Chapter 1

Steganographic Micro-architectures Employing FPGA

Magdy Saeb

Arab Academy for Science, Technology & Maritime Transport

Abstract: Real-time applications of steganography require that the processing delays are to be minimized. Therefore, hardware implementation is considered indispensable for this type of applications. In this chapter, we introduce concepts of steganographic micro-architectures for real-time data hiding employing Field Programmable Gate Arrays (FPGA). We examine video watermarking using FPGA, hardware wavelet-based data hiding, and Signature Hiding for FPGA intellectual property protection. Moreover, we discuss micro-architectures used for MPEG-4. Micro-architectures for Steganalysis and subliminal steganographic channels in cognitive radio are also discussed. We provide an explanation of the complementary relation between Steganography and Cryptography. In this respect, we show that some micro-architectures can be used for both Steganography and Cryptography data security techniques. Finally, we provide clarifications and a brief description of the FPGA technology.

1.1 Introduction and overview

Steganography hides the existence of a message while Cryptography hides the meaning of the message. Both techniques are complementary and both are essential requirements for data security. Real-time applications, such as audio and video-based data hiding, require that the encountered processing delays should be kept to a minimum. Hence, hardware implementation of Steganographic techniques is considered indispensable for this type of applications. Moreover, software implementation usually requires an added special purpose processor. This processor is usually a Digital Signal Processor chip. However, adding a steganographic component will consume only a relatively small implementation silicon area. In a large number of cases of consumer electronics, the cost, area, execution speed and power consumption comparisons are supporting the hardware solution.

One of the most commonly used media for hardware implementation is Field Programmable Gate Arrays or what is known as FPGA Technology. In the following sections, we discuss steganographic FPGA-based micro-architectures used for real-time data hiding. These micro-architectures cover a wide area of applications such as wavelet-based hiding, steganographic context techniques, video watermarking, signature hiding for intellectual property protection and video steganography.

Although steganography is distinctly separate from cryptography, we should follow the longtime advice of A. Kerckhoffs and assume that the only unknown to the opponent is a secret key. This idea protects the user from the misapprehension of applying “security by obscurity”. History has repeatedly shown that this idea is destined to fail. Hardware-based steganography and watermarking are designed keeping in mind that they will
be examined by a well-informed expert opponent. Contemporary designers rely on a secret hiding key rather on the hiding technique itself.

Chapter outline

A steganographic shuffler and hiding algorithm are presented in section 1.2. Section 1.3 discusses a micro-architecture for steganographic context. Section 1.4 examines video watermarking using FPGA. Section 1.5 considers Wavelet data hiding using Achterbahn-128. Section 1.6 discusses Zero overhead watermarking technique for FPGA designs. In section 1.7 Signature hiding techniques for FPGA intellectual property protection are discussed. Section 1.8 provides some details regarding an FPGA watermarking micro-architecture for MPEG-4. A micro-architecture for Steganalysis is discussed in section 1.9. Optimized sub-channels, or subliminal steganographic channels, in cognitive radio is explained in section 1.10. Section 1.11 provides a description of a micro-architecture that can be used for both steganography and cryptography. Finally, section 1.12 provides some notes and clarifications.

1.2 Steganographic shuffler

This shuffler design approach aims at spreading the message in an even manner over the entire multimedia cover file to emulate communication white noise [FASA04]. The shuffler randomly hides a number of bits less than or equal to the cover size with no possible location collision. However, increasing the hidden number of bits will eventually expose the existence of the message. The maximum number of bits that can be hidden in a cover, without perceptually detecting the existence of a hidden message, is called the channel or cover capability. We are using the term “channel capability” [MOCN02] instead of “channel capacity” to distinguish it from Shannon’s channel capacity based on the entropy concept [BRFO05]. To explain the method, we assume that sender Alice wants to send a hidden message to receiver Bob and both share a common secret key. In addition, we assume that the cover size is n Kbytes and we are hiding only one bit per octet of the cover, then the maximum message size is n/8 Kbytes or n Kbits. The term octet is used instead of byte since it has become customary to associate a byte with one text character rather than just eight consecutive bits. The shared key between Alice and Bob should be large enough to generate unique random addresses at least equal to the message size in bits. However to reduce the required number of addresses and the key space, two message bits are hidden in two consecutive cover locations. Therefore, if the message length is, say n bits, the required addresses or sub-keys will be n/2. The sub-key generator should provide a pseudo random set of sub-keys. In this case, a hash function can be utilized. Nevertheless to simplify the hardware, the designers developed their own simple sub-key generator and tested the output for the required degree of randomization. The conceptual block diagram is shown in Figure 1.1.

![Figure 1.1: The Shuffler conceptual block diagram [FASA04]](image-url)
The “Embed” block generates the stego-word. Each word is composed of two message bits and 14 cover bits. The two message bits are located in bit locations 0, and 8 respectively as shown in Figure 1.2. The address generator generates an address based on the input 8-bit sub-key.

![Figure 1.2: The Embed block conceptual diagram [FASA04]](image)

There are three different modes of operation of the shuffler: regular, segmented and cover-dependent shuffling. Cover-dependent shuffling has one or two-segment type. In each case, the effective address is generated differently. The results of hiding in each case are shown in Figure 1.3.

![Figure 1.3: Results of hiding in various modes of address generation [FASA04]](image)

The sub-key generator, as stated before, could have been based on repeatedly using a hash function and changing its input. The sub-key generator is expected to provide addresses that are randomly distributed, access most blocks of the segment and generate consecutive addresses that are relatively as far as possible.
1.2.1 The algorithm description

In the following few lines, we show the algorithm formal description for the hiding processes [FASA04]. The algorithm can be applied to video frames, audio files or any type of covers to hide a given message. It requires hiding uses a secret key known only to the sender and receiver. Put differently, given a message, the aim of the algorithm is to hide this message into a cover such that even if an attacker detects the existence of the message he or she will not be able to recover it without the secret key that is known only to sender and receiver. This explanation, as seen by some, may overlap with the idea of water marking, however, the algorithm is designed to practically “hide the contents” of a message. In other words, we have two levels of security and we are expanding the idea of steganography by hiding the “message existence and contents” rather than the “message existence only”. A screen shot of the software implementation for image and audio files is shown in Figure 1.4.

Algorithm: STEGOSHUFFLER

INPUT: Message M, Cover C, Key K, StateRegister SR
OUTPUT: Modulated Cover CM

Algorithm body:
Begin
1. Load a block of the message Blki into the message cache MC:
   Blki[M] → [MC];
2. Load Key into the key Cache:
   K → [KC];
3. Generate an address Adi;
4. Address memory to get one cover word CW:
   M[Adi] → CWi;
5. Hide two message bits (mi, mi+1) by replacing (C0, C8) in the cover word CW with (M0, M1+1):
   C[15:9], M1+1, C[7:1], M0 → CM;
6. Write back steganographic word:
   CMi → CWi;
7. If message cache is not empty:
   7.1 Circulate key cache one bit right: Circ1R [KC];
   7.2 Shift message cache one bit right:
      Shif1R [MC];
   7.3 Goto 3:
      GenerateAddressState → SR;
8. Else if message cache is empty:
   8.1 Load next block into message cache:
      Blki+1[M] → [MC];
   8.2 Goto 3:
      GenerateAddressState → SR;
Else if message is finished then halt;

All rights reserved ©2010 Magdy Saeb
End Algorithm.

We have applied this technique to different media files particularly audio files. Testing the resulting audio file by playing it back after hiding is performed, no perceptual audio changes were detected as long as the original sounds were present. However, it was noticed that in audio media covers with relatively long low volume or silent segments, some alteration noise can be detected by a keen and experienced listener.

**Figure 1.4:** A screen shot of the software implementation of the algorithm for image and audio files [FASA04]
1.2.2 The micro-architecture

The architecture is divided into an embedder processor and an SDRAM controller as shown in Figure 1.5. The embedder processor, depicted in this Figure, issues read and write commands to the memory, which are processed and reformatted by the SDRAM control and waits for a confirmation from memory to ensure stabilized output. The controller halts the process when hiding is complete. In the next sub-sections we discuss the various building blocks of the micro-architecture.

The Embedder processor, the Address Generator, the Memory unit, the Shuffler, the Address Extender and the Control Unit are discussed in some detail. The embedder processor generates addresses to initially cache the message and key from memory. It also generates addresses to access the cover randomly for message bit hiding. The embedder processor is composed of an address generator module, key cache, memory cache, key cache counter, message counter, message cache counter, message pointer, stegoblock, address multiplexer, control unit, and status register.

![Figure 1.5: The conceptual block diagram of the micro-architecture [FASA04]](image)

In the following few lines, a brief discussion of each of these components is provided. The address generator is composed of a shuffler, a block pointer memory and a shift & concatenate unit. The address generator receives an eight bit key and outputs an address of 17 bits as shown in Figure 1.6.
We have chosen 17 bits only in order to access an image of size no more than 128 Kbytes. It is a common size where the size of video frames, most probably, never exceeds. Certainly this depends on the video quality and the employed compression technique. The status register indicates whether to use cover information in address generation. The cover is logically divided into eight segments. The status register indicates also whether to hide in all or some of the segments. The Block Pointer Memory module consists of 64 eight-bit counters. The module takes six bits as an input to decide which one of the counters is to be incremented. The outputs of all counters are concatenated to form 512 bits and sent to the shuffler. The Shuffler module receives 512 bits from the block pointer memory module. Based on the key, it selects one of 64 pointers to be transmitted to the shift and concatenate unit. Each pointer is eight-bit in length. Therefore, the address space for each pointer is 256 words. These 256 words are taken as one block. Therefore, if each time the octet generated from the key is different from the one generated before, then the message bit will be inserted into a different block in the image. As there are only 64 pointers, only 64 blocks can be addressed. This means that only 64x256 words can be used for hiding. This problem was overcome by using the upper bits of the octet generated from the key as a segment selector. Each segment is 16384-word large.

As a result of this improvement, the message bits may be 16384 words apart in the best case and one-word apart in the worst case. This worst case will happen if a large number of octets in the key are repeated. Therefore, we have developed a short program for generating a key that covers the whole cover and selects the appropriate data and addresses in the right state to be sent to the memory. Some of the generated addresses by the different modules in the organization are less than 23 bits, which are needed to address the SDRAM. Therefore, an Address Extender is used to unify the output size to 23 bits. Finally, the control signals are generated in the hardwired control unit and provide control inputs for all integrated modules. The block diagram of the control unit and other major components is shown in Figure 1.6.

![Figure 1.6: The details of the shuffler micro-architecture [FASA04]](image)
This design is implemented and downloaded on the FPGA Spartan 2S100TQ144-6 device. As shown in Figure 1.7, on the right hand side, the design placed and routed on the FPGA chip. On the left hand side of the same Figure, one observes two images; a cover (a) and the modulated cover image (b) with no visible artifacts. Moreover, the Laplacian Filter, based on computing the average of the four neighboring pixels, is computed for both images with no traces of hiding activities.

Figure 1.7: The floor plan of the implemented stego-micro-architecture (left hand side) along with the cover (a) and the resulting modulated image (b) as well as the Laplacian filter outputs for both images and the filter output for the distorted cover (c), (d), and (e) [FASA04]

In summary, the address generator provides an output every one clock cycle. This is a major advantage as compared to SHA-based algorithms that requires 210 cycles or MD5 designs with 342 cycles. The shuffler design is a conceptually developed hardware that provides the required randomization in the embedding process. The shuffler operates in parallel with the hiding module. This saves about 25 ns for each hidden bit. The address generator is capable of generating an address in a 32,768-byte block or in multiple of these blocks based on the user preference. This allows the user to efficiently handle different image sizes. This address generator design is particularly suitable for a special-purpose processor design since it needs large sizes of busses, like 64-bit and 512-bit busses, which cannot be supported by general-purpose processors. The address generator can generate various sequences of addresses for different frames from a single 32-byte key by xoring with the cover. This is an essential requirement for video hiding schemes, since it is not realistic to ask the user for a new key for each video frame. There is a large number of testing procedures for obscurity
that are called Steganalysis Techniques. However, by proper choice of the key, one can show that the approach provides acceptable degree of data hiding with minimal distortion of the cover. This was proven utilizing the Laplacian Filter Technique. There is variation in the Laplacian filter outputs and no perceptible artifacts in the stego image when compared to the cover image. Therefore, one can conclude that the approach is refined enough to escape the watchful eyes of a passive adversary.

1.3 Architecture for steganographic context

The “ConText” technique utilizes the image noisy regions and those with abrupt gray-level changes to hide information [HFUC08]. Some researchers relate these areas of abrupt changes to edge detection where the hiding will largely take place. The messages hidden in these regions are quite difficult to detect. However, the process to locate these regions is highly repetitive and computationally expensive. To overcome these difficulties, the technique is implemented using FPGA. Accordingly, the method provides a high throughput and reduced computational time. The procedure [HFUC08], conceptually illustrated in Figure 1.8, can be summarized as follows:

Algorithm Steganographic ConText

INPUT: Image file in n-bit blocks
SUMMARY: Produces blocks with a hidden message in most noisy locations.
1. The image is divided into non-overlapping blocks of, say, 3 x 3 pixels.
2. Each 3 x 3 block is subdivided into four 2 x 2-pixel sub-blocks.
3. Each sub-block is considered valid if there are at least three different values of gray scale levels. (These levels, using 8-bit gray scale, are from 0 to 255. These are measures of the monochromatic light intensity of each pixel.)
4. The message is inserted in the 3 x 3 block center, if the four sub-blocks are valid.
5. After the hiding, the validity of the four sub-blocks is verified once more. If one or more sub-block is not valid after hiding, then the inserted bit is discarded and this part of the process is repeated again for hiding this bit to avoid losing information during the recovery process.

Figure 1.8: The hiding process where b, c, d, e are the checked four 2 x 2 sub-blocks. [HFUC08]
1.3.1 The micro-architecture

A top-down methodology was used to develop the FPGA micro-architecture. The micro-architecture Register Transfer Language (RTL) representation, resulting from using Altera, Quartus development software, is shown in Figure 1.9.

![Figure 1.9: The ConText technique RTL block diagram [HFUC08]](image)

In this implementation, the input and output ports are: **DIN** is the input of the pixel value of the cover image, **Information** is the message to be hidden, **CLK** is the clock input port and **RST** is the reset input. The **Output** port output the central pixel value of the 3 x 3 matrix after inserting the information in the in the cover object. **InfOutput** provides the extracted message of the central pixel or stego-object. **SHide** is the hiding selection activation port. **Conta** controls the input of the 8-bit message that will be hidden in the cover object. Figures 1.10 provides the details of the register file and the comparator RTL representations respectively. Figure 1.11 displays the details of the ConText sub-block.
**Figure 1.10**: The register file and the ConText block RTLs [HFUC08]

**Figure 1.11**: The ConText sub-block RTL [HFUC08]
1.3.2 Implementation results

The micro-architecture was implemented using Altera Cyclone II EP2C35F672C6 device. At a 106 MHz maximum operating frequency, the authors claim a throughput that is about 61.54 Mbps. The cover and the output stego-images are shown in Figure 1.12. The medical image shown below is given as an example only since any alterations can result in a partial loss of information leading to faulty diagnosis by the physician.

![Input cover images and output stego-images](HFUC08)

This micro-architecture, as stated by the authors, performs eight comparisons in a single clock cycle. Therefore, it is not likely that this hardware implementation can be outperformed by a software one. This Figure clearly demonstrates that the embedding process showed no perceptual traces or artifacts in the cover image. The micro-architecture, when integrated with other hardware applications, is expected to prove its validity.

1.4 Video watermarking using FPGA

Video watermarking, similar to still image watermarking, has visible and invisible types [CRUZ06]. Visible watermarking corresponds to traditional paper-based watermarking. Digital processing is required to retrieve the invisible one. If the original image is available, the effect of watermarking can be exposed by deducting the original image from the water marked one. However, the result may be different from the original watermark. In this respect, there are several types of digital watermarking schemes that are summarized in Figure 1.13.

![Various types of watermarking schemes](CRUZ06)
Capturing and then processing a video stream frame by frame, one can realize that video watermarking is an extension of still image watermarking. However, there are some differences. These differences are: no visible or audible alterations is allowed in the playback of video recordings, no effect on the compressibility of the digital contents, provides reliable detection, the implemented hardware should be of low cost, should have no effect on bit rate, can be performed directly on the compressed file, the method used should provide manageable time and space complexity and finally unauthorized removal is interdicted [CRUZ06].

1.4.1 Hardware implementation

Until recently, as shown in [CRUZ06], few hardware implementations were available. Most of the hardware was implemented in ASIC designs. However, the advances in FPGA technology have dramatically changed this situation. Software implementation usually requires an added a special digital signal processor. On the other hand, in the hardware implementation, the added watermarking component will assume only a relatively small implementation area. In a large number of studied cases of consumer electronics, the cost, area and power consumption comparisons are backing a hardware solution. In the next few lines, we discuss some of these watermarking implementations. This comparison is certainly based on commercial applications excluding unreliable freeware releases.

1.4.2 Just Another Watermarking System (JAWS)

The embedder and detector JAWS [CRUZ06] is a 0.18μm CMOS technology implementation. One of its advantages is that it works on raw uncompressed video data. This allows the user to freely select his or her own compression algorithm. The implementation has a pipelined architecture and Fast Fourier Transform (FFT) processing core. The results provided a watermarking of video streams at a rate of 30 frames per sec and 320 x 320 pixels per frame. The chip is capable of operating at 75 MHz and process a peak pixel rate of over 3 mega pixels per sec. The watermark consumes four bits per frame. The power consumption for the embedder is 60 mW and for the detector is 100 mW. Figure 1.14 provides the JAWS Embedder conceptual block diagram.

![Figure 1.14: JAWS Embedder Implementation conceptual block diagram [CRUZ06]](image-url)
1.4.3 Mohanty FPGA implementation

An FPGA-based implementation of an invisible, robust, spatial domain and still-image watermarking encoder is presented in [CRUZ06]. The hardware-based watermarking system can be implemented on an FPGA board, a TriMedia DSP, or a custom integrated circuit. The selection may be reduced to choosing an FPGA or an integrated circuit implementation. The watermarking encoder consists of a watermark generator, watermark insertion module, and a controller. The invisible watermarking algorithm inserts pseudorandom numbers to a host data. Synthesis was performed with SynplifyPro and simulations were run on ModelSim software. The FPGA device used was the Xilinx Virtex2 XCV50. It was operated at 50MHz frequency. The results show an execution time of 19.842 ns for the overall process. The block diagram of the architecture is shown in Figure 1.15.

![Figure 1.15: Block Diagram of the Mohanty FPGA Implementation. [CRUZ06]](image)

1.4.4 Hardware watermarking for surveillance systems

The system architecture developed for this purpose is demonstrated in Figure 1.16, shown below. The architecture is based on the data flow diagram that is shown in this Figure. The processing is performed on an 8 x 8 pixel block size basis. The system uses a pipelined architecture to process eight numbers in parallel. Several 8 x 8 blocks of random numbers are generated and act as pseudo-noise signal that is used for modulating the watermark bit.
Once the 64 different random numbers are generated, they are transformed using DCT and stored in a RAM. The DCT core processes a block of pixels and the result is quantized using the quantization matrix values (Q ROM). The contents stored in the RAM are then added to the quantized values if they are nonzero values. The pipelined design allows parallelization in-time for various computations. The respective computational steps are shown in Figure 1.17.

Figure 1.16: System Architecture of the surveillance water marking system [CRUZ06]

Figure 1.17: The computational steps of the system [CRUZ06]
The pseudo-random generator is implemented using a Linear Feedback Shift register (LFSR), as shown in Figure 1.18. It is a 13-bit register producing a sequence space of 8191 bits.

![Figure 1.18: The pseudo random generator](image)

The following step is to compute the Discrete Cosine Transform (DCT). Figure 1.19 illustrates the basic building blocks of this DCT. Quantization is performed following this step. The high frequency components are removed from the DCT coefficients. The process provides a compressed version of the original 8 x 8 block. Finally, the watermark addition is performed by adding the contents of the RAM to the quantized DCT coefficients of the pixels using a 12-bit adder. A control unit, previously shown in Figure 1.16, is used to organize the whole process using timing and control signals.

![Figure 1.19: The block diagram of the DCT](image)

The details of the DCT are obtained from [CRUZ06]. Figure 1.20 illustrates some of the DCT core details and the Rom-and-Accumulate structure (RAC). For more details, the interested reader is referred to [CRUZ06].
The FPGA performance, as reported in [CRUZ06], is as follows:

Latency for the pipelined-FPGA design is found to be 372 clock cycles. A number of 212 clock cycles is required for additional outputs. The time required to mark one frame for an M x M pixel resolution is given by:

\[
\text{Tot}_\text{Cycles}_\text{for1_frame} = 1 \times \text{latency} + \left(\frac{N \times M}{(8 \times 8) - 1}\right) \times \text{Throughput}
\]

1.4.1

As an example, using a 640 x 480 video, the number of cycles it would take to watermark one frame would be 1017760 clock cycles. The time it takes to watermark one frame can be obtained by multiplying the number of clock cycles by the clock period. The maximum clock frequency is 60 MHz. Thus, the minimum clock period is 16.66 ns. This results in 0.01696 seconds per frame. Therefore, the FPGA implementation can mark about 59 frames per second at the maximum clock frequency. For a standard 30 fps video the FPGA circuit duty cycle will be approximately \(\frac{1}{2}\). The power dissipation was reported to be about 90 mW. To achieve more power savings, the system frequency can be lowered up to 30 MHz to process about 30 fps. The cost of FPGA implementation is much less as compared to DSP implementations.

1.5 Wavelet data hiding using Achterbahn-128

There are two major categories of data hiding; hiding in the spatial domain or hiding in the frequency domain. The latter provides better security against passive attackers whose major objective is to detect the existence of a hidden message. In this approach [MDDE07], the message is embedded in the wavelet frequency domain by modifying a set of selected wavelet coefficients of the host image. Moreover, the utilized variable data size is
encrypted using Achterbahn-128 stream cipher before embedding takes place. The system is implemented using FPGA.

In this approach, a single-dimensional discrete wavelet transform, sometimes viewed as a repeated filter bank algorithm [MDDE07], is utilized. The input is transformed with high and low-pass filters. The low-pass filter branch generates the running average or what is known as the “trend” Discrete Wavelet Transform (DWT) coefficients of the signal. The high-pass branch generates the running difference or the “fluctuations” of the DWT coefficients. As the filter pair processes the signal, the output is carried off by a factor of two. The Haar Wavelet Transform, used in this approach, is one of these DWT techniques. The Haar transform can be summarized as follows:

Assume a discrete signal, or an analog signal \( g(t) \) that occurs at discrete instants. In addition, assume equally-spaced samples which number is \( 2^n \) where \( n \) is the number of bits used to represent the number of samples, then

\[
f = (f_1, f_2, \ldots, f_N),
\]

1.5.1

Where \( f_i = g(t_i) \ \forall \ i=1,2,\ldots, N. \)

Then the running averages are given by:

\[
C_m = (f_{2m-1} + f_{2m})/\sqrt{2},
\]

1.5.2

And the running differences are given by:

\[
d_m = (f_{2m-1} - f_{2m})/\sqrt{2},
\]

1.5.3

Where \( m = 1,2,\ldots, N/2. \)

The Haar wavelet transform is symbolically given by:

\[
f \big|^{H_1} \Rightarrow (C^2 \mid d^2)
\]

1.5.4

The energy is conserved after the transform is performed on the signal. The transform is sometimes called the “data microscope” since it reduces the bandwidth to half its original size and amplifies the signal trends by \( \sqrt{2}. \)

Filtering the signal controls the resolution of the signal, while the sub-sampling process controls the scale. Scale and frequency are inversely proportional such that higher frequencies correspond to lower or finer scales. On the other hand, lower frequencies correspond to higher or coarser scales. The filters separate the frequency bandwidth. Therefore, the filter pairs produce different resolutions, or levels, of detail. The mean coefficients are stored in the first half of the space, and the detail coefficients are stored in the latter half. The mean coefficients are then processed again through the same set of filters producing a second set of average and detail coefficients. This DWT decomposition of the signal proceeds until the sought after scale is achieved. Two-dimensional signals, such as images, are transformed using the two-dimensional DWT. Given a two-dimensional array of samples, the rows of the array are processed first with only one level of decomposition. This essentially divides the array into two vertical halves, with the first half storing the average coefficients, while the second vertical half stores the detail coefficients.
This process is repeated again with the columns, resulting in four sub-bands within the array defined by filter output. Figure 1.21 shows a single-level decomposition using the two-dimensional DWT. This process results in the well-known four classes of coefficients; the \((HH)\) coefficients represent diagonal features of the image, whereas \((HL\) and \(LH)\) reflect vertical and horizontal information. At the coarsest level, we also keep low pass coefficients \((LL)\) [MDDE07].

![Image of Haar wavelet signal decomposition]

**Figure 1.21:** The Hierarchical representation of the Haar wavelet signal decomposition [MDDE07]

**Algorithm:** Embedding Data in Wavelet Transform

**INPUT:** message and cover image file

**SUMMARY:** The message is encrypted and embedded in the wavelet domain by modifying selected Haar discrete wavelet transform (DWT) coefficients of the cover image.

1. Convert the secret message into a 1D bit stream. The information bits are encrypted using ACHTERBAHN-128 stream cipher before embedding them in the elements of the cover.
2. Before modifying the coefficients, a pseudorandom permutation of the message is used for increasing security of embedded message. The idea behind the permutation is that the permutation generator uses the stego key and produces as output different sequences of the set \(\{1, 2, 3, \ldots, \text{length (message)}\}\). This ensures that only recipients who have the corresponding secret key will be able to extract the message from a stego-object.
3. Decompose the cover image by using the one-level Haar Wavelet Transform.
4. The data sequence should be inserted into the least significant bit (LSB) of the wavelet sub-band coefficients starting from \(HH\) to \(HL\) according to the data length.
5. Insert not only the data sequence but also the data length sequence \(N\) since the receiver must know the data length in order to extract the data.
6. Once the embedding process is completed, the stego image is produced by applying the Inverse Wavelet Transform (IWT) on the modified coefficients.

The extracting process is summarized as shown in the following algorithm:

**Algorithm:** Extracting Data in Wavelet Transform

**INPUT:** Stego-image

**SUMMARY:** The stego-image is decomposed using Haar DWT and the message is recovered.

1. Decompose image by using Haar Wavelet Transform.
2. The proposed scheme is blind since with the data length (N) only, there is no need to recover the original cover image. Only the message is recovered by extracting the data length sequence N from the wavelet coefficients.

3. Extract the embedded data bits from the N LSB’s of the wavelet coefficients.

These steps are summarized in Figure 1.22 shown below.

![Figure 1.22: The wavelet-based message hiding and retrieval process [MDDE07]](image)

The system was implemented using Altera FPGA device, EP1C12Q240C8. This device contains 12060 logic elements, 52 RAM blocks (128 x 36 bits) and 249 maximum user Input and Output pins. [MDDE07]

### 1.6 Zero overhead watermarking technique for FPGA designs

Existing watermarking and fingerprinting techniques successfully embed identification information into FPGA designs to deter IP infringement [LSMP98]. However, such methods incur timing or resource overhead or both. In this section, an FPGA watermarking technique [JYPQ03] that provides zero design overhead is discussed. Due to the increasing complexity of FPGA devices and time-to-market deadlines, designers rarely build the entire design from the beginning. To protect FPGA designs, this approach deploys some inherent features of the design tools to embed watermark in the design. There are two advantages of this approach: first, the signature becomes a functional part of the design. Secondly, since no constraints are added to the design specification, the method avoids possible overhead in the implementation area and the timing delays. In this approach, the required watermarking technique was implemented using the Xilinx ISE 5.1 FPGA design platform. Specifically, the Timing Analyzer Tool was utilized to select certain nets from the design to embed the required watermark. Various FPGA development tools provide similar facilities. Therefore, this approach can be easily applied to other development tools likewise. The process is summarized in Figure 1.23.
Various constraints can be defined and added to the design to meet the specified timing or area requirements. Designers normally specify their constraints in a file called the User Constraints File (UCF). This file can be created and modified using the Constraints Editor. The configuration bit-stream files are then generated. This configuration file can then be loaded onto the on-chip RAM of the FPGA for configuring the various circuits of the device. One can control the delay on each net in the design by adding the required timing constraint on that net in this UCF. The file is integrated with the design during implementation. It affects the place and route result. Different delays on paths will force the place and route results to be different. This, in turn, will require a different set of circuits on the FPGA deployed. Therefore, the configuration bit-stream will be dissimilar. In fact, the changing of the delay on one net will affect a significant number of configuration bits. Accordingly, this provides the user with a rather unique bit-stream file as a strong proof of authorship [JYPQ03].

To demonstrate that the technique has zero overhead, a 16-bit signature “0100110101000100”, representing the ASCII code of ‘MD’ in binary, was embedded in the four selected designs. The statistical report for resource utilization, frequency requirements and bit-stream differences showed that there is no overhead on resource utilization. The overhead in timing is zero, since the delays, before and after watermarking, between flip-flops are less than the clock period. In other words, they satisfy the system frequency requirement. The watermarked design provides the same quality as the un-watermarked design to the customer. The floor plans before and after the hiding processes are shown in Figure 1.24.

Figure 1.23: The zero-overhead watermarking technique stages [JYPQ03]
Figure 1.24: The floor plans before and after hiding [JPQ03]

1.7 Signature hiding techniques for FPGA intellectual property protection

A watermark is applied to the physical layout of a digital circuit when it is mapped into an FPGA. This watermark uniquely identifies the circuit origin and yet it is difficult to detect. While this approach imposes additional constraints, experiments involving a number of large complex designs indicate that negligible effect on the performance. A watermark is applied to the physical layout [LSMP98] of a digital circuit when it is mapped into an FPGA. When the owners of an IP block believes their property has been misappropriated, they are compelled to deliver the configuration in question to an unbiased validation authority. With a special code and signature, the validation authority can reverse the signature preparation and embedding process. This is performed to identify the Configurable Logic Blocks (CLBs) used for hiding the signature using the functions defined by a secure hash function. The other required steps are: reversing the block interleaving, applying the Error Correcting Code (ECC) if necessary, decrypting the message using a known key, and finally printing out the resulting signature. If the signature matches that claimed by the IP vendor, then ownership has been established. The “netlist” file is the resulting file from translating the design file which input is the schematic editor logic diagrams. The process [LSMP98] is summarized as follows:

Algorithm: IP signature hiding

**INPUT:** netlist file, IP signature

**SUMMARY:** produces modified netlist with hidden signature

1. Read in netlist and desired signature
2. Use vendor tools to place and route unmodified netlist
3. If there is not enough spare resources for signature, then exit and retry with smaller signature
4. Process signature:
   5. Pack 8-bit ASCII into continuous 7-bit characters
   6. Encrypt signature to match “channel”, i.e. typical design, spectrum
   7. Add error correction coding
   8. Interleave ECC-encrypted blocks to combat localized tampering
   9. Embed properly-sized clique
10. Modify netlist and physical constraints to embed prepared signature
11. Execute vendor place and route tools on modified netlist
12. If (performance is too low) retry with smaller signature else terminate with success
The left hand side of Figure 1.25 is the original layout of the design with no watermark constraints. Note that the original placement does not achieve optimal logic density. Instead, unused CLBs are dispersed throughout the design. The right hand side of this Figure shows the layout with an embedded signature of 4768 bits.

![Figure 1.25: The layout without and with the watermarking constraints](image)

Experiments have proved that, even with very complex designs, a watermark can be applied and validated at this level with negligible impact on design performance and area.

### 1.8 FPGA for watermarking MPEG-4

This section deals with a VLSI architecture that implements video compression and watermarking algorithms utilizing FPGA technology [WECA07]. The micro-architecture is evaluated with video and watermarking quality metrics that will be discussed in detail in section 1.12. The video qualities of the watermarking, when comparing the peak signal to noise ratio (PSNR), the uncompressed versus the compressed versions are only 1-dB different. However, the cost involved with the compressed domain watermarking is the computational complexity of drift compensation for canceling the drifting effect. The method can be summarized as shown in Figure 1.26.
In this conceptual block diagram, the DCT is the discrete cosine transform block, IDCT is the Intra frame DCT, ME is the Quantizer, ZZ is the Zigzag scanning reorder, and the Entropy Encoder is used to perform data compression.

**1.9 Hardware-based Steganalysis**

Steganalysis can be considered as an attack on steganography where the opponent aim is to detect the existence of a hidden message. Software-based approaches are numerous; however, they hardly can follow the high speed network throughputs. Real-time Steganalysis tools are essential component in the monitoring of large data traffic. The FPGA-implemented micro-architecture is designed to detect least Significant Bit steganography [SPWP07]. The micro-architecture utilizes RS [ref] Steganalysis algorithm to detect message hidden in gray scale or color images. The algorithm is implemented in a parallel mode and lookup tables are also used to improve the expected throughput and reduce execution speeds. This algorithm [SPWP07] is summarized as follows:

**Algorithm:** Stego-detection

**INPUT:** The cover image

**SUMMARY:** detects Least Significant Bit (LSB) hiding in eight-bit gray-scale and 24-bit colored images by inspecting differences in the number of regular and singular groups for the LSB and shifted LSB plane.

1. The cover image is M x N pixels with pixel values from the set P. For example for the 8-bit gray-scale image P = {0, .., 255}.
2. Embedding starts with dividing the image into disjoint groups of $n$ adjacent pixels $(x_1, x_2, \ldots, x_n) = \sum_{i=1}^{n} |x_{i+1} - x_i|$

3. Define an Invertible operation $F$ on $P$ that is called flipping that is a 2-cycle permutations of gray levels. This operation is given by: $F_1(x) = F_1(x + 1) - x \forall x, F_0$ defined as the identity permutation $F(x) = x \forall x \in P$.

4. A discrimination function $f$ and the flipping operation $F$ are used to define three types of pixel groups- $R$, $S$, and $U$:

- **Regular groups:** $G \in R \iff f(F(G)) > f(G)$
- **Regular groups:** $G \in S \iff f(F(G)) < f(G)$
- **Regular groups:** $G \in U \iff f(F(G)) = f(G)$

In these expressions, $F(G)$ means that we apply the flipping function $F$ to the components of the vector $G = (x_1, \ldots, x_n)$. We may wish to apply different flipping to different pixels in the group $G$. We can capture the assignment of flipping to pixels with a mask $M$, which is an $n$-tuple with values $-1, 0, 1$.

5. Define the flipped group $F(G)$ as $(FM(1)(x_1), FM(2)(x_2), \ldots, FM(n)(x_n))$. The purpose of the flipping $F$ is perturbing the pixel values in an invertible way by some small amount, thus simulating the act of invertible noise adding.

6. The stego analyzer is based on how the number of regular and singular groups changes with the increased message length embedded in the LSB plane.

7. In typical pictures, adding a small amount of noise (for example, flipping by a small amount) will lead to an increase in the discrimination function rather than a decrease. Thus, the total number of regular groups will be larger than the total number of singular groups. This bias allows for lossless imperceptible embedding of a potentially large amount of information.

8. The RS steganalytic method is to estimate four curves of the RS diagram and calculate their intersections. Their shapes vary depending on the cover image.

9. For a message length $p$ in pixels, the number of R and S groups correspond to $R_M(p/2)$, $S_M(p/2)$, $R_{M/2}$, $S_{M/2}$. The factor of one half is based on the fact that only half of the bits will be flipped if the message is random bit stream.

10. Flipping the LSBs of all pixels in the image and compute the number of R and S groups, four points on a given curve are found.

11. The process is repeated by randomizing the LSB plane to estimate the location of the middle points $R_M(1/2)$ and $S_M(1/2)$ from the statistical samples.
12. The secret message length \( p \) is then computed from the x-coordinate of the intersection point. This is the root of the quadratic equation given by:
\[
2(d_1 + d_0) x^2 + (d_0 - d_1 - 3 d_0) x + d_0 - d_1 = 0,
\]
where
\[
d_0 = R_M(p/2) - S_M(p/2),
\]
\[
d_1 = R_M(1 - p/2) - S_M(1 - p/2),
\]
and the result \( p = x / (x - \frac{1}{2}) \)

The algorithm was accidentally discovered while research was performed on lossless data embedding. The micro-architecture of the data hiding detector is based on this algorithm and is constructed from the following four building blocks:

- The address generator (AG)
- Block memory
- Reconfigurable steganography detect engine (RSDE)
- Main control unit (MCU)

These building blocks are shown in Figure 1.27.

![Figure 1.27: Conceptual block diagram of the least significant bit (LSB) steganography detector [SPWP07]](image)

The address generator organizes the data and feed it them to the steganography detect engine. The color image pixel data are stored in the memory unit forming a three-dimensional data space. Several address modes are supporting the RS algorithm that divides the image into several pixel groups with different division modes. A data bus of 32-bit width is used and two addressing modes are supported in this design. In the first implementation, the AG supports regular three dimensional mappings and two dimensional mappings that appear in gray-scale images. The AG block diagram is shown in Figure 1.28. The five registers hold the address parameter vector APV for address calculation. Depending on the scan mode, the Address Calculation Unit (ACU) generates the address. The address bus is 22-bit wide.
1.10 Multimedia transmission over cognitive radio

Sensing the frequency unused part of the frequency spectrum without interfering with original spectrum users. The technique utilizes a secondary network, which utilizes the access of available spectrum aided by software or cognitive radio [KXCH08]. The idea was previously known as an auxiliary or a subliminal (steganographic) channel. The practicality of such a steganographic channel has become apparent with the development of FPGA software-based radio or what is commonly known these days as cognitive radio.

In this approach, the primary user traffic is modeled as a Poisson process. Selection of the appropriate channels and the study of the trade-offs between link reliability, spectral efficiency and coding overhead are provided. The cognitive cycle consists of three major components:

- Sensing RF stimuli
- Cognition/spectrum management
- Actions

Each component has certain tasks to accomplish. These tasks, called cognitive cycle, are shown in Figure1.29. The model uses spectrum pooling concept to select a set of sub-channels (SC) to establish a communication link. A cognitive radio can transmit packets over noncontiguous spectrum bands. This will help streaming multimedia traffic over multiple paths with a higher overall throughput to the user. This requires coordination between SCs. Digital fountain codes are used to accomplish this objective. The use of such codes achieves two objectives simultaneously. The SCs are to be distributed with no need for coordination and it acts as a channel code that overcomes the effect of loss caused the primary user interference and other channel conditions.
The spectrum pooling concept is based on allowing the secondary use by dividing the channel into many sub-channels to constitute a secondary user link in such a way that interference from the primary user channel is minimized. The secondary user link will degrade gradually rather a sudden breakdown with the arrival of the primary user. This basic idea is shown in Figure 1.30.

In the proposed model, if some of the secondary sub-channels are occupied by the primary users during transmission, then the decoder waiting time is relatively long. However, for multimedia applications, which are delay sensitive if the delay exceeds a maximum allowable delay then a decoding error will take place. Therefore, depending on the arrival rate of the primary users, one can find the optimum number of sub-channels to maximize the effective throughput and spectral efficiency. The maximum tolerable delay $D_{\text{max}}$ for the decoder is shown in Figure 1.31.
In the proposed model, an optimal number of sub-channels, or a subliminal channel in steganographic terms, maximizes the secondary user spectral efficiency for the same primary user traffic. Another extension, which we will not discuss here is the use optimal coding schemes to improve spectral efficiency.

1.11 Relation between Steganography and Cryptography

The idea of hiding messages in a media cover, which is known as Steganography can be extended to be used as an encryption technique. This is called Hybrid Hiding Encryption Algorithm [FASA05]. The algorithm uses simple plaintext hiding in a random bit string called the hiding vector. The word “Hybrid” indicates the encryption is based on ideas borrowed from Steganography. The algorithm hides a number of bits from the plain text message (M) into an N-bit long random vector (V). The locations of the hidden bits are determined by the key (K). The formal description of the algorithm [FASA05] is shown below.

Algorithm: MHHEA

INPUT: plain text message M , key matrix KLx2 , scrambled key matrix KNLx2 where
K_{ij} \in \{0,1,2,3,4,5,6,7\} \forall i =1,2, \forall j =0,..., L; L \geq 15

SUMMARY: The aim of the algorithm is hiding a number of bits from plain text message (M) into a random vector (V) of bits. The locations of the hidden bits are determined by the key KLx2

OUTPUT: encrypted file

Algorithm Body:

1. i :=0, m:=0
2. M[0]: = first digit in M file
3. while (M[m] \neq EOF) [EOF: End Of File]
4. i: = i mod L
5. Generate 16-bit randomly and set them in V Vector
6. if (K_{i,1} \geq K_{i,2}) then
7. z: = K_{i,1}
8. K_{i,1}: = K_{i,2}
9. K_{i,2}: = z
10. // Scramble the hiding location using the high order bits of the hiding vector
11. K_{Ni,1}: = V[K_{i,2}+8 down to K_{i,1}+8] XOR K_{i,1}
12. K_{Ni,2}: = K_{Ni,1}+(K_{i,2}-K_{i,1}) mod 8
13. if (K_{Ni,1} \geq K_{Ni,2}) then
14. z: = K_{Ni,1}
15. K_{Ni,1}: = K_{Ni,2}
15. \( KN_{i,2} = z \)
   // Scramble the message bits using the original key
16. \( q = 0 \)
17. \( \text{for } j = KN_{i,1} \text{ to } KN_{i,2} \)
18. \( q \leftarrow q \mod 3 \)
19. \( \text{if } (M[m] \neq EOF) \text{ then do} \)
20. \( V[j] = M[m] \text{ XOR } K_{i,1}[q] \)
21. \( m \leftarrow m + 1 \); next m in M file
22. \( q \leftarrow q + 1 \)
23. \( \text{end do} \)
24. \( \text{next } j \)
25. \( \text{Save } V \text{ in output file} \)
26. \( i \leftarrow i + 1 \)
27. \( \text{end while;} \)
End algorithm.

In this algorithm, we have scrambled the location of the message to overcome constant chosen plain-text attack. It is clear from this algorithm that one can view the complementary relation between steganography and cryptography. If the cover is randomly generated and data is hidden in this cover using a shared key then the methodology is basically an encryption technique or cryptographic technique. If the secret message is hidden in multimedia cover in a certain bit-plane then the algorithm is pure steganography. However, using a key to hide the message in certain locations is nothing but borrowing ideas from cryptography. If the message is first encrypted then is hidden, this will provide maximum security since it is using both cryptography and steganography at the same time. In the next section, we discuss a micro-architecture that is based on the previously mentioned algorithm, which can be used for both steganography and cryptography.

1.11.1 The micro-architecture

In this section, the micro-architecture with its operation details is provided. The machine operation takes place through six basic states. These are summarized as follows. The initial state “Init” holds back the execution of the successive states until the “Go” signal is triggered and furthermore resets all hardware modules. In the following state “LMsg”, the 32-bit input plaintext is buffered for the other modules to operate on. The key is buffered into sixteen four-bit pairs of registers in the “LKey” state. The key is saved in pairs of integers. One part of the key is XOR-ed with a part of the random vector V as described in the algorithm. After the scrambling of the key, the new key points to the locations of the substitution procedure as depicted in Figure … The aim is designing an architecture that replaces the whole number of bits determined by the key in parallel rather in serial to improve the overall performance. The location of the replaced bits is determined randomly based on the generated sub-key. In this respect, two design alternatives are possible. In the first one, a variable connection between the register containing the random hiding vector and the register with the scrambled plaintext is required. Nevertheless, this approach is rather difficult. Therefore, in this modified design, the connection is fixed but the plaintext is rotated to be aligned with the bits that are to be replaced in the hiding vector. A special rotation scheme is used in this architecture [ref#]. This has provided considerable savings in time and implementation area. The limited FPGA implementation area places a barrier on the size of the input plaintext required to be rotated. This fact has led to the splitting of the 32-bit input into two 16-bit parts. Each part is taken into a buffer inside the “Message Alignment” Module at a time during the “LMsgCache” state. The plaintext is subsequently aligned in the “Circ” state. Afterwards, the encryption or replacement procedure is performed in “Encrypt” state. These two states are interleaved in a chain of cycles until the whole 16-bit plaintext is encrypted. Consequently, the encryption
process takes two clock cycles per one key pair regardless of the number of bits replaced. The micro-architecture is subdivided into six modules as shown in Figure 1.32.

**Figure 1.32:** The block diagram of the hybrid Hiding Encryption Algorithm [FASA05]

The implementation floor plan I shown in Figure 1.33 and Table 1 provides a comparison with other techniques regarding throughput and implementation area.

**Figure 1.33:** The HHEA FPGA implementation floor plan [FASA05]

**Table 1.1:** A performance comparison between various implementations [FASA05]
This micro-architecture allows the user to choose between steganography and cryptography by selecting the appropriate input without changing the hardware. Consequently, we have bridged the gap between cryptography and steganography. This micro-architecture can also be combined with the Steganographic Shuffler (STS), shown in [FASA04], for shuffled-type steganography. The micro-architecture provides the highest functional density excluding the YAEA algorithm. Without a doubt, different algorithms have different degrees of security. However, this approach has demonstrated that with proper adaptation of the algorithm to hardware implementations, one can arrive at higher degrees of functional density and overall better performance. The complementary nature between Cryptography and Steganography is illustrated in this work with a modified micro-architecture that can be used for both techniques.
1.12 Notes

As shown in section 1.11, the relation between Steganography and Cryptography is of a complementary type. Hiding bits in a random vector can be used to scramble messages. That is Steganographic techniques can be used for encrypting messages. One can even visualize that reversing bits to hide a message is, in principle, an encryption technique. In this case, the key is the location pointer to the altered octets. Put differently, some cryptographic algorithms have used bit-hiding techniques to encrypt plaintext messages. Therefore, in these cases steganographic micro-architectures can be efficiently used as encrypting devices.

Multi-media files such as video frames, with their inherently large hiding capability, provide an opportunity to hide relatively large messages. However, the delay resulting from the hiding process has to be controlled. In this case, the hiding key can serve two purposes; selecting the octets to be altered and selecting which video frames to be the message-carrying medium. Therefore, two levels of security are obtained by concealing the message-carrying frame and the message bit-location.

Real-time execution of steganographic algorithms provides a multitude of security applications. To evaluate the perceptibility of steganography both subjective testing and quality-based metrics are used. These quality-based metrics, normally used for video compression, are the Signal-to-noise ratio (SNR) and the Peak-signal-to-noise ratio (PSNR). For the pixels of the cover image $P_i$ and the stego-image $Q_i$, the PSNR is given by:

$$\text{PSNR} = 20 \log_{10} \left[ \frac{\text{max}_i |P_i|}{\text{square root}((1/n) \cdot \Sigma_i (P_i - Q_i)^2)} \right]$$  \hspace{1cm} 1.12.1

The nominator absolute value is rarely needed since the pixel values are normally positive. The denominator is the root mean square error. For a gray-scale eight-bit image the numerator is 255. For color images, the luminance component is used. This metric can be used for comparing different hiding schemes. However, other factors should be taken into account such as the embedding capability of the image or the video stream. Whether all frames will be used for hiding or a group of frames that is selected based on a secret shared key. Other factors that should be considered in the design of various steganographic micro-architectures are: the processing time, the implementation area and the cost. Usually the processing delay is expressed as the throughput in bits per second. The implementation area is expressed in terms of the number of Configurable logic Blocks (CLBs) or logic elements (LEs). Both factors can be combined in one performance metric that is called the “functional density” metric ($F_D$). This metric is found by dividing the output, which is the throughput in Mega bits per second (Mbps) by the input or the implementation area in CLBs or LEs. The resulting equation for the functional density metric is given by:

$$F_D = \frac{T}{A}$$  \hspace{1cm} 1.12.2

Where $F_D$ is the functional density metric in Mbps/CLB (LE), $T$ is the throughput in Mbps, and $A$ is the implementation area in CLBs or LEs.

In multi-media data bases, video marking based on a descriptor extracted from the original media file provides a fast and reliable solution for visual information retrieval. Furthermore, real-time steganography, when used on public or satellite channels, can serve to relay secret information to an anonymous shared key owner who can unveil the hidden message. Anonymous receiving of these subliminal communications has a plurality of applications in information security and information operations.
References


Appendix A1: Field programmable gate arrays (FPGA)

A technology invented by Xilinx Corporation in the late eighties. It is a semiconductor device that can be reconfigured (reprogrammed) by the user in the field; hence the name “Field Programmable”. The device can be reprogrammed using different methods. These methods are:

- A schematic editor, where the logic circuit is actually described by the designer using a specially-designed circuit diagram editor,
- A source code of a hardware description language (HDL) such as:
  - VHDL
  - Verilog
  - System-C
  - ABEL
- A Finite State Machine Editor (FSM)

These design entry methods are summarized in Figure A1.1, shown below.

![Figure A1.1: Design Entry techniques](image)

The FPGA fabric is built from an array of configurable logic blocks CLBs and, in the basic device, a set of input and output ports. Other elements that have been later added; a reduced instruction set processor (RISC Processor), and an array of random access memory RAM components. This basic configuration is shown in Figure A1.2.
In the basic model, each CLB is constructed using a lookup table (LUT), a flip-flop and a multiplexer (MUX). The CLB typical architecture is shown in Figure A1.3.

Figure A1.2: Basic FPGA configuration

Figure A1.3: The basic CLB architecture

Different types of FPGAs have different number of CLBs or different number of equivalent logic gates. The capacity of an FPGA device is determined by the number of logic gates it holds. This number has been increasing quite rapidly since the early nineties when the number of logic gates was in the order of few thousands (4000-9000 logic gates). Present-day technology provides millions of logic gates on one FPGA device. The number of designs adopting FPGAs has increased from less than 10,000 designs in the early nineties to more than 100,000 designs in the year 2009. The market size has expanded from about $15 million in the mid eighties to about $3 billion in the year 2009. The market and technology dynamics are
changing the design implementation technology from Application Specific Integrated Circuits (ASIC) to FPGAs. This is due to the following factors:

- The complexity of developing ASIC-based digital logic circuits
- Decreasing R&D resources and personnel
- Slow time-to-market financial losses
- Increase of Integrated Circuits (IC) development costs
- Poor economies are deriving low-cost technologies such as FPGAs

FPGAs are applied in most fields where digital logic is used. This includes:

- Special purpose processors
- Defense and Aerospace
- Digital Signal Processing DSP
- Cryptography/Encryption
- Steganography
- Data Communications
- Cognitive radio (software-defined radio)
- Instrumentation

The design process using FPGA is summarized in Figure A1.4 shown below.

The designer enters his design using a schematic editor or a HDL, then a compiler or a synthesizer is used to translate this design into a gate-level netlist. This gate-level netlist can be used for functional verification using a logic simulator and also as an input to the Place-and-Route software. This software provides the extraction and timing analysis. If any problem is detected in these phases, the designer may have to go back to his/her original source to fix the problem and repeat the process.
In a design flow, the designer simulates the design at various stages. The register transfer language description in VHDL or Verilog is created by creating test vectors to simulate the system and observe the results. After the netlist file is generated, it is translated to a gate-level description. Simulation is repeated at this stage to assure that the design is conforming to the timing and area specifications provided by the designer. Finally, the design is mapped onto the FPGA where propagation delays are taken into account.

The design tools are provided by Xilinx, Altera, and Actel. Each one of these corporations has different FPGA device families. As an example: Xilinx provides Xilinx 4000, Spartan and Virtex families of FPGAs. Altera provides Flex 8000, Apex and Cyclone families. Actel provides PROASIC.