DESIGN AND IMPLEMENTATION OF A SECRET KEY STEGANOGRAPHIC MICRO-ARCHITECTURE EMPLOYING FPGA

Hala Farouk, Magdy Saeb
Arab Academy for Science, Technology & Maritime Transport
School of Engineering, Computer Department
Alexandria, Egypt

Abstract – In the well-known "prisoners' problem", a representative example of steganography, two persons attempt to communicate covertly without alerting the warden. One approach to achieve this task is to embed the message in an innocent-looking cover-media. In our model, the message contents are scattered in the cover in a certain way that is based on a secret key known only to the sender and receiver. Therefore, even if the warden discovers the existence of the message, he will not be able to recover it. In other words a covert or subliminal communication channel is opened between two persons who possess a secret key to reassemble its contents. In this article, we propose a video or audio steganographic model in which the hidden message can be composed and inserted in the cover in real-time. This is realized by designing and implementing a secret key steganographic micro-architecture employing Field Programmable Gate Arrays FPGA.

Keywords: Steganography, data hiding, FPGA, architecture, covert communications, subliminal channel.

I. INTRODUCTION

Quite recently, information hiding techniques have gained extended attention in a number of application areas, namely watermarking, fingerprinting, captioning, steganography and covert channels [1]. Moreover, a rather new and interesting application of data embedding is granting users with different access levels to the data [2].

In this work, we present a hardware implementation of a secret-key steganographic algorithm. The basic idea of our algorithm is selecting the hiding bits in a pseudorandom manner as a function of a secret key to increase obscurity. We compare the performance of this algorithm to former algorithms found in the literature [3, 4]. By means of this performance comparison, as shown in section 5, we demonstrate that our approach has some clear advantages in applications utilizing real-time systems.

The paper is organized as follows: in section 2 we provide a summary of the algorithm. Section 3 provides a brief description of our proposed micro-architecture. In section 4, we discuss the simulation and implementation results. A performance analysis and comparison with other implementations are shown in section 5. Finally a summary and our conclusions come into view in section 6.

II. THE ALGORITHM

In the following few lines, we provide a summary of our algorithm that can be applied to video frames, audio files or any type of covers to hide a given message. This message hiding uses a secret key known only to sender and receiver.

ALGORITHM STEGO
[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.]

Input: Message M, Cover C, Key K, State Register SR
Algorithm Body:
Begin
1. Load a block of the message Blk_i
into the message cache MC:
Blk_i [M] → [MC];
2. Load Key into the key Cache:
K → [KC];
3. Generate an address Ad_i;
4. Address memory to get one cover word CW:
M [Ad_i] → $CW_{i;}$
5. Hide two message bits (m_i, m_{i+1})
by replacing (C_0, C_8) in the cover word CW
with (M_i, M_{i+1}):
C [15:9], M_{i+1},C[7:1],M_i → CM ;
6. Write back steganographic word:
CM_i → $CW_{i;}$
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:
If message not finished
8.1 Load next block into message cache:
Blk_{i+1} [M] → [MC];
8.2 Goto 3:
GenerateAddressState → SR;
Else if message is finished then halt;
End Algorithm.
Output: Modulated Cover CM

III. THE MICRO-ARCHITECTURE

The architecture is divided into an embedder processor and an SDRAM controller as shown in Figure 1. The embedder processor 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 sections we discuss the various building blocks of our proposed micro-architecture.

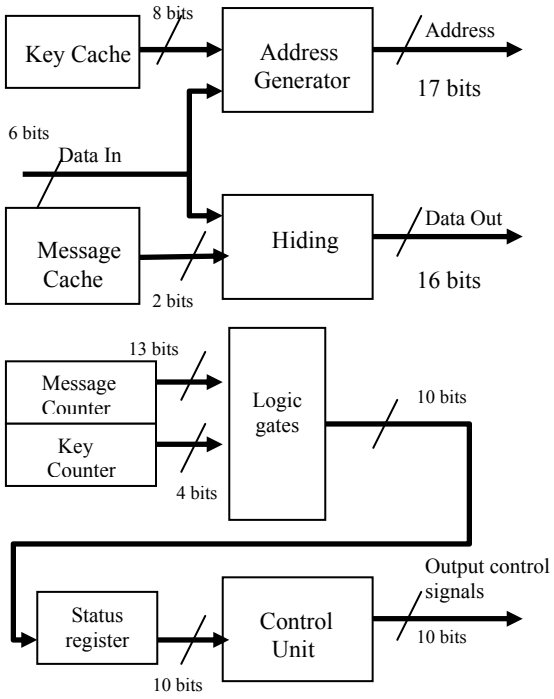

Figure 1: Micro-architecture block diagram

3.1 The Embedder Processor Overview and Organization

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. In the following subsections, we provide a brief discussion of each of these components.

3.1.1 Address Generator

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 2. 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 video frames most probably never exceed.

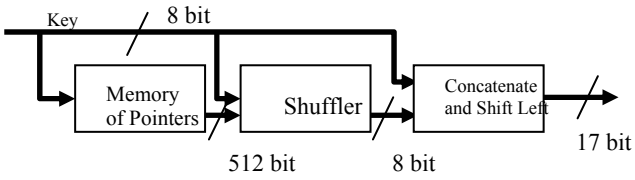

Figure 2: Address generator block diagram

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 segments or in some of them.

3.1.2 Block Pointer Memory

This module consists of 64 eight-bit counters. The module takes six bits as an input to decide which one of the counters to be incremented. The outputs of all counters are concatenated to form 512 bits and sent to the shuffler.

3.1.3 Shuffler

This module receives 512 bits from the block pointer memory module and based on the key, it selects one of 64 pointers to be transmitted to the shift and concatenate unit. Each pointer is eight bits. Therefore, the address space for each pointer is 256 words. We consider these 256 words 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 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 image and attempts all blocks evenly. This new key will also avoid large repetition in the key octets. The histogram in Figure 3 is not uniform and the key doesn't cover the whole key space. This is an indication of a "bad" key. This key will make hiding biased to a certain area in the image. The histogram in Figure 4 is almost uniform and the key covers 100% of the key space, which proves that all image blocks will be attempted evenly. This key is shared by the sender and the receiver to reassemble the message at the receiver side, as mentioned before.

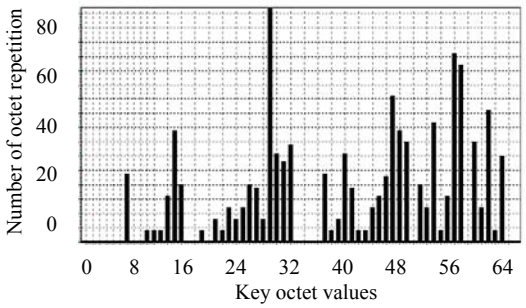

Figure 3: Histogram of the key octet values before improvement

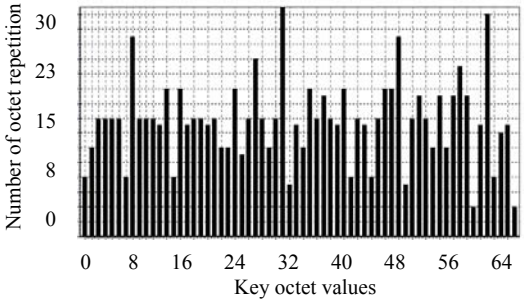

Figure 4: Histogram of the key octet values after improvement

3.1.4 Shift & Concatenate Unit

This unit forms the 17 bit output by shifting and concatenating the shuffler's output and some parts of the key.

3.1.5 Stegoblock

The stego unit acquires one memory word and hides two bits in the zero and eight location.

3.1.6 Status and State Registers

The 10-bit status register contains a *run* bit, set by the reset push button on the FPGA board and reset when message counter is zero and message cache is empty, a *cover-dependence* bit, set by the user to indicate the desire of making the addresses generated to be unique for each cover, a two *segment* bits, set by the user to limit the area of hiding in the cover, a *memdone* bit, set by the SDRAM when it is ready to send the requested word or when it has finished writing the prompted word, a *msgcntrzero* bit, set by the message counter when it arrives to zero value, a *msgcacheempty* bit, set by the message cache when it is empty, a *msgcachefull* bit, set by the message cache to show that it is full, a *loadkey* bit, set by the control unit after loading the key into key cache, and finally a *keycachefull* bit, set by the key cache when it is full.
The 3-bit state register is given new values by the control unit and in turn it indicates the state of the program. The state can be one of the following:

- Message load
- Key load
- Control word load
- Address generation
- Hiding
- Waiting for a reset

3.1.7 Message Cache (MC)

The message cache is organized as a rather large shit register. The message to be embedded into the cover image is saved consecutively starting from location 131,072. Blocks of this message is cached during the hiding process. The message cache stores, and then shifts the message bit by bit to the right. These shifted out bits are to be hidden in the cover word by the stegoblock.
The message cache improves performance as it saves eight memory calls per bit, during the hiding process. The memory used in this design is a synchronous dynamic RAM, which requires more cycles in the read and write operations than the static RAM. The dynamic RAM on the other hand is larger and can support large images as is needed in our case. As shown in the coming sections the SDRAM is divided into 512 columns and 4096 rows. The addressing of consecutive words in same memory row requires three clock cycles. The addressing of words spaced out in different memory rows requires from 8 to 9 clock cycles. Since, changing the address from the cover location to the message location in the memory requires from 8 to 9 clock cycles, it is better to address the message consecutively and then address the cover. This improves performance and is achieved by caching the message before the embedding process.

3.1.8 Key Cache (KC)

The key cache holds the entire key, which is 32 bytes. It is designed to circulate its contents to the right, bit by bit, and always outputs the least significant eight bits. The key is responsible for the block that will be chosen for the hiding to take place and therefore choosing a "good" key is not a trivial matter. Some issues on the key where discussed in section 3.1.3.

3.1.9 Counters

The key cache counter (KC) and the message cache counter (MC) are used in the loading interval of the message and key cache in order to acquire the exact number of memory words needed. As the XSA100 did not allow us to create a message cache as large as the message, a message counter is needed to know whether to load another message block into cache or the whole message is already processed.

3.1.10 Address and Data Multiplexers

These multiplexer are controlled by the control unit in order to select the appropriate data and addresses in the right state to be sent to the memory.

3.1.11 Address Extender

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.

3.1.12 Control Unit

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 is shown in Figure 5.

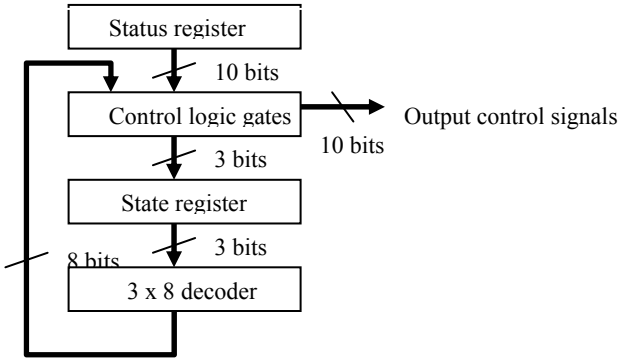

Figure 5: Control unit block diagram

It consists of one decoder, a state register, and a number of control logic gates. The status register supplies the control unit with the needed signals. The outputs of the state register are decoded into eight state signals explained above. Based on some logic operations and on the state signals, a 10-bit output is generated by the control unit. This output consists of a *msgptrinc* bit, which increments the address of the next message block to be loaded into cache, a *msgcacheload* bit that controls message cache loading, a *memwr* and a *memrd* bit that control the read/write signal of the SDRAM, a *setloadkey* bit that controls loading of the key cache, a *msgcachecntrinc* bit that increments the message cache counter, a *generateaddr* bit that enables the generation of a new address for hiding, and a 3-bit *addressselector* bus that controls the address multiplexers explained above.

3.2 SDRAM Controller

The XSA100 Spartan 2 Xilinx FPGA board has a 16 M Byte synchronous dynamic random access memory mounted on it. The SDRAM controller allows the interface with the large capacity SDRAM. The 16M SDRAM is organized as 4096 row x 512 column x 4 banks. Therefore, the 23-bit address is divided in three parts, naming column (9 bits), row (12 bits) and bank (2 bits). The SDRAM is word addressable. Each word is 16 bits.
The basic SDRAM commands are:

- Mode register set command
- Row address strobe and bank active command
- Pre-charge
- Column address and write command
- Column address and read command
- Auto refresh command

A sequence of these commands comprises the primitive operations of read, write and refresh. The circuit realizations of all of the above are shown in Appendix B.

IV. SIMULATION & IMPLEMENTATION RESULTS

At the start, the circuit is at the default state waiting for a reset signal. During the reset signal the whole circuit is initialized. Once the reset is driven low the hiding process commences. The loading of message cache is presented in Figure 6. Note that the memory address bus, *memaddr22* bus, has the value 20000hex at the start. This value is equal to 131072 in decimal format, which is the address of the first message word in the memory as mentioned in section 3.1.7.

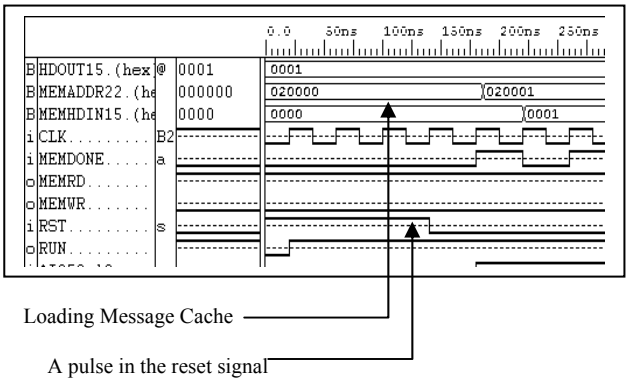

Figure 6: Simulation of message cache loading

Finally, the hiding process is illustrated in the simulation results as shown in Figure 7. The step in the embedding process is requesting a cover word from the memory. Then, the memory drives the *memdone* bit high to indicate end of transaction.
Afterwards, the cover word, which in this case is 5544hex, is to be received by the embedder processor. Two bits are replaced by two message bits, namely the bits in the zero and eighth location. This results in a modulated cover word that is equal to 5445hex. This modulated cover word is written back to memory. Following this operation, a new address is prompted to the memory to start the hiding process all over again for two new message bits. Based on the simulation results in Figure 7, the new address requires 5 ns to take place. On the other hand, according to timing results shown later in Appendix A, the shuffler circuit requires 30.846 ns in order to generate one address. This time difference is due to the fact that the address generator module works in parallel during the embedding process. Hence, the new address is complete inside the address generator module waiting for the new embedding sequence to start. Consequently, only 5 ns were required to output the prepared generated address.

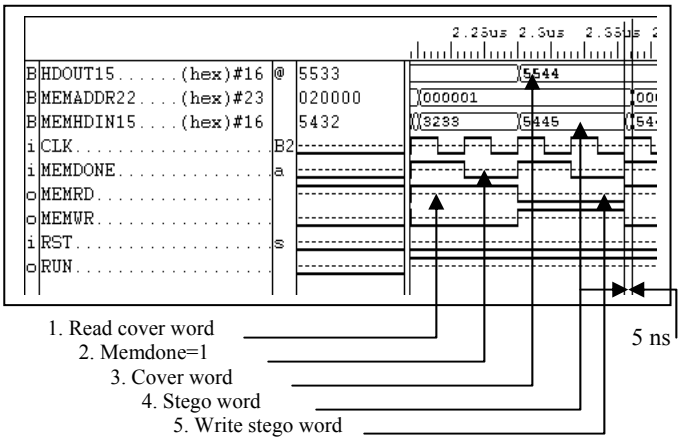

Figure 7: Simulation of the hiding process

V. PERFORMANCE ANALYSIS

As described in the previous sections, the algorithm selects the cover words in a pseudorandom fashion as a function of a secret key, to hide in it the message bits. The algorithm presented in references [3, 5] is implemented using hash functions. In reference [4], a hardware design of the MD5 and Secure Hash Algorithm (SHA) a hash function is implemented. As shown in these references, the MD5 running at 60.2 MHz, takes 66 cycles to generate a hash number, while the SHA, running at 38.6 MHz requires 22 cycles. The algorithm in reference [3] consists of three hash operations, one division, four modulo operations, one multiplication and four addition operations. If we assume using a Montgomery multiplier [6] and that the input is nine-bit large, then the total number of cycles needed for generating an address using this technique, will be 342 cycles using the MD5 and 210 cycles using the SHA as shown in Table 1 . The table shows that our shuffler, the module that generates memory addresses in a pseudorandom order, requires only one cycle. The throughput in the last row in Table 1 demonstrates the higher throughput obtained when applying our algorithm. This throughput is calculated for the operations required to hide only one

message bit. The throughput is computed as in [3] by the following equation:

$$S = 1 \times F / C \qquad (1)$$

Where S is the Throughput, F is the clock frequency and C is the number of cycles.

Table1. Performance comparison

	Shuffler	Design using SHA	Design using MD5
Number of cycles	1	210	342
Frequency in MHz	35.4	38.6	60.2
Throughput in Mbps	1.576	0.174	0.170

5.1 Testing for Obscurity

The information is hidden using the designed chip in the sense that it is perceptually and statistically undetectable. To prove this, a sample of the cover and its modulated version is shown in Figure 8. As it is apparent from comparing these two pictures, there is hardly any visible significant difference. This modulated cover is statistically analyzed using a discrete Laplacian Filter [7]. The result of this type of analysis is shown in Figure 9b. If the embedding process adds noise that is statistically quite different from true random noise, then the output of the Laplacian filter will lose the high peak illustrated in Figure 9a and instead two lower peaks with high side fluctuations will appear as shown in Figure 9c.

(a) *(b)*

Figure 8: Hiding using designed chip. (a) Cover image. (b) Modulated cover image.

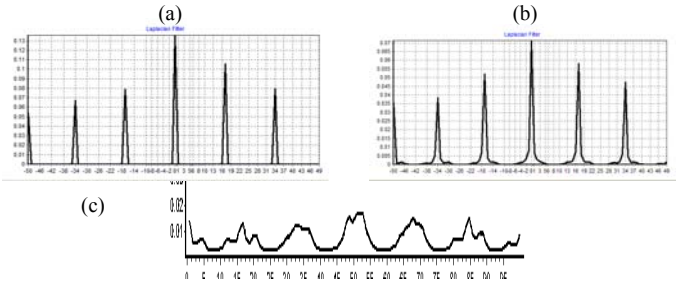

Figure 9: Laplacian Filter. (a) Filter output of cover image. (b) Filter output of modultated cover image. (c) Filter output of distorted cover

SUMMARY AND CONCLUSION

Motivated by the need for a fast hardware implementation suitable for real-time applications, we have provided a micro-architecture of a secret-key steganographic FPGA implementation. The distinctive features of this design are as follows:

- The address generator generates an output every clock cycle. This is a major advantage as compared to SHA-based algorithm that requires 210 cycles or MD5 designs with 342 cycles.
- The shuffler design is our 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.

We realize that there is a large number of testing procedures for obscurity that are called steganalysis techniques. However, we have shown by proper choice of the key, that our approach has provided an acceptable degree of data hiding with minimal distortion of the cover. This was proven utilizing the Laplacian Filter Technique. We believe that our approach is refined enough to escape the watchful eyes of a passive adversary (the warden). Comparing this architecture with other types of steganographic algorithm implementations, we have demonstrated the dominance of our algorithm in time-critical applications.

REFERENCES

[1] F. Petitcolas, R. Anderson, and Kuhn, "Information hiding—a survey," IEEE proceedings, special issue on protection of multimedia content, vol. 87, No. 7, pp.1062-1078, July 1999.

[2] D. Swanson, M. Kobayashi, A. Tewfik, "Multimedia Data Embedding and Watermarking Technologies," proceedings of the IEEE, vol. 86, no. 6, June 1998, pp.1064-1087.

[3] T. Aura, "Invisible communication," HUT Network Seminar in Helsinki Institute of Technology, 1995.

[4] J. Diez, S. Bojanic, L. Stanimirovic, C. Carreras, O. Nieto-Taladriz, "Hash algorithms for cryptographic protocols," 10th Telecommunications forum TELFOR'2022, Belgrade, Yugoslavia, November 26-28, 2002.

[5] M. Luby, Ch. Rackoff, "How to construct pseudorandom permutations from pseudorandom functions," SIAM Journal on Computing, 17(2), pp.373-386, April 1988.

[6] F. Tenca, K. Koc, "A scalable architecture for modular multiplication based on Montgomery's algorithm," IEEE Transactions on Computers, to appear, 2003.

[7] S. Katzenbeisser, F. Petitcolas, Information hiding techniques for steganography and digital watermarking, Computer security series, Artech House, 2000.

APPENDIX A: IMPLEMENTATION RESULTS

Target Device: x2s100
Target Package: tq144
Target Speed : -6
Mapper Version: spartan2 -- C.22

The Embedder Module: Timing Summary:

Minimum period: 52.516ns (Maximum frequency: 19.042MHz)
Maximum combinational path delay: 53.822ns
Maximum net delay: 9.853ns

Device utilization summary:

Number of External GCLKIOBs	2 out of 4	50%
Number of External IOBs	43 out of 92	46%
Number of SLICEs	1197 out of 1200	99%
Number of DLLs	2 out of 4	50%
Number of GCLKs	1 out of 4	25%
Number of TBUFs	2 out of 1280	1%

The Shuffler Module:

Timing Summary:

Minimum period: 28.254ns (Maximum frequency: 35.393MHz)
Maximum combinational path delay: 30.846ns
Maximum net delay: 13.745ns

Device utilization summary:

Number of External GCLKIOBs	1 out of 4	25%
Number of External IOBs	40 out of 92	43%
Number of SLICEs	750 out of 1200	62%
Number of GCLKs	1 out of 4	25%

The SDRAM Controller Module: Timing Summary:

Minimum period: 23.851ns (Maximum frequency: 41.927MHz)
Maximum net delay: 7.122ns

Device utilization summary:

Number of External GCLKIOBs	2 out of 4	50%
Number of External IOBs	57 out of 92	61%
Number of SLICEs	115 out of 1200	9%
Number of DLLs	2 out of 4	50%
Number of GCLKs	1 out of 4	25%
Number of TBUFs	2 out of 1280	1%

The Total Design

Timing Summary:

Minimum period: 48.811ns (Maximum frequency: 20.487MHz)
Maximum combinational path delay: 49.783ns
Maximum net delay: 11.980ns

Device utilization summary:

Number of External GCLKIOBs	2 out of 4	50%
Number of External IOBs	43 out of 92	46%
Number of SLICEs	1195 out of 1200	99%
Number of DLLs	2 out of 4	50%
Number of GCLKs	1 out of 4	25%
Number of TBUFs	2 out of 1280	1%

APPENDIX B: CIRCUIT REALIZATIONS

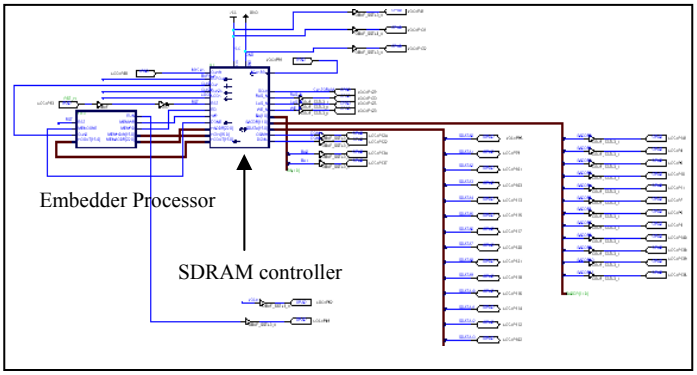

Figure B-1: Top-level view of the embedder processor and SDRAM control

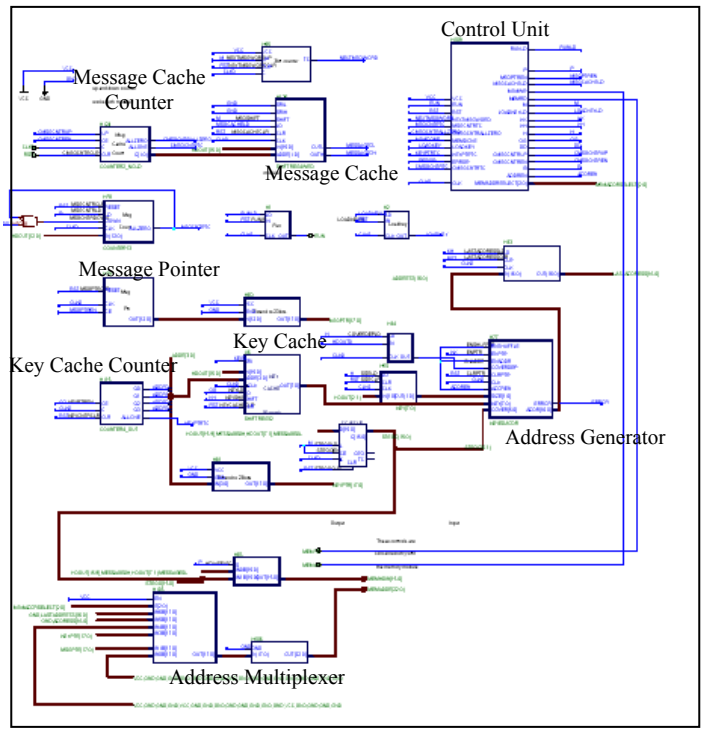

Figure B-2: Embedder Processor's inside view

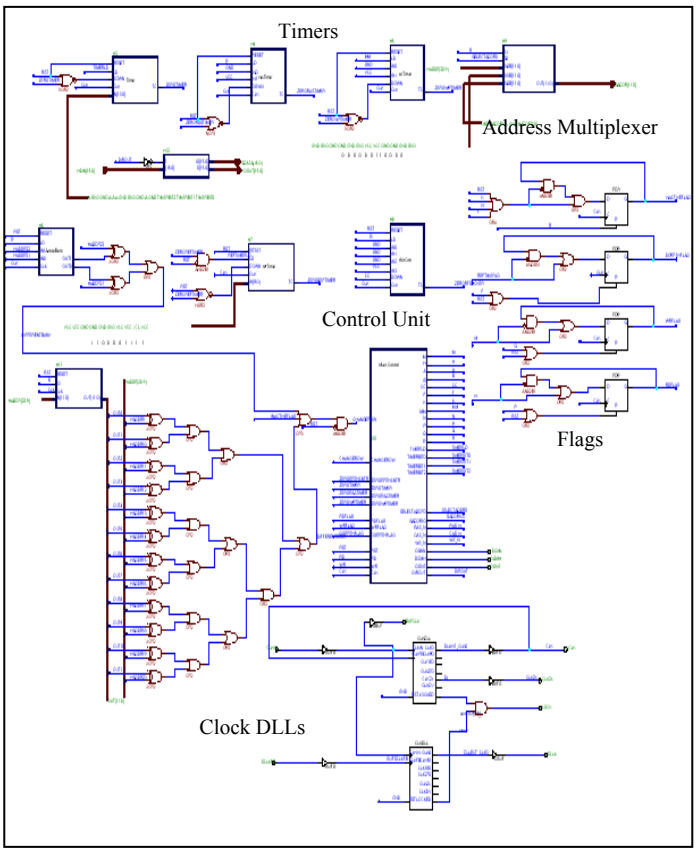

Figure B-3: SDRAM controller's inside view