, *MaxCrit* and η are parameters that control the weightage of slack's impact on congestion-delay trade-off, in the cost function Equation 2.

The congestion cost function for VPR is defined as follows:

$$Cong(n) = b(n).h(n).p(n). \quad (4)$$

b(n) is the base cost of using a routing node. The default base costs are as given in the Table 1.

Table 1: Default base cost of routing resources

$RoutingResource, n$	$BaseCost, b(n)$
Wire segment	1
Logic block output pin	1
Logic block input pin	0.95
Source	1
Sink	0

The base costs are normalized to the average delay of the routing resource. The driving capacity of a pass transistor switch is less compared to the buffered switches, so for high fanout nets we should use more buffered switches. The DynamicBaseCost function will bias the router accordingly. For a net with high fanout, the function increases the base cost of using wires that connect to other wires via pass transistor switches. Consider a net with a fanout of k, the new base cost $b_{updated}(n)$ is computed as,

$$b_{updated}(n) = b(n)\sqrt{k}. \tag{5}$$

The DynamicBaseCost function will increase the utilization of wires connected through buffered switches, for a net with high value of k. This will also enable the different sinks of a Net i to share the partial routing without overloading the source.

The present congestion penalty, *p(n)*, is updated every time a net is ripped and re-routed. The value of *p(n)* is given by the equation:

$$p(n) = 1 + max(0, [occupancy(n) + 1 - capacity(n)]p_{fac}). \tag{6}$$

The historic congestion cost, *h(n)*, is updated at the end of a routing iteration. The value of *h(n)* is given by the equation:

$$\begin{aligned} h(n) &= 1 \text{ , for first routing iteration} \\ &= h(n) + max(0, [occupancy(n) - capacity(n)]h_{fac}) \text{ , for remaining iterations} \end{aligned} \tag{7}$$

The values of h_{fac} and p_{fac} controls the routing schedule of VPR's routing. *Capacity(n)* gives the maximum capacity of the routing resource Node n, and *occupancy(n)* represents the current demand of the resource n.

VPR uses A* search function, α controls the weightage given to the expected cost, during the wave expansion phase described in Figure 10. Setting α to zero will reduce

it to the Dijkstra search function.

The wave expansion algorithm has been optimized for speed. VPR does not allow the net routing to expand more than three channels outside its net bounding box. Unlike Pathfinder, VPR does not empty the expanded wavefront after reaching a net sink. Instead, it assigns a zero cost to all the nodes that connect this sink to the partial routing tree and begins expanding around these nodes. This saves the time to re-expand the entire wavefront every time a new sink is added to the net. Although, this does not affect the final result quality, it does reduce the execution time of the router.

In brief, the major modifications in VPR include:

- New congestion cost function and routing schedule
- Dynamic base cost update
- A* search algorithm for wave expansion
- Elmore delay to estimate the delay
- Modification of wavefront expansion technique for better speed

CHAPTER IV

WATERMARKING AND SIGNATURE GENERATION

Digital watermarking is a security technique that inserts a unique mark in a design to identify the author of the design. The VSI Alliance [10] describes a foreseeable enterprise model for the re-usable designs where an independent agency will carry records of IP ownership, labeling, and digital signatures. This enterprise is similar to the current proprietary music distribution system where the fees for the use of music are collected from the users and distributed to the music owners as royalty.

The VSI Alliance group have further discussed three fundamental techniques for IP protection. The first technique is a deterrent approach where the author tries to deter the illegal acts by filing patents, copyrights, and maintaining trade secrets. The second technique is a protection approach where the author takes active measure to protect their own IP by using license agreement and encryption mechanism. This does secure the IP to certain extent, but we still need a mechanism to detect the breach of trust. The third technique described is to detect and trace the design theft. This requires ability to embed tags, digital signature, watermark, and fingerprinting for a design. The detection scheme should have a high degree of confidence, so that it would be acceptable as a proof in the court of law.

A good watermarking scheme should posses the following properties:

1. Difficult to remove without damaging the IP
2. Resistance to tampering
3. Low area and timing overheads on design
4. Transparent

5. Strong proof of authorship

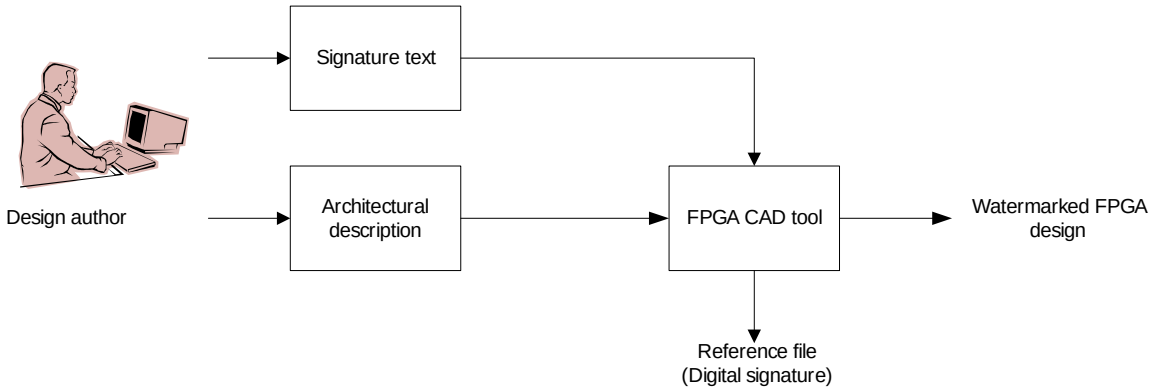
6. Low false positive rate

4.1 Security Model

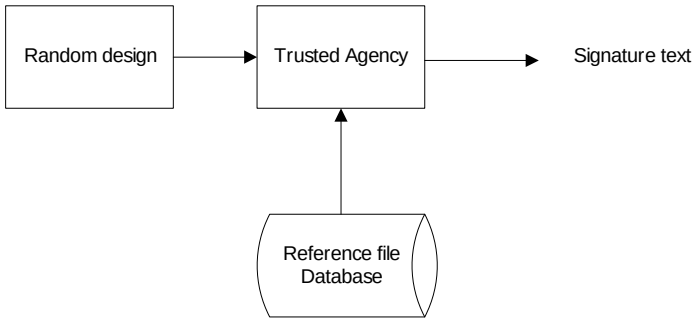
We propose the security model shown in Figure 11 for the FPGA design protection. A unique mark that identifies the design author is inserted within the routing of the design forming the watermark. The process of embedding the mark is integrated in the CAD tool, and it is completely transparent to the end user. The next step is watermark verification. The CAD tool generates a digital signature of the watermarked design and stores it in a reference file. The design author is responsible to register his design with the trusted agency and submit the configuration file, the reference file, and the signature text used for watermarking. The trusted agency maintains a database of the reference files. It can pick a suspicious design from the market and verify it with the signature database to identify the owner. The number of unique watermarks that our technique can insert is very large, thus the author can choose to generate a different watermarked version of the design for each of his customers. The signature text can include the name of the customer along with the author's name, this will allow to trace the source of IP leakage. Let us discuss the watermarking and the digital signature generation process in detail.

4.2 Embedding Watermark in the Routing

We propose to embed a watermark at the design routing stage. Our technique may be classified with a set of constraint-based watermarking techniques [3][9]. We specify the routing constraints that are directly correlated to the signature text defined by the author. The routing generated under these constraints is a function of the signature bitstream. The watermark, if added during the early stage of the CAD flow, might be removed or distorted due to the design optimizations. In our watermarking technique,



(a) Watermarking

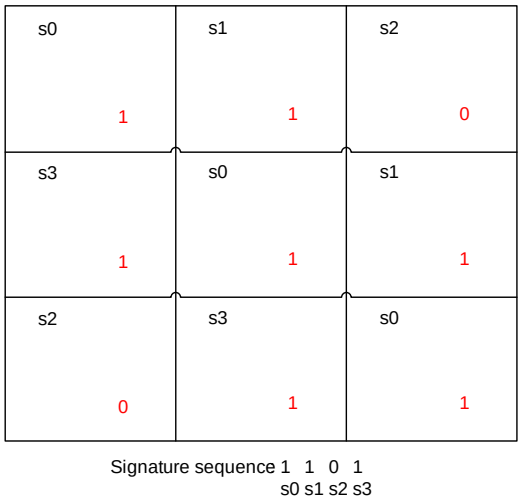


(b) Verification

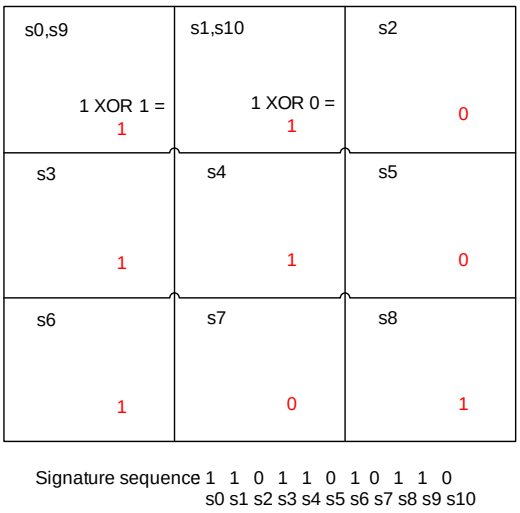
Figure 11: Security model

we introduce the mark towards the last stage of the CAD flow, hence, it is guaranteed to be passed over to the final design bitstream. The complex structure of the routing helps to hide the signature, and thus makes the watermark transparent to the user.

At this stage, since the design has already been placed, we can determine the bounding box for the design. The bounding box is the minimum sized box that can encompass all the logic blocks used by the design. The signature text is converted into ASCII and one sign bit is assigned per FPGA tile that is enclosed within the bounding box. As shown in Figure 12a, for short signatures, the signature stream is repeated until it covers all the FPGA tiles. On the other hand, if the length of the signature stream is more than the number of FPGA tiles, then the overflowed bits are XORed. As Figure 12b shows, the length of the sequence is 11 bits, and the size of the FGPA is (3x3) 9 tiles. The first overflowing bit s9 is XORed with s0, and the



(a) Sequence length less than number of tiles



(b) Sequence length more than number of tiles

Figure 12: Signature spreading on the FPGA

result is mapped to the first tile.

Let S be the signature bit assigned to the *Tile[x][y]*, and hence, to every routing resource Node n within the tile. This means that all the logic block input and the output pins, and the horizontal and vertical wire segments of the *Tile[x][y]* will have a sign bit *S[n]* associated with it. Let us route a connection from the source to the Sink j on Net i. The cost of adding a Node n to the net's routing tree is given by Equation 2. We add a new term, called the sign cost which is given by:

$$SignCost(n) = (1 - e^{\frac{-(1-Crit[i][j])}{h(n)}}).b(n).S(n). \quad (8)$$

This term introduces an artificial cost that adds up with the cost of the node given by Equation 2. The extra cost for using the resources that belong to the tiles having the sign bit set to one will initially force the routing tree to diffuse out of these tiles. Ideally, we want the watermarking technique to not affect the critical delay paths and gradually increase the pressure to diffuse on the non-critical paths, in the order of increasing slack ratio. This relationship can be mapped using an

exponential term. The criticality factor in the power of the exponential term will reduce the *SignCost* to zero, for the critical path connections. This ensures that watermark insertion has minimal side-effects on the circuit speed. The *SignCost* value will increase exponentially for nets having a small *Crit[i][j]* value. This will force the non-critical nets to try a different node, and hence, deviate from the original routing path. The original routing path refers to the path this net would have taken in the absence of a watermark. The sign cost is not a real penalty, so it should not completely subdue the effect of original cost function. The *h(n)* in the denominator of the power prevents the watermarking technique from throttling the historically congested resources due to the *SignCost*. The base cost, b(n) will normalize the *SignCost* value to the remaining terms in the cost function. Finally, the *SignCost* term reduces to zero, if the node is assigned a sign bit zero.

From a higher level view, adding the sign cost can be viewed as modifying the routing resources available within a tile. From router's perspective, a tile with sign bit of one would appear as if there are less resources available in that tile. As a result the FPGA fabric appears to the router like an heterogeneous architecture with different sized tiles. The resources available to the router could also have been modified by removing certain nodes from the routing resource graph. But, this would force hard constraints on the router that could affect the critical path delay. Instead, our proposed watermarking technique will dynamically squeeze the resources with an attempt of not penalizing the critical paths.

Let us assume a signature of l bits used as a watermark in a Design D. The Design D fits within a bounding box of size a (in number of tiles). The above information can be used to calculate an approximate value for the number of unique watermarked instances (x) that can be created:
if $l < a$ then,

$$x = 2^l. \tag{9}$$

if $l > a$ then,

Probability of signature collusion, $p_c = 1 - \frac{2^a}{2^l}$.

$$x = 2^a. \tag{10}$$

From Equations 9 and 10, we observe that more watermarks could be embedded in larger designs. Similar, to any other security technique, a longer signature would guarantee more strength; however, there would be a risk of signature collusion, when the signature length exceeds the design size a. Alternately, the upper bound set by design size a could be increased by mapping the signature bits at a lower granularity of individual channels or wire segments within a tile.

The probability of false positive for a good watermark should ideally be zero. A false positive means mistakenly declaring the two configuration files belonging to the same watermarked design, when they are actually different. Our watermark is inserted in the hard IP; thus, finding a match between the configuration files means that all the nets in the the two designs have matching routes. Therefore, all the switches, and the logic block input and output pins used by both the designs are exactly the same. Which means that the constraints imposed during the placement and routing of the design were identical. Such a security flaw can be avoided by better selection of the signature text used for watermarking. The signature text should include a design name, a timestamp or revision number, and the customers' name.

Given a Design D, that is watermarked with two non-colluding signatures. If the configuration files of the two designs collude then it implies that the set of constraints imposed by the watermarks were identical. Note that the signatures are different, so the constraints should be different. However, the constraints imposed by signature are soft constraints, that is, we allow the router to ignore the constraints under two cases: if it is routing a net on the critical path or the resource is historically congested. This could occur when the two signatures differ in only those bits that map to a tile where all resources are either historically congested or being used by critical net connections

only. Let us call a tile that shows these properties, a throttled tile (T-tile). Let the Design D be routed on a FPGA having W wire segments per tile. Let h be the probability of a node being historically congested. If the circuit has C critical nets out of the total of N nets, then the probability of finding a T-tile is $(\frac{C}{N} + h)^W$. Generally, $N >> C$ and W is in order of 10's. The combined probability of finding a T-tile, and a pair of signatures that differs in bit positions of only T-tiles is very low.

The design effort required to insert the watermark is quite low. The watermarking scheme is included as an optional feature of the CAD tool. Unlike many other watermarking schemes, our technique does not require any post-processing steps to insert the signature. The time taken to convert the signature to ASCII and assign it to the FPGA tiles is negligibly small and could be ignored. Apart from modifying the cost function, the rest of the routing process remains the same. In short, integrating watermarking in the CAD tool requires little design effort and the time taken to route the design remains approximately the same to that of routing a non-watermarked design.

4.3 Digital Signature Generation

In the last section, we discussed an important technique of inserting a watermark in the design. It is equally important to verify the existence of the signature in the design, hence we generate a digital signature of the watermarked design. The digital signature of an FPGA design will capture the unique features of the design. This technique is similar to a lossy compression where a complex FPGA design can be completely characterized by the digital signature. Some techniques have been proposed to create a power signature, a noise signature, and a signature based on LUT content analysis [7]. The trusted agency, described in Section 4.1 will store the digital signatures of all the watermarked designs, we call these signatures as the reference files. A suspicious design is checked against the reference file database to

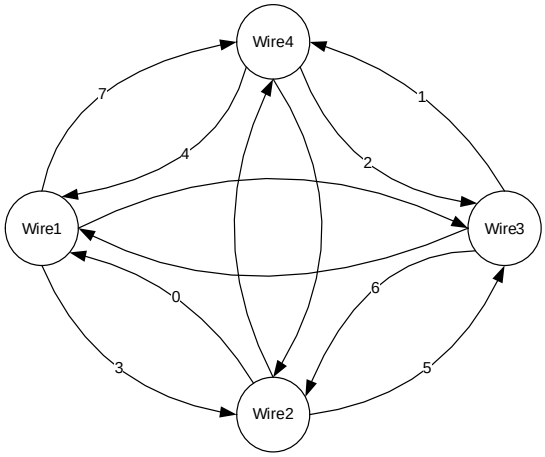


Figure 13: Graphical representation of S-block junction

identify the watermark.

The reconfigurability of an FPGA can be attributed to the large number of switches inside the FPGA. Our digital signature technique uses the switch positions of a sufficient number of S-block junctions to identify the IP core. The graph for an S-block junction with an F_s value of three, is shown in Figure 13. Each incoming signal can be connected to the three wires. The graph edges show the valid connections. The number associated with each graph edge will be used for generating the reference file. The significance of the numbers is to identity each switch configuration with a unique number.

The probability, P_s for a specified switch to be ON is *0.5*, assuming that each switch connection is independent. Figure 14 shows three possible connections from the source node to the sink node. Consider the path that breaks from the straight line connection at Node (7,2) and goes through Node (7,1). Now given that the location of the sink is (0,2) is known, we can predict that the subsequent switch connections should lead the path down and then to right. Similarly, for the path to the sink that breaks at Node (4,2) and goes through Node (4,3), we can predict that it should go down and then to the left, to reach the sink Node (0,2). In short, while routing a connection between two given points, there is a dependence between the switch

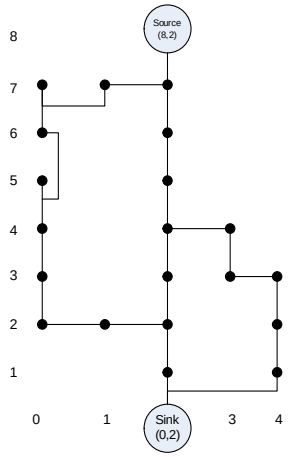


Figure 14: An example to demonstrate switch connection dependence

connections on that path. However, the switch connections on two different nets can be assumed to be independent.

Let us pick one switch connection from m routed nets of the Design *D1* and write their co-ordinates in the reference file. Next, let us match the switch connections at the same co-ordinates on a different Design *D2*. The reference file represents the universal set of switches that we shall try to match. The probability of success for each individual trial; that is, the chances of finding a switch ON is 0.5. The probability of success for k samples is given by the binomial theorem:

$$P_k = \binom{m}{k}.(P_s)^m. \tag{11}$$

For a modest signature of length 64, the probability of finding a complete match is 5.4×10^{-20}. The probability that at least half the signature bits will match is given by substituting $k = 32$, which gives $P_k = 0.1$. The strength of the digital signature is pretty good even with a fairly small sample of 64 switches. To increase the strength of the digital signature, we should capture more features within a design.

According to the binomial theorem's definition, if the sample size is less than 10% of the universal set then the samples may be assumed to be independent. This means that for long nets with more than 20 switches, we can select 2 switches on it, and

assume them to be mutually independent.

We will discuss the implementation details of the watermarking and the digital signature techniques in the next section.

CHAPTER V

IMPLEMENTATION

We developed a prototype of our watermarking model using VPR open source place and route tool [16]. We modified the routing algorithm to introduce the watermarking feature, this was done by adding the signature cost function given by Equation 8 to the original cost function. A digital signature is generated post-processing using the route file, but this process is not on the critical path. The VPR tool accepts a technology mapped circuit netlist and FPGA architecture file as inputs. The FPGA architecture file describes all the parameters of an island-type FPGA, such as the segment length, F_c, F_s, the logic block inputs and outputs, the switch block architecture, the type of switches, the resistance and the capacitance values used for delay computation. The architecture file specifies the parameters for one FPGA tile. Since, FPGA has a symmetric architecture, VPR will generate the routing-resource graph for the entire FPGA structure by replicating the tile. VPR generates the placement file and the routing file as its outputs. The following two FPGA architectures shown in Table 2 have been used to generate the results.

Table 2: FPGA architectures used for simulations

Parameter	Architecture-1	Architecture-2
LUTs per tile	1	1
No. of inputs per LB	4	4
No. of outputs per LB	1	1
F_c LB output	1	1
F_s LB input	0.75	0.75
F_c pad	1	1
Segment length	1	1,4
Switch type	buffered	buffered & pass transistor

Since our results are dependent on the routing, we have used a simple FPGA architecture of just 1 LUT per tile for both the architectures while modifying the interconnection parameters. For the Architecture-1, segments of length 1 with buffered switches are used. For the Architecture-2, we used only 50% of the total segments of length 1 having buffered switches. The remaining 50% segments are of length 4, which means they run across 4 logic blocks. Half of the length 4 segments uses pass transistor switches and the other 50% uses buffered switches. The Architecture-2 adds more constraints for the router, this architecture also reduces the number of candidate switches for digital signature generation.

We set VPR to route for the minimum channel width. VPR uses binary search to find the least possible channel width to route the design. The channel width and the critical path delay are the parameters used to compare the quality of the routing. Since the critical path delay can affect the speed of our design, another feature in our tool lets the user specify an upper bound on the critical path delay of the watermarked design. The user specifies this as a percentage of the critical path delay for the non-watermarked design. We generate the place and route results for a set of 20 MCNC benchmark circuits. The circuits are watermarked with two signature texts: $pratikmarolia$ and $author\_marslabANDcustomer\_xyz$.

The digital signature is generated by processing the route file of the watermarked design. To minimize the false positives between the different versions of the same design, we avoid using the nets that are routed same in the non-watermarked and watermarked versions of the design. The assumption being that most of the critical nets will be left unchanged during watermarking process. So first, we identify the non-matching nets and then identify the ON switches at each S-junction on these nets. We assign a number between 0 to 7, to the different switches in a S-junction and add it to a queue, one queue for each pattern. The graph edge numbers shown in Figure 13 are equivalent to switch numbers on a S-junction. Next, we use the

same signature text used for watermarking insertion and convert it to ASCII. The signature bitstream is divided into groups of three, and a switch node's co-ordinates from corresponding queue is popped out and placed in the reference file, as shown in Figure 15. We propose to use the same signature text for the watermarking and the digital signature extraction process, so that every watermarked design can have a unique digital signature. This method will prevent any claims of ghost signature.

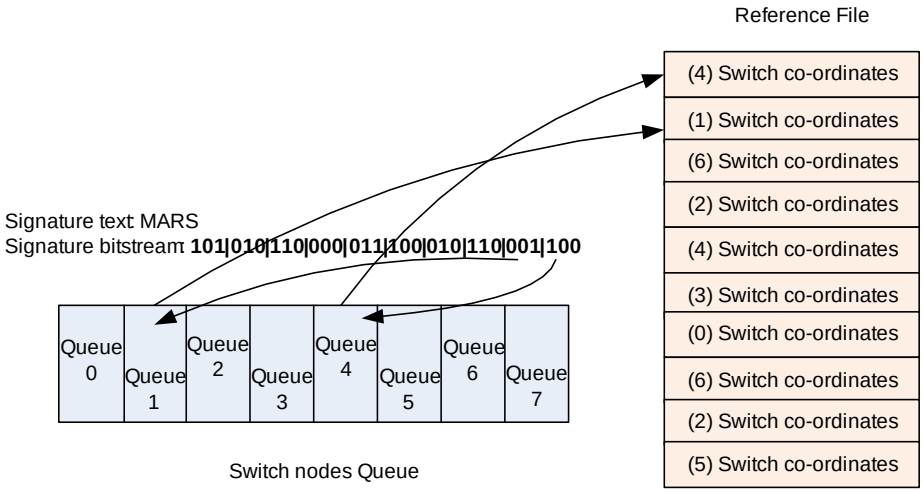


Figure 15: Generating a Reference file

To improve the strength against tampering attack, we add redundancy to the digital signature. This is done by adding some redundancy in the signature to co-ordinates mapping function. Alternatively, an ECC may be added to the signature stream, before using it to generate a reference file.

The simulation results are shown in the following section.

CHAPTER VI

EXPERIMENTAL RESULTS

Table 3: MCNC benchmark circuit parameters

CircuitName	*Blocks*	*Nets*	*CLBs*	*Inputs*	*Outputs*
alu4	1544	1536	1522	14	8
apex2	1919	1916	1878	38	3
apex4	1290	1271	1262	9	19
bigkey	2133	1936	1707	229	197
clma	8527	8445	8383	62	82
des	2092	1847	1591	256	245
diffeq	1600	1561	1497	64	39
dsip	1796	1599	1370	229	197
elliptic	3849	3735	3604	131	114
ex1010	4618	4608	4598	10	10
ex5p	1135	1072	1064	8	63
frisc	3692	3576	3556	20	116
misex3	1425	1411	1397	14	14
pdc	4631	4591	4575	16	40
s298	1941	1935	1931	4	6
s38417	6541	6435	6406	29	106
s38584.1	6789	6485	6447	38	304
seq	1826	1791	1750	41	35
spla	3752	3706	3690	16	46
tseng	1221	1099	1047	52	122

Figure 16 shows a comparison of the minimum critical path delays between the non-watermarked design and the watermarked design. We have generated the results for two signature texts: *pratikmarolia*, and *author_marslabANDcustomer_xyz*. Figure 17 shows the minimum channel width required to route the benchmark circuits. On average, 70% of the circuits could be watermarked with the same channel width. This means that the watermarked design can fit on the same FPGA chip as the original design. From Figure 16, we see that the delay overhead for certain circuits like ex1010, s298, elliptic, s38417 is more than 15%. This might not be acceptable for

ceratin hard realtime applications. We could limit the delay penalty by increasing the channel width on the FPGA architecture. The channel width on commercial FPGAs is sufficiently wide. Xilinx XC4000 E which was released in 1999, has a 24 vertical wire segments and 18 horizontal wire segments [2]. The circuit pdc experienced more than 30% increase in critical path delay on Architecture-2 due in part to the fact that the wateramrked design has been routed over a smaller channel width than the original design.

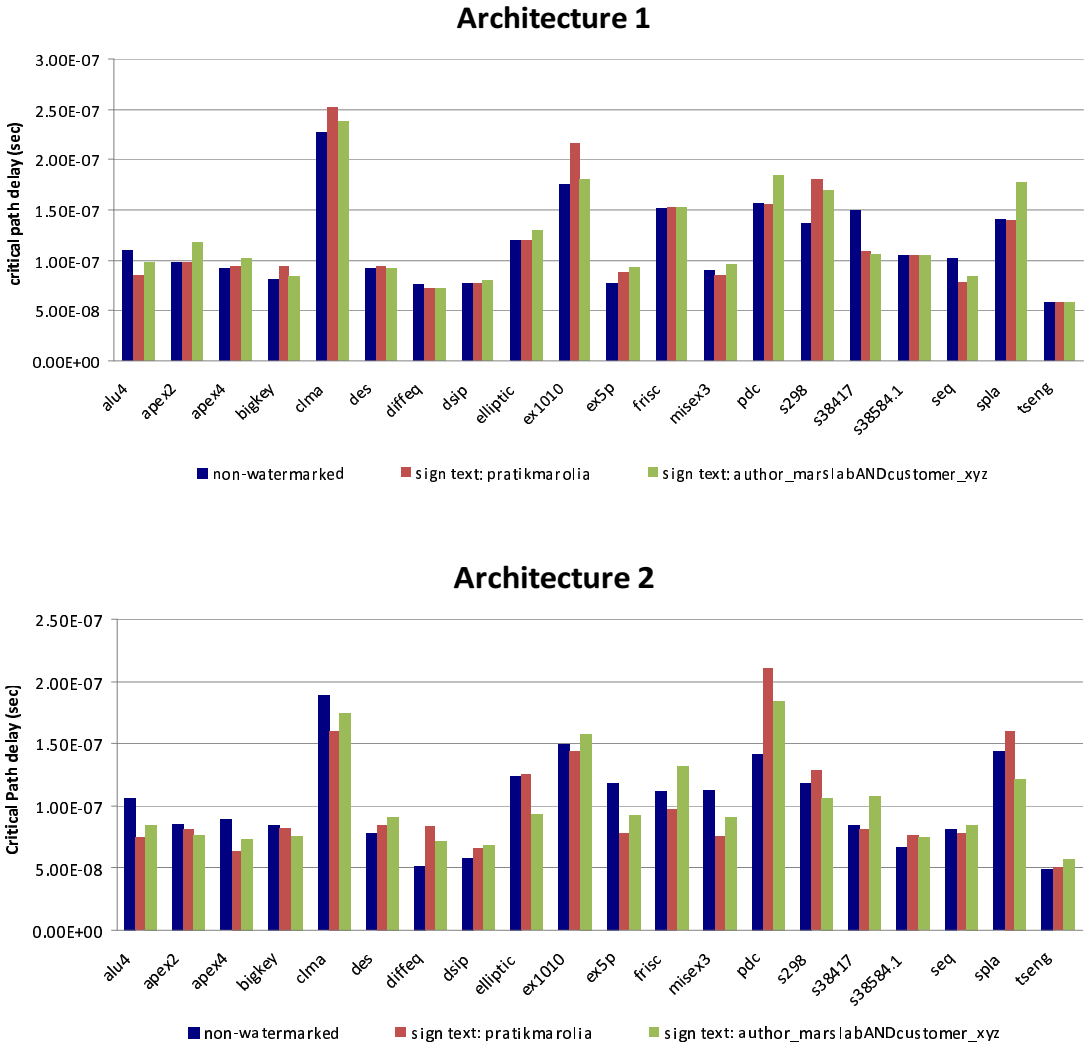


Figure 16: Effect of Watermarking on circuit speed

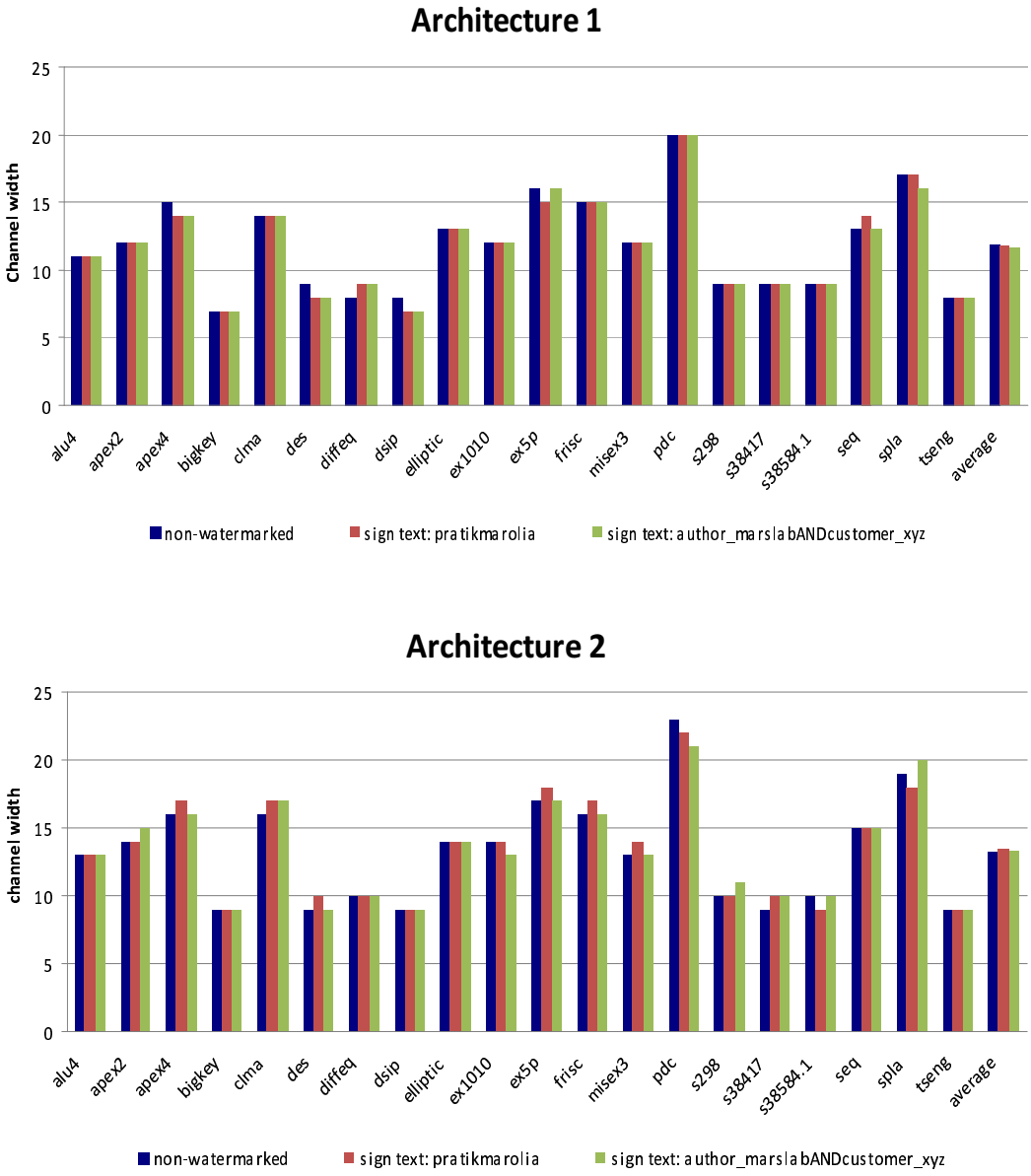


Figure 17: Effect of Watermarking on minimum channel width, to route the circuit

Figure 18 shows the actual routing area overhead for watermarking the design. Our watermark is applied after the placement stage; hence, it would be justified to compare the wire overhead to get an estimate of area penalty due to watermarking. The results show an average negative wiring overhead which means that fewer wire segments are used to route the watermarked design.

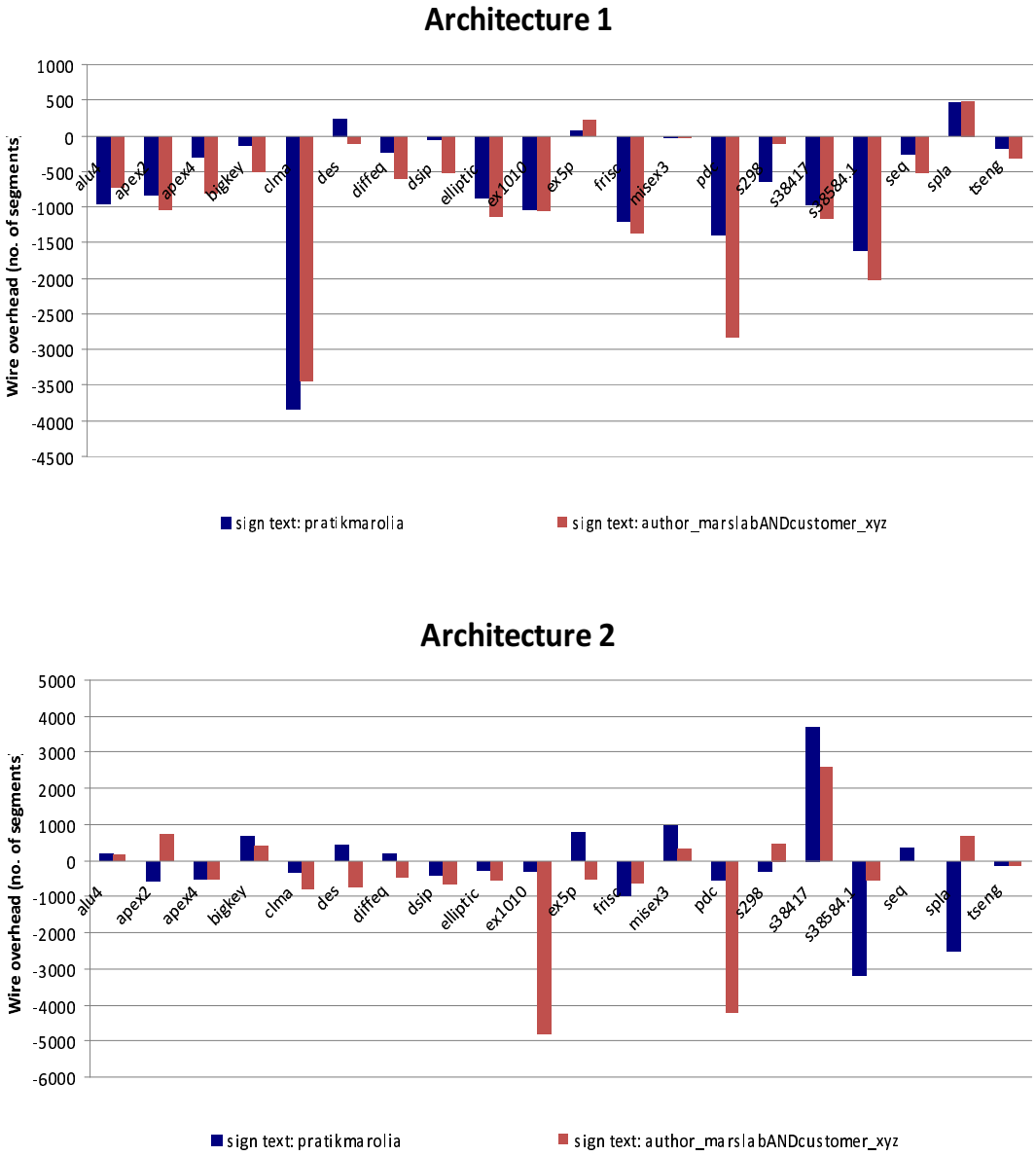


Figure 18: Wire overhead for embedding the Watermark

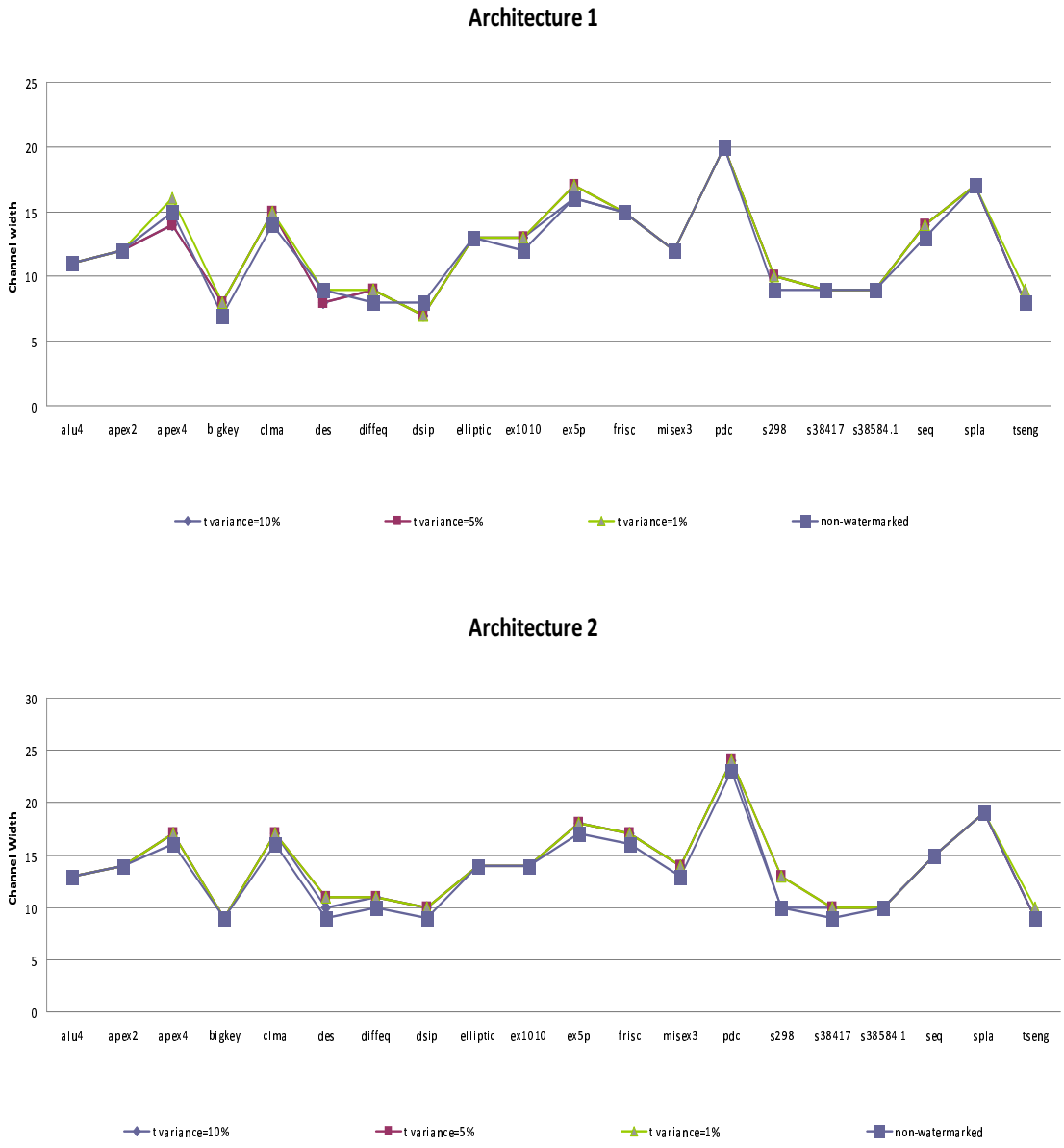


Figure 19: Trade-off analysis between circuit speed and channel width

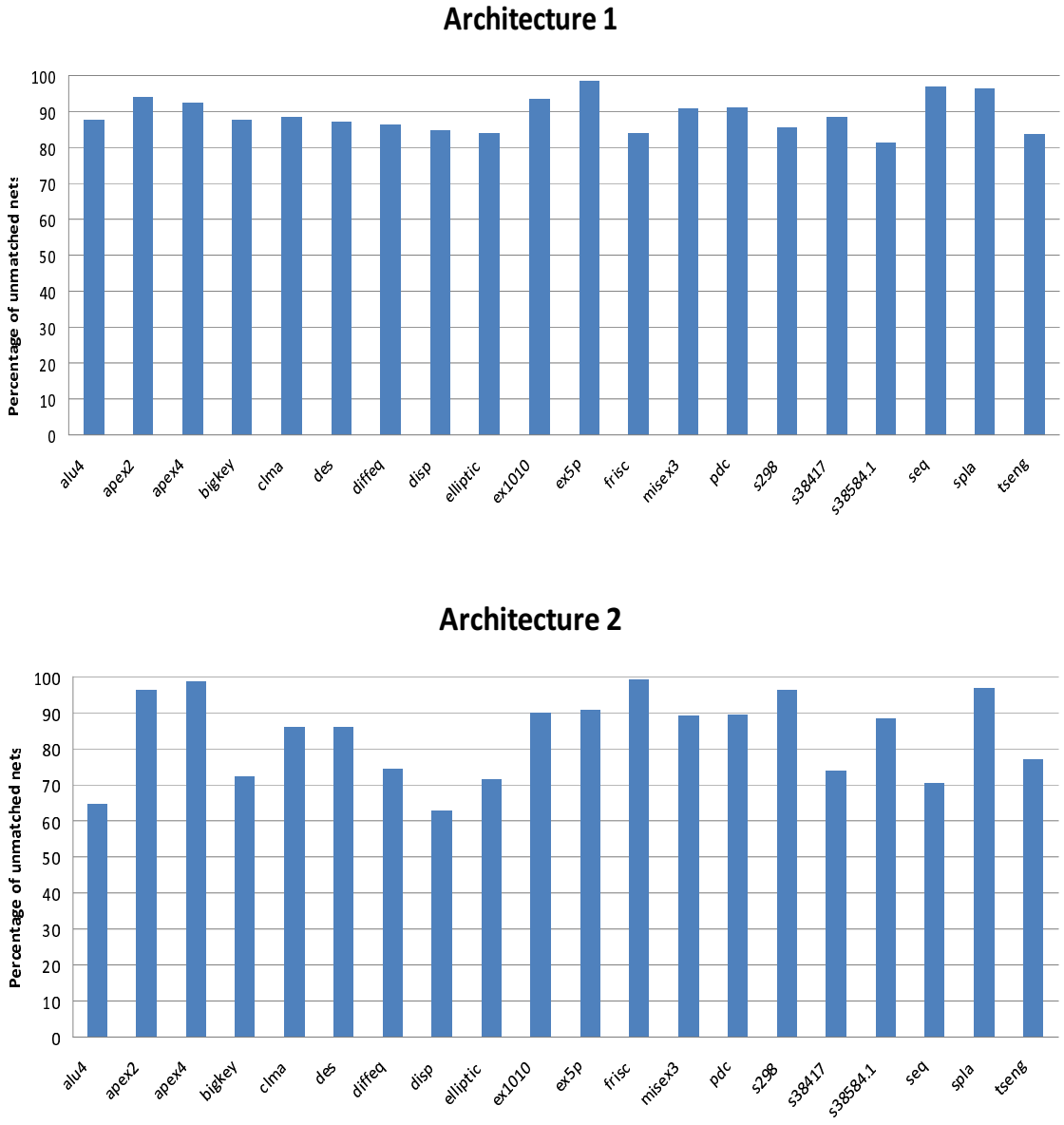


Figure 20: Percentage of unmatched nets between two different watermarks

We specified an upper bound on the critical path delay to study the increase in channel capacity required to offset the cost of watermarking. The delay bounds are specified as a percentage increase over the original design delay. We ran simulations for 10%, 5%, and 1% delay bounds. Figure 19 shows that by increasing the channel width by one, we could meet the timing constraints for all the benchmark circuits.

Finally, to verify the strength of our watermarking technique, we compare routing results to find the nummber of nets that were routed differently under a different signature text. Figure 20 shows the percentage of nets that were routed differently. Approximately, 75% of the nets in a circuit took different paths with different watermark signatures.

The results support our objective of minimizing the impact of watermark insertion on the circuit performance and the area overhead. From Figure 20, we can also conclude that a large number of watermarks can be embedded on a given circuit design.

CHAPTER VII

CONCLUSION

We discuss the problem of digital IP protection for FPGA designs. The time and effort required to design complex circuits is quite high; thus, it makes sense to re-use these complex designs. IP developers can sell their designs and collect revenue. However, it is necessary to protect the IP rights of the IP developer. License agreement, patent, and design encryption helps to protect the IP, however it cannot detect an IP theft. We address this problem by using a watermarking technique that can detect the IP developer of a design. This technique inserts a unique mark by modifying the routing of a design and making it a function of the author's signature. This watermarking method is robust and has a low design overhead.

Our results show that for most of the circuits there is no area overhead due to watermarking on an FPGA design; however, there could be an increase in the critical path delay. Our objective is to minimize the watermarking overhead on any critical circuit parameters. If the circuits were routed with channel width of one greater than the minimum channel width for non-watermarked design, then watermarking has a zero performance overhead.

This watermarking technique might fail for very small designs, because it will not have enough nets to modify. But, the idea of watermarking arises out of the need to exchange complex designs that could be reused. A simple design would be easy to create, and hence it will not have much commercial value.

Inserting a second watermark will be difficult. Firstly, because the same signature is used for watermarking and signature generation, which means that the two processes are inherently tied together. So without reverse engineering the entire design,

it will not be possible to embed a new watermark. Secondly, the FPGA vendors do not disclose the format of configuration files which makes it extremely difficult to reverse engineer the configuration bitstream.

In conclusion, this thesis presents a routing-based watermarking technique to establish proof of authorship for FPGA designs. It discusses in detail the routing algorithm used in VPR and modifications made to incorporate our watermarking scheme. Finally, the critical path delay results for a set of MCNC benchmark circuits shows that a watermarked design has less than 10% speed overhead at the minimum channel, for most of the circuits. The idea of using the switch pattern from the configuration bitstream to capture digital signature of a design is a novel contribution.

REFERENCES

[1] "Virtex-E Field Programmable Gate Array, Product specification," in *Xilinx*.

[2] "Xilinx 4000E and 4000X Series FPGA Product specifications," in *Xilinx*, 1999.

[3] A. B. Kahng, J. Lach, W. H. Mangione-Smith, S. Mantik, and I. L. Markov, "Watermarking Technique for Intellectual Property Protection," in *Proceedings of 35th Design Automation Conference*, 1998.

[4] Adarsh K. Jain, Lin Yuan, Pushkin R. Pari, and Gang Qu, "Zero Overhead Watermarking Technique for FPGA Designs," in *ACM Great Lakes Symposium on VLSI*, 2003.

[5] Amr T. Abdel-Hamid, Sofiene Tahar, and El Mostapha Aboulhamid, "IP Watermarking Techniques: Survey and Comparison," in *Proceedings of 3rd IEEE International Workshop on Systems-on-Chip for Real-time Applications*, 2003.

[6] C. Y. Lee, "An Algorithm for Path Connections and its Applications," in *IRE Transactions on Electronic Computers*, 1961.

[7] Daniel Ziener, Stefen Abmus, and Jurgen Teich, "Identifying FPGA IP-Cores based on lookup table content analysis," in *International Conference on Field Programmable Logic and Applications*, 2006.

[8] E. Dijkstra, "A Note on Two Problems in Connexion with Graphs," in *Numerische Mathematik*, 1959.

[9] Greg Wolfe, Jennifer L. Wong, and Miodrag Potkonjak, "Watermarking Graph Partitioning Solutions," in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2002.

[10] Intellectual Property Protection Development Working Group, "Intellectual Property Protection: Schemes, Alternatives and Discussion," VSI Alliance Group, 2000.

[11] John Lach, William H. Mangione-Smith, and Miodrag Potkonjak, "Signature Hiding Techniques for FPGA Intellectual Property Protection," in *IEEE/ACM International Conference on Computer-Aided Design*, 1998.

[12] John Lach, William H. Mangione-Smith, and Miodrag Potkonjak, "Fingerprinting Techniques for Field-Programmable Gate Array Intellectual Property Protection," in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2001.

[13] Larry McMurchie and Carl Ebeling, "Pathfinder: A Negotiation-Based Performance-Driven Router for FPGAs," in *Proceedings of third International ACM Symposium on Field-Programmable Gate Arrays*, 1995.

[14] Ravi Nair, "A Simple yet effective Technique for Global Wiring," in *IEEE Transactions on Computer-Aided Design*, 1987.

[15] Tingyuan Nie, Tomoo Kisaka, and Masahiko Toyonaga, "A Watermarking System for IP Protection by a Post Layout Incremental Router," in *Proceedings of 42nd Design Automation Conference*, 2005.

[16] Vaughn Betz and Jonathan Rose, "VPR: A New Packing, Placement and Routing Tool for FPGA Research," in *International Workshop on Field Programmable Logic and Applications*, 1997.

[17] Vaughn Betz, Jonathan Rose, and Alexander Marquardt, *Architecture and CAD for Deep-Submicron FPGAs*. Kluwer Academic Publishers, 1999.