IP protection for DSP algorithms' FPGA implementation.

Wei Dai
University of Windsor

IP Protection for DSP Algorithms' FPGA Implementation

by

Wei Dai

A Thesis
Submitted to the Faculty of Graduate Studies and Research
Through the Department of Electrical and Computer Engineering
In Partial Fulfillment of the Requirements for
The Degree of Master of Applied Science at the
University of Windsor

Windsor, Ontario, Canada
August 2004

Your file Votre référence
ISBN: 0-612-96386-1
Our file Notre référence
ISBN: 0-612-96386-1

1008405

Abstract

With today's system-on-chip (SOC) technology, we are able to design larger and more complicated application-specific integrated circuits (ASICs) and field programmable gate array (FPGA) in shorter time period. The key point of the success of SOC technology is the reuse of intellectual property (IP) cores. Consequently the copyright protection for these IP cores becomes the major concern for the development pace of SOC technology.

Watermarking technology has been proved to be an effective way of copyright protection. In this thesis, the author presents two new watermarking algorithms respectively at algorithm level and FPGA layout level. The simulations and implementation results show that the new proposals have much less design and hardware implementation overheads, lower watermark embedding and extraction cost, as well as higher security strength, compared to the previously proposed methods.

Acknowledgements

Here I would like to give my sincere thanks to everyone who has help me for the successful completion of this thesis.

Firstly I would like to thank my co-supervisors, Dr.H.K.Kwan and Dr.H.Wu. Dr.H.K.Kwan has rich experience in DSP algorithm development as well as FPGA implementation. He gives me detail guidance of the research direction. Dr.H.Wu shares his strong experience in data hiding and data encryption during my research procedure. He helps me to select the thesis topic, supervises me through the thesis writing procedure and gives a lot of valuable suggestions.

Thanks also go to the committee members, Dr.K.Tepe and Dr.H.Hu, for their helpful advice to improve this thesis.

Finally, I also thank my friends in VLSI lab for their kind help.

Table of Contents

List Of Figures

List Of Tables

List Of Abbreviations

ASIC	Application-Specific Integrated Circuit
CDMA	Code Division Multiple Access
CLB	Configurable logic block
DCT	Discrete Cosine Transform
DSP	Digital Signal Processing
FIR	Finite Impulse Response
FPGA	Field Programmable Gate Array
LUT	Look-up Table
IP	Intellectual Property
IPP	Intellectual Property Protection
PCB	Printed Circuit Board
RAM	Random Access Memory
ROM	Read Only Memory
SOC	System On Chip
VHDL	Very-high-speed Integrated Circuit Hardware Description Language
VLSI	Very Large Scale Integration

Chapter 1: Introduction

1.1. Research Motivations

In the past decade there has been an explosion in the use and distribution of digital contents such as music, movies, electronic books, software, etc, over Internet. Electronic commerce applications and online services have also grown rapidly. But at the same time, digital products and services providers concern themselves about the unrestricted duplication and dissemination of copyrighted materials [1][2]. For example, the lack of effective protection methods for copyright content was the main reason for the delayed introduction of Digital Video Disks (DVDs) [1]. Two complementary technologies have been developed as countermeasures against copyright violation, one is encryption and the other is watermarking [1].

More recently, watermarking has been shown as an effective way to protect intellectual property (IP) of hardware design. In the past few years, the system-on-chip (SOC) technology has made it possible and efficient to design complicated DSP ASICs and FPGAs. The main reason of the success of the SOC technology is the reuse of IP cells. The copyright protection for the IP cells becomes a vital issue for further development of the SOC technology. Encryption technology can provide copyright protection during the transmission of the IP cells from the vendor to the customer. It is obvious that encryption is not enough for the complete IP protection since there is no protection for the IP cells at the customer's end after the customer receives and decrypts the encrypted IP cells. Watermarking technology has been used to complement encryption by embedding a piece of secret information into the original data to generate the watermarked data. It is assumed that the watermarked data still keep their proper content or function and this secret information cannot be removed from the watermarked data without damaging the proper content/function of the data.

Several watermarking schemes for DSP algorithms and/or their ASIC/FPGA designs have been presented since late 1990's [3][4][5][6][7][8][9][[10][11]. However, many of the watermarking schemes have been shown not secure by recent proposals of attacking schemes [8] and not efficient due to large overheads and performance degradation [3][[9][10][11]. Many IP vendors do not use any watermarking scheme and are still trying to find an efficient enough watermarking scheme to be used for the customer's end. Market leaders of ASIC CAD tool and IP providers like Cadence and Synopsys have built up research teams focusing on ASIC IP's watermarking technology [4][12][13][14][15][16].

Two pieces of research work have been proposed in this thesis. Firstly, we apply the method of modifying the least significant bit (LSB) to FIR filter and some other DSP algorithms. The simulation results show that the watermarked filter/DSP algorithms have very low magnitude distortion. Secondly, a novel watermarking scheme at FPGA layout layer is proposed. It provides IP protection for FPGA design by utilizing the location information of the used FPGA cells. Compared to the previous methods for FPGA, the proposed watermarking scheme has extremely low overheads and high security strength.

1.2. The Organization of Thesis

There are five chapters in this thesis. In Chapter 1, the motivation of developing new DSP algorithms' FPGA watermarking scheme is given. Chapter 2 introduces watermarking technology, watermarking application fields and requirements, current ASIC/FPGA as well as DSP algorithms' watermarking technologies. Chapter 3 presents the filter coefficients' LSB watermarking scheme. Chapter 4 proposes the FPGA cell locations' watermarking scheme. Several FPGA IPs, like FIR filter, comb filter, Code Division Multiple Access (CDMA) match filter, discrete cosine transform (DCT) IPs are used to verify the performance of watermarking scheme. The comparisons between the FPGA cell location modification watermarking scheme and the current available DSP algorithms as well as ASIC/FPGA watermarking schemes are also given in chapter 4. Conclusions are drawn in chapter 5 based on the simulation and analysis presented in the previous chapters. Several possible future work directions are also mentioned.

Chapter2: Review of Watermarking Technology

In this chapter, we introduce the concepts of watermarking and information hiding technologies. Then the application fields and general requirements for watermarking technology are discussed. Finally, a review of the current ASIC/FPGA as well as DSP algorithms' watermarking schemes is given. A few watermarking attack methods are also introduced.

2.1.Watermarking Technology Basics

2.1.1. Current Copyright Protection Technologies

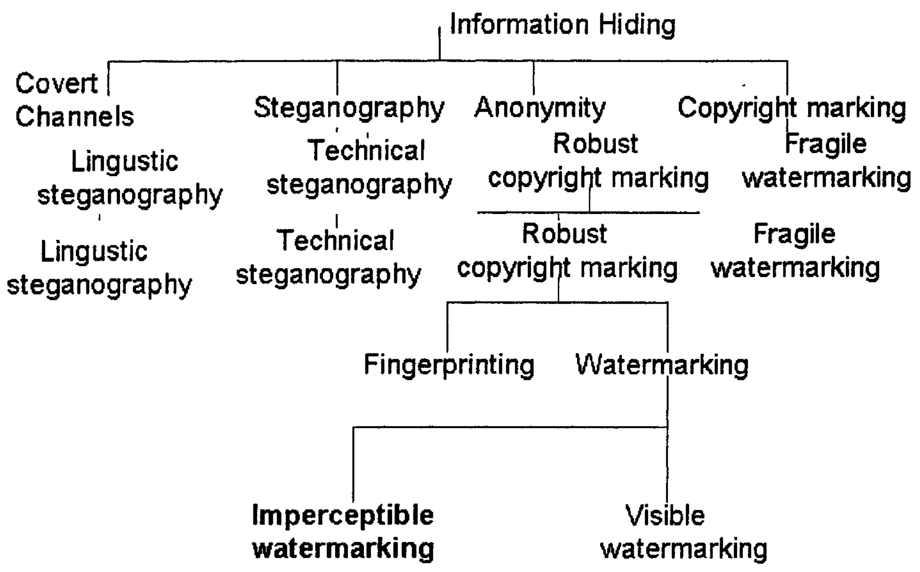
Great effort has been made for copyright protection in the past years. We have seen some legislation act like Digital Millennium Copyright Act, which was effective on October 28,1998, in U.S.A [1]. The European Union is also preparing similar intellectual property rights protection for digital multimedia products [1]. To provide copyright protection for digital data, two complemented methods have been developed: *encryption* and *watermarking*.

Encryption technology can be used to protect digital data during the transmission from the sender to the receiver. After the receiver has obtained the encrypted data and decrypted it, the decrypted data is no longer protected. Unauthorized copies may be made from the decrypted data. For example, after downloading the music files from the Internet, the customer can use the given key to decrypt the music files and make copies by himself, and then gives them to his friends. The encryption technology cannot protect the copyright anymore [1].

Watermarking techniques can be used to complement encryption by embedding secret information, which is imperceptible to the receiver, called *watermark*, into the original data. *Watermark* is a piece of information embedded within other objects that shows the identity of copyright owner. Watermarks tell us who is the owner of the IP for the object [1][2]. Watermarking is an old technology that has been used for centuries. This technology was firstly developed to protect valuable paper documentations, like bank notes, commercial contracts as well as cash. In this thesis, the discussion of watermarking technology will be limited to the digital watermarking field. For example, if a customer violates copyright protection by making illegal copies of a watermarked image and freely distributing them among his friends, then the original copyright owner can suit this customer and show the copyright information embedded in the image as the watermark in the court [1][2].

2.1.2. Watermarking as a Sub-field of Information Hiding

Figure 1 Classification of Information Hiding Technology [2]



Information hiding includes covert channels, steganography, anonymity and copyright marking, see Figure 1. *Covert Channels* means the hidden secret channels used by two parties for communication. *Anonymity* is the technology that investigates the hiding of information sender's true name. *Steganography* means hiding information in other information. This is the technology to conceal the existence of information within other data [2]. *Steganography* is different from *Cryptography*. *Cryptography* is the technology to protect the content of messages. People use encryption algorithms to encrypt data and use decryption algorithms to decrypt the encrypted data [2].

We can see there are two subsets under copyright marking, one is robust copyright marking and the other is fragile watermarking. *Robust copyright watermark* is the watermark that is infeasible to be removed or modified without destroying the object at the same time [2].

Fragile watermark is the watermark that will be destroyed as soon as the object is modified too much [2]. We can see that the watermarking technology belongs to robust copyright marking. *Watermarking* is the technology or procedure to embed a digital watermark into a digital IP and detect/extract it from the digital IP that has been watermarked [2].

Fingerprinting technology also belongs to robust copyright marking. *Fingerprint* is the serial number information hidden in the original data. *Fingerprinting* is the technology or procedure to embed fingerprint into the original digital IP and detect/extract it from the digital IP that has been fingerprinted [2]. Fingerprints can enable the intellectual property owners to identify which customer broke his license agreement and supplied the property to unauthorized third parties [2].

This thesis focuses on the imperceptible watermarking method since imperceptible watermark will arouse less suspicion and attacks. Until recently, information hiding technology received much less attention from the research community and industry than cryptography (encryption belongs to cryptography). But this condition has changed. The first academic conference on the watermarking subject was organized in 1996 [2]. The main driving strength is concern over copyright protection. Law enforcement and counter intelligence agencies are also very interested in understanding these technologies so as to detect and trace hidden copyright and serial number information [2]. In this area rapid strides are being made constantly but general theories are still very tentative.

Copyright marking, as opposed to Steganography, has some additional requirement of robustness against possible attacks. Copyright marks can be either visible or invisible, depending on the specific application case. But most of the literature has focused on invisible (or transparent) digital watermarks that have wider applications [2]. Watermarking in paper is a very old anti-counterfeiting technology; more recent innovations include special ultraviolet (UV) fluorescent inks used in printing traveler's cheques. In this paper, the discussion will focus on digital watermarking techniques.

2.1.3. Watermarking Application Fields

The watermarking technology can be used for the following purposes:

Copyright Protection: The owner can embed a watermark with copyright information in the original data. Under the condition that there is any copyright argument or copyright violation, the original owner can show the copyright information within the watermark to the judge at court [1][2].

Fingerprinting: The copyright owner or IP vender also can embed different watermarks called fingerprints in the copies submitted to different customers. So they can trace the source of illegal copies [1][3][5]. For example, the music publisher can embed different serial number to the CDs that will be sold by different retail agents in different countries. If the music publisher finds some illegal copies of the CD products, the publisher can trace the source of the illegal copy by extracting the watermark.

Copy Protection: The watermark information can stop the copy procedure during unauthorized copy. For example, for some software, there are copy protection codes embedded. When the customer try to make illegal copies, these copy protection bits will make the computer to stop execute the copy instructions [1].

Authentication: Fragile watermarks (watermarks that will be modified or removed without difficulty with the modification of the original data) can be used to check the authenticity of the data. The author of the electronic books can embed large amount of fragile watermarks into the whole electronic book. So if some of the watermarks have been removed, that means some sections of the books have been alternated [1].

2.1.4. Watermarking Embedding and Extraction Schemes

The watermarking procedure is to add watermark that contains copyright information into the original data by using watermark embedding algorithms. The embedded data are the message that one wishes to send secretly. Sometimes secret or public keys are also used to encrypt the original copyright information. The key is used to control the hiding processing so as to restrict detection and/or recovery of the embedded data to parties who know it [2]. After applying the watermarking algorithm we will have the marked data with watermark inside.

Figure 2 Watermark Embedding scheme [2]

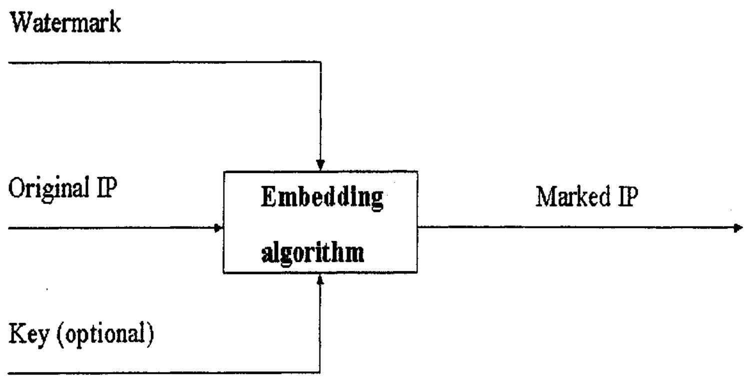
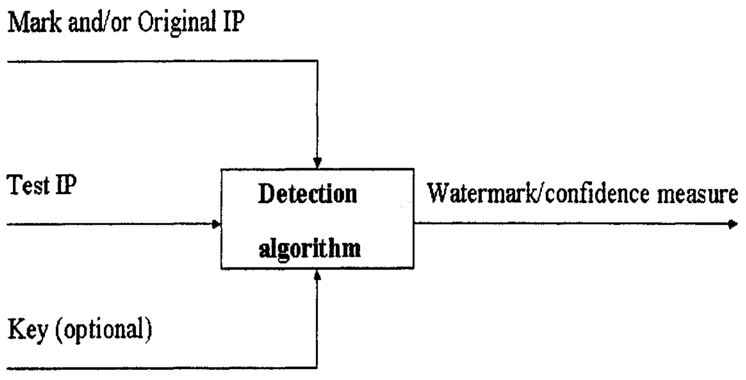


Figure 3 Watermark Extraction Scheme [2]



In Figure 3, the watermark inside the IP will be extracted. Let us use the digital photo image as an example. The watermark extraction procedure is to use the watermark and the original image, as well as the image that will be tested. After comparing the original image and the image under test, we can find the watermark inside the tested image, which is the copyright confidence. Other watermark extraction methods are similar to this scheme.

2.1.5.General Watermarking Requirements

A few general requirements for a watermarking scheme have been proposed [1][2][3][4]:

Perceptual Transparency: A watermark is imperceptible if humans cannot distinguish the original data and the data with watermark. Since by comparing the original data and the watermarked data, people can easily find the difference, that is the watermark, we should assume that the customer does not have access to the original data [1].

Payload of the Watermark: This is the amount of the information that can be stored in a watermark. For specific data object, if we can embed more information into the original data, we say this watermark-embedding scheme has higher payload. A good watermark-embedding scheme should provide high payload [1].

Robustness: This refers to the difficulty to remove/modify the watermark without degrading or changing the original data. To remove the watermark, the potential attacker should firstly try to find the watermark. If the watermark is very hard to find, very hard to forgery and very hard to remove, we say that this kind of watermarking scheme has good robustness [1][2][3]. To remove or modify the watermark must require the knowledge of a secret, like the secret key value, the watermarking procedure and watermarking algorithm details. The watermark should survive all attacks that do not degrade the IP's perceived quality. Some attacks include re-sampling, re-quantization, dithering, compression and the combination of them.

Security: The security of watermarking scheme means the watermarking scheme's ability against watermark detection/forgery/modification. We should assume that the method to encrypt and watermark the data is known to the public and the security must lie in the choice of a key [1].

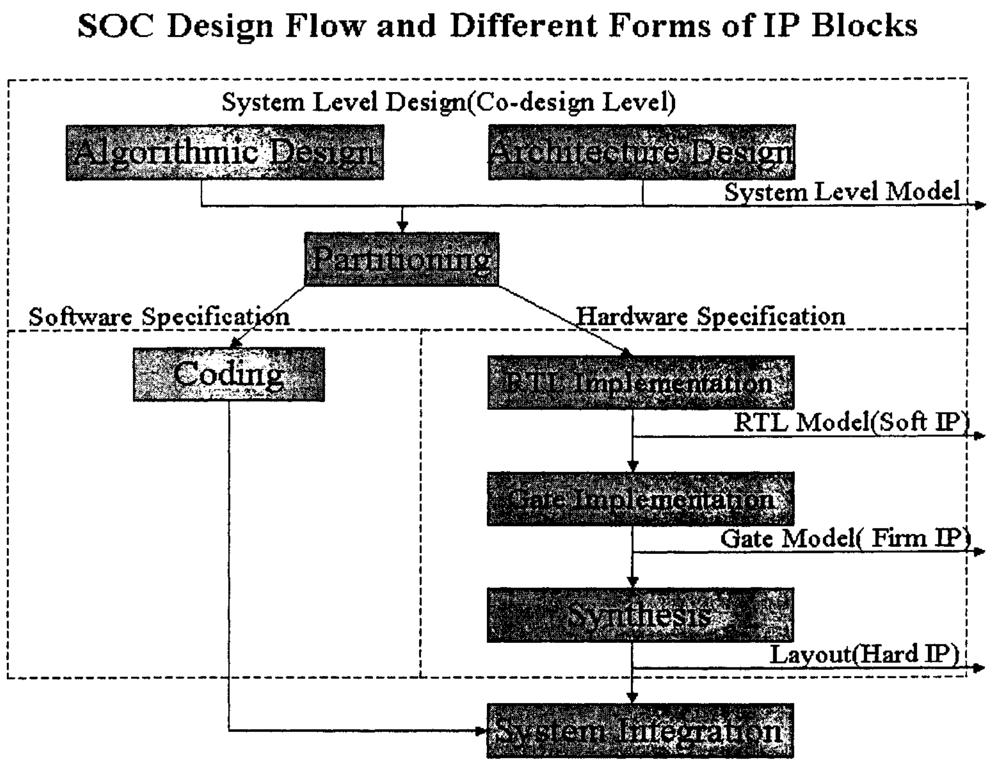
Independency: If multiple marks are inserted in a single object, then they should not interfere each other.

Most watermarking applications have a sharp tradeoff between robustness and watermark embedding efficiency. This make a single watermarking scheme meeting the requirements of all applications to be difficult [2]. At the other side, most real case applications do not require all of the properties for general watermarking schemes.

2.2.ASIC and FPGA Watermarking Technology

2.2.1. Where We Need to Add Watermark

Figure 4 SOC Design Flow and Different Forms of IP Blocks [17]



A watermark can be added at different levels such as algorithm level, architecture level, register level, layout level, etc. At a lower level, payload is usually higher. A watermark can be added at multiple levels to make an attack even harder.

Figure 4 shows the current SOC design flow and different types of IP blocks that need to be protected. For each level, there should be different copyright protection schemes. For watermarking technology, the watermarking schemes of these IPs locating at different levels are also different. The watermark embedded at the higher level will be carried to the lower levels. For example, if we add watermark to the RTL model (soft IP), the watermark will be carried to the lower levels, like gate level and layout level. This is the reason we prefer to add watermark to the design level as high as possible. The reason is if any attacker wants to remove the watermark and get the high level design, he needs to remove the watermark embedded at each lower level, if the attacker can only access the lowest level of IPs.

The watermark embedded at higher level is often easy to be identified and removed, if the IP is provided with high-level form modules. Human beings normally will feel much easier to understand high-level design data. For example, the attacker will feel more comfortable to understand the structure of the software program written in C or Java, comparing to the layout data of all binary format (FPGA bit-stream programming data or ASIC layout data).

2.2.2. The Importance of ASIC/FPGA IP Watermarking

The fast development of system-on-chip technology increases the importance of IP reuse. Now we can integrate the complete system on a single silicon chip that has all the functions of those chips on a printed board produced several years ago. Reusable virtual components or IP blocks are most effective for the purpose of reducing design cycle time, as well as decreasing design risk. IP owners want to make sure that their IP products will not be illegally redistributed. Consumers also want to make sure that the IPs they buy are legitimate. FPGAs become good candidates for fast-to-market products like DSP cores for wireless communication and computer network applications. Distribution of the IP cores through the Internet is the commercial fact but it also increases the risk of IPs being stolen and forgery [17]. The IP provider can use some technologies like JAVA applets and JHDL to protect their IPs during the evaluation of the IPs by potential customers. Based on the JHDL design tool, these JAVA applets allow structure viewing, circuit simulation and netlist generation for some cases. Applets can be customized to provide varying levels of visibility and functionality [18].

Watermark technology is also needed for AISC/FPGA co-design and co-verification, which means from the right beginning there should be considerations for the integration of different types of design blocks (including software codes, digital blocks, analog blocks, as well as IPs with the form of Very-high-speed Integrated Circuit Hardware Description Language (VHDL)/Verilog source code, black box schematic block or layout hard blocks) [17][19]. There should be copyright protection schemes for all these kinds of design IPs.

IP vendors can protect their copyright by encrypting their source codes (VHDL/Verilog) when these codes are sent to the buyers. These modules can be loaded into authorized simulators or synthesis tools without making the source code visible to the system designer who uses the IP blocks. In this case, the CAD tool maintains the safety of the copyright. But in practice, this protection is often broken with attacking the CAD tool (simulators, synthesis tool) directly. We need to find extra methods to protect the design IPs. Watermarking technology has shown its ability for ASIC/FPGA IP protection.

The methods of watermarking for digital images, music and video are quite different from the methods to protect DSP algorithms and ASIC/FPGA designs. The research about the DSP algorithm and ASIC/FPGA watermarking just began several years ago and still is a relatively new area. But we can see this area begin attract more researchers and some market leader ASIC/FPGA design software companies like Cadence that also begins to support such kind of research activities.

2.2.3. ASIC/FPGA Watermarking Evaluation Criteria

Watermarking is a process that hides or embeds data into a design IP that can help deter theft and counterfeiting. The watermark serves as evidence of ownership. A complete ASIC/FPGA IP protection scheme based on watermarking consists of two phases: *watermark synthesis* and *watermark detection* [4][12].

The *synthesis* phase is fully characterized by [4]:

(a) Algorithms translating design features onto a unique watermark

(b) Tr: the worst case time required to forge and/delete the watermark

(c) Pu: the odds that a design carries an unintended watermark in part or in its totality.

The *detection* phase is characterized by [4]:

(d) Pm: the probability of a detection miss

(e) Pf=Pu: the probability of a false alarm.

Typical specification of a complete IP protection scheme could be [4]:

Tr	>= 2 years,
Pu=Pf	<= 1E-30
Pm	<= 1E-6

To evaluate certain ASIC/FPGA and DSP algorithms' watermarking scheme, some major factors need to be considered. The following factors have combined the requirements presented in [4][12] as well as the general requirements for watermarking schemes [1][2]. These evaluation criteria will be used in the following chapters.

Embedding Efficiency: This item shows the watermarking schemes' efficiency. For example for the same filter, watermarking scheme A can embed maximum 8-bit information and watermarking scheme B can embed 16-bit information, we say watermarking scheme B has better embedding efficiency.

Embedding Cost: This item reflects the effect of the watermarking procedure on the original ASIC/FPGA or DSP algorithm. If the watermarking scheme introduces comparatively small timing delay increase, hardware cost increase, algorithm complexity and/or software computational time's increase, we say this scheme has low embedding cost.

Design Overhead: This item shows the extra time for the design with watermarking steps. If the watermarking steps will add small extra design time, we say this watermarking scheme has low design overhead.

Extraction Cost: This one is related to the difficulty, complicity, as well as the time of watermark extraction. If the extraction procedure needs less effort, for example, does not need complicate test equipments, people having strong electronics background or extra software, we will say this scheme's watermark extraction cost is low.

Extraction Miss Probability: This is the probability that a watermark cannot be extracted. This item equals to Pm.

Probability of Coincidence: This item shows the probability that one watermarked IP will carry the whole or partial watermark as another IP that has not been watermarked. The smaller this number, the better. Normally this number should be less than 1E-30 for copyright protection applications. This item equals to Pu and Pf.

Security Strength: This item shows the watermarking scheme's ability against potential attacker's attempt of detecting, counterfeiting and remove of the watermark. The higher the ability, the better the security strength. Also the more transparent the watermark, the better the security strength. The value of Tr is one of the indications of this specification.

Applied Area: This item is related to the application fields of the watermarking scheme. Some watermark schemes can be used for ASICs' IP protection, others are for FPGAs only. Some are for digital designs and others are for analog designs. The wider the applied area, the better the watermarking schemes.

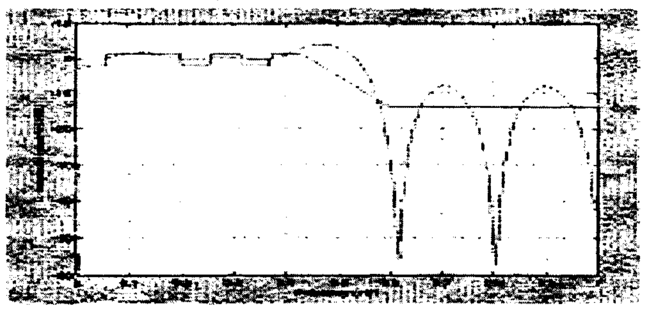
2.3. Filter Watermarking Schemes

The following three filter watermarking schemes have been proposed: filter magnitude modification, filter tap's equal-replacement and windowing function watermarking. In this section, the watermarking procedures and the major properties of these three schemes are mentioned.

2.3.1. Filter Magnitude Modification [9]:

Firstly, the designer separates the filter stop or pass-band to several equal width zones (i.e., seven zones). Then he modifies the filter magnitude response according to the watermark bits. If the bit is 1, he will decrease the filter magnitude response by x dB (here x is a small number like 0.1). If the bit is 0, he will increase the filter magnitude response by x dB. After that, he uses the modified filter magnitude response as the design constrains input to the design tool. Finally he obtains the filter coefficients. Now the filter's magnitude response decided by the new coefficients will contain the watermark information that is 7-bit as 0110101.

Figure 5 The Magnitude Response of the Filter with 7-bit Watermark [9]:



Watermarking performance analysis for filter magnitude modification scheme:
The embedding efficiency is less than 8 bit/31taps=0.258 bit/tap. The authors do not give out clear number of the maxim number of watermark bits, which can be hidden into the filter's magnitude response. But the authors do mention that for the hiding of 7-bit watermark, the 31-tap filter is not long enough in terms of filter tap number. The filter with 41-taps can be used to embed such 7-bit watermarks. Since the authors divide the filter's pass band's magnitude response and modify the magnitude response, the filter's order may increase. But the authors of [9] do not mention other simulation results with more watermarking bits, the author of this thesis can not exactly evaluate the relation between the watermark length and the filter's order, as well as the hardware cost increase. There is 7% hardware increase for the embedding of 7-bit watermark at algorithm level only.

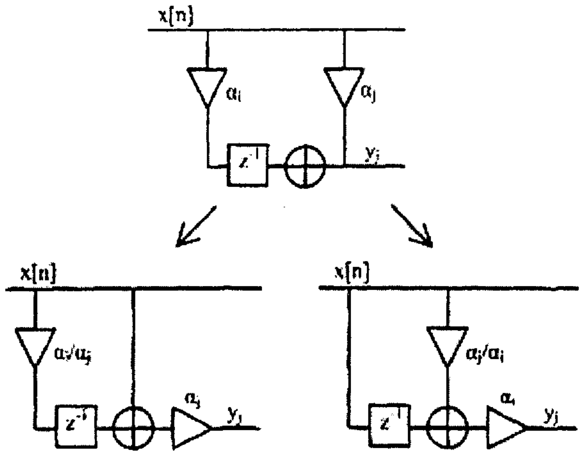
From [9], the designer needs to re-design the filter for embedding the watermark. So the design overhead is high. The authors of this paper do not mention the watermark extraction flow. It seems that we can plot out the filter's magnitude response and check the magnitude response, then find the watermark bits related to the magnitude ripples. And if the original filter already has magnitude ripples in pass-band, it will be more difficult to extract the watermark from its magnitude response.

The authors of paper [9] do not provide any information about the possible attack mode analysis and security strength analysis. The filter performance degradation is related to the filter magnitude change value. If the filter magnitude change decreases, the filter performance degradation will decrease, but the filter taps number will increase. This scheme adds watermark at algorithm level, the highest level for filter design, so it is hard to be removed from lower level of filter's implementation, like logic, layout or circuit level. This watermarking scheme increases the complexity of filter design. It also increases the order of the filer, since we introduce new ripple constrains to the filter magnitude response, As a result, the hardware cost increases by +7%.

2.3.2. Filter Tap's Equal Function Replacement Method [9]

The designer firstly designs the filter with the original performance specification then he replaces the filter taps by using equal filter structure replacement. There are three equal function filter structures, A, B and C. When the watermarking bit is 0, the designer will use B to implement this tap. When the watermarking bit is 1, the designer will use C to implement this tap. When there is no watermark to be embedded, the designer will keep on using A.

Figure 6 Filter Equal Function Replacement [9]



Watermarking performance analysis:

Since the author can use one tap to embed one bit of watermark information, the embedding efficiency is 1 bit/tap. After watermarking, the hardware area increases by 29%, which is primarily due to a higher internal word length. The higher word length is the result of dividing coefficients introduced by equal filter tap function replacements. The increase of hardware area will also increase the power consumption of the filter, since we increase the amount of operations introduced by dividing operations. The design of the filter with watermark needs to replace the standard filter taps with equal functional tap module. This introduces the non-regularity for the filter's hardware design and will increase the time and effort of VHDL coding.

The authors of the paper do not mention the details of how to extract the watermark. The extraction of the watermark needs the extractor to review the filter's implementation layout or logic circuit design or VHDL/Verilog source code, extract the logic functions of the filter and extract the watermark according to the equal function replacement watermarking rules. This will require considerable time and effort. The probability of coincidence of this kind will be zero if only the author of that paper uses this watermarking method. Since the tap structure of the filters will be all the same if the filter is designed in traditional way.

The authors do not mention the security strength of this method. Layout reverse engineering may be able to remove this kind of watermark, as long as the layout processing technology is visible by the attacker (normally 0.5um or wider metal wire width). This method can be applied to both ASIC and FPGA design. There is no filter response performance degradation, since the authors use equal functional module to do the replacement.

This scheme makes watermarking at algorithm level, so it is hard to remove at the lower levels and there is no degradation of filter response performance. But this method increases hardware cost dramatically by +29. It also makes the filter structure to be not uniform and increases the design time for implementation on ASIC or FPGA.

2.3.3. Windowing Function Watermarking [10][11]

Suppose $W(n)$ is the original windowing function, where $1 <= n <= N$.

Firstly the designer adds random noise to $W(n)$ to obtain $Wm(n)$:

$Wm(n)=W(n)+a * r(n), 1<=n<=N$. Here a is a small number like 0.0001.Then the designer adds watermarking bits c (n) to Wm (n):

$$Wc(n) = \begin{cases} Wm(n), & 1 <= n <= i-1 \\ Wm(n)+b*c(n-i+1), & i <= n <= i+P-1 \\ Wm(n), & i+P <= n <= N/2 \\ Wc(N+1-n), & N/2+1 <= n <= N. \end{cases}$$

The designer will let b=0.0001. The starting bit of the P-bit watermark code sequence c (n-i+1) is bit i of Wm (n). The sequence Wc (n), n=1,N, is the new window function which contains the watermark information.

Watermarking Performance Analysis:

For a classical windowing function with length n, we can embed n bits of watermark information, so the embedding efficiency is 1 bit/tap. Since the embedding cost is just to add random noise sequence as well as watermark bits to the original windowing function, the embedding procedure is simple and fast at software computational level. The embedding cost is low. There is no design overhead for this method. Since the watermark is added to the windowing function after it has been designed.

The authors do not mention the details of watermark extraction flow. But we can see that the watermark can be extracted by comparing (subtraction operation) the watermarked windowing function with the original windowing function. Then by subtracting the random noise sequence, we can extract the watermark. To do this is simple and fast at software level. The extraction cost is low. The length of the watermark decides the probability of coincidence. For the watermark with length n, it equals to 1/(2^n).

The authors do not provide detail information about the application fields. But if this method is only carried at algorithm level, the attacker may be able to add extra random noise to the windowing function and remove the watermark. So the security strength is low. Since this method is developed specifically for windowing function, it can be applied to windowing function or similar structure functions. The authors do not mention this method can be applied to other DSP algorithms' ASIC/FPGA implementation or not.

There will be some degree of performance degradation, which is caused by the modification of the windowing function. The degradation is decided by the comparative ratio of the windowing function's modification value and the original function value. This method embeds watermark at algorithm level that is hard to remove at a lower levels. Its watermarking scheme is simple and direct. But this method increases design complexity and the security strength is low.

2.4. FPGA Watermarking Schemes

In this section, two kinds of FPGA watermarking schemes are described. The first one uses FPGA spare LUTs to embed watermarking information and the second one is to manipulate the FPGA bit-stream programming data directly.

2.4.1.FPGA Fingerprinting by Using Spare LUTs (look-up-table)

This method for watermarking FPGA uses the unused lookup table (LUT) bits to encode the signature bits. In order to further hide the signature, the constrained configurable logic blocks (CLBs) are incorporated into the design with unused interconnections and neighboring CLB inputs. Each unused LUT bit can be used to encode one bit of the mark. The inputs of the marked CLB are taken from the passing signals in adjacent routing channels and outputs are routed to neighboring don't care inputs. Upon inspection, it is not apparent which CLB has been marked [5][6][20][21].

Watermarking Performance Analysis:

From the embedding method, every LUT (look-up-table) can be used to embed 16 bits of watermark [21], so the embedding efficiency is 16 bit/LUT. When the FPGA device has two LUTs in one FPGA slice, the embedding efficiency equals to 32 bit/slice. The embedding cost for this method is not too large. For embedding 7 bits of watermark, the hardware usage increase is less than 1%. [5] The author does not mention the timing delay change before and after watermarking.

To use this method, the designer needs to assign the locations of the unused LUTs, then place and route the original design around these watermarked unused LUTs. The authors of [5] provide the extraction procedure information. The watermark extractor needs to find the locations of the watermarked LUTs and read out the contents of these LUTs. The length of the watermark decides the probability of coincidence. If the length of the embedded watermark is m, the probability of coincidence is 1/(2^m).

The authors of paper[5] do not give out the analysis of possible attack methods and the security strength of this method. Another paper [8] points out that one attack method could be used to successfully detect the watermark embedded in this way. Since this watermarking method uses unused LUTs to embed the watermark, there will be no logic change connections between these LUTs and the original logic design. So the attacker may be able to scan the logic changes happening inside the FPGA and separate the unused LUTs whose output has no effected on the logic changes of other parts. Since there is available attach scheme to break this kind of watermarking, new FPGA watermarking method need to be developed.

2.4.2. FPGA Programming Bit-stream Data Watermarking [3]

This FPGA watermarking method involves substituting watermark bits for some of the bits in the configuration bit stream that control multiplexers for the unused CLB outputs. This method involves the modification of the FPGA bit stream programming data. But in practice, this kind of knowledge is not open to the FPGA design engineers. So the application area of this scheme is limited.

2.5. ASIC Watermarking Schemes

In this section, several ASIC watermarking methods are mentioned. Compared with FPGA watermarking technology, there are more papers about ASIC watermarking methods. ASIC watermarking methods at different design stages like logic synthesis, gate level and layout level have been proposed.

2.5.1. Finger-marking by Transistor Finger W/L (width/length) Watermarking

In paper [3], the author presents a method called finger marking. It can be applied for integrated circuit design at the physical (layout) design level. The watermark is embedded in the transistor layout, making this method applicable to digital, analog and mixed-signal SOC (system-on-chip) designs [3]. The transistors in the circuit design are uniquely ordered based on their connectivity. The random bit-stream is then embedded into the transistor geometries to serve as an indelible mark in the circuit. If the value of the bit-stream bit is 0, the designer then uses even number of transistor fingers. For example: for a N-transistor with design specification of W/L=300um/5um, he uses 2 parallel

150um/5um fingers. If the value of the bit-stream bit is 1, the designer uses odd number of transistor fingers. For example, for a N-transistor with design specification of W/L=300um/5um, he uses 3 parallel 100um/5um fingers.

This method can be applied to analog circuit design and easy to be understood by the designer. But this watermarking scheme increases the design effort to implement the layout. It is easy to be removed by layout reverse engineering. The reason is that for analogy design, the layout processing technology is much bigger than pure digital designs. It makes it easier for the attacker to observe the layout by optical devices(micorscope, for example). We can extract the circuit diagram from the layout then re-arrange the finger configurations to remove the watermarking information. But this kind of reverse engineering need at nearly the same time to re-design the layout. So this method still has its value.

2.5.2. Hierarchical Watermarking

Charbon [13][14][15] presents a hierarchical watermarking technique for ASIC designs. It involves the unique mapping of design topological information onto a sequence of symbols called a topological signature. His method has limited applicability in designs that use more than just standard cells and lacks robustness analysis[4][12]. Charbon and Torunoglu also present the watermarking technology of sequential functions [22]. This method will impose a digital watermark on the state transition graph of the synchronous circuits [22][23].

2.5.3. Protocols for IP Protection

Protocols for IP protection have also been developed for hiding data at the combinational logic's behavioral synthesis level [16]. This approach involves embedding a watermark as a set of design constrains. The synthesis tool results in solution satisfying both the original design specifications and the additional set of constrains. Constraint-based techniques have been proposed for watermarking at different stages of the VLSI (very large scale integration) design process. This method can be used well for pure-digital layouts, but need to be complemented with another scheme when applied for SOC (for example, digital-analog-mixed-signal) design. Watermarking protocols added at ASIC layout place and route stage has also been developed [7].

2.6. Attack Methods Targeting IP Protection Routines with Watermarking

Several attack methods have been mentioned by different authors [8][17].

An attack method called logic redundancy detection uses logic scan method to scans the logic transitions happening inside the ASIC or FPGA and try to find the redundancy in the design. Then the attacker may be able to locate and remove the redundant logics and the watermark that is implemented by using these redundant logics [8]. Another attack method is finite state machine reduction. This method is trying to reduce the unused transitions in the state machine transitions that have been used to embed watermark [17].

The hardware remove method will physically remove the FPGA and ASIC chip that contains watermark. No existing watermarking scheme can avoid this kind of attack. But this method also makes the attacker away from the usage of the IP. So the copyright protection purpose also has been realized [17].

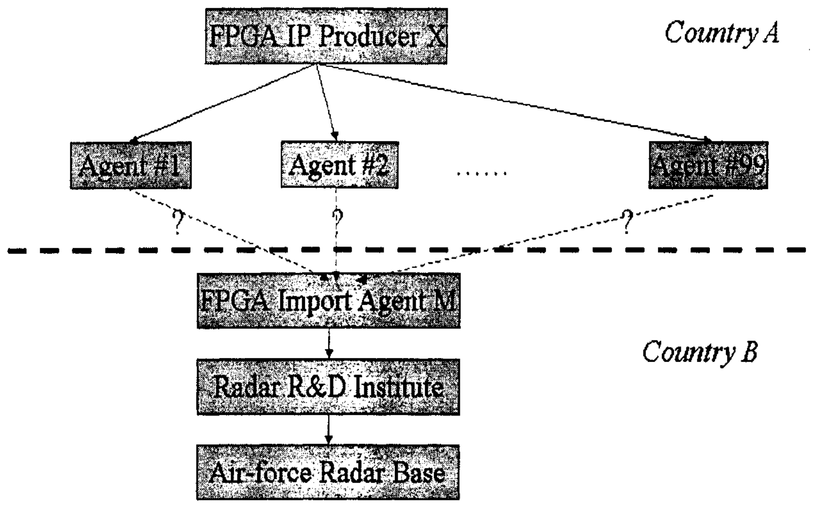
Other attack methods include encryption system attacks. This one is the attack of the DES and AES cryptographic algorithms. This method is a big challenge of the security of many watermarking schemes. The designer needs to keep an eye on the most advanced technology of encryption. And the attack for ASIC logic synthesis level watermarking tries to extract the logic synthesis constrains that contain watermark information. ASIC layout level reverse engineering is to recover the original circuit design by photographing the layout, then carry human or computer vision detection for layout to logic extraction.

2.7. Importance of Developing New Watermarking Schemes for DSP Algorithms' FPGA Implementation

The author selects the DSP algorithm's FPGA implementation's watermarking as the topic of the thesis. The reason is the DSP algorithm's FPGA implementations are widely used for the consumer electronics as well as telecommunication applications. And the protection of this kind of IPs is important. To show the importance of the FPGA watermarking technology, two real scenarios are used to show its application for FPGA based IPs' fingerprinting and watermarking.

Scenario One: Who Have Sold the FPGA Chips to Our Enemy?

Figure 7 Who Have Sold the FPGAs to Our Enemy?



In this scenario, FPGA IP producer X in country A produces FPGA chips and develops high performance DSP IP soft cores. This company also sells FGPAs with DSP IP cores implemented inside. Company X sells their products through 99 retail agents. The customer officers of country A find some mail packages that contained company X's FPGA products. These mail packages are mailed to the electronics retail agent M in country B. But the names and addresses of the sender are false. FBI officers do not find any valuable human being's fingerprints of the sender, except those fingerprints of the post officers'. So the question here is: How the FBI officer can identify which sales agent has involved into the illegal export of the FPGAs? Is there any technology that can make this task much easier?

FPGA watermarking technology is one of the good candidates that can be used to fingerprint the FPGAs. For this kind of applications, the watermark is used as fingerprints to identify different objects. For our example, there are 99 retail agents and we need at lest 99 different fingerprints to identify all the sales agents. So we need 8-bit watermark length, since 2's power 8 is 128. The idea is each sales agent will receive different watermarks that have been embedded into the FPGAs containing DSP IP cores. Although the functions of these FPGAs are the exactly the same, the watermark is different. And all the watermarks have been documented by company X. In this way, the FBI officer will feel easy to trace the source of the illegal exported FPGAs, by contacting company X and extracting the watermark inside the FPGAs. In chapter 4, the author's implementation results of such kind of 8-bit watermark into some real world DSP IPs are presented. The simulations are used to verify the effect of watermark embedding. The simulation results are encouraging.

Scenario Two: Is Mr. Z the DSP IP Thief?

Mr. Z is the technician who works for company X that produces FPGAs and DSP IP cores. Mr. Z's job is to program the FPGA chips with the programming data that is stored in the PC. Mr. Z has ability to read and copy the data. Some sales persons of company X report that another company, company Y, also sales the similar products. So the managers of company X want to know weather Mr. Z has sold the FPGA programming data to company Y. If it is true, how can company X suite Mr. Z, as well as company Y in the court? What kind of evidence will help company X to win the case? Insiders make many high-tech IP stealing cases; the law procedures for these cases are normally long and complicated, partly due to the high-tech background. This makes the suite process time and money consuming.

So is there any technology that will simplify the situation and help to protect the right of the original IP copyright owner? As mentioned in chapter one, watermarking technology can also be used for copyright information embedding. The watermark length of this kind of application will sometime be longer than the watermark's length for fingerprinting usages. For this case, we need 128-bit watermark, to provide strong enough witness for copyright provident. In chapter 4, the simulation results of embedding this kind of 128-bit watermark are presented.

From the review of the current available DSP algorithms' and FPGA implementation watermarking schemes, the area of watermarking for DSP algorithms is relatively new. Although some people have done a lot of innovative work in this area, the proposals they have mentioned still have space for improvement, and the application fields of those watermarking schemes need to be extended. DSP algorithms' FPGA implementation is today's hot-point in the consumer electronics market. FPGAs can provide hardware based fully parallel data processing ability, which makes the FPGAs good candidates for ultra-high speed and high volume throughout DSP applications, like wireless applications, wide band network, etc.

The filter magnitude modification scheme will introduce 7% hardware increase. The filter taps' equal replacement also have 29% hardware increase after watermarking. And the windowing function' watermarking scheme may be broken by adding false watermark at algorithm level. The current available FPGA watermarking methods also have some limitations. The look-up-table (LUT) watermarking scheme has been broken and the FPGA bit stream modification scheme is not fit for those engineers who do not have knowledge of the format of FPGA bit-stream data. The FPGA producers also announce that they will not expose such bit-stream format information to the public, for confidential reason and IP protection considerations. So to develop new and better watermarking schemes that can protect DSP algorithms' FPGA implementation becomes important.

In chapter 3, the FIR filter coefficients' modification watermarking scheme will be presented. The idea to modify the original image bitmap data and windowing function coefficients has been proposed by several papers [1][10][11]. The author of this thesis extends the application area of this idea to filter and other DSP algorithms. By embedding the coefficients into ASIC and FPGA hardware level, the security strength of watermarking scheme has been increased. In chapter 4, the watermarking through FPGA cell locations' constrain is introduced. This is an innovative watermarking scheme that works at FPGA layout level. This scheme uses the unique layout structure of FPGA to embed watermark information into the FPGA cells' layout coordinates.

Chapter 3: FIR Filter Coefficient Modification Watermarking

3.1. Watermarking Scheme

For the filter coefficients' LSB watermarking scheme, a designer who is responsible for the watermark embedding firstly needs to have the filter's coefficients' file. The designer will use the watermark bits to replace the filter coefficients' LSBs. After embedding of the watermark, the designer needs to run simulation of the filter with the watermarked coefficients. If the simulation results are acceptable, the designer will save the filter's coefficients to a new file and the watermarking flow ends here. For the watermark extraction, the designer needs to read out the filter's coefficients, find the watermark locations and extract their LSBs. After that, he can obtain the watermark bit sequence.

Let a FIR filter design be given by:

Y(k)=A0*x(k)+A1*x(k-1)+......+An-1*x(k-N-1), where k=0,1,......,N-1.

Here A0, A1,......, An-1 are filter coefficients. A watermark, for instance, is 10001010. Watermark embedding process is to replace the filter coefficients' LSBs with the watermark bits. Table 1 shows the procedure of watermark embedding by replacing filer coefficients' LSBs.

Table 1 FIR Filter Coefficients' LSB Watermarking Scheme

Original Filter Coefficients	Modified Filter Coefficients	Watermark bits
A0: 1111-1111-1100-0100	A0: 1111-1111-1100-010**1**	**1**
A1: 1111-1111-0100-1101	A1: 1111-1111-0100-110**0**	**0**
A2: 1111-1111-1010-1001	A2: 1111-1111-1010-100**0**	**0**
A3: 0000-0000-1110-1010	A3: 0000-0000-1110-101**0**	**0**
A4: 0000-0011-0111-1000	A4: 0000-0011-0111-100**1**	**1**
A5: 0000-1000-1101-1111	A5: 0000-1000-1101-111**0**	**0**
A6: 0000-0001-1111-1010	A6: 0000-0001-1111-101**1**	**1**
A7: 1111-1110-0101-0110	A7: 1111-1110-0101-011**0**	**0**

The watermark embedding flow of filter coefficient modification is concluded in the following paragraph:

Scheme1.a:Watermark Embedding by Filter Coefficients' Modification

Input: Filter design specifications, watermark bits that need to be embedded

Output: Watermarked filter design

```
Step 1.Design filter by using filter design specifications.
Step 2:Obtain digitized filter coefficients with specific word length.
Step 3:Modify filter's coefficients' LSBs according to watermark bits
Step 4:Use the modified filter coefficients as the simulation inputs of the filter design tool.
    Verify weather the filter's response satisfies the design specification or not.
    4.1.If yes, output the watermarked filter's coefficients, stop the procedure.
    4.2.If no, check weather we can increase the filter's coefficients or not.
        4.2.1.If yes, go to Step 2.
        4.2.2.If no, stop the procedure.
```

The watermark extraction procedure is as following: The watermark extractor first reads out the filter's coefficients and get the LSBs of the coefficients. The binary sequence of the filter coefficients' LSBs is the watermark information. For the improvement of the watermark scheme's security, the filter's coefficients need to be solidified into FPGA hardware by using ROM (read only memory) programming or hard-wiring (connect the signal lines with power or ground). So the designer needs to either read out the filter's coefficients from the ROM or find the internal hard-wired signals' value that are used as filter's coefficients. The following scheme shows the watermark extraction procedures.

Scheme1.b: Watermark Extraction for Filter Coefficients' Modification

Input: Filter coefficients (binary format), watermark bits that need to be verified

Output: Embedded watermark/copyright confidence

```
Step 1. Extract the filter coefficients LSBs.
Step 2:Read out the LSBs within the specific range where the watermark supposed to be
added.
Step 3:Compare the LSBs read from the filter coefficients with the original watermark
bits, check how many bits match and get their match rate.
```

3.2. Simulation of Coefficients' LSB Watermarking

To evaluate the effectiveness of proposal one, FIR filter LSBs' watermarking, the author makes the simulation to check the filter performance degradation after watermarking. Commercial filter design and simulation tool is used to carry the simulation. Here are the filter's original design specifications:

Direct form II transposed equal ripple low pass FIR filter

Order: 8

Fs: 48 KHz

Fpass: 9.6 KHz

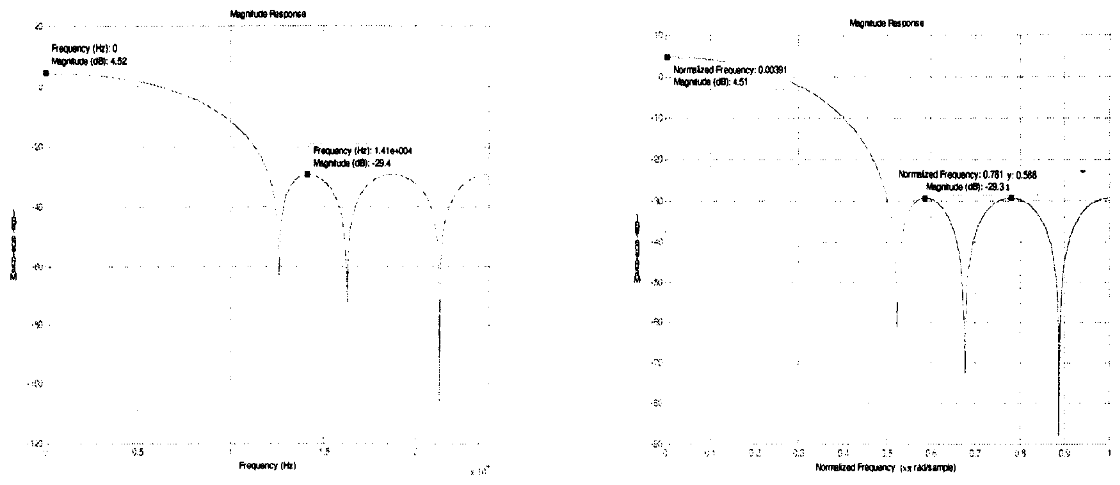
Fstop: 12 KHz

Wpass: 1 dB

Wstop: 20 dB

The Matlab FDA (Filter Design and Analysis) tool is used to design the original filter and create the filter magnitude response before and after the watermarking. By reading out the magnitude response after watermarking, the difference of Wpass and Wstop of the filter before and after watermarking are less than 0.1 dB.

Figure 8 FIR Filter's Magnitude Response Before (Left) and After Watermarking (Right), 16-bit Coefficients



From the simulation results, we can see that FIR filter coefficients' LSB watermarking method will introduce some degree of performance degradation. The author also carries the simulation for the watermarking of the same FIR filter with 8-bit coefficients. But the simulation results show that the magnitude distortion is too large and unacceptable. So for 8-bit coefficients case, the LSB watermarking scheme introduces too large magnitude response change.16-bit watermarking case works well, with quite small magnitude response degradation. Coefficients' LSB watermarking method can also be extended to other DSP algorithms that have a sequence of parameters.

3.3.Watermarking Performance Analysis

To evaluate the watermarking scheme's performance for proposal one, the author uses the evaluation items that have been introduced in chapter 2. The evaluation results will also be used for the summarizations that are made at the end of this chapter.

Embedding Efficiency: 1 bit/tap. From the watermarking method, we can embed one bit of information with each filter coefficient. The maxim number of watermarking bits that can be embedded to the filter's coefficients is equal to the number of the filter's taps.
Embedding Cost: The embedding procedure is simple and direct. The designer can modify the filter's coefficients' LSBs by using normal text editors, like notepad or so. There is no hardware or timing punishment for this method.
Design Overhead: There is no design overhead for the filter design itself. The watermark is added after the filter design has been finished. But the designer who embeds the watermark needs to run simulation after watermarking to verify the filter's performance degradation is acceptable or not.
Extraction Cost: The extraction procedure is direct and simple. People just need to read out the filter's coefficients' LSBs and get the watermark bit sequence.
*Probability of Coincidence (Pu):*If we use all available coefficients' LSB for watermarking, the probability of coincidence equals to 1/(2^n)), where n is the number of filter taps. For the 8-bit watermark and 8-tap filter case, this number equals to 1/256. For 128-bit watermark, this number equals to 3.4E-38.

Security Strength: For those DSP algorithms whose coefficients have been programmed into non-erasable memory (ROM) or implemented through hard wiring, this method provides very strong security strength. The attacker will have no way to remove the watermark embedded inside the coefficients' LSBs unless he removes the ASIC or FPGA chip.

Applied Area: This method can be applied to filter algorithms, as well as DSP algorithms' ASIC/FPGA implementation that have sequential parameters. The filter coefficients or other parameters should be solidified into the hardware.

Filter Performance Degradation: The filter performance degradation is decided by the filter coefficients' word-length. For 16-bit FIR filter case, the magnitude degradation is less than 0.1dB.

Probability of Detection Miss (Pm): From the watermark detection procedure, we can see that the probability of a detection miss is 0.The extractor can always successfully extract the watermark information by reading out the filter's coefficients' LSBs.

Table 2 Comparisons between Filter Coefficient Modification Watermarking and Current Filter Watermarking Schemes

	Magnitude Modification	**Tap's Equal Replacement**	**Windowing Function Watermarking**	**Filter Coefficient Modification Watermarking**
Filter Performance Degradation	Medium	Small	Small	Small
Hardware Usage Increase	+7%	+29%	N/A	0%
Design Overhead	High	Medium	Low	Low
Extraction Cost	Low	Medium	Low	Medium
Probability of Coincidence	Low	Low	Low	Low
Security	Medium	Medium	Low	High

The proposed filter coefficient's modification watermarking scheme shows better performance than other current available filter watermarking schemes with most performance items. The filter's hardware usage increase is 0% and is the major strong point for this new proposal. The design overhead is also low due to little additional design effort required for watermark embedding. The probability of coincidence is low due to the large amount of watermark information that can be embedded inside the filter's coefficients. The security strength is quite high because of the hardware solidation of the watermark information, so to remove the watermark is physically impossible. All in all, the proposed filter coefficients' modification watermarking scheme shows good watermarking performance and has good potential for practical DSP algorithms' watermarking.

Chapter 4: Watermarking Through FPGA Cell Location Constrain

The FPGA cell location constrain watermarking scheme uses the FPGA cells' x and y coordinates to embed the watermark information. This method belongs to the physical constrain watermarking method. By doing so, each selected FPGA layout cell for watermarking can embed 2 bits of information. Several real world IPs like FIR filter, CDMA match filter, discrete cosine transform (DCT) are used to embed 8-bit watermark. The simulation results are used to verify the watermarking performance. Varied length watermark embedding are carried for the FIR filter to study the relationship between design overhead and watermark length.

4.1. Watermarking Scheme

The watermarking flow for this scheme is firstly set up some kind of watermark embedding rules. Then the designer needs to select the FPGA layout cells that will be used for watermarking. After that, the designer makes the original implementation of the IP without watermarking. After this step, the selected FPGA cells will be moved according to the watermarking rules. And the FPGA implementation procedure will be carried again with these new layout location constrains. The finished layout of the FPGA will contain the watermark information that the designer has embedded by constraining the cells' layout locations.

Let a FIR filter design be given by:

Y(k)=A0*X(k)+A1*X(k-1)+......+An-1*X(k-N-1),where k=0,1,....N-1.

Suppose we need to embed a watermark consists of 8 bits as 10001010. Let the RAM (random access memory) cells to store Ai be RAM_i, i=0,1,...,7. A RAM cell in FPGA layout can be identified by its coordinates (x, y). The watermark embedding rules are described as following:

If the bits are 00, we choose both x and y as even.

If the bits are 01, we choose x as even and y as odd.

If the bits are 10, we choose x as odd and y as even.

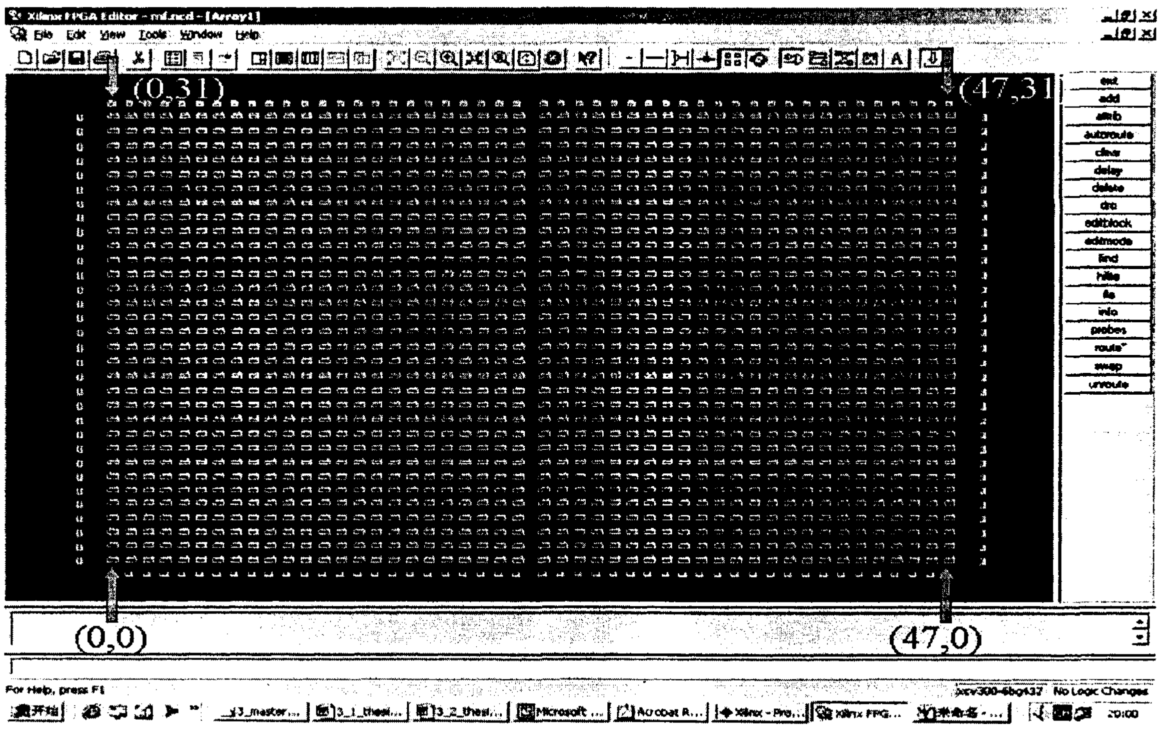
If the bits are 11, we choose both x and y as odd.

Table 3 shows the procedure of embedding watermark to RAM cells' location coordinates.

Table 3 FPGA Cell Location Watermarking

RAM_i for Storing Ai	Watermarking Bits	New Location of the Chosen RAM Cell: (x, y)
RAM_0	**1**	**(1,4)**
	0	
RAM_1	**0**	**(4,4)**
	0	
RAM_2	**1**	**(3,2)**
	0	
RAM_3	**1**	**(3,0)**
	0	

Figure 9 FPGA Layout with 48*16 Slices Array



For FPGA devices, the whole layout is divided into certain number of slices. (see figure 9) There are route path between slices. Each slice has some kind of logic resources, like 16-bit RAMs, look-up tables, D-type flip-flops and so on. Different types of FPGA devices will have different number of slices available, as well as the logic cells' type and amount within each slice. For watermarking purpose, we can use the logic cells like 16-bit RAM, D-type flip-flop or other kind of logic cells to embed the watermark information. The selection of the cells should satisfy that each cell used for watermarking does not occupy more than one FPGA slice.

Scheme 2.a.Watermark Embedding Flow of FPGA Cell Location Modification

```
Input: Original FPGA logic and layout design and design specifications, watermark bits
        need to be embedded
Output: Watermarked FPGA layout design
Step 1.Try to find FPGA layout RAM cells that have sequential relations from the logic
view.
        1.1.If we can find, from these RAM layout cells,
            1.1.1.Try to find layout cells that occupy less than 1/4 FPGA slices. If we can
                  find them, go to step 2.
            1.1.2.Try to find layout cells that occupy less than or equal 1 FPGA slices.
                  If we can find them, go to step 2.
                  If we cannot, stop the procedure.
        1.2.If we cannot, stop the procedure.
Step 2.Select layout cells locating at loose place and route area.
        2.1.From the cells we find in step 1,try to find those cells locating at unused
            place and route areas.
            2.1.1.If we can find them, go to step 3.
            2.1.2.If we cannot, go to step 2.2.
        2.2.From the cells we find in step 1,try to find those cells locating at loose
        place and route areas.
            2.2.1.If we can find them, go to step 3.
            2.2.2.If we cannot, use the cells we find in step 1 and go to step 3.
```

Step 3.From the layout cells we find in step 2, select layout cells locating at parallel logic paths.

    3.1.If we can find them, go to step 4.
    3.2.If we cannot, use the cells we find in step 2 and go to step 4.

Step 4.From the layout cells we find in step 3, select layout cells locating at non-critical logic paths.

    4.1.If we can find them, go to step 5.
    4.2.If we cannot, use the layout cells we find in step 3 and go to step 5.

Step 5.For every selected watermarking cell, try to find new location for re-locating that satisfies watermarking bit and watermarking rules.

```
5.1.Try to find spare layout slice within one slice distance.
    5.1.1.If we can find it, move the layout cell to the new location. Go to step 6.
    5.1.2.If we cannot , try to find spare layout slice within two slices distance.
          If we can find it, move the layout cell to the new location. Go to step 6.
    .......
    5.1.n.If we cannot , try to find spare layout slice within n slices distance. Here
          n is the largest slice number of possible movement.
        If we can find it, move the layout cell to the new location. Go to step 6.
5.2.If we cannot find it, try to swap this layout cell with another layout cell that is
    not selected as watermarking cell and within one slice location.
    5.2.1.If we can find it, move the layout cell to the new location. Go to step 6.
    5.2.2.If we cannot , try to find such layout cell within two slices distance.
          If we can find it, move the layout cell to the new location. Go to step 6.
    .......
    5.2.n.If we cannot , try to find spare layout slice within n slices distance. Here
          n is the largest slice number of possible swap .
        If we can find it, move the layout cell to the new location. Go to step 6.
        If we cannot, stop the procedure.
```

Step 6.Run place and route with the watermarked layout cell locations.

Step 7.Read out the place and route report after watermarking. Verify weather the watermarked layout implementation satisfies the design specification or not.

```
7.1.If yes, output the layout and create the bit-streaming data, stop the procedure.
7.2.If no, check weather all possible layout cell candidates for watermarking have
    been tried or not.
    7.2.1.If yes, stop the procedure.
    7.2.2.If no, go to step 1 and find new watermarking candidates.
```

The above paragraph shows the watermarking flow of FPGA RAM cell locations' watermarking schemes. This watermarking flow can also be applied to DSP algorithms' watermarking. The requirement is that the DSP algorithm has some kind of sequence of parameters that can be implemented with hardware. If the DSP algorithm does not have such kind of sequential parameters, only the FPGA cell location modification proposal can be applied.

Here the FIR filter's coefficients' LSB watermarking and the RAM cell locations' watermarking are used as examples. The proposal of FIR filter coefficient modification watermarking also can be extended to any DSP algorithms that have sequential structure of parameters and the parameters can be solidified into the FPGA hardware (For example, the filter's coefficients are programmed into the ROMs within FPGA chips. Another way to solidify the parameters with hardware is hard wiring. For example, when the filter's coefficients' bit is 0, this bit-line will be connected with ground. When the filter's coefficients' bit is 1, this bit-line will be connected with power.) The second proposal can be extended to any pure digital logic IP's FPGA implementation as long as these digital IPs have some kind of sequential structure of layout cells. The DES/AES encryption function is provided by the FPGA implementation tool [16].

Scheme 2.b.FIR Filter FPGA Cell Location Watermark Extraction Flow

```
Input: Watermarked FPGA layout design, original watermark bits  that have be embedded
Output: Watermark embedded within FPGA layout design
Step 1.Find the layout locations of RAM cells storing FIR filter coefficients
Step 2. Extract RAM cells' (x, y) location labels.
Step 3. Using cell location watermarking rules to extract the embedded watermark.
```

Step 4. Compare the embedded watermark with the original watermark, get copyright confidence measure.

The above scheme shows the watermark extraction flow. Here the FIR filter's coefficients' LSB watermarking and FPGA RAM cell location watermarking schemes have been used to embed the watermarks inside the FIR filter's coefficients and RAM cell locations' coordinates. The extractor needs to extract the watermarks inside the FPGA layout as well as FIR filter's coefficients. The extractor will be given the complete original logic design files containing all the schematics and placement/route files. The watermark extractor will also be given the FPGA design and debug software that are used to design and implement the original FIR filter design.

During the watermark extraction flow, the extractor firstly needs to find the layout locations of the RAM cells that store FIR filter's coefficients. The extractor can use the original logic and layout design files to search these RAM cells between logic schematic and layout. Then he will be able to extract the RAM cells' x and y coordinates. After that the extractor can use the RAM cells' x and y coordinates to get the embedded watermark information, by referencing the watermark embedding rules. After that he also can read out the RAM cell's contents by applying correct address and read signals to the FPGA chip and get the FIR filter's coefficients. After reading out the coefficients' LSBs, he gets the watermark that has been embedded into the FIR filter's coefficients LSBs.

4.2.FPGA Cell's Location Watermarking Simulation Procedures and Results

This section describes the simulation results for embedding 8-bit watermark to different FPGA devices as well as embedding different length watermark (from 8-bit to 128-bit) into the FIR filter design. The watermarking procedures are described step by step. To compare with the results from paper [21], the author of this thesis also selects the Xilinx FPGA device and its implementation tools.

4.2.1. Embedding 8-bit Watermark: Virtex-II device

The FIR filter has the following specifications:

Single rate single channel FIR filter

Number of taps: 8

Coefficient width: 16-bit

Coefficient data type: signed

Coefficients reload ability: yes

Input data width: 16-bit, signed

Output data width: 32-bit, signed

Implementation type: fully serial, conventional structure

The FPGA implementation software tool environments are:

Design and Implementation tool: Xilinx ISE (integrated silicon environment) 4.2

Design Flow: VHDL

Synthesis tool: XST VHDL

Floor plan tool: Xilinx Floor Planer

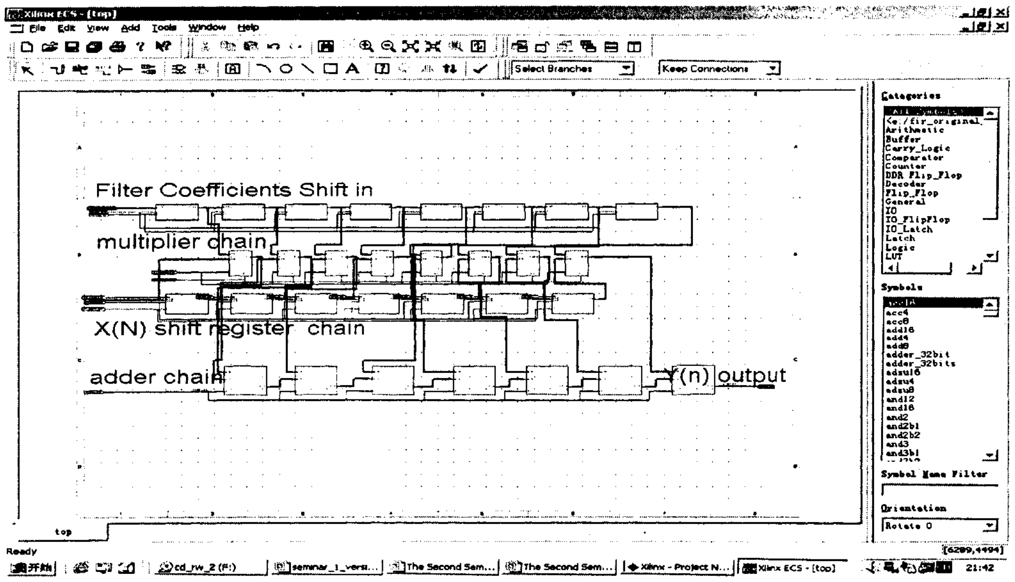
Place and route optimization goal: timing

Target FPGA device: Xilinx Virtex-II, XCV250, speed grade –6

Simulation Procedures:

Step1: Input the FIR filter design into the ISE design tool.

Figure 10 FIR Filter's Schematic Diagram



Step 2:Synthesis the FIR filter using XST VHDL

Step 3:Place and route the FIR filter without location constrains. After this step, we will get the FPGA place and route result with timing optimization.

Step 4:Suppose the registers that store filter's coefficient **A0**'s 4 LSBs are register 0-3, respectively. In this case, the registers will be used to embed the watermark. The registers are used to implement the function of RAM cells. We will re-assign these 4 registers with following new slice locations:

Register0: Slice (13,26)-------->(13,26)

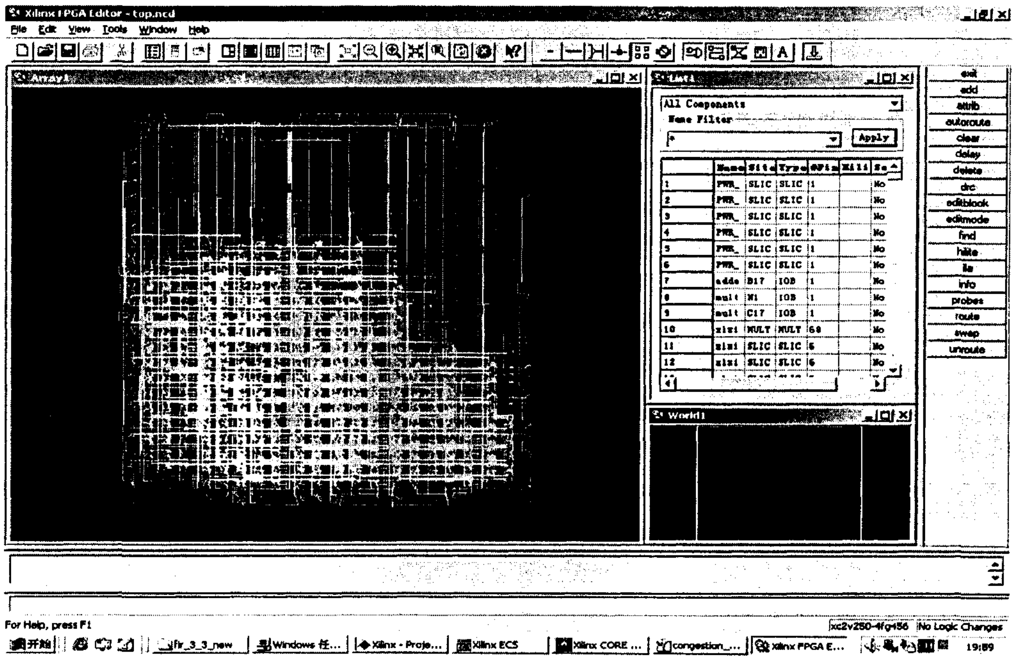
Register1: Slice (13,26)-------->(12,22)

Register2: Slice (5, 23)--------->(5, 20)

Register3: Slice (5, 23)-------->(5, 20)

Step 5:Run placement and route again with watermark location constrains, get the new layout with watermark inside. In this case, we only move the cells that we need to use for watermarking and do not change the locations of those cells that have not been used for watermarking.

Figure 11 Original Layout of the FIR Filter, Virtex-II Device

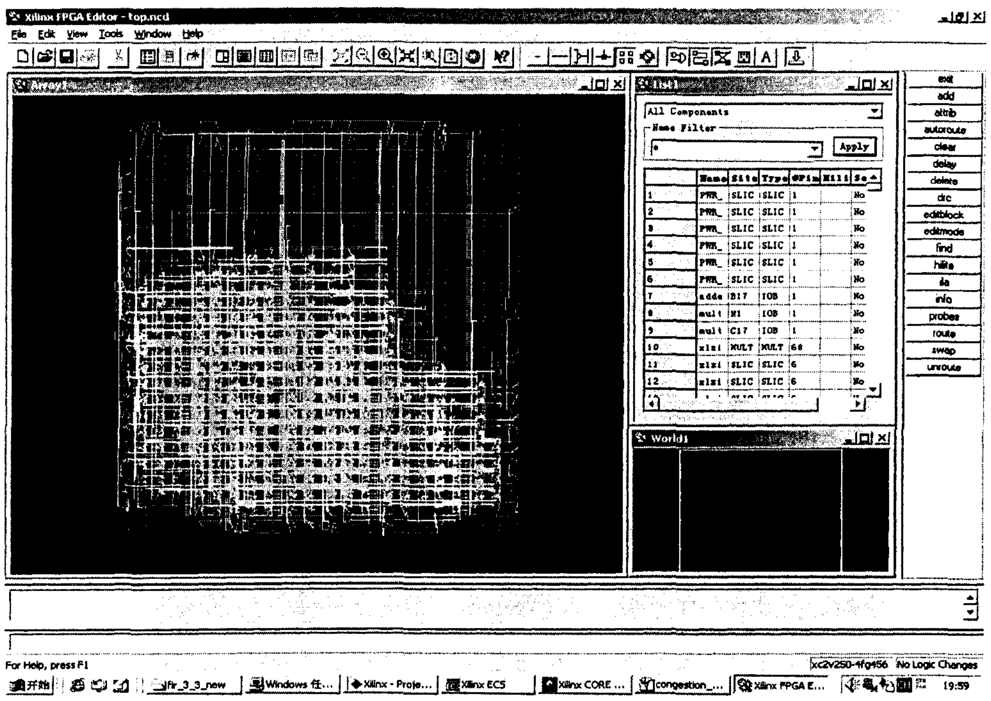


All Components

Name Filter

	Name	Site	Type	#Pin	Hili	Se
1	PWR_	SLIC	SLIC	1		No
2	PWR_	SLIC	SLIC	1		No
3	PWR_	SLIC	SLIC	1		No
4	PWR_	SLIC	SLIC	1		No
5	PWR_	SLIC	SLIC	1		No
6	PWR_	SLIC	SLIC	1		No
7	adde	D17	IOB	1		No
8	mult	N1	IOB	1		No
9	mult	C17	IOB	1		No
10	xixi	MULT	MULT	68		No
11	xixi	SLIC	SLIC	6		No
12	xixi	SLIC	SLIC	6		No

Figure 12 Watermarked Layout of the FIR Filter, 8-bit Watermark, Virtex-II Device



	Name	Site	Type	#Pin	Hili	Se
1	PWR_	SLIC	SLIC	1		No
2	PWR_	SLIC	SLIC	1		No
3	PWR_	SLIC	SLIC	1		No
4	PWR_	SLIC	SLIC	1		No
5	PWR_	SLIC	SLIC	1		No
6	PWR_	SLIC	SLIC	1		No
7	adde	B17	IOB	1		No
8	mult	N1	IOB	1		No
9	mult	C17	IOB	1		No
10	xlxi	MULT	MULT	68		No
11	xlxi	SLIC	SLIC	6		No
12	xlxi	SLIC	SLIC	6		No

From the layouts of the FIR filter before and after watermarking, it is quite hard to find the locations of the watermark by only observing the watermarked layout. This is the property of the transparency of the watermark. This property makes it more difficult to detect and remove the watermark from the FPGA layout by potential attackers. (Here we suppose the potential attacker does not have access to the original layout of the FIR filter. Because by comparing the original and after watermarking layouts, the difference can be easily identified. If the attacker already has access to the original layout, it is not necessary for him to attack the watermark anymore.)

The selection of cells' new locations is decided by the author with referencing the original optimization layout location placement results to make as small distance change as possible. This will help to achieve better timing performance. After implementing both the design, the author gets the timing reports, (see appendix A and B), to find the detailed timing delay information. The author also gets the place and route reports (see appendix C and appendix D) that show the hardware usage information before and after watermarking.

The following table shows the results of the two implementations on **Virtex II** device, before watermarking and after watermarking, with 8-bit watermark embedding.

Table 4 FIR Filter Cell Location Watermarking, 8-bit, Virtex-II Device

	Maxim net delay	Minim Period	Max Frequency	Hardware cost (slices)
Before watermarking	4.441 ns	9.289 ns	107.654 MHz	488/1536
After watermarking	4.441 ns	9.392 ns	106.474 MHz	489/1536

So after watermarking, the FIR filter's

Timing delay increases by :(9.392-9.289)/9.289 = + 1.109 %

Hardware cost increase by :(489-488)/488 = + 0.205 %

The movement of the registers introduces the timing delay increase from their timing optimization results. And the hardware cost increase is caused by the movement of registers to spare slices. Before watermarking, these registers share slices with other logic cells. When certain register is moved to a spare slice and cannot share with other logic cells that are not used for watermarking, the hardware cost will increase by one FPGA slice. If we can find such slice that can satisfy the watermarking rule and is shared by the logic cells that will not be used for watermarking, the hardware cost will not change.

4.2.2. FIR Filter Cell Location Watermarking, 8-bit watermark, Virtex II Pro device

Table 5 FIR Filter Cell Location Watermarking, 8-bit, Virtex II _PRO Device

	Maxim net delay	Minim Period	Max Frequency	Hardware cost (slices)
Before watermarking	3.210ns	5.593ns	178.795MHz	488/1536
After watermarking	2.947ns	5.540ns	180.505MHz	490/1536

So after watermarking, the FIR filter's

Timing delay increases by :(5.540-5.593)/5.593 = - 0.948 %

Hardware cost increases by (490-488)/488 = + 0.410 %

4.2.3. FIR Filter Watermarking, Watermark Length Varying from 8 bits to 128 bits, Virtex II Device

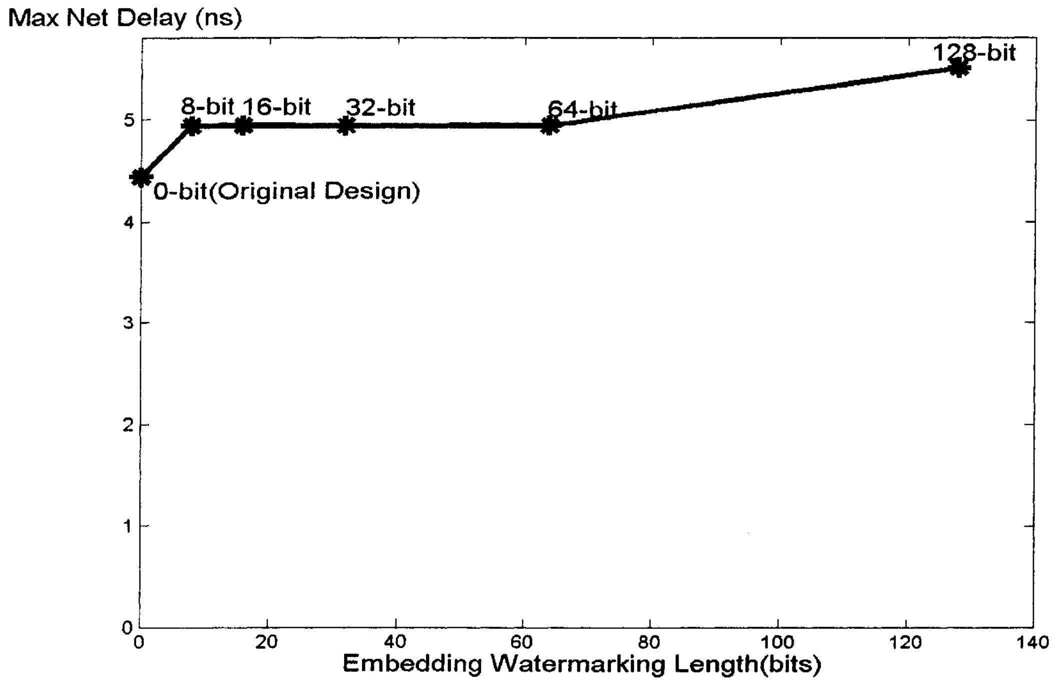
To study the relationships of the length of the watermark and the FPGA implementation results, several implementations of the watermark whose length varies from 8 bits to 128 bits are carried. Firstly the author uses Matlab to create 128 bits of random number sequence. Then the author uses the FIR filter's registers that store the coefficients to embed the watermark, using watermark embedding rules. The registers are selected and used from the register that store FIR filter's coefficient A0, then the register that store FIR filter's coefficient A1, so on and so forth.

Table 6 FIR Filter Cell Location Watermarking, 8-bit to 128-bit, Virtex II Device

Watermark length	Max net delay (ns)	Min Period (ns)	Max Frequency (MHz)	Hardware cost (slice)
0	4.441	9.289	107.654	488
8 bits	4.934	13.866	72.119	489
16 bits	4.941	13.866	72.119	488
32 bits	4.934	13.866	72.119	495
64 bits	4.941	13.866	72.119	498
128 bits	5.504	13.866	72.119	510

The following figure shows the relationship between watermark length and maxim net delay:

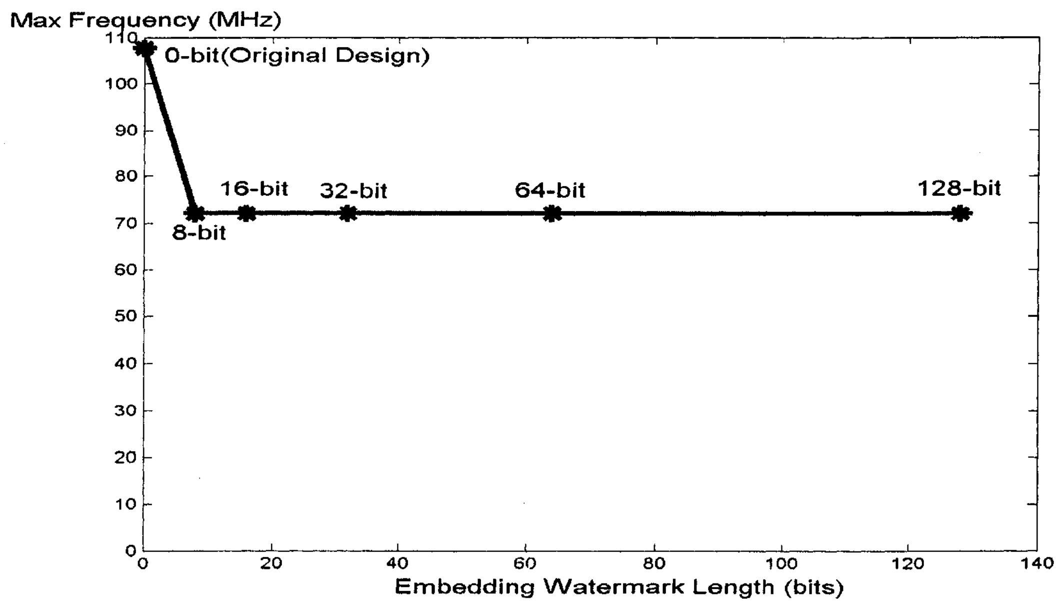
Figure 13 FIR Filter Watermark Length & Max Net Delay Relation



From Figure 16 we can see that when the watermark length changes from 8 bits to 64 bits, the max net delay keeps nearly the same. As the author explains later in this chapter, the increase of the timing delay that happens at different logic path will not accumulate all together according to simple addition operation relation. As long as the timing increase happening within watermarked logic paths is smaller than the max net delay path's timing delay, the max net delay for the FPGA will not change. If the later on watermark cells' movement introduces net delay increase that overpasses the original max net delay, the final max net delay for the FPGA chip will increase further.

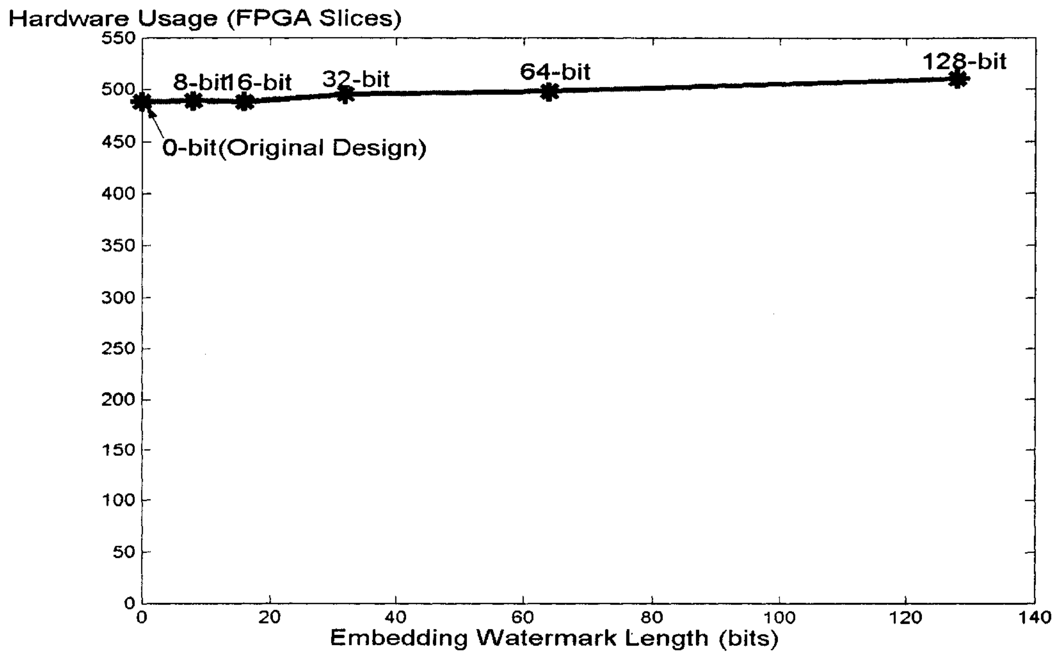
The following figure shows the relation between watermark length and Maxim frequency:

Figure 14 FIR Filter Embedding Watermark Length and Max Frequency Relation



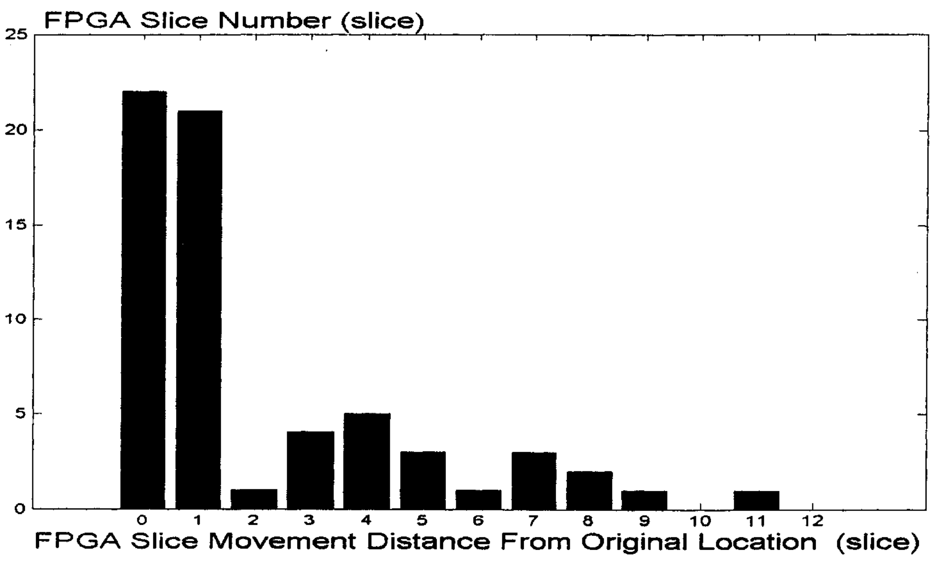
From figure 14, we can see that the max frequency does not change when the embedding watermark length changes from 8-bit to 128-bit. The max frequency is decided by the minim period of the FPGA chip.

Figure 15 FIR Filter Embedding Watermark Length and Hardware Usage Relation



From figure 15, the after all tendency is that the hardware usage of the FPGA after watermarking increases when the watermark length increases. When we need to embed longer watermark, we need more spare FPGA slices to re-assign the watermarking FPGA cells. If we cannot find the slices that we can share the watermarking FPGA cells with the cells that have not been selected for watermarking usage, the hardware usage will increase.

Figure 16 FIR Filter Cell Movement Distance Distribution, 128-bit Watermark



The cell movement distance used here means the difference between the cell's original x coordinate and the cell's x coordinate after watermarking, or the difference between the cell's original y coordinate and the cell's y coordinate after watermarking, whatever is the larger. To embed 128-bit watermark information, we need 64 FPGA cells for location watermarking usage. From Figure 19, we can see that 70%(45 cells out of 64 cells) of the cell movements happen within one slice distance. And the longest movement is 11 slices.

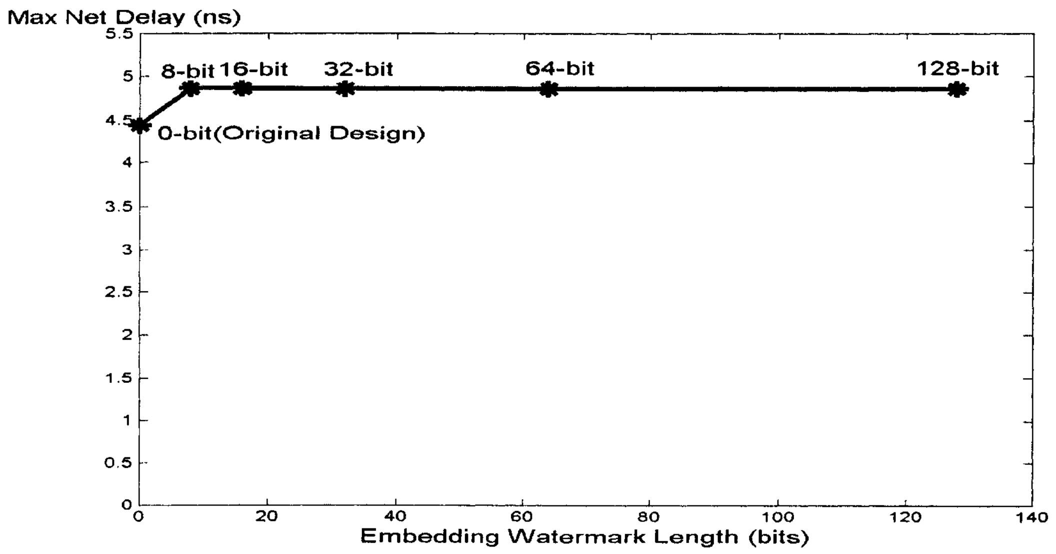
4.2.4. FIR Filter Watermarking, Watermark Length Varying from 8-bit to 128-bit, Virtex II Device, Reverse Sequence

In this simulation, several implementations of the watermark whose length varies from 8 bits to 128 bits are also carried. The author uses the FIR filter's registers that store the coefficients to embed the watermark in reverse sequence. The registers are selected begin from the registers that store FIR filter's coefficient A15 (Firstly we will use the register that stores A15's MSB, then we will use the register that store A15's 15th bit, so on and so forth.) After we use up the registers that store A15, we will use registers that store FIR filter's coefficient A14. Finally we will use the registers for storing A12. (For 128 bits watermark embedding, we need 64 registers all together.)

Table 7 FIR Filter Cell Location Watermarking, 8-bit to 128-bit, Virtex II Device, Reverse sequence

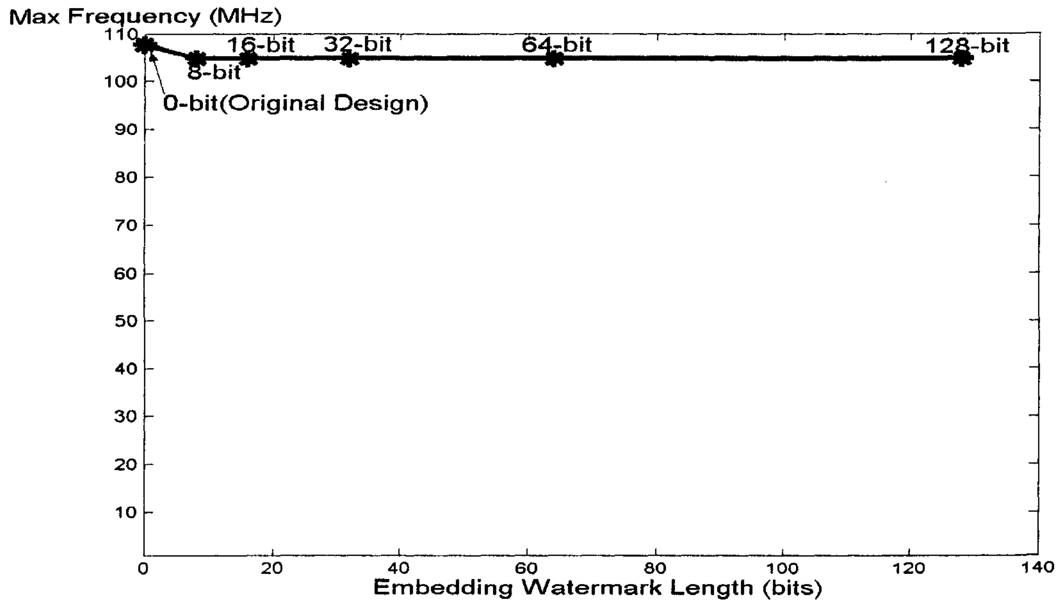
Watermark length	Max net delay (ns)	Min Period (ns)	Max Frequency (MHz)	Hardware cost (slice)
0	4.441	9.289	107.654	488
8 bits	4.864	9.554	104.668	489
16 bits	4.864	9.554	104.668	489
32 bits	4.859	9.554	104.668	492
64 bits	4.859	9.554	104.668	494
128 bits	4.859	9.554	104.668	503

Figure 17 FIR Filter Watermark Length & Max Net Delay Relation, Reverse Sequence



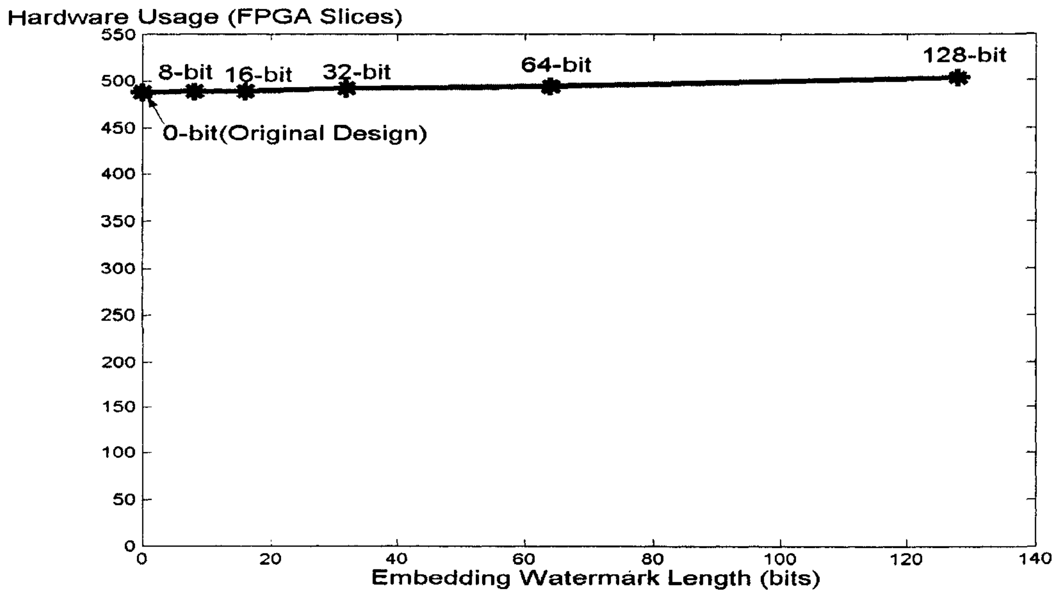
From Figure 17 we can see that when the watermarking length changes from 8-bit to 128-bit, the max net delay keeps nearly the same.

Figure 18 FIR Filter Embedding Watermark Length and Max Frequency Relation, Reverse Sequence



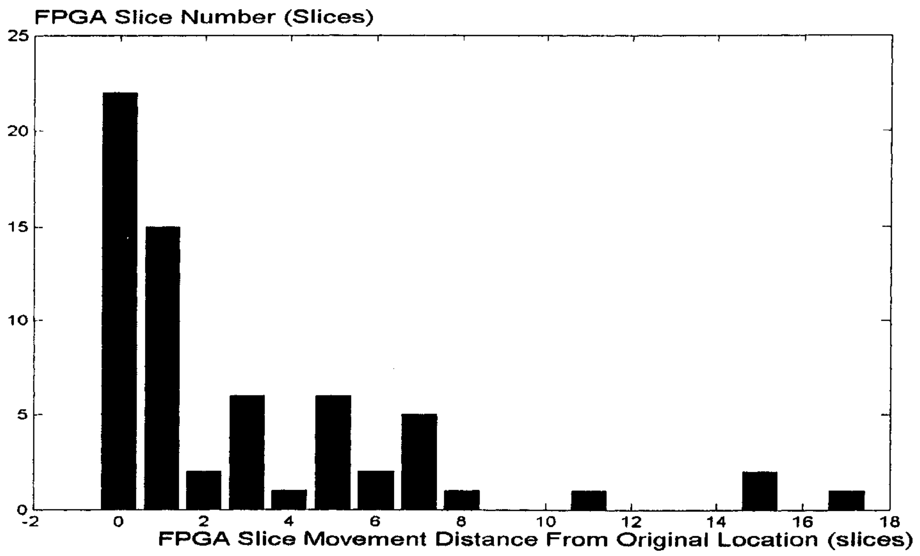
From figure 18, we can see that the max frequency does not change when the embedding watermark length changes from 8-bit to 128-bit. The max frequency is decided by the minim period of the FPGA chip.

Figure 19 FIR Filter Embedding Watermark Length and Hardware Usage Relation, Reverse Sequence



From figure 19, the tendency is that the hardware usage of the FPGA after watermarking increases when the watermark length increases. Comparing with figure 18, we can see that for embedding same length watermark, the reverse embedding method cause less hardware usage increase.

Figure 20 FIR Filter Cell Movement Distance Distribution, 128-bit Watermark, Reverse sequence



4.3. Watermarking Simulations for Other FPGA IPs

To evaluate the performance of the FPGA cell locations' watermarking scheme, the author also makes several simulations for other types of DSP algorithms' watermarking. These DSP algorithms include comb filter, CDMA match filter, DCT and waveform synthesizer. These DSP algorithms are among the most widely used DSP IP cores. Another non-DSP IP is also used to verify the applicability of this proposal for general form digital IP's FPGA implementation's watermarking.

4.3.1. Comb Filter Watermarking, 8-bit watermark

Comb filters are widely used in image processing and signal up-sampling/down-sampling. Here are the comb filter IP's specifications:

Comb filter type: Interpolator

Input bus width: 16-bit

Number of stages: 4

Sample rate change: 16000

Differential delay: 2

The following is the comb filter watermarking results with 8-bit watermark.

Table 8 Comb Filter Watermarking, 8-bit watermark

	Min Period	Max frequency	Max combinational path delay	Max net delay	Hardware cost (slice)
Before watermarking	7.568ns	132.135MHz	7.911ns	3.918ns	501/1536
After watermarking	7.538ns	132.661MHz	8.526ns	4.087ns	504/1536

So after watermarking, the comb filter's

Timing delay increases by :(7.538-7.568)/7.568 = - 0.396 %

Hardware cost increases by (504-501)/501 = + 0.599 %

4.3.2.CDMA Match Filter Watermarking, 8-bit Watermark

The CDMA match filter is also watermarked with 8-bit watermark.

The following table shows the implementation results before and after watermarking:

Here the mem-16s cells (16-bit memory blocks) are used for watermarking.

Table 9 CDMA Match Filter Watermarking, 8-bit watermark

	Min Period	Max frequency	Max combinational path delay	Max net delay	Hardware cost (slice)
Before watermarking	14.113ns	70.857MHz	N/A	9.329ns	2133/3072
After watermarking	13.870ns	72.098MHz	N/A	8.868ns	2133/3072

So after watermarking, the CDMA match filter's

Timing delay increases by :(13.870-14.113)/14.113 = - 1.722 %

Hardware cost increase by: (2133-2133)/2132 = + 0 %

After watermarking, the FPGA's max frequency has been improved by 1.722% and the hardware usage does not change. The hardware usage does not change since the original mem-16 cells do not share slice with any other logic components. So the movements of these mem-16 cells do not increase the hardware usage.

4.3.3.DCT Watermarking, 8-bit Watermark

The DCT IP is widely used for image processing. Here we use the 1-D DCT IP for watermarking.

This DCT IP has the following specifications:

Operation mode: Forward DCT (1-D)

Number of points: 16 enable symmetry

Performance: 9-clock cycles/sample

Coefficients width: 8-bit

Input data width: 8-bit, signed

Precision control: Round

Result width: 20

Table 10 DCT Watermarking, 8-bit Watermark

	Min Period	Max frequency	Max combinational path delay	Max net delay	Hardware cost (slice)
Before watermarking	7.068ns	141.483MHz	7.960ns	4.970ns	1173/1536
After watermarking	7.804ns	128.139MHz	8.469ns	4.775ns	1173/1536

So after watermarking, the DCT's

Timing delay increases by :(7.804-7.068)/7.068 = +10.4 %

Hardware cost increase by: (1173-1173)/1173 = + 0 %

4.3.4. Direct Waveform Synthesizer Watermarking, 8-bit Watermark

The direct digital synthesizer can produce sine and cosine waveforms, without using look-up tables. It uses the iteration operations to create the sine and cosine waveforms.

Its design specification are shown below:

Output width: 16-bit

Function :Output both sine and cosine waveforms

Memory type: Distributed ROM

Pipelined operation: Yes

Phase increment: Register based

Data width: 16-bit

Phase angle width: 4-bit

Phase offset: None

Noise shaping: No

In this case, the author uses the 16-bit distributed ROMs to embed the watermark information.

Table 11 Direct Waveform Synthesizer Watermarking, 8-bit Watermark

	Min Period	Max frequency	Max combinational path delay	Max net delay	Hardware cost (slice)
Before watermarking	4.3000ns	232.558MHz	6.919ns	3.864ns	34/1536
After watermarking	4.261ns	234.687MHz	6.919ns	4.404ns	37/1536

So after watermarking, the direct waveform synthesizer's

Timing delay increases by :(4.261-4.300)/4.300 = + 0.907 %

Hardware cost increase by (37-34)/34 = + 8.82 %

4.3.5. Board Interface Circuit Watermarking, 8-bit Watermark

From the discussion and simulation procedure, the FPGA cell location's watermarking method can also be used for other types of IPs that have sequential logic structures.(For example, the IP we need to watermark has 8 registers in serial.)The watermarking simulation for non-DSP IP is also carried. The IP here is the interface circuit between the FPGA chip and personal computer, for test and debug purpose. The implementation results are shown as below:(Four D-type flip-flops are used to carry the 8-bit watermark information)

Table 12 Board Interface Circuit Watermarking, 8-bit Watermark

	Min Period	Max frequency	Max combinational path delay	Max net delay	Hardware cost (slice)
Before watermarking	10.410ns	96.061MHz	13.484ns	2.602ns	192/256
After watermarking	10.548ns	94.805MHz	13.948ns	2.602ns	192/256

So after watermarking,

Timing delay increases by :(10.548-10.410)/10.410 = +1.326 %

Hardware cost increase by (192-192)/ 192 = +0 %

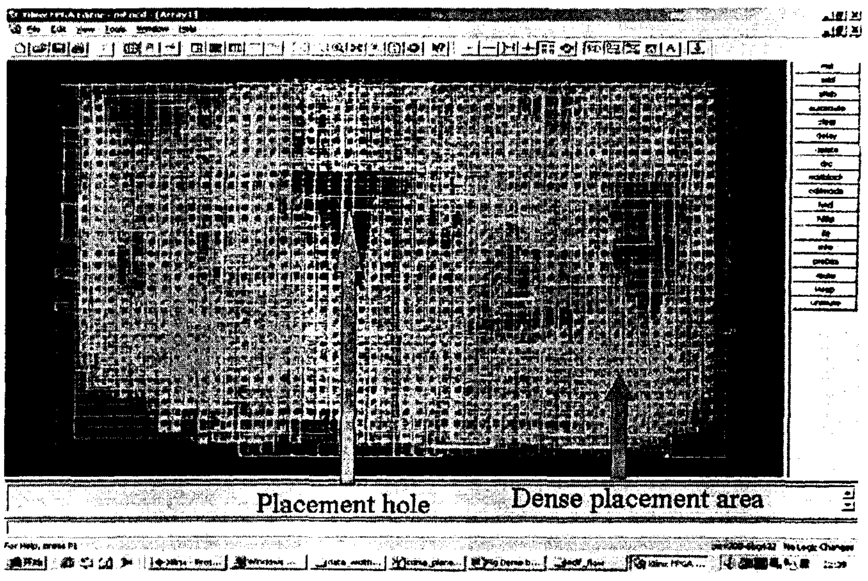
There is 1.326% maxim working frequency decrease and the hardware usage does not change. The hardware usage does not change since the original D-type flip-flops do not share FPGA slice with other components. So the movements of these D-type flip-flops do not increase the hardware usage.

4.4. Detail Watermarking Steps for Implementation Optimization

From the simulation, the author concludes that the following watermarking rules might help the designer to optimize the FPGA's timing performance after watermarking. These guild-lines give the designer some direction for implementation optimization during the watermarking procedure.

The designer should try to use the FPGA cells that locate along the border around the loose placement areas. Some times we will also have placement holes that in this area no FPGA hardware resource has been used. The designer should try to avoid using the FPGA cells that locate inside the dense place and route area. (We call these dense place and route area hot areas, just like the traffic jam areas in modern big cities of human being). To make such kind of selections will give us more selections for the new locations of watermark cells and may improve implementation results. If we use the cells that are inside the hot area, we will feel more difficult to find available new locations for watermarking within small slice movement distance.

Figure 21 Holes and Dense Placement Areas of the CDMA Match Filter



The designer also should try to use cells that locate at different logic paths.

Figure 22 FPGA Cell Selection Strategy

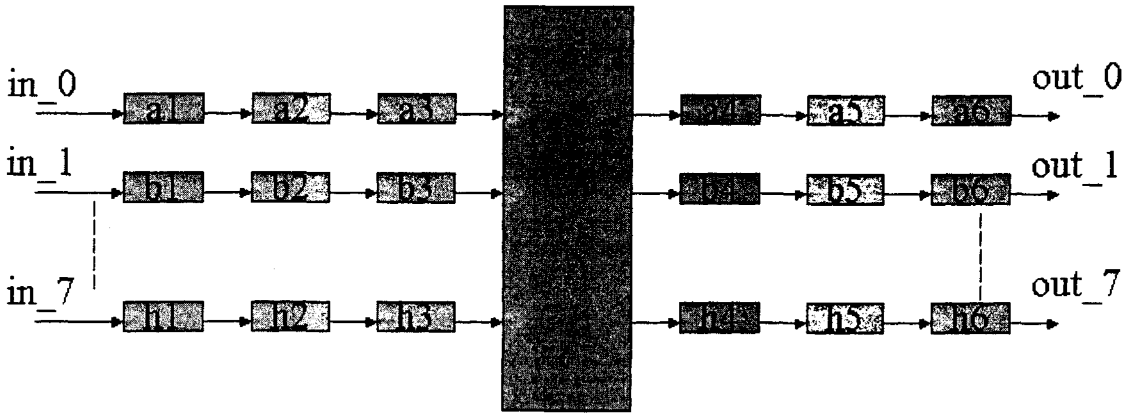
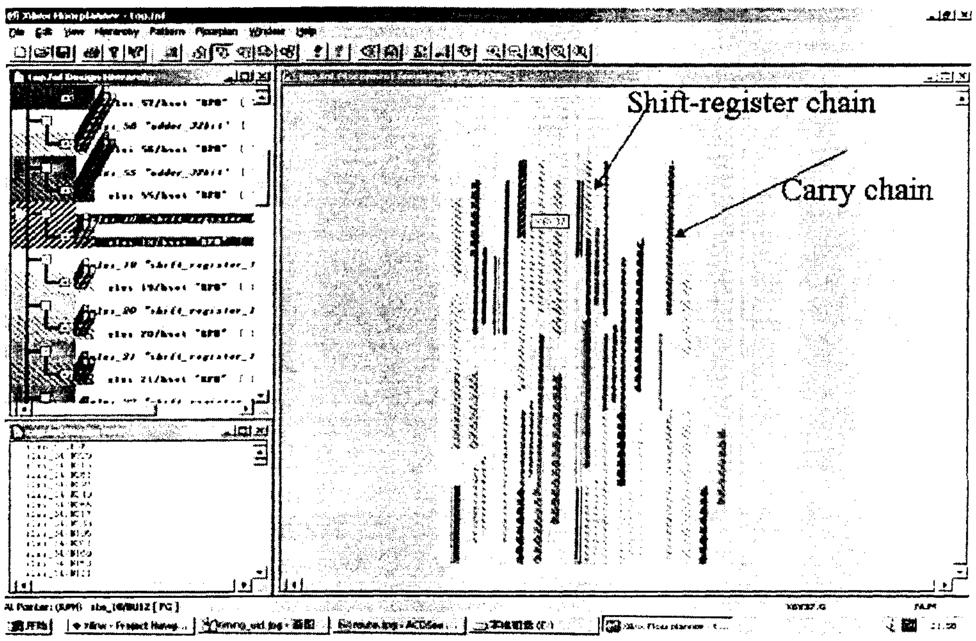


Figure 22 shows the logic structure of an IP. We need to use some of the cells in this IP to embed the watermark. In order to reduce the timing delay as much as possible, we need to select the cells smartly. Suppose the modification of the cells (a1, b1...) in this IP will introduce equal amount of timing delay Td to the final layout implementation. We should firstly use the cells locate on different path in parallel. For this case, we can use a1, b1, c1, ..., h1. Then we use a2, b2, c2, ..., h2. Then we will use a3, b3, c3, ..., h3. Finally we will use a6, b6, c6, ..., h6. The reason for this selection method is that the timing delay happening within the serial path will accumulate according to addition relation. And it will increase the total delay continuously. If we use a1, a2 and a3, we will get 3Td for the final timing delay increase. But if we use a1, b1 and c1, we will have Td only for the final timing delay increase. For digital logic design, the final maxim working frequency is decided by the slowest path's maxim timing delay. This makes us to use cells for watermarking in parallel instead of in serial. But the fact is every IP design and its FPGA implementation is different, the effect of moving the cells will be different. This makes the problem more complicated. But from the simulation results, most of the cells' movements happen within 2 slices' distance, so this method still has its value to some extent.

The designer should try to use small layout modules instead of big and solid layout cells. There is relationship between the layout design style and the watermarking difficulty. The higher the hardware usage percentage, the more difficult the watermarking procedure will be. When the FPGA hardware usage percentage increases, we will have less spare layout slices for re-assignment of those watermark cells. And we need to move slices with longer distance to find the locations that satisfy the watermarking embedding rules. The worst case is that 100% of the FPGA slices have been used. We have no way but try to swap the locations of the FPGA layout cells to re-assign the watermark cells. If we use big and solid layout cells, for example, the layout module that occupies 16 slices and can not be separated or re-placed and re-routed, we will have to overpass those large blocks during watermarking. And the average slice movement distance will increase. This will cause worse timing performance. In figure 23,we can see some kind of large and solid layout cells like carry-chain and shift-register-chain. When we carry the watermarking operations, we need to overpass these cells to find the available new slices to assign the cells for watermarking. This increases the watermarking difficulty.

Figure 23 FIR Filter Placement Diagram with Pre-solidified Layout Modules



The designer should try to find the slice to assign the watermarking component (register, for example) within one slice distance from the optimized location. If there is more than one slice candidate available, for fast embedding, he should arbitrarily choose one. For best result, he should try all the available candidates and run implementation for each case and select the best one. If he cannot find the available slice within one slice distance, he should try to find such a slice within two slices distance away from the optimized slice location. Such kind of search should be carried until one slice has been found or the search area has touched the boundary of FPGA chip. This means the search step has failed. Under this condition, the designer may use cell swap technology to solve the problem. The designer should try to avoid using the components that are large logic blocks like multipliers. We should use those small components like gates or latches that can be implemented within single FPGA slice. The designer should also try to avoid using the components that are located on the critical path.

We have two different after-watermarking placement flows for the FPGA cell locations watermarking scheme after layout cell's selection.

The steps for first one are:

1.Place and route the original FPGA IP. 2.Select the cells we prefer to use for watermarking. 3.Re-assign layout slice locations for these cells that have not been used by the original placement. Tell the place and route tool "do not touch all cells" during following implementation process. So the place and route tool will not change the locations of other cells that have not been selected for watermarking. 4.Run the implementation flow again and get the final layout with watermark.

The steps for second one are:
1.Place and route the original FPGA IP. 2.Select the cells we prefer to use for watermarking. 3.Re-assign layout slice locations for these cells, without considering these locations have been used by the original placement or not. In this step, the designer only use the smallest slice movement rule to make the selection. Tell the place and route tool "do not touch these watermarked cells" during implementation process. Let the place and route optimization tool to decide whether change the locations of other cells that have not been selected for watermarking or not. The optimization goal is timing. 4.Run the implementation flow again and get the final layout with watermark.

The difference between the first and second one is whether we allow the tool to change the locations of the cells that we do not use for watermarking. If time is allowed, the designer can try both methods and select the one with better timing implementation results.

4.5. Watermarking Performance Analysis

Embedding Efficiency: 2 bits/cell. Since we can embed 2 bits to one FPGA cells' x and y coordinates.
Embedding Cost: The embedding of watermark by assign FPGA cells with new locations is simple and fast. The designer can use the FPGA placement tool to carry this operation. No extra software is needed.
Design Overhead: For the 8-bit watermark and the FIR filter with 8 taps, the timing increase is 1.1% and the hardware cost increase is 0.2% (Virtex-II device case).
Extraction Cost: The extraction of the watermark within the FPGA layout is fast and simple. We can use the design tool to load in the FPGA layout data after place and route,

as well as the original design schematic files. Then we can use the search function of the FPGA CAD tool to find the locations of the cells we have used for watermarking from schematic diagram. Then by using the embedding rules we can extract the watermark information from the cells' layout locations. The time to extract 8 bits of watermark from the FIR filter layout is less than 5 minutes and no extra software is needed.

Probability of Coincidence (Pu): This figure is decided by watermark length. For 128-bit watermark, the probability of coincidence is: 1/(2^128)=3.4*10^(-38).

Security Strength: For anyone who wishes to detect/remove/modify/forgery the watermark information, they have several ways to do the attack. The first way is to observe the FPGA layout directly and try to find the watermark inside the FPGA layout. This method is quite difficult with today's FPGA processing technology. The reason is by using this method the optical microscope must be used to take the photos of the FPGA layouts or using reverse analysis CAD tools to analysis the FPGA layouts. But the optical microscopes cannot easily be used to observe the layouts that are fabricated under 0.5um technology, which is the observation limit of normal optical microscopes. (The observation limit of optical microscopes is decided by the visible light's wavelength. By using shorter wavelength's light, like x-ray, the observation limit of the microscope can be extended. But such kind of x-ray microscopes are much more expensive. For today's FPGA technology, most products are fabricated using 0.13um technology and the most advanced devices using 0.09um technology, like IBM and Xilinx's products.) Even the attacker can find more advanced technology that allows them to observe the FPGA layouts directly; they also cannot see the logics under 5 levels of metal wirings. And not like ASICs, the wiring of FPGAs is naturally very messy, as we can see from the layout after place and route. So technically the reverse engineering by optical observation of FPGA layouts is really difficult.

The second way the attacker can select is to attack the FPGA programming data and try to reverse out the logic netlist, do the place and route again, with new location constrains. By this way they can remove the embedded watermark within FPGA layout. By doing so, the attacker firstly need to break the 3-key 128-bit AES encryption algorithm. There is no known efficient enough algorithm to break AES encryption algorithm so far. Even the attacker can find the keys and can recover the original FPGA programming data; they still need to extract the logic netlist by using the FPGA programming binary data. Since the FPGA programming data only provide the connection information of the FPGA, the attacker need the detail format knowledge of the FPGA bit-stream data to extract the logic netlist from the FPGA programming bit -stream data.

The FPGA factories keep this kind of knowledge confidential and do not expose to the public. And there is no hanker tool have been reported so far that can reverse the FPGA programming data to logic netlist by 100%. Some tools can reverse the logic netlist from FPGA programming bit-streams to 80% of the whole logic within reasonable computer hours. And the rest 15% is very difficult to extract and the final 5% is nearly impossible to extract.

Applied Area This method can be applied to FPGA implementations of DSP algorithms

Filter or DSP Algorithm Performance Degradation: There will not be any filter performance degradation, since we do not change the algorithm of the filter or DSP algorithms.

Probability of Detection Miss (Pm): From the watermark detection procedure, we can see that the probability of a detection miss is 0. Since by providing the complete design files including the logic netlist and placement files, route files, the watermark extractor can always extract the watermark by using the search and debug functions of the FPGA design tool.

Table 13 Comparisons Between FPGA Cell Location Watermarking and State-of-art FPGA/ASIC Implementation Watermarking Methods

	Using Spare LUT	Bit-stream Data Modifica-tion	Hierarchical Watermark-ing	Watermark-ing by Using Protocols	Finger-mark-ing	Proposal Two
Add Level	Layout	Layout	RTL	RTL	Layout	Layout
Embedding Cost	Medium	Medium	High	Medium	High	Medium
Design Overhead	Medium	Medium	Medium	Medium	High	Low
Extraction Cost	Medium	High	High	High	Low	Medium
Probability of Coincidence	Low	Low	Low	Low	Medium	Low
Security	Low	High	High	Medium	Low	High
Applied Area	FPGA	FPGA	ASIC (digital)	ASIC (digital)	ASIC (mix-signal)	FPGA

The proposed FPGA cell locations' watermarking scheme shows good watermarking performance comparing with current ASIC/FPGA watermarking schemes. Since the watermark is added at layout level, the lowest level of FPGA design, the robustness of watermark has been improved. The design overhead is low since the watermarking procedure is carried after the original design has been finished. The extraction cost is not that high, because the person to extract the watermark can use the search function of the FPGA design CAD tool. The probability of coincidence is quite low due to the large amount of watermark information we can embed into the FPGA layout. And no extra software is required for the watermark embedding and extraction, making this scheme fit for companies wish to watermark their design without buying new software. All in all, the proposed FPGA cell location's watermarking scheme shows good overall watermarking performance and may be commercially used for protection of today's FPGA designs.

Chapter 5: Summary of Contributions and Possible Future Work

In this thesis, two new watermarking schemes have been proposed. The first one is the filter coefficient modification watermarking. This scheme embeds watermark information into the filter's coefficients. The watermarked filter's coefficients are then embedded into the FPGA design by hardwiring or ROM programming. The second proposal is the FPGA layout cell locations' watermarking. This method embeds the watermark information into the FPGA cells' layout location coordinates. The simulation results show that the both schemes have relatively high performance and are very suitable for practical FPGA IP copyright protection.

The main contributions can be summarized as follows:

- Have extended the still image watermarking method of the pixel's LSB modification to filter and other DSP algorithms (at algorithm and lower levels). We have shown that the new method has much high security strength since the watermarked data can be hardwired or written into ROM.
- Have proposed a novel FPGA layout cell location watermarking scheme that can be used for FPGA design IP protection. It has been shown by the simulation results the proposed scheme has better watermarking performance, extremely low overheads and high security strength, compared to the previous FPGA methods.
- Have made in depth state-of-art literature survey in the areas of IP protection and watermarking technology.

Future work may include ASIC layout watermarking, FPGA watermark embedding optimization, printed circuit board (PCB) designs' protection, and FPGA/ASIC place and route algorithms that integrate watermark embedding and optimization procedures.

References:

[1] G.C.Langelaar, I.Setyawan, R.L.Lagendijk, "Watermarking Digital Image and Video Data, A State-of-the-Art Overview", *IEEE Signal Processing Magazine*, September 2000, pp. 20-46

[2] F.A.P.Petitcolas, R.J.Anderson, and M.G.Kuhn, "Information Hiding-A Survey", *Proceedings of the IEEE*, Vol 87,No.7, July 1999, pp.1062-1078

[3] D.L.Irby, R.D.Newbould, J.D.Carothers, J.J.Rodriguez, and W.T.Holman, "Low Level Watermarking of VLSI Design For Intellectual Property Protection", *ASIC/SOC Conference, 2000, Proceedings, 13th Annual IEEE International*

[4] E.Charbon, "Hierarchical Watermarking in IC Design", *In Proceedings, IEEE Custom Integrated Circuit Conference*, pp 295-298

[5] J.Lach, W.H.M.Smith, and M.Potkonjak, "Fingerprinting Techniques for Field-Programmable Gate Array Intellectual Property Protection", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* Vol 20, No.10, October, 2001,pp 1253-1261

[6] A.B.Kahng, J.Lach, W.H.Smith, S.Mantik, I.L.Markov, M.Potkonjak, P.Tucker, H.Wang and G. Wolfe, "Constraint-Based Watermarking Techniques for Design IP Protection", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* Vol 20,No.10, October, 2001, pp. 1236-1252

[7] A.B.Kahng, S.Mantik, I.L.Markov, M.Potkonjak, P.Tucker, H.Wang and G.Wolfe, "Robust IP Watermarking Methodologies for Physical Design", *Design Automation Conference 1998,Proceedings,* pp. 782-787

[8] T.Van and Y.Desmedt, Department of Computer Science, Florida State University, "Cryptanalysis of UCLA Watermarking Schemes for Intellectual Property Protection", *F.A.P. Petitcolas (ED.): IH 2002, LNCS 2578, 2003,* pp. 213-225.

[9] A.Rashid, J.Asher, W.H.Mangione-Smith and M.Potkonjak, "Hierarchical Watermarking for Protection of DSP Filter Cores", *IEEE 1999 Custom Integrated Circuits Conference,* pp. 39-42

[10] R.Chapman, T.S.Durrani, A.P.Tarbert, "Watermarking DSP Algorithms for System On Chip Implementation", *Signal Processing Division, Electronics, Circuits and Systems, 1999,Proceedings of ICECS'99,* The 6th IEEE International Conference on, Volume 1, pp. 377-380,Vol 1.

[11] R.Chapman and T.S.Durrani, "IP Protection of DSP Algorithms for System on Chip Implementation", *IEEE Transactions on Signal Processing,* Vol 48,No.3, March 2000.

[12] E.Charbon, "Intellectual Property Protection Via Hierarchical Watermarking", Cadence Design Systems Inc, San Jose, California, 1998

[13] E.Charbon and I.Torunoglu, "Copyright Protection of Design Based on Multi Source IPs", *Computer-Aided Design,* 1999,*Digest of Technical Papers, 1999 IEEE/ACM International Conference,* pp. 591-595

[14] E.Charbon and I.Torunoglu, "Watermarking Layout Toplogies", *Design Automation Conference 1999, Proceedings of ASP-DAC 99*, pp. 213-216,Vol 1.

[15] E.Charbon and I.Torunoglu, "Copyright Protection of Designs Based on Multi Source IPs", *Computer-Aided Design, Digest of Technical Papers, IEEE/ACM International Conference* , pp. 591 - 595,1999.

[16] I.Hong, Synopsys Inc, M.Potkonjak, Department of Computer Science, University of California, Los Angeles, CA, "Behavioral Synthesis Techniques for Intellectual Property Protection"

[17] A.T.Abdel-Hamid, S.Tahar and E.M.Aboulhamid, "IP Watermarking Techniques: Survey and Comparison", *Proceedings of the 3rd IEEE International Workshop on System-on-chip for Real-Time Applications*, 2003

[18] M.J.Wirthlin and B.McMurtrey, "IP Delivery for FPGAs Using Applets and JHDL", *Design Automation Conference, Proceedings,* 39th,page 2-7, 2002

[19] G.Bollano, G.Cesana, S.Claretto, L.Licciardi, M.Parlini, M.Turolla, "Merging Hardware and Software: Intellectual Property Cores for Internet Applications", G.*Bollano, S.Clarett, E.Filippi, A.Toriell, M.Turolla,* Custom Integrated Circuits Conference, Proceedings of the IEEE, pp. 537 - 540, 2000.

[20] J.Lach, W.H. M.Smith, M.Potkonjak, "Signature Hiding Techniques for FPGA Intellectual Property Protection", *Computer-Aided Design,* 1998.ICCAD98, *Digest of Technical Papers 1998, IEEE/ACM International Conference,* 1998, pp.186-189

[21] J.Lach, W.H.M.Smith, M.Potkonjak, "Robust FPGA Intellectual Property Protection Through Multiple Small Watermarkings", *Design Automation Conference, 1999,proceedings*, 36th, pp. 831-836

[22] I.Torunoglu and E.Charbon, "Watermarking Based Copyright Protection of Sequential Functions", *IEEE Journal of Solid-State Circuits,* Vol 35,No3, 2000, pp. 434-440

[23] A.L.Oliverira, "Robust Techniques For Watermarking Sequential Circuit Designs", *Design Automation Conference, 1999,Proceedings,* 36th, 1999, pp. 837-842

[24] *Virtex-II 1.5V Field-Programmable Gate Arrays, Advanced Product Specification,* Xilinx 2001

Appendix A Timing Report for the FIR Filter's Original Implementation

```
--------------------------------------------------------------------------------
Release 4.2i - Trace E.35
Copyright (c) 1995-2001 Xilinx, Inc.  All rights reserved.

trce -e 3 -l 3 -xml top top.ncd -o top.twr top.pcf

Design file:          top.ncd
Physical constraint file: top.pcf
Device, speed :        xc2v250,-6 (ADVANCED 1.96 2002-01-02)
Report level:          error report
================================================================================
Timing constraint: Default period analysis
 16528 items analyzed, 0 timing errors detected.
 Minimum period is   9.289ns.
--------------------------------------------------------------------------------
Timing constraint: Default net enumeration
 1726 items analyzed, 0 timing errors detected.
 Maximum net delay is   4.441ns.
--------------------------------------------------------------------------------
All constraints were met.
Data Sheet report:
--------------------------------------------------------------------------------
All values displayed in nanoseconds (ns)
Setup/Hold to clock multiplier_clk
---------------+------------+------------+
               | Setup to   | Hold to    |
Source Pad     | clk (edge) | clk (edge) |
---------------+------------+------------+
multipler_sclr |   2.836(R)|   0.660(R)|
xn<0>          |   0.432(R)|   0.000(R)|
xn<10>         |   0.350(R)|   0.000(R)|
xn<11>         |   0.169(R)|   0.013(R)|
xn<12>         |   0.846(R)|   0.000(R)|
xn<13>         |   0.770(R)|   0.000(R)|
xn<14>         |   0.440(R)|   0.000(R)|
xn<15>         |   0.085(R)|   0.097(R)|
xn<1>          |  -0.319(R)|   0.501(R)|
xn<2>          |   0.281(R)|   0.000(R)|
xn<3>          |   0.460(R)|   0.000(R)|
xn<4>          |  -0.567(R)|   0.749(R)|
xn<5>          |  -0.572(R)|   0.754(R)|
xn<6>          |   0.563(R)|   0.000(R)|
xn<7>          |  -0.386(R)|   0.568(R)|
xn<8>          |  -0.151(R)|   0.333(R)|
xn<9>          |  -0.119(R)|   0.301(R)|
---------------+------------+------------+

Setup/Hold to clock xlxn_150
---------------+------------+------------+
               | Setup to  | Hold to   |
Source Pad     | clk (edge) | clk (edge) |
---------------+------------+------------+
xlxn_138<0>    |   0.017(R)|   0.165(R)|
xlxn_138<10>   |  -0.008(R)|   0.190(R)|
xlxn_138<11>   |  -0.266(R)|   0.448(R)|
xlxn_138<12>   |   0.670(R)|   0.000(R)|
xlxn_138<13>   |  -0.438(R)|   0.620(R)|
xlxn_138<14>   |   0.029(R)|   0.153(R)|
xlxn_138<15>   |  -0.475(R)|   0.657(R)|
xlxn_138<1>    |   0.299(R)|   0.000(R)|
xlxn_138<2>    |  -0.093(R)|   0.275(R)|
xlxn_138<3>    |  -0.340(R)|   0.522(R)|
xlxn_138<4>    |  -0.564(R)|   0.746(R)|
xlxn_138<5>    |  -0.334(R)|   0.516(R)|
xlxn_138<6>    |   0.158(R)|   0.024(R)|
xlxn_138<7>    |  -0.385(R)|   0.567(R)|
xlxn_138<8>    |  -0.317(R)|   0.499(R)|
xlxn_138<9>    |  -0.116(R)|   0.298(R)|
xlxn_147       |   2.314(R)|   0.371(R)|
---------------+------------+------------+
Setup/Hold to clock xn_load_clk
---------------+------------+------------+
               | Setup to  | Hold to   |
Source Pad     | clk (edge) | clk (edge) |
---------------+------------+------------+
xlxn_39        |   0.184(R)|   0.000(R)|
xlxn_44        |  -0.852(R)|   1.034(R)|
xlxn_45        |  -1.138(R)|   1.320(R)|
xlxn_50        |  -0.834(R)|   1.016(R)|
xlxn_52        |  -0.426(R)|   0.608(R)|
xlxn_54        |   0.186(R)|   0.000(R)|
xn<0>          |   0.596(R)|   0.000(R)|
xn<10>         |  -0.359(R)|   0.541(R)|
xn<11>         |   0.456(R)|   0.000(R)|
xn<12>         |  -0.114(R)|   0.296(R)|
xn<13>         |   0.397(R)|   0.000(R)|
xn<14>         |   0.664(R)|   0.000(R)|
xn<15>         |   0.592(R)|   0.000(R)|
xn<1>          |   0.045(R)|   0.137(R)|
xn<2>          |   0.731(R)|   0.000(R)|
xn<3>          |   0.751(R)|   0.000(R)|
xn<4>          |  -0.380(R)|   0.562(R)|
xn<5>          |  -0.179(R)|   0.361(R)|
xn<6>          |  -0.378(R)|   0.560(R)|
xn<7>          |  -0.090(R)|   0.272(R)|
xn<8>          |   0.146(R)|   0.036(R)|
xn<9>          |  0.190(R)|  0.000(R)|
xn_preload     |  1.503(R)|  1.253(R)|
xn_serial_in   | -0.008(R)|  0.190(R)|
---------------+------------+------------+

Clock adder_clk to Pad
---------------+------------+
               | clk (edge) |
Destination Pad|   to PAD   |
---------------+------------+
yn<0>          |   8.597(R)|
yn<10>         |   8.855(R)|
yn<11>         |   8.671(R)|
yn<12>         |   8.921(R)|
yn<13>         |   8.560(R)|
yn<14>         |   8.405(R)|
yn<15>         |   9.125(R)|
yn<16>         |   9.430(R)|
yn<17>         |   8.860(R)|
yn<18>         |   8.736(R)|
yn<19>         |   8.698(R)|
yn<1>          |   8.633(R)|
yn<20>         |   8.653(R)|
yn<21>         |   8.586(R)|
yn<22>         |   9.175(R)|
yn<23>         |   8.519(R)|
yn<24>         |   8.557(R)|
yn<25>         |   8.612(R)|
yn<26>         |   8.697(R)|
yn<27>         |   8.040(R)|
yn<28>         |   8.620(R)|
yn<29>         |   8.536(R)|
yn<2>          |   8.233(R)|
yn<30>         |   8.326(R)|
yn<31>         |   8.988(R)|
yn<3>          |   8.449(R)|
yn<4>          |   8.567(R)|
yn<5>          |   8.701(R)|
yn<6>          |   8.518(R)|
yn<7>          |   8.333(R)|
yn<8>          |   8.589(R)|
yn<9>          |   8.346(R)|
---------------+------------+
Clock to Setup on destination clock adder_clk
---------------+---------+---------+---------+---------+
               | Src:Rise| Src:Fall| Src:Rise| Src:Fall|
Source Clock   |Dest:Rise|Dest:Rise|Dest:Fall|Dest:Fall|
---------------+---------+---------+---------+---------+
adder_clk      |    5.151|         |         |         |
multiplier_clk |    5.431|         |         |         |
---------------+---------+---------+---------+---------+

Clock to Setup on destination clock multiplier_clk
---------------+---------+---------+---------+---------+
               | Src:Rise| Src:Fall| Src:Rise| Src:Fall|
Source Clock   |Dest:Rise|Dest:Rise|Dest:Fall|Dest:Fall|
---------------+---------+---------+---------+---------+
multiplier_clk |    9.393|         |         |         |
xlxn_150       |    3.003|         |         |         |
xn_load_clk    |    3.135|         |         |         |
---------------+---------+---------+---------+---------+

Clock to Setup on destination clock xlxn_150
---------------+---------+---------+---------+---------+
               | Src:Rise| Src:Fall| Src:Rise| Src:Fall|
Source Clock   |Dest:Rise|Dest:Rise|Dest:Fall|Dest:Fall|
---------------+---------+---------+---------+---------+
xlxn_150       |    2.769|         |         |         |
---------------+---------+---------+---------+---------+

Clock to Setup on destination clock xn_load_clk
---------------+---------+---------+---------+---------+
               | Src:Rise| Src:Fall| Src:Rise| Src:Fall|
Source Clock   |Dest:Rise|Dest:Rise|Dest:Fall|Dest:Fall|
---------------+---------+---------+---------+---------+
xn_load_clk    |    2.991|         |         |         |
---------------+---------+---------+---------+---------
WARNING:Timing:2554 - Clock nets using non-dedicated resources were found in this
design. Clock skew on these resources will not be automatically  addressed during path
analysis. To create a timing report that analyzes
clock skew for these paths, run trce with the '-skew' option.
The following clock nets use non-dedicated resources:
 xn_load_clk_IBUF     adder_clk_IBUF multiplier_clk_IBUF
 xlxn_150_IBUF
Timing summary:
---------------

Timing errors: 0  Score: 0
Constraints cover 16528 paths, 1726 nets, and 2993 connections (100.0% coverage)
Design statistics:
 Minimum period:   9.289ns (Maximum frequency: 107.654MHz)
 Maximum net delay:   4.441ns

Analysis completed Tue Mar 09 21:29:03 2004
--------------------------------------------------------------------------------
```

Appendix B Timing Report for the FIR Filter Implementation with 8-bit Watermarking

```
Release 4.2i - Trace E.35
Copyright (c) 1995-2001 Xilinx, Inc.  All rights reserved.
trce -e 3 -l 3 -xml top top.ncd -o top.twr top.pcf
Design file:          top.ncd
Physical constraint file: top.pcf
Device,speed:          xc2v250,-6 (ADVANCED 1.96 2002-01-02)
Report level:          error report

WARNING:Timing:2491 - No timing constraints found, doing default enumeration.
Timing constraint: Default period analysis
 16528 items analyzed, 0 timing errors detected.
 Minimum period is   9.392ns.
Timing constraint: Default net enumeration
 1726 items analyzed, 0 timing errors detected.
 Maximum net delay is   4.441ns.
All constraints were met.
Data Sheet report:
All values displayed in nanoseconds (ns)
Setup/Hold to clock multiplier_clk
---------------+------------+------------+
               | Setup to   | Hold to    |
Source Pad     | clk (edge) | clk (edge) |
---------------+------------+------------+
multipler_sclr |   2.836(R)|   0.660(R)|
xn<0>          |   0.432(R)|   0.000(R)|
xn<10>         |   0.350(R)|   0.000(R)|
xn<11>         |   0.169(R)|   0.013(R)|
xn<12>         |   0.846(R)|   0.000(R)|
xn<13>         |   0.770(R)|   0.000(R)|
xn<14>         |   0.440(R)|   0.000(R)|
xn<15>         |   0.085(R)|   0.097(R)|
xn<1>          |  -0.319(R)|   0.501(R)|
xn<2>          |   0.281(R)|   0.000(R)|
xn<3>          |   0.460(R)|   0.000(R)|
xn<4>          |  -0.567(R)|   0.749(R)|
xn<5>          |  -0.572(R)|   0.754(R)|
xn<6>          |   0.563(R)|   0.000(R)|
xn<7>          |  -0.386(R)|   0.568(R)|
xn<8>          |  -0.151(R)|   0.333(R)|
xn<9>          |  -0.283(R)|   0.465(R)|
---------------+------------+------------+

Setup/Hold to clock xlxn_150
```

```
---------------+------------+------------+
               | Setup to   | Hold to    |
Source Pad     | clk (edge) | clk (edge) |
---------------+------------+------------+
xlxn_138<0>    |  0.017(R)|   0.165(R)|
xlxn_138<10>   | -0.008(R)|   0.190(R)|
xlxn_138<11>   | -0.266(R)|   0.448(R)|
xlxn_138<12>   |  0.838(R)|   0.000(R)|
xlxn_138<13>   | -0.423(R)|   0.605(R)|
xlxn_138<14>   | -0.025(R)|   0.207(R)|
xlxn_138<15>   | -0.219(R)|   0.401(R)|
xlxn_138<1>    |  0.299(R)|   0.000(R)|
xlxn_138<2>    | -0.093(R)|   0.275(R)|
xlxn_138<3>    | -0.424(R)|   0.606(R)|
xlxn_138<4>    | -0.564(R)|   0.746(R)|
xlxn_138<5>    | -0.334(R)|   0.516(R)|
xlxn_138<6>    |  0.158(R)|   0.024(R)|
xlxn_138<7>    | -0.385(R)|   0.567(R)|
xlxn_138<8>    | -0.317(R)|   0.499(R)|
xlxn_138<9>    | -0.116(R)|   0.298(R)|
xlxn_147       |  1.321(R)|   0.560(R)|
---------------+------------+------------+

Setup/Hold to clock xn_load_clk
---------------+------------+------------+
               | Setup to   | Hold to    |
Source Pad     | clk (edge) | clk (edge) |
---------------+------------+------------+
xlxn_39        |  0.184(R)|   0.000(R)|
xlxn_44        | -0.852(R)|   1.034(R)|
xlxn_45        | -1.138(R)|   1.320(R)|
xlxn_50        | -0.834(R)|   1.016(R)|
xlxn_52        | -0.426(R)|   0.608(R)|
xlxn_54        | -0.319(R)|   0.501(R)|
xn<0>          |  0.596(R)|   0.000(R)|
xn<10>         | -0.359(R)|   0.541(R)|
xn<11>         |  0.456(R)|   0.000(R)|
xn<12>         | -0.114(R)|   0.296(R)|
xn<13>         |  0.397(R)|   0.000(R)|
xn<14>         |  0.664(R)|   0.000(R)|
xn<15>         |  0.592(R)|   0.000(R)|
xn<1>          |  0.045(R)|   0.137(R)|
xn<2>          |  0.731(R)|   0.000(R)|
xn<3>          |  1.185(R)|   0.000(R)|
xn<4>          | -0.380(R)|   0.562(R)|
xn<5>          | -0.179(R)|   0.361(R)|
xn<6>          | -0.378(R)|   0.560(R)|
xn<7>          | -0.090(R)|  0.272(R)|
xn<8>          |  0.146(R)|  0.036(R)|
xn<9>          |  0.156(R)|  0.026(R)|
xn_preload     |  1.675(R)|  1.253(R)|
xn_serial_in   | -0.008(R)|  0.190(R)|
---------------+------------+------------+

Clock adder_clk to Pad
---------------+------------+
               | clk (edge) |
Destination Pad|   to PAD   |
---------------+------------+
yn<0>          |  8.597(R)|
yn<10>         |  8.855(R)|
yn<11>         |  8.671(R)|
yn<12>         |  8.921(R)|
yn<13>         |  8.560(R)|
yn<14>         |  8.405(R)|
yn<15>         |  9.125(R)|
yn<16>         |  9.430(R)|
yn<17>         |  8.860(R)|
yn<18>         |  8.736(R)|
yn<19>         |  8.698(R)|
yn<1>          |  8.633(R)|
yn<20>         |  8.653(R)|
yn<21>         |  8.586(R)|
yn<22>         |  9.175(R)|
yn<23>         |  8.519(R)|
yn<24>         |  8.557(R)|
yn<25>         |  8.612(R)|
yn<26>         |  8.697(R)|
yn<27>         |  8.040(R)|
yn<28>         |  8.620(R)|
yn<29>         |  8.536(R)|
yn<2>          |  8.233(R)|
yn<30>         |  8.326(R)|
yn<31>         |  8.988(R)|
yn<3>          |  8.449(R)|
yn<4>          |  8.567(R)|
yn<5>          |  8.701(R)|
yn<6>          |  8.518(R)|
yn<7>          |  8.333(R)|
yn<8>          |  8.589(R)|
yn<9>          |  8.346(R)|
---------------+------------+

Clock to Setup on destination clock adder_clk
```

```
---------------+---------+---------+---------+---------+
               | Src:Rise| Src:Fall| Src:Rise| Src:Fall|
Source Clock   |Dest:Rise|Dest:Rise|Dest:Fall|Dest:Fall|
---------------+---------+---------+---------+---------+
adder_clk      |    5.151|         |         |         |
multiplier_clk |    5.431|         |         |         |
---------------+---------+---------+---------+---------+

Clock to Setup on destination clock multiplier_clk
---------------+---------+---------+---------+---------+
               | Src:Rise| Src:Fall| Src:Rise| Src:Fall|
Source Clock   |Dest:Rise|Dest:Rise|Dest:Fall|Dest:Fall|
---------------+---------+---------+---------+---------+
multiplier_clk |    9.393|         |         |         |
xlxn_150       |    3.003|         |         |         |
xn_load_clk    |    3.187|         |         |         |
---------------+---------+---------+---------+---------+

Clock to Setup on destination clock xlxn_150
---------------+---------+---------+---------+---------+
               | Src:Rise| Src:Fall| Src:Rise| Src:Fall|
Source Clock   |Dest:Rise|Dest:Rise|Dest:Fall|Dest:Fall|
---------------+---------+---------+---------+---------+
xlxn_150       |    2.787|         |         |         |
---------------+---------+---------+---------+---------+

Clock to Setup on destination clock xn_load_clk
---------------+---------+---------+---------+---------+
               | Src:Rise| Src:Fall| Src:Rise| Src:Fall|
Source Clock   |Dest:Rise|Dest:Rise|Dest:Fall|Dest:Fall|
---------------+---------+---------+---------+---------+
xn_load_clk    |    2.804|         |         |         |
---------------+---------+---------+---------+---------+
WARNING: Timing: 2554 - Clock nets using non-dedicated resources were found in this
design. Clock skew on these resources will not be automatically addressed during path
analysis. To create a timing report that analyzesclock skew for these paths, run trce with
the '-skew' option.The following clock nets use non-dedicated resources:
 xn_load_clk_IBUF    adder_clk_IBUF multiplier_clk_IBUF
 xlxn_150_IBUF
Timing summary:
--------------------------------------------------------------------------------
Timing errors: 0  Score: 0
Constraints cover 16528 paths, 1726 nets, and 2995 connections (100.0% coverage)
Design statistics:
Minimum period:  9.392ns (Maximum frequency: 106.474MHz)
Maximum net delay:  4.441ns
Analysis completed Tue Mar 09 21:43:53 2004
--------------------------------------------------------------------------------
```

Appendix C Place and Route Report Before Watermarking of FIR Filter

```
------------------------------------------------------------------------
Release 4.2i - Par E.35
Copyright (c) 1995-2001 Xilinx, Inc.  All rights reserved.
Tue Mar 09 21:26:49 2004
par -f _par.rsp

Constraints file: top.pcf
Loading design for application par from file par_temp.ncd.
   "top" is an NCD, version 2.37, device xc2v250, package cs144, speed -6
Loading device for application par from file '2v250.nph' in environment
D:/Xilinx.
Device speed data version:  ADVANCED 1.96 2002-01-02.
Device utilization summary:
 Number of External IOBs          78 out of 92     84%
 Number of LOCed External IOBs    0 out of 78       0%

 Number of MULT18X18s              8 out of 24     33%
 Number of SLICEs                488 out of 1536   31%

Overall effort level (-ol):  5 (set by user)
Placer effort level (-pl):   5 (set by user)
Placer cost table entry (-t): 1
Router effort level (-rl):   5 (set by user)
Extra effort level (-xe):    0 (set by user)

Starting Clock Logic Placement.  REAL time: 12 secs
Finished Clock Logic Placement.  REAL time: 12 secs
Automatic resolution of clock placement was successful.
It was not necessary to constrain the placement of any of the logic driven by
the global clocks with the current clock placement.
========================================================================
## Automatic clock placement completed.
========================================================================
Starting clustering phase.  REAL time: 13 secs
Finished clustering phase.  REAL time: 15 secs
Starting Mincut Placer.  REAL time: 15 secs
Finished Mincut Placer.  REAL time: 16 secs
Dumping design to file top.ncd.
Dumping design to file top.ncd.
Total REAL time to Placer completion: 54 secs
Total CPU time to Placer completion: 53 secs
0 connection(s) routed; 2993 unrouted active, 7 unrouted PWR/GND.
Starting router resource preassignment
Completed router resource preassignment. REAL time: 55 secs
Starting iterative routing.
Routing active signals.
..............
```

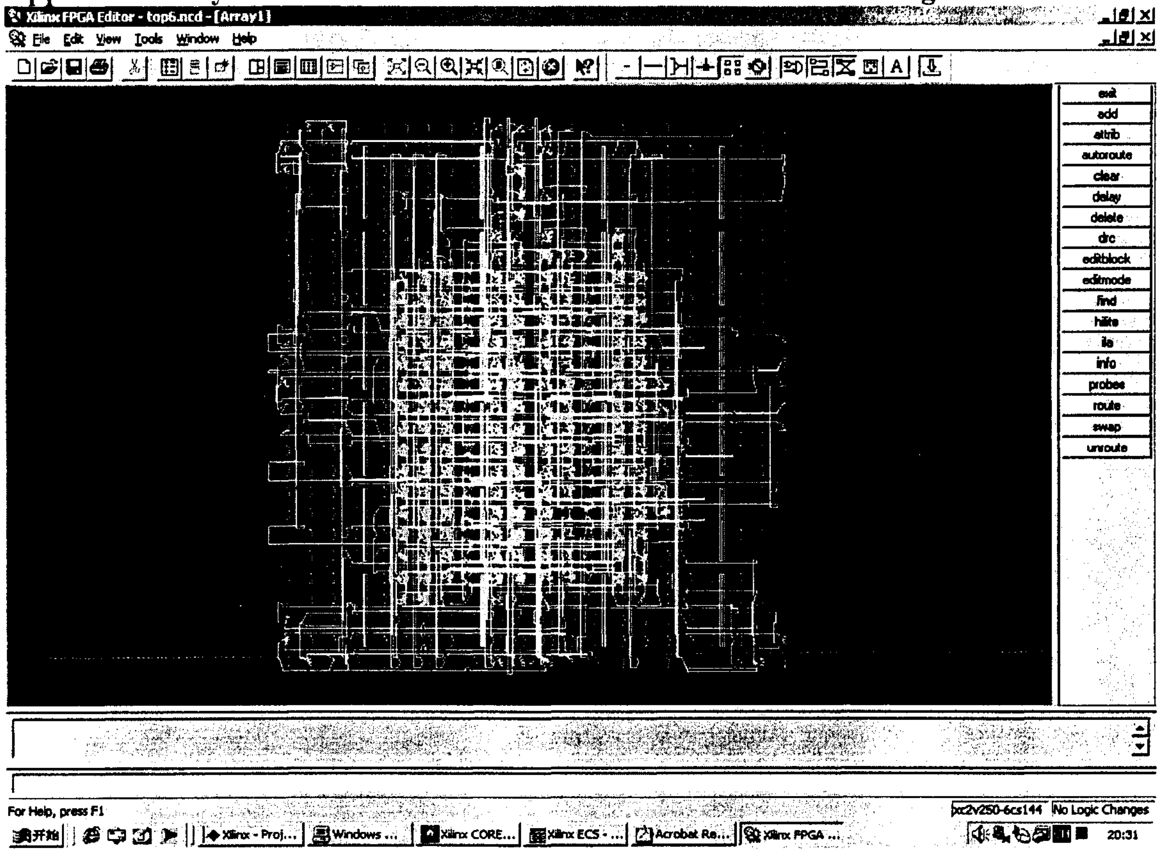
```
End of iteration 1
3000 successful; 0 unrouted; (0) REAL time: 1 mins 15 secs
Total REAL time: 1 mins 15 secs
Total CPU  time: 1 mins 15 secs
End of route.  3000 routed (100.00%); 0 unrouted.
No errors found.
Completely routed.

This design was run without timing constraints.  It is likely that much better
circuit performance can be obtained by trying either or both of the following:
  - Enabling the Delay Based Cleanup router pass, if not already enabled
  - Supplying timing constraints in the input design


Total REAL time to Router completion: 1 mins 16 secs
Total CPU time to Router completion: 1 mins 16 secs
Generating PAR statistics.
The Delay Summary Report
The Score for this design is: 169
The Number of signals not completely routed for this design is: 0
 The Average Connection Delay for this design is:        1.182 ns
 The Maximum Pin Delay is:                               4.441 ns
 The Average Connection Delay on the 10 Worst Nets is:   2.576 ns

   Listing Pin Delays by value: (ns)
    d < 1.00   < d < 2.00  < d < 3.00  < d < 4.00  < d < 5.00  d >= 5.00
   ---------  ---------  ---------  ---------  ---------  ---------
       1253       1421        223         93         10          0

Dumping design to file top.ncd.


All signals are completely routed.

Total REAL time to PAR completion: 1 mins 24 secs
Total CPU time to PAR completion: 1 mins 23 secs

Placement: Completed - No errors found.
Routing: Completed - No errors found.

PAR done.
```

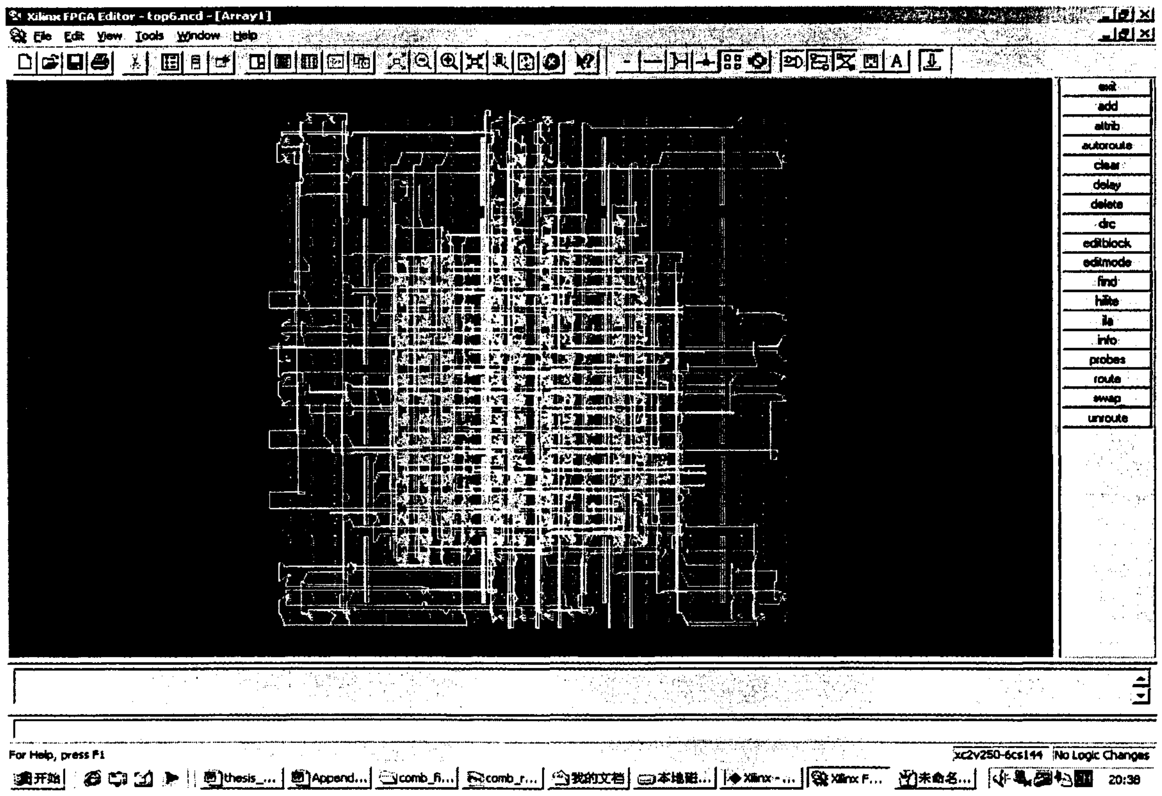
Appendix D Place and Route Report of FIR Filter with 8-bit Watermark

```
Release 4.2i - Par E.35
Copyright (c) 1995-2001 Xilinx, Inc.  All rights reserved.


Tue Mar 09 21:38:33 2004

par -f _par.rsp


Constraints file: top.pcf

Loading design for application par from file par_temp.ncd.
  "top" is an NCD, version 2.37, device xc2v250, package cs144, speed -6
Loading device for application par from file '2v250.nph' in environment
D:/Xilinx.
Device speed data version:  ADVANCED 1.96 2002-01-02.

Device utilization summary:
Number of External IOBs          78 out of 92     84%
Number of LOCed External IOBs   78 out of 78    100%
Number of MULT18X18s              8 out of 24     33%
Number of SLICEs               489 out of 1536   31%

Overall effort level (-ol): 5 (set by user)
Placer effort level (-pl): 5 (set by user)
Placer cost table entry (-t): 1
Router effort level (-rl): 5 (set by user)
Extra effort level (-xe): 0 (set by user)

Starting Clock Logic Placement.  REAL time: 23 secs
Finished Clock Logic Placement.  REAL time: 23 secs
Automatic resolution of clock placement was successful.
It was not necessary to constrain the placement of any of the logic driven by
the global clocks with the current clock placement.
=================================================================
## Automatic clock placement completed.
=================================================================
Dumping design to file top.ncd.
Starting Optimizing Placer.  REAL time: 29 secs
Optimizing
Swapped 0 comps.
Xilinx Placer [1] 301752   REAL time: 30 secs
Finished Optimizing Placer.  REAL time: 30 secs
Dumping design to file top.ncd.
Total REAL time to Placer completion: 31 secs
Total CPU time to Placer completion: 30 secs
```

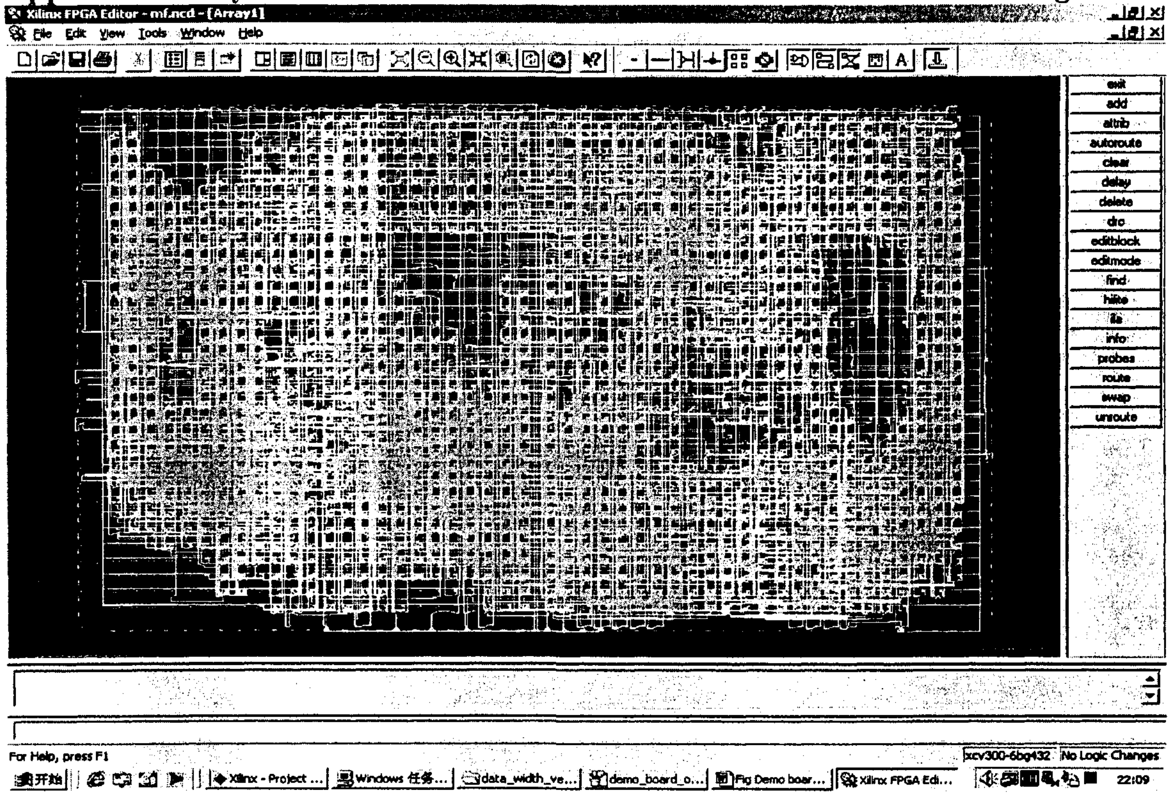
```
0 connection(s) routed; 2995 unrouted active, 7 unrouted PWR/GND.
Starting router resource preassignment
Completed router resource preassignment. REAL time: 32 secs
Starting iterative routing.
Routing active signals.
............
End of iteration 1
3002 successful; 0 unrouted; (0) REAL time: 52 secs
Total REAL time: 52 secs
Total CPU  time: 52 secs
End of route.  3002 routed (100.00%); 0 unrouted.
No errors found.
Completely routed.

This design was run without timing constraints.  It is likely that much better
Circuit performance can be obtained by trying either or both of the following:
  - Enabling the Delay Based Cleanup router pass, if not already enabled
  - Supplying timing constraints in the input design


Total REAL time to Router completion: 53 secs
Total CPU time to Router completion: 52 secs
Generating PAR statistics.
   The Delay Summary Report
   The Score for this design is: 167
The Number of signals not completely routed for this design is: 0
   The Average Connection Delay for this design is: 1.179 ns
   The Maximum Pin Delay is: 4.441 ns
   The Average Connection Delay on the 10 Worst Nets is: 2.463 ns

   Listing Pin Delays by value: (ns)
    d < 1.00   < d < 2.00  < d < 3.00  < d < 4.00  < d < 5.00  d >= 5.00
   ---------  ---------  ---------  ---------  ---------  ---------
      1251       1425        225         91         10          0

Dumping design to file top.ncd
All signals are completely routed.
Total REAL time to PAR completion: 1 mins
Total CPU time to PAR completion: 59 secs
Placement: Completed - No errors found.
Routing: Completed - No errors found.
```

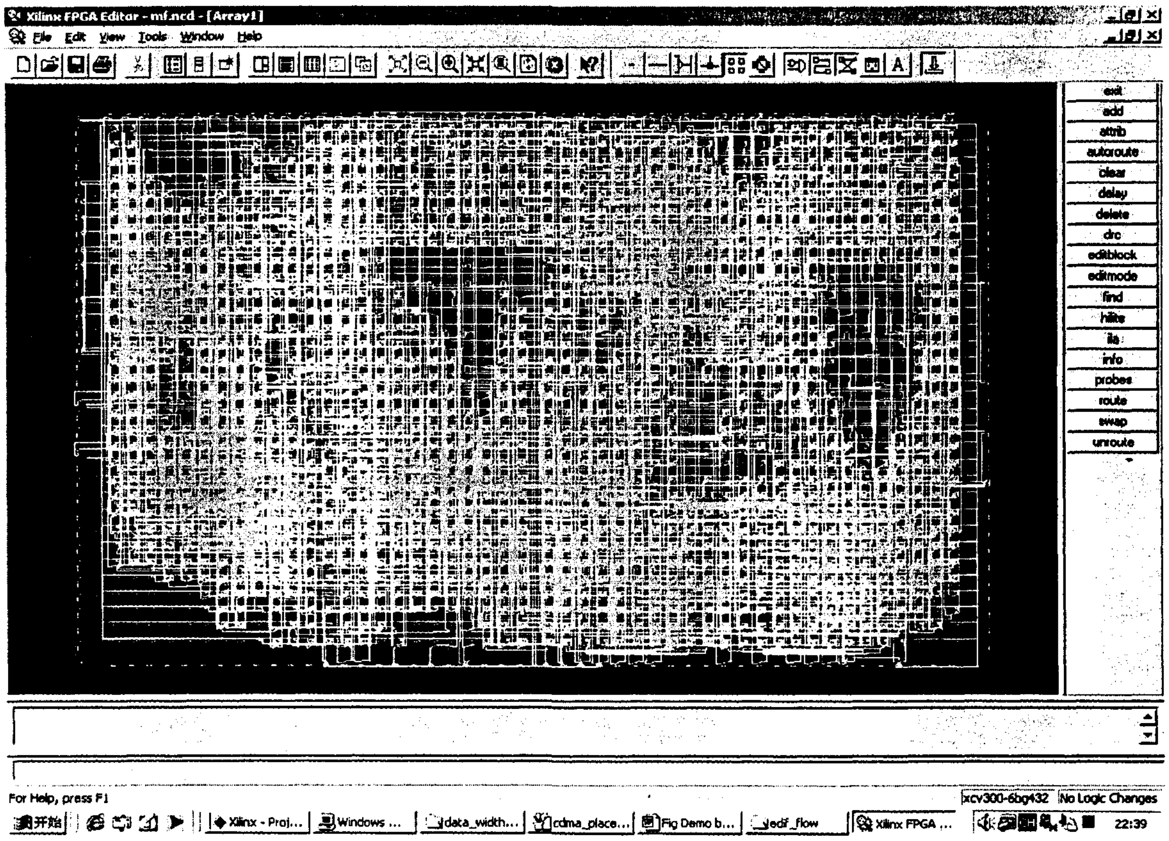
Appendix E Layout of Comb Filter Before and After Watermarking



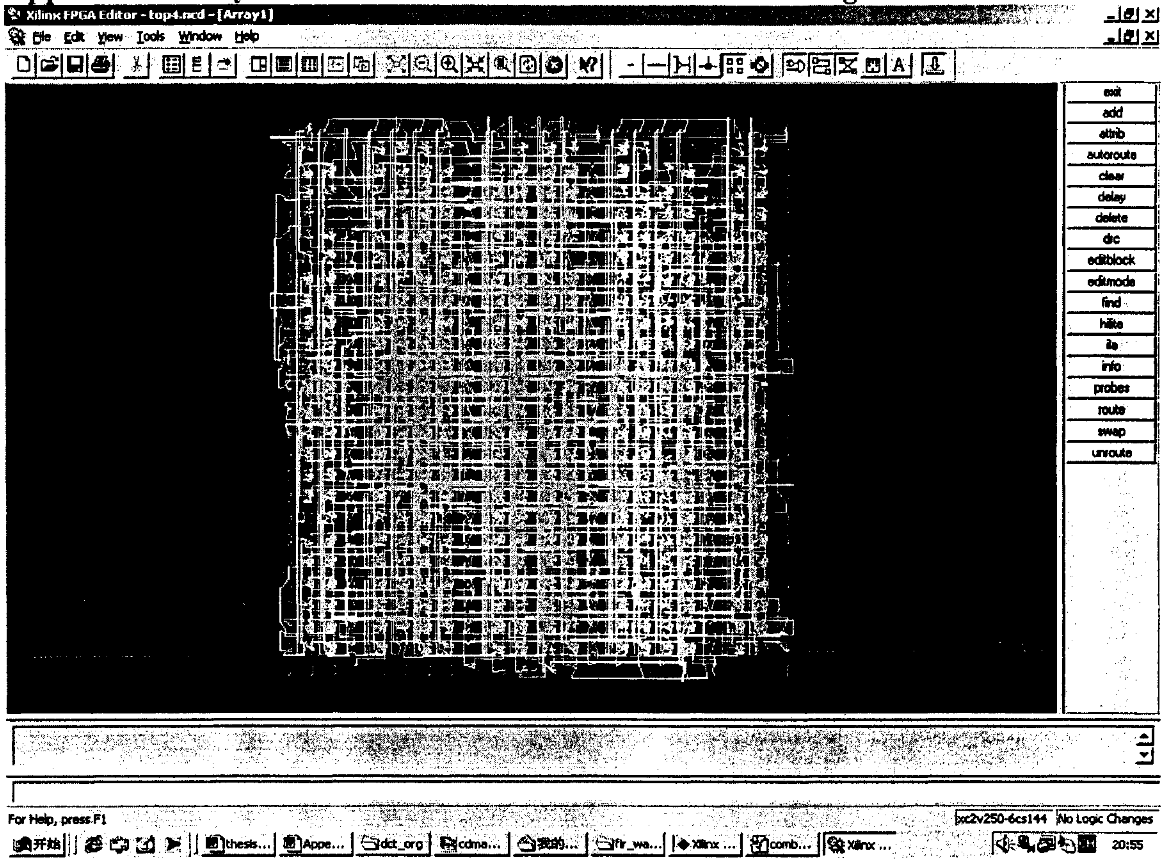


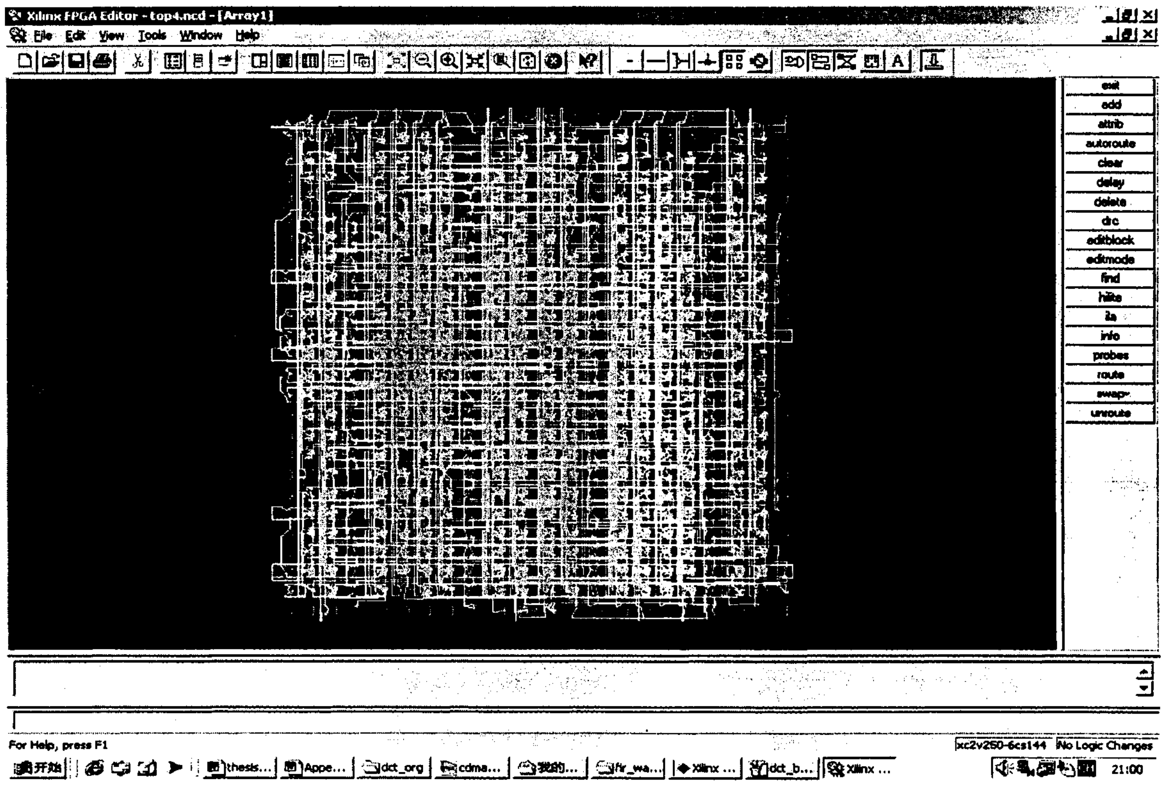
Appendix F Layout of CDMA Match Filter Before and After Watermarking





Appendix G Layout of DCT Before and After Watermarking





Vita Auctoris

Name:	Wei Dai
Place of Birth:	Wuxi, China
Year of Birth:	1974
Education:	M.A.Sc, University of Windsor, Canada 2002-2004
	B.Eng, Nanjing University of Science and Technology, Nanjing, China, 1992- 1996