Chapter 07

Key-triggered Hash-chaining based Encoded Hardware Steganography for Securing DSP Hardware Accelerators

Anirban Sengupta
Computer Science and Engineering
Indian Institute of Technology Indore

This chapter describes a multi-encoding driven key-triggered hash chaining based hardware steganography approach for securing DSP hardware accelerators which uses multiple layers of encoding and key-based parallel switch blocks to drive multiple secure hash chaining blocks in the algorithm. The presented approach is highly robust against fraud ownership claim and piracy threats. The chapter is organized as follows: Section 7.1 provides some introduction to the research problem, followed by some discussion on other selected approaches in Section 7.2; Section 7.3 describes the presented hash-chaining based hardware steganography approach. Section 7.4 presents the design process of securing FIR filter using this hardware steganography process; Section 7.5 presents the KHC-Stego tool of this corresponding key-triggered hash chaining hardware steganography approach; Section 7.6 discusses the analysis on case studies; Section 7.7 concludes the chapter while Section 7.8 presents some exercise for readers.

7.1. Introduction

The rage of development is increasing the demand of electronics systems in the market. To cope up with the increasing demand, the design of reusable intellectual property (IP) cores is automated through design automation tools to reduce the design time and effort (Sengupta *et al*., 2019a). Moreover, the design of highly complex IP cores such as digital signal processing (DSP) and multimedia makes the role of automated high level synthesis (HLS) process more significant (Sengupta *et al*., 2010; Schneiderman, 2010; Plaza and Markov; 2015). The HLS process accepts the algorithmic description of an application and generates its equivalent register transfer level or Verilog/VHDL description (which is referred as soft IP core) (Chakraborty and Bhunia, 2019). Further, the journey of an electronics design from its algorithmic description to an end integrated circuit (IC) involves various design houses forming a design chain. More explicitly, an IP core design process is outsourced to a separate design house in the supply chain for SoC integration later. The various design houses involved in the design supply chain are situated globally and may not be trustworthy (Yasin *et al*., 2016). This is because an attacker in the untrustworthy design house may pirate the IP core and sell the counterfeited/cloned IPs/ICs in the market to earn illegal revenue. Moreover, a dishonest IP

buyer may fraudulently claim the IP ownership, deceiving the genuine owner. This renders an IP core design vulnerable against 'fraudulent claim of ownership' and piracy threats (Zhang, 2016; Sengupta, 2016; Sengupta, 2017; Sengupta *et al*., 2019b).

To nullify the false ownership claim and to enable the detection of counterfeiting/cloning, designer's secret-mark can be embedded into an IP core design (Colombier and Bossuet, 2015; Newbould, 2002; Sengupta *et al*., 2018; Sengupta *et al*., 2019c; Sengupta and Mohanty, 2019). However, the embedded secret constraints (secret-mark) are also required to be highly secured so that an attacker would not be able to nullify the purpose of embedding secret-mark by extracting it (Sengupta, 2020). In order to enhance security of DSP cores, Sengupta and Rathor, 2020 proposed an '*encoding and key-driven hash-chaining based hardware steganography*' approach that produces a highly robust stego-mark which is arduous to be extracted or regenerated by an adversary. Hence the adversary fails to evade counterfeit detection process by embedding stego-mark of genuine author post regeneration or extraction of it. The encoding and hash-chaining driven hardware steganography approach (Sengupta and Rathor, 2020) ensures robustness of the secret-mark by generating it through a large size secret-key, designer chosen encoding rules, encoded bitstreams and number of iterations of round function in each hash-unit of the chaining process.

7.2. Discussion on Selected Approaches

In the field of protection of reusable IP cores for DSP hardware accelerators, there are two categories of approaches available in the literature viz. firstly, hardware (IP) watermarking approaches (Sengupta and Bhadauria, 2016; Sengupta and Roy, 2017; Koushanfar *et al*., 2005; Le Gal and Bossuet, 2012) which are signature based mechanisms and secondly, hardware (IP) steganography approaches (Sengupta and Rathor, 2019a; Sengupta and Rathor, 2019b) which are signature free techniques. For example in hardware watermarking approaches (Sengupta and Bhadauria, 2016; Sengupta and Roy, 2017), multi-variable signature encoding algorithm was employed to generate the watermarking constraints for implanting into high-level synthesis phase of DSP hardware accelerator design; while (Koushanfar *et al*., 2005) employed binary digit combination to form vendor signature, followed by implanting watermarking constraints during high-level synthesis. Further, (Le Gal and Bossuet, 2012) uses mathematical relationships between numeric values as inputs and outputs at specified times to generate watermarking constraints. On the other hand, hardware steganography approaches (Sengupta and Rathor, 2019a; Sengupta and Rathor, 2019b) derive secret relationships from cover design data (colored interval graph of architectural synthesis phase) of DSP hardware design process and constructs encoding algorithms to form secret stego-constraints for implanting, without requiring IP vendor signature at all. The threat model for both the category of techniques is same and intends to primarily prove fraudulent claim of ownership as well as detect pirated IP cores such as counterfeited/cloned ICs/IPs in the design chain.

However, it is well established that in the context of DSP hardware accelerators, steganography provides better designer control in terms of secretly implanted design constraints and incurs lesser design overhead than hardware watermarking approaches (Sengupta, 2020). This is vastly due to the inherent nature of the IP steganography algorithms which neither depends on the strength of the vendor signature chosen nor its combination. Instead, it is dependent on the stego-constraints converter algorithm and its equivalent secret encoding algorithm. Nevertheless, in both existing IP steganography approaches such as (Sengupta and Rathor, 2019a; Sengupta and Rathor, 2019b) and watermarking approaches (Sengupta and Bhadauria, 2016; Sengupta and Roy, 2017; Koushanfar *et al*., 2005; Le Gal and Bossuet, 2012), proof of ownership could still be relatively easier from an attackers perspective due to non-existence of robust secret keys and multi-layered encoding algorithms. On the contrary, in the switch based encoding driven hardware steganography approach (Sengupta and Rathor, 2020), the secret stego-constraints are generated through interweaving of hash-chaining process, switch based hash units, robust stego-keys, multiple encoding algorithms and steganography embedder algorithm. The complexity of encoding and key-driven hash-chaining based steganography (Sengupta and Rathor, 2020) algorithm makes it infeasible for an attacker to generate/extract the implanted secret stego-constraints to prove IP ownership. The switch based encoding driven steganography (Sengupta and Rathor, 2020) provides stronger security than hardware watermarking approaches in terms of lower probability of co-incidence (reflecting more digital evidence), greater designer control (due to no dependency on signature) and lower design overhead.

7.3. Encoding and Key-driven Hash-chaining based Hardware Steganography Methodology

1. Threat Model

A vendor's IP core is susceptible to fraudulent claim of ownership by a dishonest buyer. The encoding and key-driven hash-chaining based hardware steganography approach (Sengupta and Rathor, 2020) handles the 'fraudulent claim of ownership' threat for IP cores by embedding vendor's secret stego-mark (secret steganography information) into the design. The embedded robust stego-mark also enables the detection of counterfeited/cloned IPs/ICs.

2. High Level Description

The encoding and key-driven hash-chaining based steganography methodology (Sengupta and Rathor, 2020) generates secret stego-constraints (steganography information) through key-triggered hash-chaining steganography process. The generated stego-constraints are embedded during register and functional unit (FU) allocation phase of the HLS process. This results into a stego-embedded DSP hardware design which is secured against the fraudulent claim of ownership and counterfeiting/cloning threats.

The overview of the switch based encoding driven steganography approach (Sengupta and Rathor, 2020) is shown in Fig. 1. As shown in the figure, following are the inputs to the approach (a) DSP application in the form of data flow graph (DFG) (b)

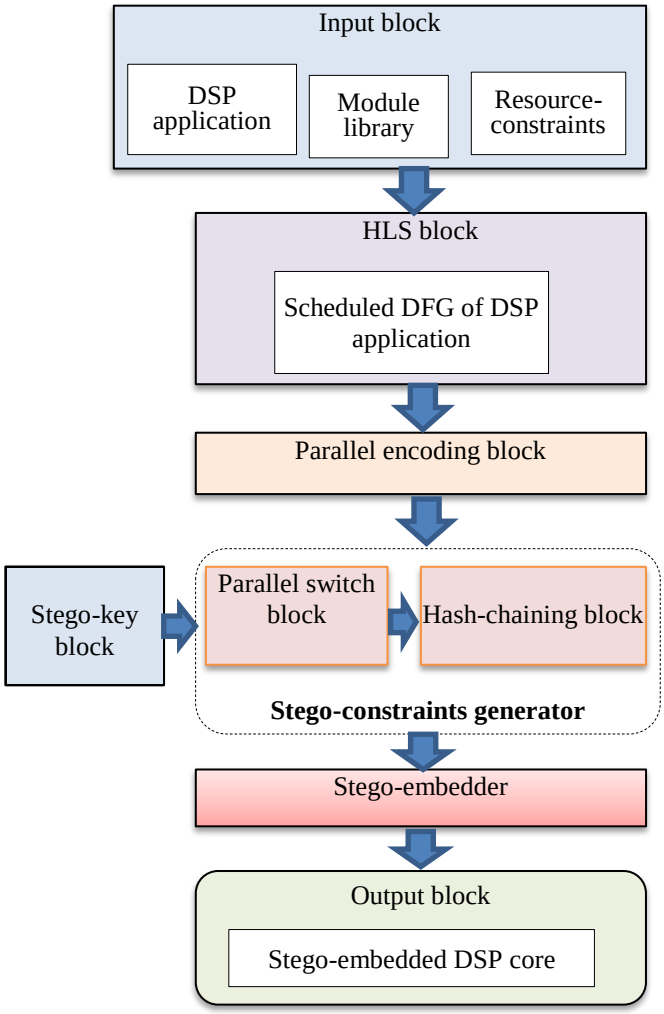

Fig. 1. *Overview of key-triggered hash-chaining based steganography (Sengupta and Rathor, 2020)*

resource constraints (c) module library. These inputs are fed to the HLS block whose output is a scheduled DFG of the DSP application. The role of this block is to schedule the DFG based on designer specified resource constraints. Further, the scheduled DFG is fed as input to the parallel encoding block. The output of this block is a number of bitstream representations of the respective DSP application. The role of this block is to encode the scheduled DFG of DSP application into a number of unique bitstreams based on designer chosen encoding rules. Thereafter, all encoded bitstreams are fed to the stego-constraints generator block whose output is the stego-constraints to be implanted into the DSP core, as secret steganography information. The role of this block is to generate stego-constraints through two sub-blocks viz. parallel switch block and hash-chaining block as shown in Fig. 1. Another input to the stego-constraints generator is the output of stego-key block which provides keys required to generate the stego-constraints. These stego-keys are fed to the parallel switch block along with the encoded bitstreams generated from the parallel encoding block. The role of each switch in the parallel switch block is to select an encoded bitstream (out of all available) based on specific stego-key value. The output of parallel switch block is the selected bitstreams (based on vendor stego-key

values) which are fed to the hash-chaining block as input. All encoded bitstreams generated from parallel encoding block are also directly fed to the hash-chaining block. The role of hash-chaining block is to generate stego-constraints through a chain of hash-units employed inside. The final output of hash-chaining block is the stego-constraints which are fed to the stego-embedder block as input. The stego-embedder block embeds the stego-constraints during the register and functional unit (FU) allocation phase of HLS. The output of stego-embedder block is the steganography embedded DSP hardware accelerator.

3. In-depth Description of Key-triggered Hash-chaining based Hardware Steganography

This section discusses the key-triggered hash-chaining steganography approach (Sengupta and Rathor, 2020) in details. Fig. 2 depicts the details of generating stego-embedded DSP core through key-triggered hash-chaining based steganography. As shown in the figure, the entire process is divided into the following blocks below:

(a) Input block

This block contains the inputs fed to the key-triggered hash-chaining based steganography approach as shown in Fig. 2. The DSP application to be secured can be fed as input in the following forms: C/C++ code, transfer function or intermediate representation such as DFG. Here, DFG representation of DSP application is used as input to the key-triggered hash-chaining steganography methodology (Sengupta and Rathor, 2020). Besides DFG, module library and resource constraints are also fed as inputs to the HLS block.

(b) HLS block

This block performs HLS process on the DFG of the DSP application. First, the operations present in the DFG are scheduled using a scheduling algorithm into control steps (Q), based on designer specified resource/functional unit (FU) constraints. This results into a scheduled DFG of the DSP application, which is fed as input to the next block.

(c) Parallel encoding block

This block encodes the scheduled DFG of the DSP application into manifold bitstream representations by applying encoding rules (Sengupta and Rathor, 2020). Each encoding rule is capable to encode the DSP application into a unique bitstream. Length of each bitstream is equal to the number of operations in the DSP application. For a DSP application having 'n' operations, total possible encoded bitstreams are 2^n and the length of each bitstream is 'n'. Out of 2^n, a designer can generate 'k' number of encoded bitstreams (for implementation) by applying k-encoding rules (where, $1 \leq k \leq 2^n$) as shown in Fig. 2. These 'k' encoded bitstreams are fed as input to the parallel switch block (these inputs are represented by green lines in Fig. 2) and hash-chaining block (these inputs are represented by red lines in Fig. 2). The nine (k=9) encoding rules (shown in

Table 1) to generate nine encoded bitstreams of the DSP application have been proposed by Sengupta and Rathor, 2020. Because of total 2^n possible bitstream representations of the DSP application, an attacker is required to put extremely high

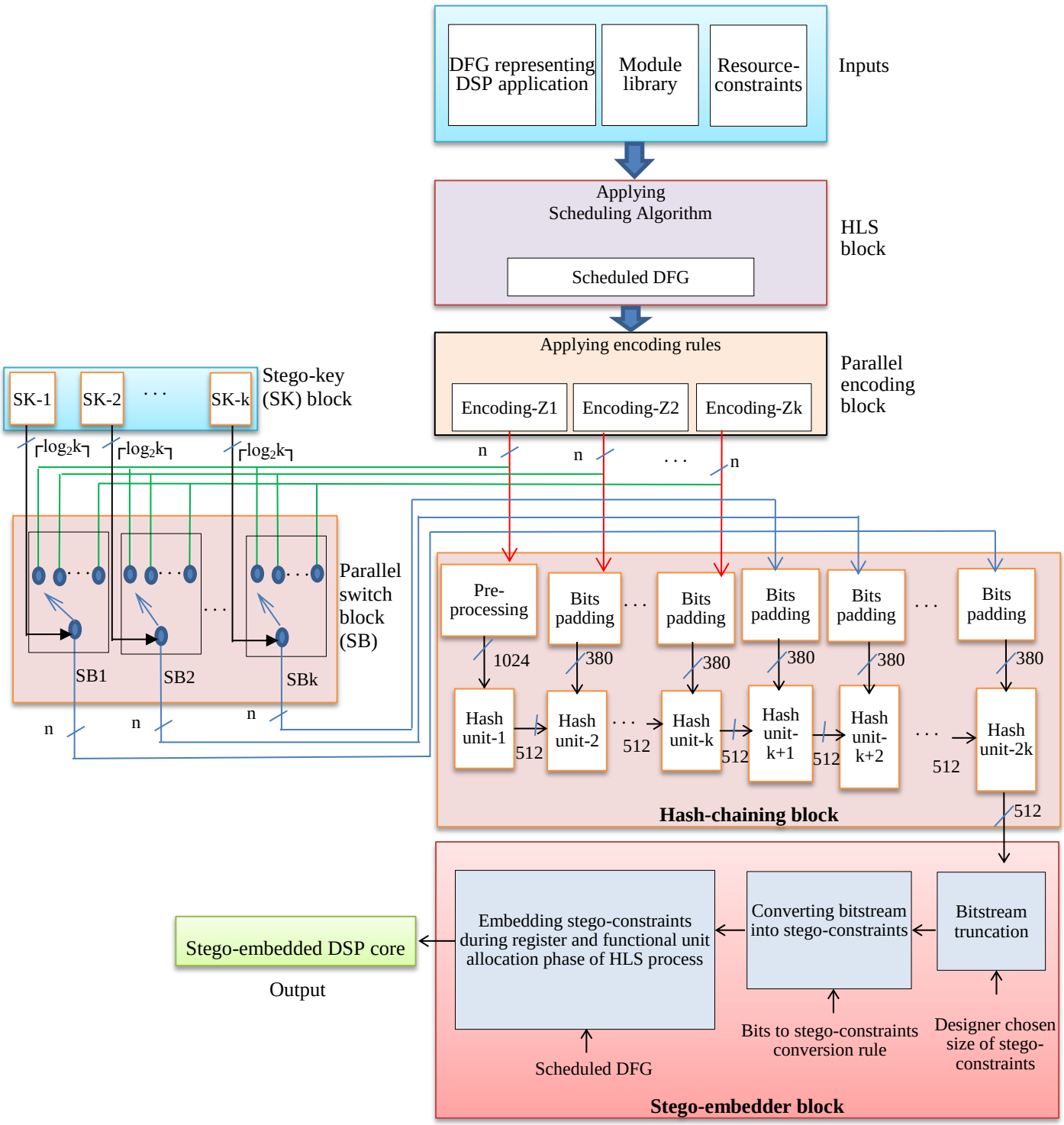

Fig. 2. *Details of securing DSP cores using key-triggered hash-chaining based steganography (Sengupta and Rathor, 2020)*

effort to find out the actual encoded bitstreams used in generating the secret stego-constraints (or stego-mark).

Further, the encoding rules applied by the designer also remain unknown to the attacker. This renders almost infeasible for an attacker to find the encoded bitstreams used in subsequent blocks for generating stego-constraints.

(d) Parallel switch block

For ‘k’ number of encodings chosen by the designer, total k-parallel switches are used inside the parallel switch block as shown in Fig. 2. Each switch block (SB) selects one out of ‘k’ encoded bitstreams based on the stego-key value. Each switch operates on a distinct stego-key value; hence total ‘k’ stego-keys are required. Each switch can be implemented using a multiplexer of size k:1; where a stego-key acts as the select lines/bits and the size of select line/ stego-key is $\lceil \log_2 k \rceil$ bits. The selected k-bitstreams through all k-switches are fed to the hash-chaining block (these inputs are represented by blue lines in

Table 1. *Encoding rules to encode a DSP application into bitstreams*

Encoding	Encoding Rules (operation (O), control step (Q))	Encoded bit
Z1	If O# and corresponding Q # are both even	0
	Otherwise	1
Z2	If O# and corresponding Q# are of same parity (both even or both odd parity)	0
	If O# and corresponding Q# are of different parity	1
Z3	If O# and corresponding Q # are both odd	0
	Otherwise	1
Z4	If O# and corresponding Q# are of different parity	0
	If O# and corresponding Q# are of same parity	1
Z5	If O# and corresponding Q# are both prime	0
	Otherwise	1
Z6	If O# and corresponding Q# are both prime	1
	Otherwise	0
Z7	If GCD of O# and corresponding Q# is 1	0
	If GCD of O# and corresponding Q# is not 1	1
Z8	If (O#) mod (corresponding Q#) is 0	0
	If (O#) mod (corresponding Q#) is not 0	1
Z9	If Q# is equal to 2^{nd} odd sequence of O#	0
	Otherwise	1

Fig. 2).

(e) Stego-key block

This block provides stego-keys to operate the k-switches in the parallel switch block. As discussed in the previous step, the size of each stego-key used to operate single switch is $\lceil \log_2 k \rceil$ bits. Since, there are total k-parallel switches, therefore the total size of stego-key becomes $k \times \lceil \log_2 k \rceil$ bits. Since the maximum possible encoded bitstreams are $k=2^n$ (as discussed earlier in parallel encoding block), therefore a parallel switch block can have maximum 2^n switches and size of each stego-key becomes $\lceil \log_2 2^n \rceil = n$. Hence, the maximum possible key size (Max key size) is given by following equation:

$$\text{Max key size} = (\text{Max number of switches}) \times (\text{size of each stego key})$$

$$\text{Max key size} = 2^n \times \lceil \log_2 2^n \rceil = n \times 2^n \text{ bits} \qquad (1)$$

Because of very large size stego-key, it is very hard for an attacker to find out the correct key used to generate the stego-constraints (or stego-mark). This strengthens the security achieved through key-triggered hash-chaining based steganography approach.

(f) Hash-chaining block

This block is responsible for generating the stego-constraints in the form of hashed bitstream. A chain of total 2k hash-units (HUs) is employed inside the hash-chaining block, where k is the total number of encodings chosen by the designer. Each hash unit is fed with the input of 1024-bit and generates output of 512 bits. Each hash unit employs 'round function computation (RFC)' process (Sengupta and Rathor, 2019c) to generate 512-bit hash-output. The number of rounds the RFC process is run for each hash unit is specified by the designer. The number of rounds varies from 1 to 80. Because of designer specified number of rounds, the hardness of finding the hash-output and consequently stego-constraints enhances from an attacker's perspective.

Further in the key-triggered hash-chaining steganography approach (Sengupta and Rathor, 2020), two different processes are used to feed input to the hash units:

(1) Pre-processing: This process is used to feed 1024-bit input to the hash unit-1 (HU_1) only. The rule of constructing 1024 bits through pre-processing is given by following equation:

$$(1024 \text{ bits})_{HU_1} = (n \text{ bits of encoded bitstream obtained from encoding } Z1) \,\&\, '1' \,\&\, (\text{sequence of } (896 - n - 1) \text{zeros}) \,\&\, (128 \text{ bit representation of the length } 'n') \qquad (2)$$

(2) Bit padding: This is the process to form 380-bit input to the remaining hash units (HU_2 to HU_{2k}). To form the 380 bits, the designer chosen combination of (380-n) bits is padded before the n bits of the encoded bitstream. Thus the size of

encoded bitstreams (corresponding to encoding Z2-Zk) fed to the hash-units (HU_2 to HU_{2k}) becomes 380 bits which are only known to the designer and unknown to an attacker.

Further, the total 1024-bit input to the remaining hash units (HU_2 to HU_{2k}) is formed using following equation:

$$(1024\text{ bits})_{HU_{2-2k}} = (512\ bit\ output\ of\ previous\ hash\ unit)\&\text{"1000"}\ \&\ (380\ bit\ output\ of\ bit\ padding)\& \\ (128\ bit\ representation\ of\ the\ length\ of\ previous\ hash\ output) \quad (3)$$

The 2k hash units employed in the hash-chaining block are categorised into two types viz. regular hash units and key-triggered hash units.

(i) Regular hash units (HU_1 to HU_k):

The first 'k' hash units in the hash-chaining block are the regular hash units. These hash units (HU_1 to HU_k) exploit encodings Z1 to Zk respectively to generate hash-output. 1024-bit input for HU_1 is formed using (2) and 1024-bit input for HU_2 to HU_k is formed using (3).

(ii) Key-triggered hash units (HU_{k+1} to HU_{2k}):

The last 'k' hash units in the hash-chaining block are the key-triggered hash units. This is because a particular encoding (out of k-encodings) used by a key-triggered hash unit is selected based on the specific value of stego-key. The 1024-bit input for HU_{k+1} to HU_{2k} is formed using (3).

It is important to note here that the total hash-units (2k) employed in the hash-chaining block are two times of the number of encodings (k) chosen by the designer. And total key-triggered hash-blocks (k) are equal to the number of encodings (k) chosen by the designer. The 512-bit hash-output of the hash-chaining block is fed as input to the next block.

(g) Stego-embedder block

This block is responsible for embedding stego-constraints into the design during HLS process. The entire process is explained

Table 2. *Rules to convert a hash-bitstream into corresponding stego-constraints*

Bit	**Conversion rule**
'0'	An edge is embedded between node pair <even, even> in to CIG
'1'	If operation (O)# is odd, then it is allocated to FU of vendor type 1 (V1) and if O# is even, then it is allocated to FU of vendor type 2 (V2)

using following three sub-processes:

(i) Bitstream truncation: This process truncates the 512-bit hash-bitstream (obtained from hash-chaining block) to the designer specified size of stego-constraints.

(ii) Conversion into stego-constraints: All '0' and '1' bits in the truncated bitstream are converted into corresponding stego-constraints by applying conversion rules as shown in Table 2. Thus obtained stego-constraints are fed to the 'embedding stego-constraints' process.

(iii) Embedding stego-constraints: Inputs to this process are scheduled DFG and stego-constraints to be embedded. This process embeds stego-constraints into the DSP core design during register allocation and FU allocation phase of HLS process. The stego-constraints corresponding to bit '0' are embedded during register allocation and those corresponding to bit '1' are embedded during FU allocation. To embed stego-constraints corresponding to bit '0' during register allocation phase, a colored interval graph (CIG) framework is leveraged. A CIG represents graphically the assignments of all storage variables of the design to the minimum possible registers, where each register is denoted using a distinct color and storage variables are denoted using nodes in the CIG (Sengupta and Bhadauria, 2016). An edge between two nodes in the CIG indicates that the respective storage variables/nodes are essentially executed through two distinct registers/colors. The storage variables of the design can be extracted out from the scheduled DFG, where all primary and intermediate inputs/output of the design are assigned to variables. The stego-constraints corresponding to bit '0' are embedded as extra edges (constraint edges) in the CIG complying with the condition that two nodes connected through an edge should be of different colors. To satisfy this condition, alteration of node colors in the CIG may be required to perform. The added constraint edges into the CIG are reflected in the scheduled DFG through the reallocation of storage variables to the registers/colors.

Further, the stego-constraints corresponding to bit '1' are embedded by reallocating the operations in scheduled DFG to a particular vendor type, which is determined based on the rule mentioned in Table 2. Thus, each embedded stego-constraint corresponding to bit '1' indicates the allocation of an operation to a fixed vendor type.

The embedding stego-constraints (stego-mark) results into a stego-embedded DSP core. By detecting the embedded steganography during forensic detection, false claim of ownership of the IP core can be nullified and counterfeited/cloned IPs/ICs can be detected.

4. Detection of Steganography

To nullify the fraudulent claim of ownership, the stego-constraints (or secret steganography information) are detected in the design under test. To do so, the RTL datapath of the design is inspected and the information about the inputs of multiplexers

associated with each register (which tells about the assignment of storage variables to the registers) and vendor type allocation of each operation is collected. This information is matched with the designer's stego-constraints generated through the key-triggered hash-chaining steganography approach (Sengupta and Rathor, 2020). If both matches, then the presence of stego-constraints of the genuine owner are verified in the design, thus ownership is awarded to the genuine owner. Fig. 3 depicts the steganography detection process in detail.

5. Security from an Attacker's Perspective

A dishonest IP buyer/user (attacker) can claim the ownership of the design, deceiving the genuine owner. If secret information (stego-mark) of the genuine owner is not embedded into the design, then the attacker can easily realize his/her malicious intent of claiming IP ownership fraudulently. The embedded stego-mark thwarts an attacker to do so. However an attacker may try to claim stego-mark (stego-constraints) as his/her own, defeating the purpose of steganography. In order to do so, the attacker is required to find the secret stego-constraints embedded into the design. The key-triggered hash-chaining based steganography enhances the attacker's effort of finding stego-constraints manifold. The attacker needs to find out the valid stego-key and the encoded bitstreams used to generate the stego-constraints. The attacker's maximum effort of finding the stego-key (e_{sk}^{m}) through brute-force is given as follows:

$$e_{sk}^{m} = 2^{\text{Max } key\ size}$$

$$e_{sk}^{m} = 2^{n \times 2^{n}} \qquad (4)$$

Where, maximum stego-key size is obtained using (1).

Further, the total encoded bits used in hash-chaining block to generate stego-constraints are calculated using following equation:

$$\text{Total encoded bits } = 380 \times (\#\text{ of hash units})$$

$$\text{Total encoded bits } = 380 \times (2\text{k}) \qquad (5)$$

Where, 380 is the size of encoded bits fed to each hash unit in the hash-chaining block and 2k is the number of hash units which are two times of number of encodings (k) chosen by the designer. Hence, the attacker effort (e_{H}^{eb}) to find out the encoded bits through brute-force is given as follows:

$$e_{H}^{eb} = 2^{total\ encoded\ bits}$$

$$e_{H}^{eb} = 2^{380 \times (2\text{k})} \qquad (6)$$

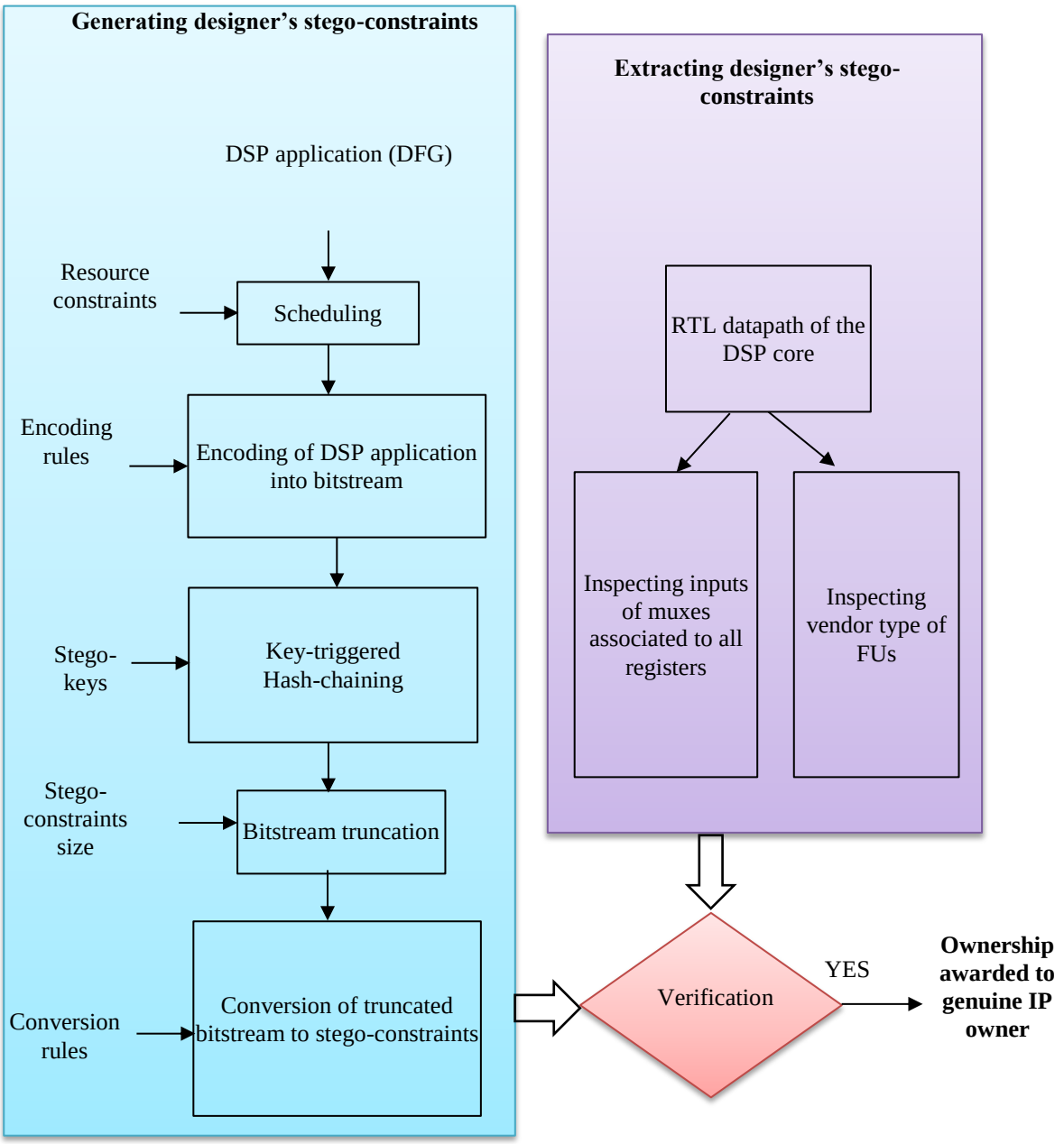

Fig. 3. *Detection process of steganography (Sengupta and Rathor, 2020)*

Where, total encoded bits are calculated using (5).

Hence, attacker's total effort in determining the stego-constraints embedded into the design is quantified as follows:

$$\text{Total effort} = e_{sk}^{m} \times e_{H}^{eb}$$

$$\text{Total effort} = 2^{n \times 2^{n}} \times 2^{380 \times (2k)}$$

$$\text{Total effort} = 2^{(n \times 2^{n} + (380 \times 2k))} \qquad (7)$$

It is evident from (7) that an attacker requires a huge effort in order to find out the stego-constraints embedded into the design. This renders the generated stego-mark using key-triggered hash-chaining steganography approach (Sengupta and Rathor, 2020) highly robust and secured.

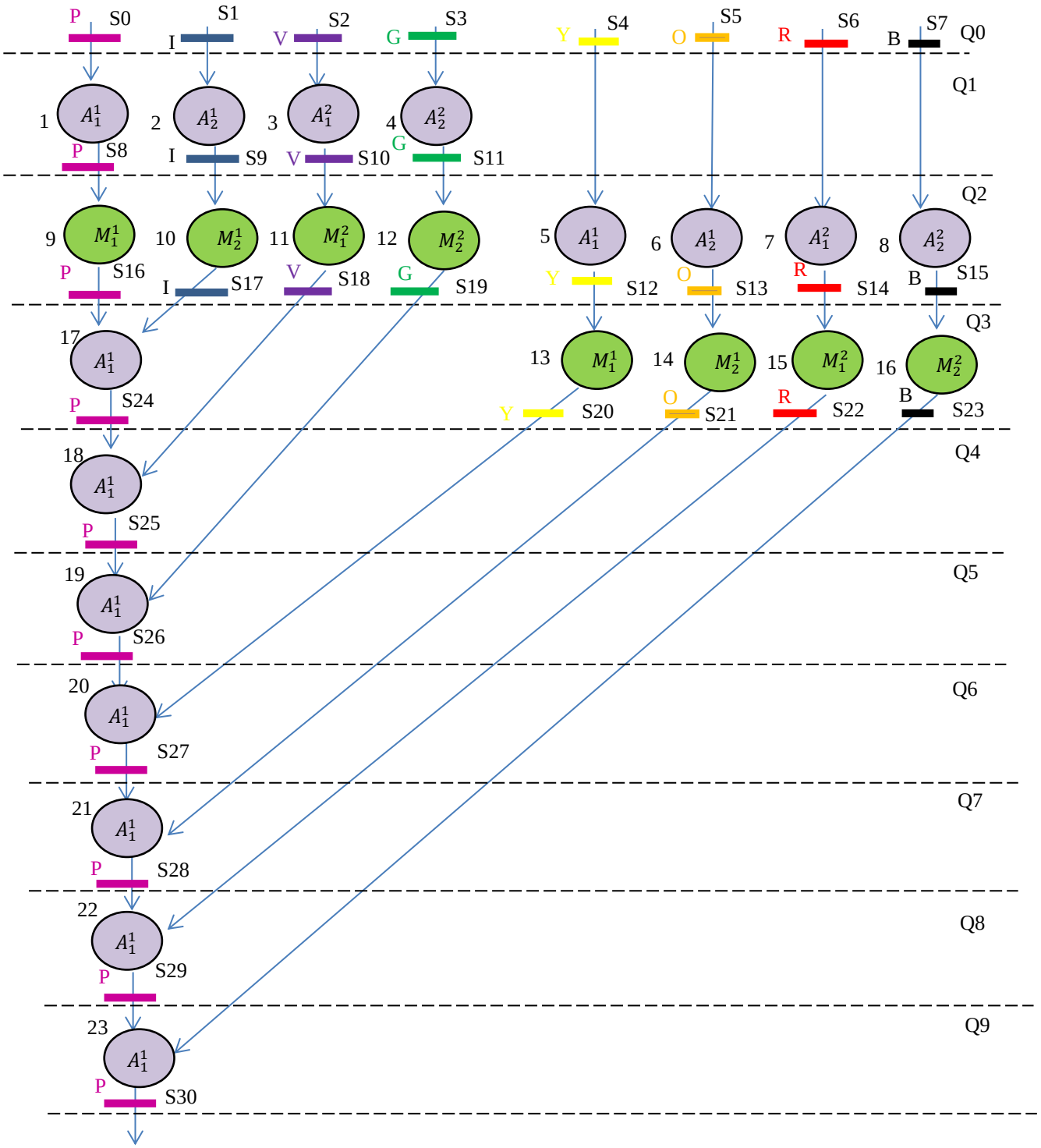

Fig. 4. *Scheduled and resource allocated DFG of FIR filter application*

7.4. Design Process of Securing FIR Filter using Encoding and Key-driven Hash-chaining Steganography

Digital FIR (finite impulse response) filters are widely used in electronics applications such as digital communication for de-noising. Here, a FIR filter core is chosen to illustrate the hash-chaining based steganography approach (Sengupta and Rathor, 2020). The FIR filter application in the form of DFG is fed as input along with module library and resource (FU) constraints of 4 multipliers (M) and 4 adders (A). The steps of generating a stego-embedded FIR filter core using key-triggered hash-chaining based steganography are as follows:

1. *Generating scheduled DFG of FIR application*: The DFG of FIR application is scheduled based on designer chosen resource (FU) constraints of 4 M and 4 A. The scheduled DFG is shown in Fig. 4. As shown in the figure, 23 operations of the FIR applications are executed through 9 control steps (Q). Operations have been allocated to FUs from two different vendors (V1 and V2). Note: For a FU instance (M_j^i *or* A_j^i), instance number and vendor type are denoted by the subscript 'j' and superscript 'i' respectively. Since chosen constraints of multiplier are 4, therefore two instances of multiplier of vendor V1 and another two instances of multiplier of vendor V2 have been used to execute multiplication operations in a control step. Similarly, chosen constraints of adder are 4, therefore two instances of adder of vendor V1 and another two instances of adder of vendor V2 have been used to execute addition operations in a control step. Further, the primary and intermediate inputs and outputs of the design are stored using total 31 storage variables (S0-S30). The corresponding CIG of the FIR application is shown in Fig. 5, where nodes S0-S30 indicate the storage variables of the design. Table 3 captures the allocation of storages variables to registers (colours) into different control steps.
2. *Generating encoded bitstreams*: By applying nine (k=9) encoding rules (Sengupta and Rathor, 2020), the scheduled DFG of FIR application is encoded into nine unique bitstreams as shown in Table 4. As there are n=23 operations in the FIR application, the length of each bitstream is 23.
3. *Pre-processing and bit padding of encoded bitstreams*: The first bitstream generated using encoding rule Z1 is pre-processed to form 1024 bits using (2). The remaining eight bitstreams (each of size '23') generated from encoding rules Z2-Z9 are converted to 380 bits by padding designer chosen combination of 357 bits before them. Thus obtained eight bitstreams of 380-bit each are converted to 1024 bits using (3). Now all nine bitstreams are of length 1024 bits and ready to be fed to the hash-chaining block.
4. *Feeding bitstreams to hash-chaining block*: The first bitstream of 1024-bit (corresponding to encoding-Z1) is fed to the hash unit-1 (HU_1). The remaining eight bitstreams (corresponding to encoding-Z2 to Z9) are fed to the regular and key-triggered hash units. Please note that as 9 encodings are chosen by the designer, therefore total 9×2=18 hash units are employed in the hash-chaining block. Where, HU_1- HU_9 are the regular hash units and HU_{10}- HU_{18} are the key-triggered

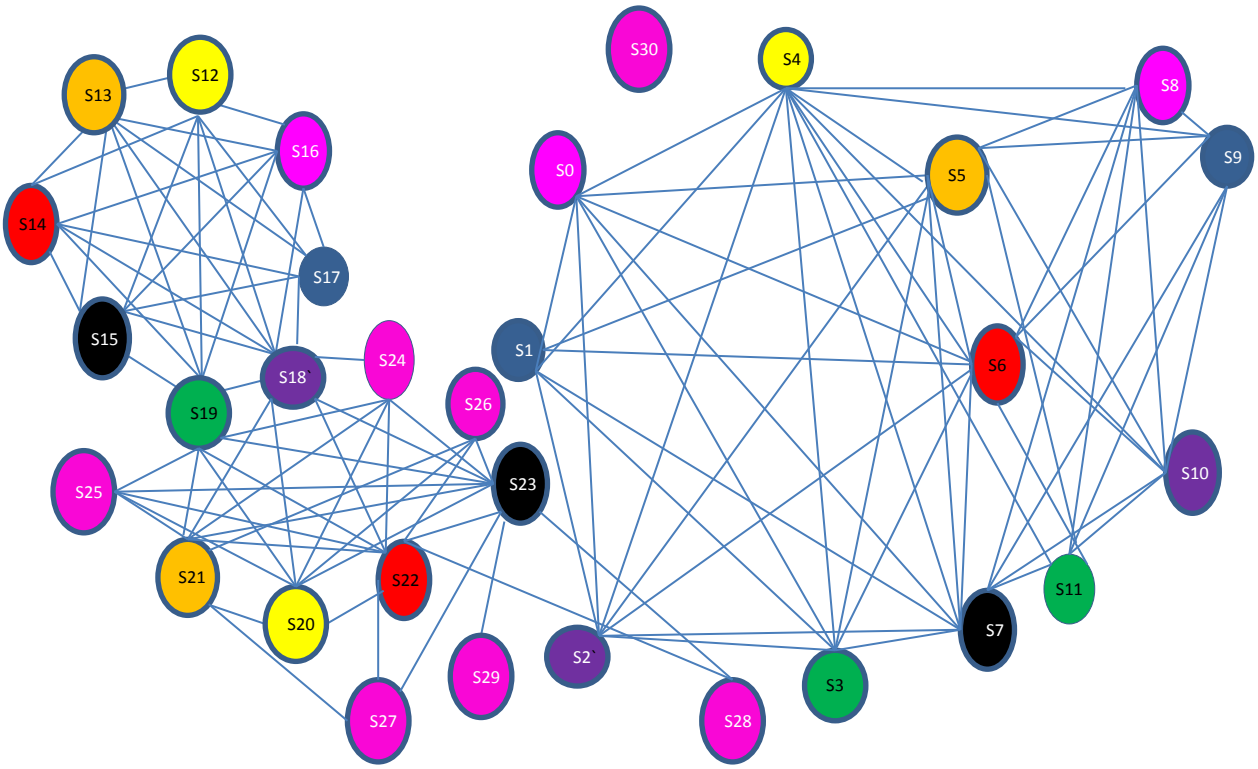

Fig. 5. *CIG of FIR filter hardware accelerator (IP core) before steganography*

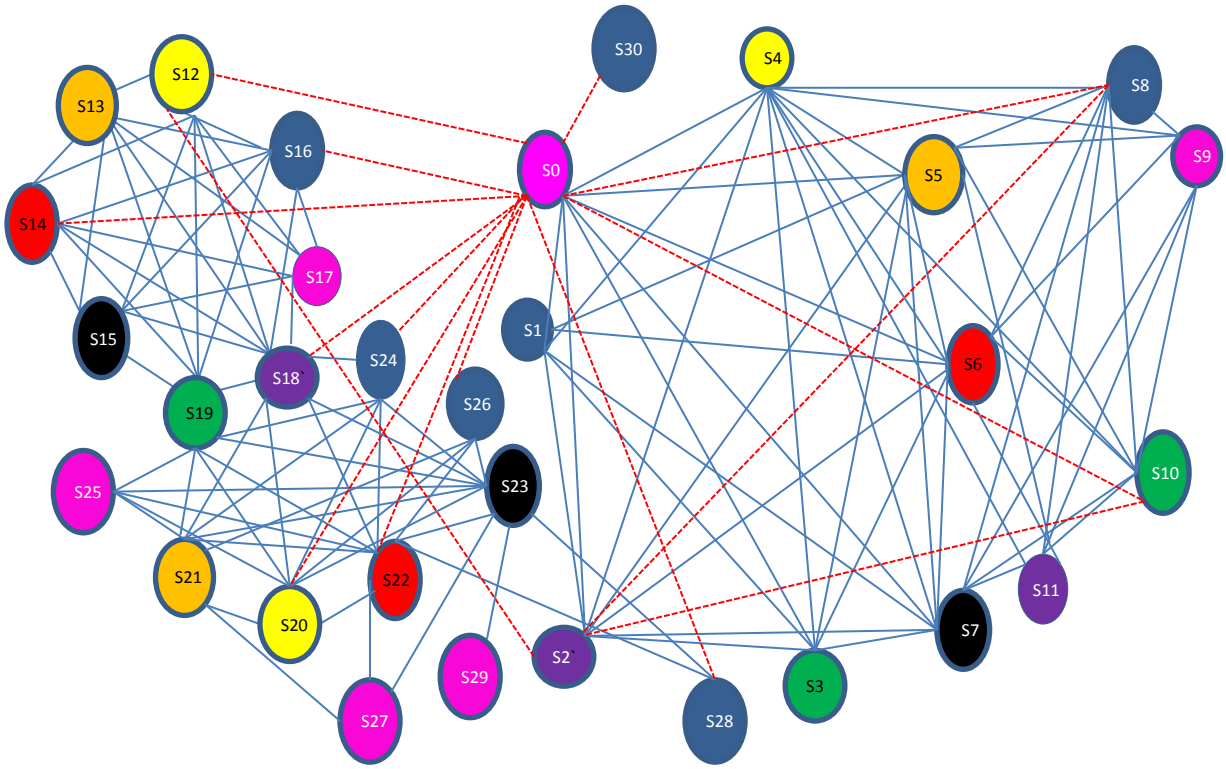

Fig. 6. *CIG of FIR filter hardware accelerator (IP core) after steganography*

hash units. All nine bitstream are directly fed to the nine regular hash units (HU_1- HU_9) respectively. However, the

selection of a specific bitstream (out of 9) fed to a particular key-triggered hash unit is based on a stego-key value of size $\lceil \log_2 9 \rceil = 4$ bits. The overall all stego-key used (in this demonstration) to select the bitstreams fed to the all nine key-triggered hash units are as follows: "1000-0111-0110-0101-0100-0011-0010-0001-0000".

5. *Generating final hash-bitstream of 512-bit long*: Based on the designer chosen stego-key values and the 1024-bit inputs

Table 3. *Register allocation of FIR before implanting steganography*

Q	Q0	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9
P	S0	S8	S16	S24	S25	S26	S27	S28	S29	S30
I	S1	S9	S17	--	--	--	--	--	--	--
V	S2	S10	S18	S18	--	--	--	--	--	--
G	S3	S11	S19	S19	S19	--	--	--	--	--
Y	S4	S4	S12	S20	S20	S20	--	--	--	--
O	S5	S5	S13	S21	S21	S21	S21	--	--	--
R	S6	S6	S14	S22	S22	S22	S22	S22	--	--
B	S7	S7	S15	S23	S23	S23	S23	S23	S23	--

Note, P, I, V, G, Y, O, R and B indicate eight different registers (colours) used to store storage variables S0-S30

Table 4. *Encoded bitstreams corresponding to FIR filter core using nine encoding rules*

Encodings	Encoded bitstream
Z1	11111010101011111010101
Z2	01011010101001010000000
Z3	01011111111101010101010
Z4	10100101010110101111111
Z5	11110101110101110101111
Z6	00001010001010001010000
Z7	00000101010100100101110
Z8	00001010101011011111011
Z9	11111111111111111111111

fed to the hash units, the hash-chaining block generates 512-bit long hash-bitstream which is as follows: "01000011001000100001000000010010110010111010101011001010101001001011001110101001101011110001100100110011010011111000001000001010110111111010010110010111101110011101111101010100010110100001100010100110010001010100001101011110100110100000101111011111011011001010011011011011111100011101100111110111100101110100111000110011110101110010101010110100101000001100000011111001001101011010010010101001010100001111010111111100000010000000001011001100000110111111111011100010110100111010010011111100111010000101000110000"

6. *Hash-bitstream truncation*: The hash-bitstream is truncated to the designer chosen size of stego-constraints. For the stego-constraints size (W) of 26, the truncated bitstream is as follows: "01000011001000100001000000".
7. *Conversion of bits to stego-constraints*: The truncated bitstream contains 20 zeros and 6 ones. Stego-constraints corresponding to 20 zeros are as follows (based on conversion rule shown in Table 2):

   <S0,S2>, <S0,S4>, <S0,S6>, <S0,S8>, <S0,S10>, <S0,S12>, <S0,S14>, <S0,S16>, <S0,S18>, <S0,S20>, <S0,S22>, <S0,S24>, < S0,S26>, <S0,S28>, <S0,S30>, <S2,S4>, <S2,S6>, <S2,S8>, <S2,S10>, <S2,S12>

   These stego-constraints are embedded into the CIG in the form of additional edges.

Table 5. *Register allocation of FIR AFTER implanting steganography*

Q	Q0	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9
P	S0	**S9**	**S17**	--	S25	--	S27	--	S29	--
I	S1	**S8**	**S16**	**S24**	--	**S26**	--	**S28**	--	**S30**
V	S2	**S11**	S18	S18	--	--	--	--	--	--
G	S3	**S10**	S19	S19	S19	--	--	--	--	--
Y	S4	S4	S12	S20	S20	S20	--	--	--	--
O	S5	S5	S13	S21	S21	S21	S21	--	--	--
R	S6	S6	S14	S22	S22	S22	S22	S22	--	--
B	S7	S7	S15	S23	S23	S23	S23	S23	S23	--

Further, stego-constraints corresponding to 6 ones are as follows (based on conversion rule shown in Table 2):

O1-> V1, O2->V2, O3-> V1, O4->V2, O5-> V1, O6->V2

i.e. odd operations (O1, O3, O5) are assigned to vendor type-1 (V1) and even operations (O2, O4, O6) are assigned to the vendor type-2 (V2).

8. *Embedding stego-constraints during HLS process*: The constraint edges (stego-constraints corresponding to bit '0') are embedded into the CIG one by one. During embedding of an edge as a stego-constraint, the condition that the respective two nodes should be of different color must be taken care of. If it is not so, then the color of one of the node is modified

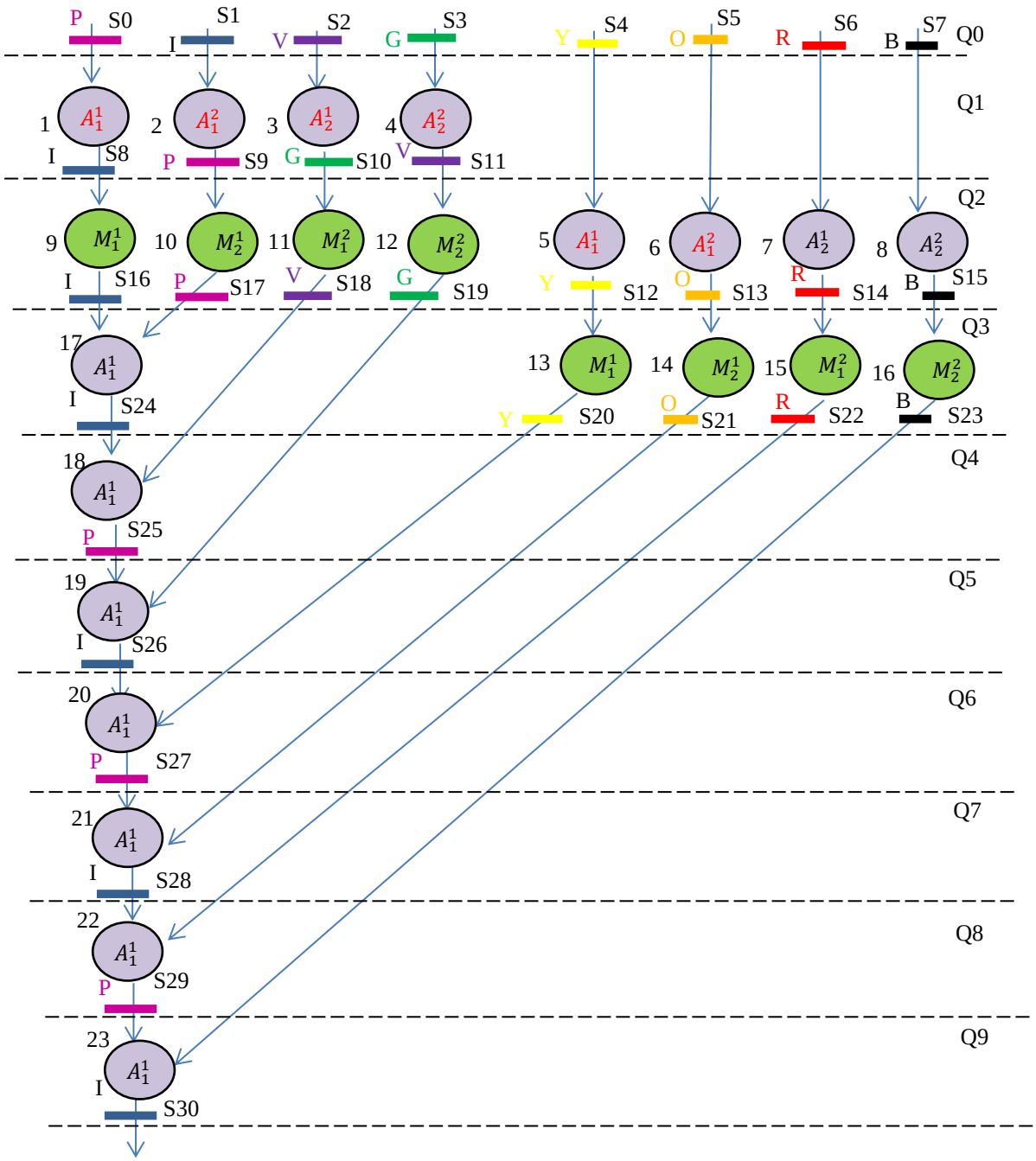

Fig. 7. *Scheduled DFG of FIR filter application after embedding steganography*

in order to accommodate that edge. For example, before embedding the constraint edge <S0,S8>, both nodes (S0 and S8) are of same color as shown in Fig. 5. However, embedding the edge <S0,S8> results into change in the color of node S8 from Pink (P) to Indigo (I) as shown in Fig. 6. The Fig. 6 shows the CIG after embedding all constraint edges. The embedded constraint edges have been drawn as red dotted lines in Fig .6. As shown in the figure, the color of storage variables/nodes S8, S9, S10, S11, S16, S17, S24, S26, S28 and S30 are modified to embed all the constraint edges. Table 5 captures the register allocation of storage variables into different control steps post embedding stego-constraints. The storage variables subjected to register re-allocation post embedding constraints have been marked grey in the table. Further, to embed the constraints corresponding to all 6 ones, reallocation of operations (O1-O6) to the particular vendor type is performed based on the rule shown in Table 2. The scheduled DFG of FIR application with embedded stego-constraints (corresponding to both 0s and 1s) is shown in Fig. 7. As shown in the figure, the FUs (corresponding to O1-O6) highlighted in red color indicate the embedded stego-constraints corresponding to bit '1'. Embedding stego-constraints produces a stego-embedded FIR filter core which is secured with designer's robust stego-mark.

7.5. Key-triggered Hash-chaining Driven Steganography Tool for Securing Hardware Accelerators

Authors have developed a ***KHC-Stego tool*** (Key-triggered Hash-chaining Driven Steganography tool) to simulate and analyse the functionality of key-triggered hash-chaining based steganography approach (Sengupta and Rathor, 2020) for DSP hardware

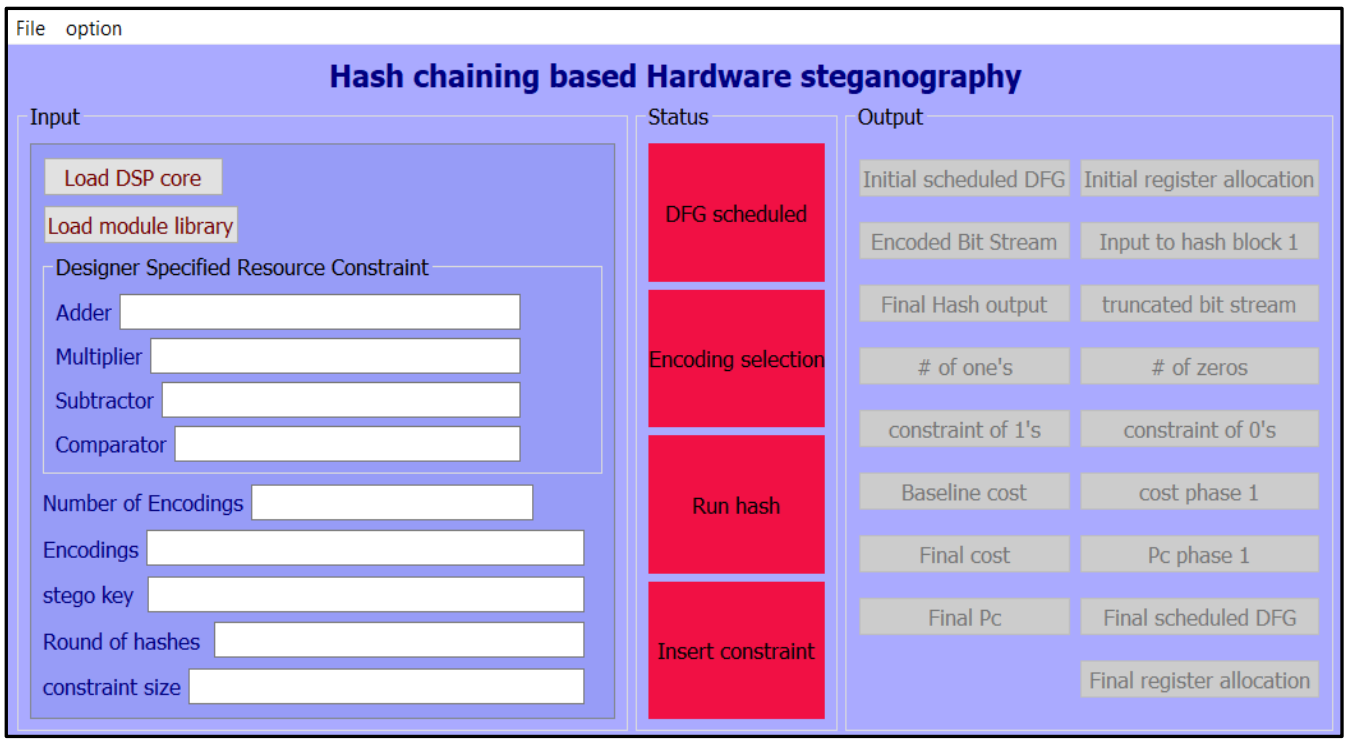

Fig. 8. *A snapshot of key-triggered hash-chaining driven steganography tool*

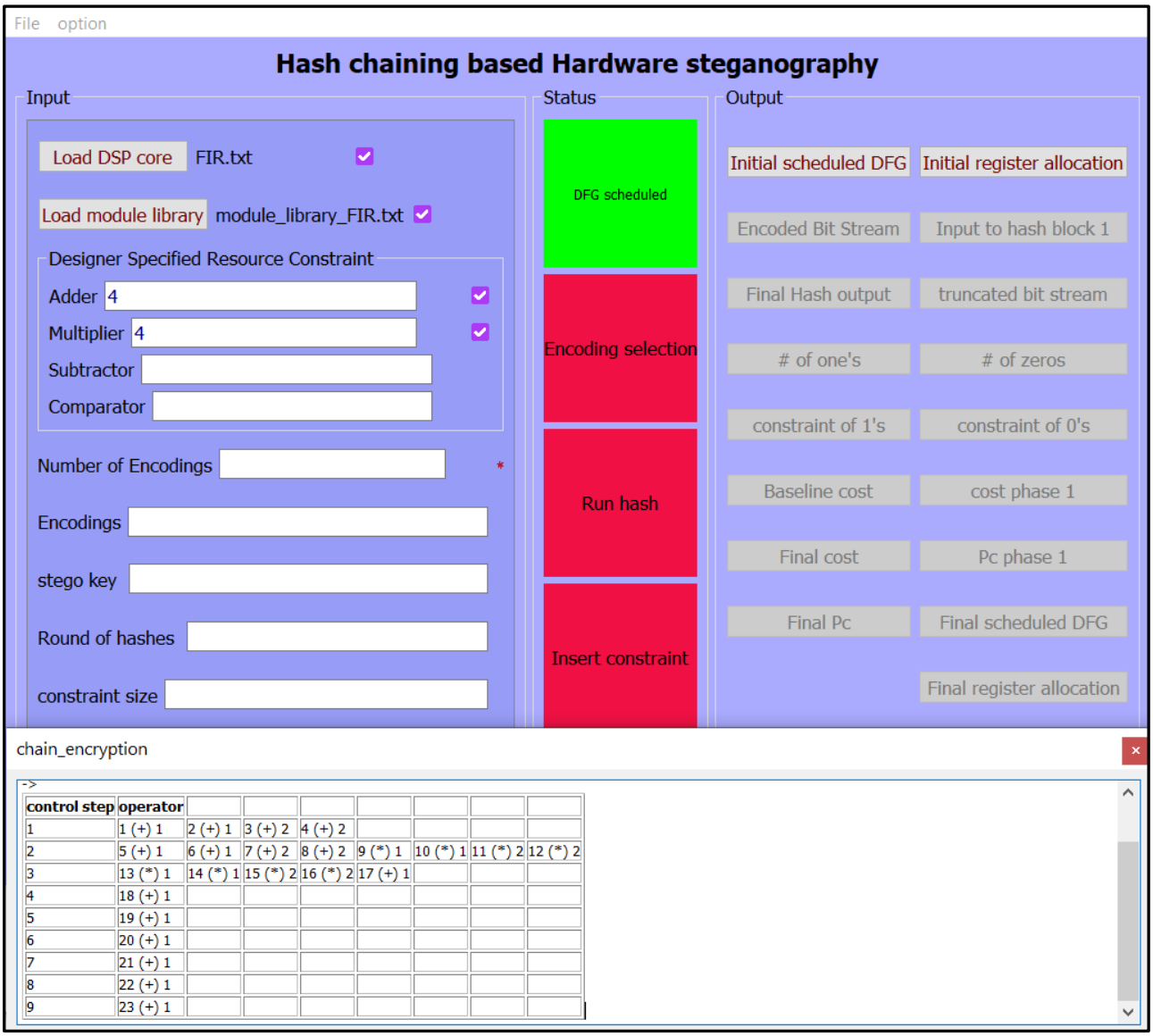

Fig. 9. *Snapshot post entering input DSP application, library and constraints size*

accelerators. This tool provides a friendly graphical interface to designers. A snapshot of the graphical user interface (GUI) of the tool is shown in Fig. 8. The left portion of the tool shows the panel for providing required inputs to the tool, right portion shows the panel with output buttons to see the intermediate and final outputs of the key-triggered hash-chaining based steganography approach. The panel in the middle shows the status of the intermediate steps viz. CDFG scheduling of DSP application, encoding selection, executing hash-chaining process and insert constraints, as shown in Fig. 8. Initially, these status bars remain red. Upon applying the inputs, the respective status bar turns green. The KHC-Stego tool accepts the DSP application input in the form CDFG along with module library and resource constraints. The tool shows output of intermediate steps of key-triggered hash-chaining based steganography, finally generated stego-constraints, the security metric in terms of probability of coincidence, pre

Fig. 10. *Snapshot of tool showing different encodings to be chosen*

and post-steganography design cost. Further, it also shows scheduling and registers allocation pre and post-embedding steganography constraints, onto the output window.

Let's generate all the intermediate and final output of key-triggered hash-chaining based steganography approach for FIR filter core using the KHC-Stego tool. We will provide the same inputs and stego-keys used during the demonstration of FIR filter core discussed in section 7.4. Here, we can match the output generated with the tool and that obtained in the demonstration. First of all, inputs CDFG of FIR filter core, resource constraints of 4 adder & 4 multipliers and module library, are fed to the tool as shown in Fig. 9. On clicking on the button "Initial scheduled DFG" on output panel, the scheduled DFG becomes available onto the output window shown at the bottom of the tool. As the DFG gets scheduled, the respective status bar ("DFG scheduled" shown in Fig. 9) turns green. Whereas, the remaining status bar are still red as shown in Fig. 9. Further, number of encodings is entered in the tool.

Fig. 11. *Snapshot of tool showing encoded bitstreams of scheduled DFG of FIR filter*

The tool pops up a window which shows the definitions of nine different encodings, as shown in Fig. 10. Further the encoding numbers are entered in the tool as shown in Fig. 11. Upon clicking on the output button "Encoded Bitstream" at output panel, the nine different encoded bitstreams of scheduled DFG of FIR filter are available onto the output window as shown in Fig. 11. Further, stego-key value for each key-driven hash block and number of rounds for all eighteen hash blocks in the hash-chain are entered in the tool as shown in Fig. 12. The final hash output is also shown in Fig. 12 on to the output window. Further, Fig. 13 shows that the constraints size=26 has been fed as input and the final truncated bitstream is made available on to the output window by clicking on the button "truncated bitstream". The number of zeros and ones in the truncated bitstream can also be

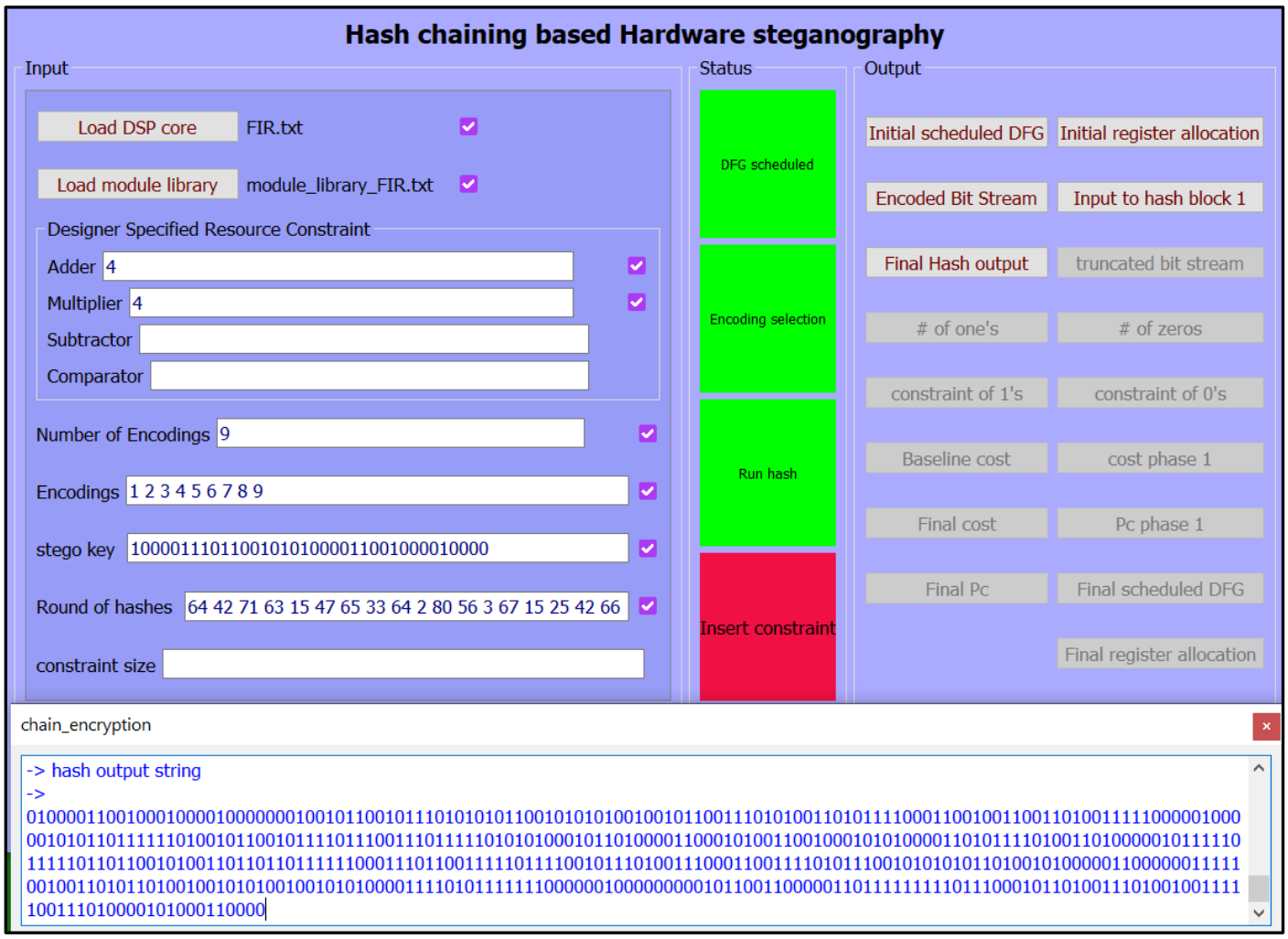

Fig. 12. *Snapshot of tool post entering encoding numbers, stego-key and rounds of hash*

shown at output window by clicking on the respective buttons on output panel. As shown in all the snapshots of the tool, same inputs and stego-keys that are used in the demonstration of FIR filter core in section 7.4 have been fed here. The tool produces the desired outputs that match with the demonstration of FIR filter core in section 7.4. Further, Fig. 14 shows the register allocation post-embedding stego-constraints corresponding to '0' bits. Here, values under the column headings 1 to 8 show the storage variable (S) number and the heading of the column show the register number, where Pink, Indigo, Violet, Green, Yellow, Orange, Red and Black registers have been denoted by the number 1, 2, 3, 4, 5, 6, 7 and 8 respectively. The rows heading (0 to 9) show the control step numbers. As shown in the figure, this register allocation post-embedding constraints matches with that demonstrated in section 4.7. Further, Fig. 15 shows the scheduling of CDFG of FIR filter post-embedding key-triggered hash-chaining based steganography. This scheduled CDFG captures the impact of embedding stego-constraints corresponding to bit 1' in the form of FU vendor re-allocation. Additionally, design cost pre and post-embedding steganography can be seen at output

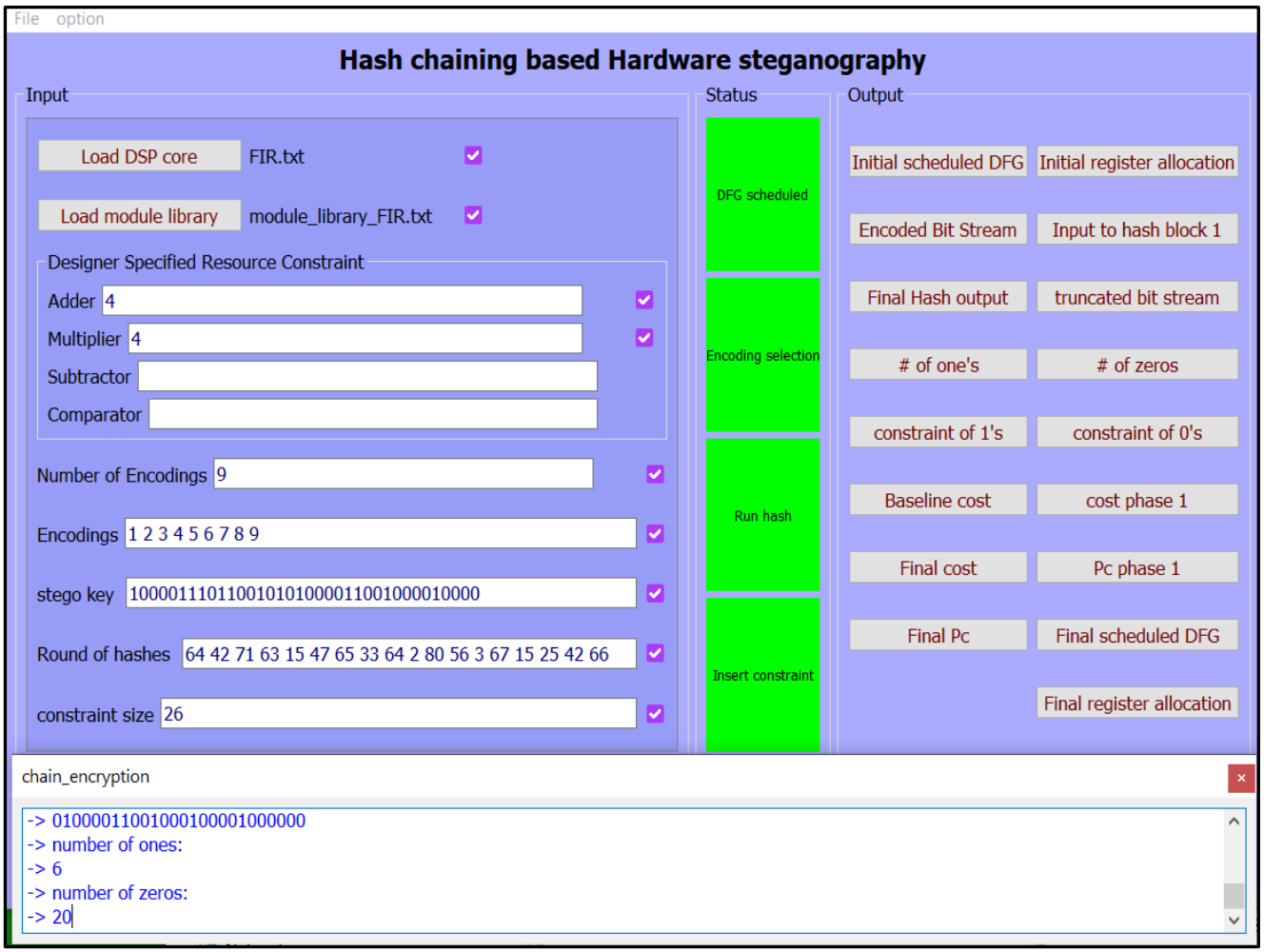

Fig. 13. *Snapshot of tool post entering constraint size*

window by clicking on the respective buttons on output panel as shown in Fig. 16. Further, the value of probability of coincidence metric can also be seen at output window of the tool as shown in Fig. 17.

Thus the key-triggered hash-chaining based steganography approach can be simulated and analysed using the KHC-stego tool developed by the authors. This tool is useful for analysing various kinds of DSP hardware accelerator applications.

7.6. Analysis on Case Studies

The case studies of different DSP applications have been analysed to show the security achieved through key-triggered hash-chaining steganography approach (Sengupta and Rathor, 2020). Further a security comparison, in terms of key strength, has been shown with respect to contemporary watermarking and steganography approaches. Furthermore, the impact of key-triggered hash-chaining steganography approach on design cost has been assessed by comparing it with the baseline design cost (before steganography). The security and design cost analysis along with comparative assessment are discussed in following sub-sections:

1. ***Security Analysis*** *(Sengupta and Rathor, 2020)*

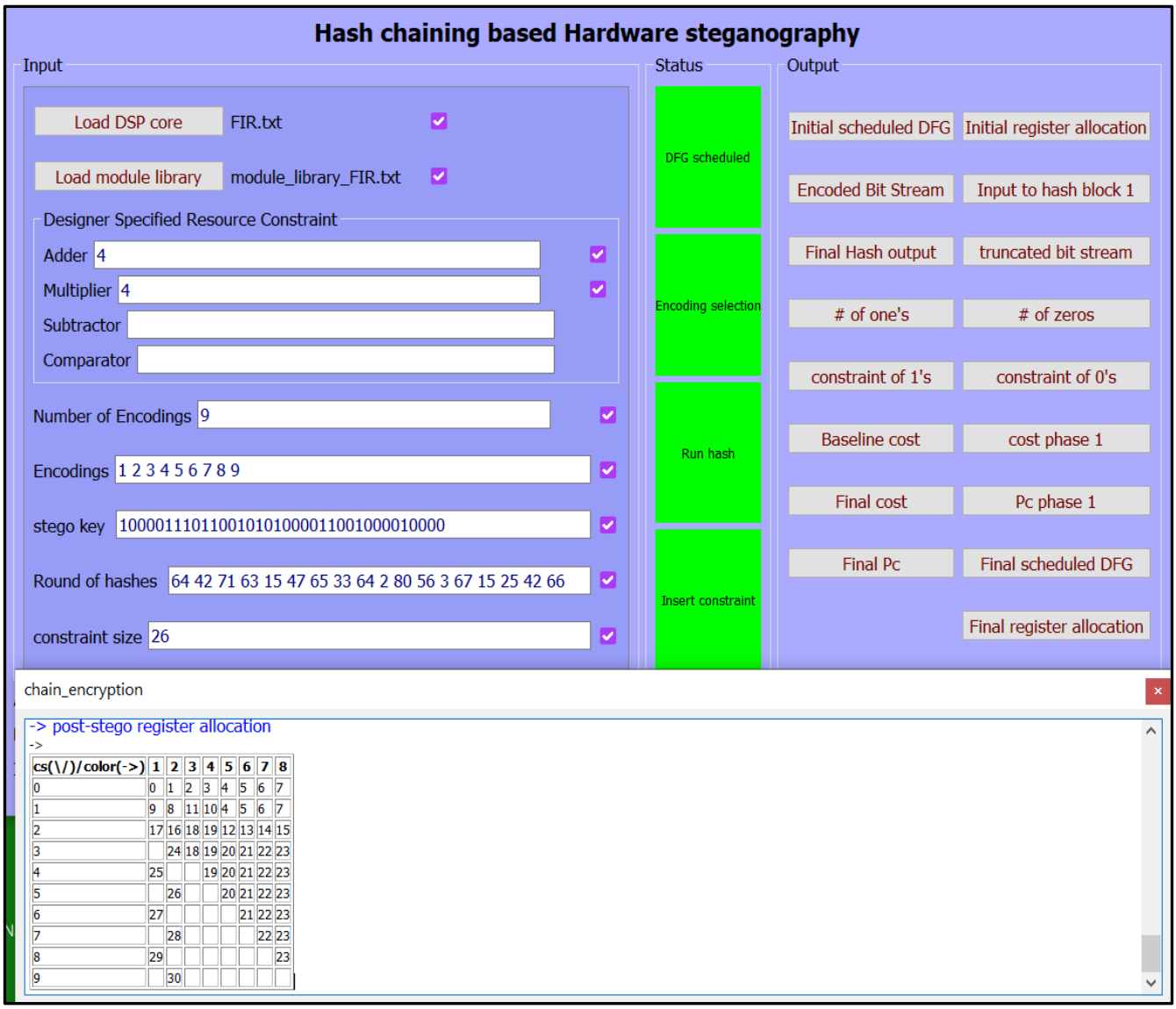

Fig. 14. *Snapshot of tool showing register allocation post embedding constraints*

The key-triggered hash-chaining steganography approach (Sengupta and Rathor, 2020) targets the security of DSP cores against false claim of ownership and counterfeiting/cloning threats. The false claim of ownership can be nullified by embedding significant amount of digital evidence which shows the proof of ownership. The strength of ownership proof is assessed in terms of probability of coincidence (Pc) which represents the probability of coincidently finding designer's stego-mark in a non-steganography embedded design. The Pc metric is evaluated using following formula (Sengupta and Rathor, 2019b):

$$Pc = \left(1 - \frac{1}{h}\right)^{k1} \times \left(1 - \frac{1}{\pi_{j=1}^{m} N(Uj)}\right)^{k2} \qquad (8)$$

Where, h indicates the number of colours/registers in the CIG of DSP hardware accelerator design before steganography and k1 indicates the number of stego-constraints embedded during the register allocation phase (i.e. number of 0s embedded). Further, k2

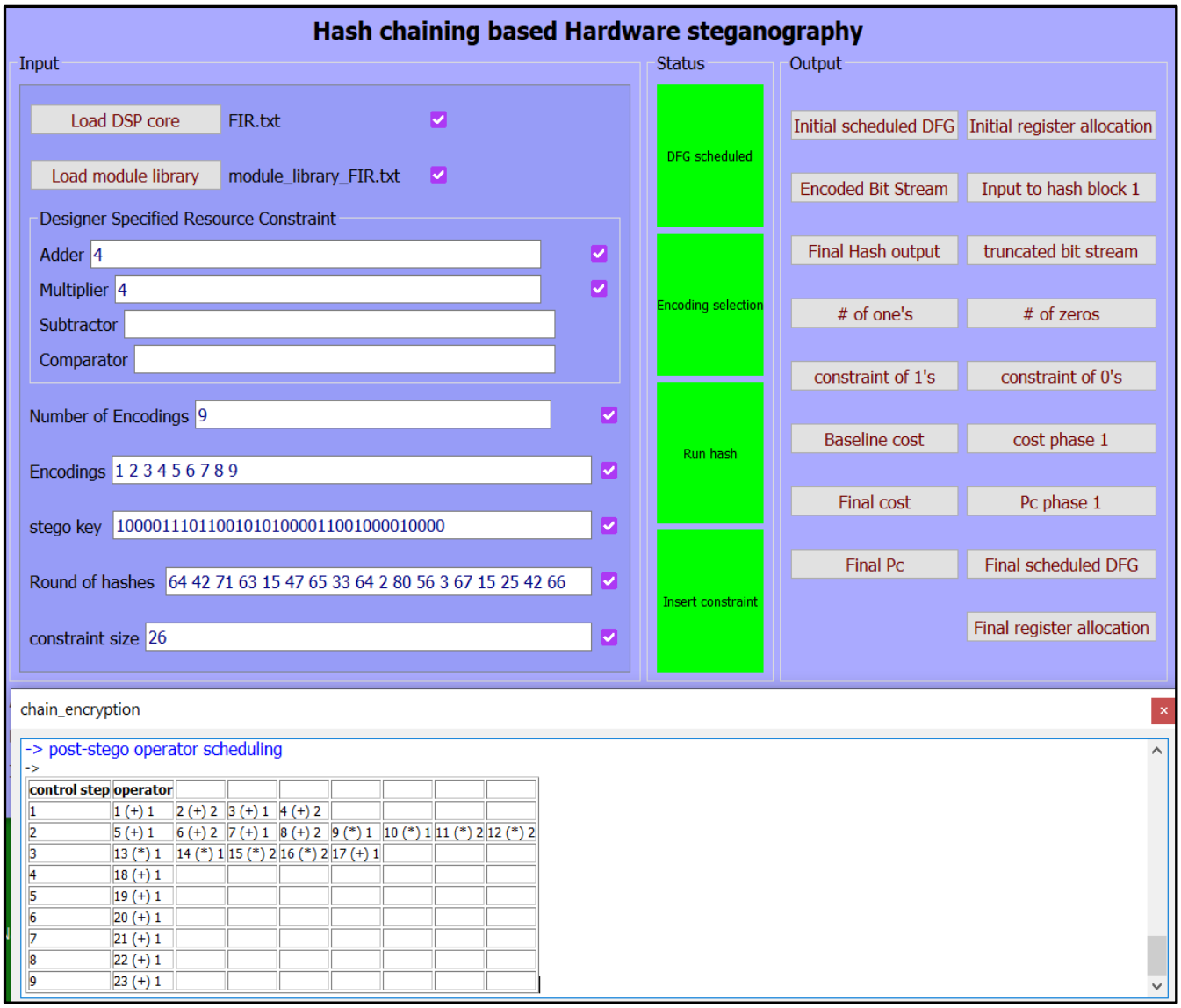

Fig. 15. *Snapshot of tool showing scheduling and allocation post embedding constraints*

indicates the number of stego-constraints embedded during the FU resource allocation phase (i.e. effective number of 1s embedded), N(Uj) indicates the number of resources of FU type Uj and m indicates the total types of FU resources present in the DSP hardware accelerator design. Here, k1(number of stego-constraints embedded during register allocation) and k2 (number of stego-constraints embedded FU allocation phase) indicate the amount of digital evidence hidden within the design. Higher the value of k1 and k2 (i.e. higher digital evidence), lower is the Pc value which in turn indicates the higher strength of ownership proof. Table 6 shows the number of stego-constraints k1 and k2 and total constraints size W=k1+k2 for various DSP applications. Further, Fig. 18 shows probability of coincidence achieved through key-triggered hash-chaining steganography approach (Sengupta and Rathor, 2020) post embedding stego-constraints (k1) corresponding to bit '0'. Fig. 19 shows the final probability of coincidence achieved post embedding stego-constraints (k1+k2) corresponding to bits '0' and '1'. Further, variation

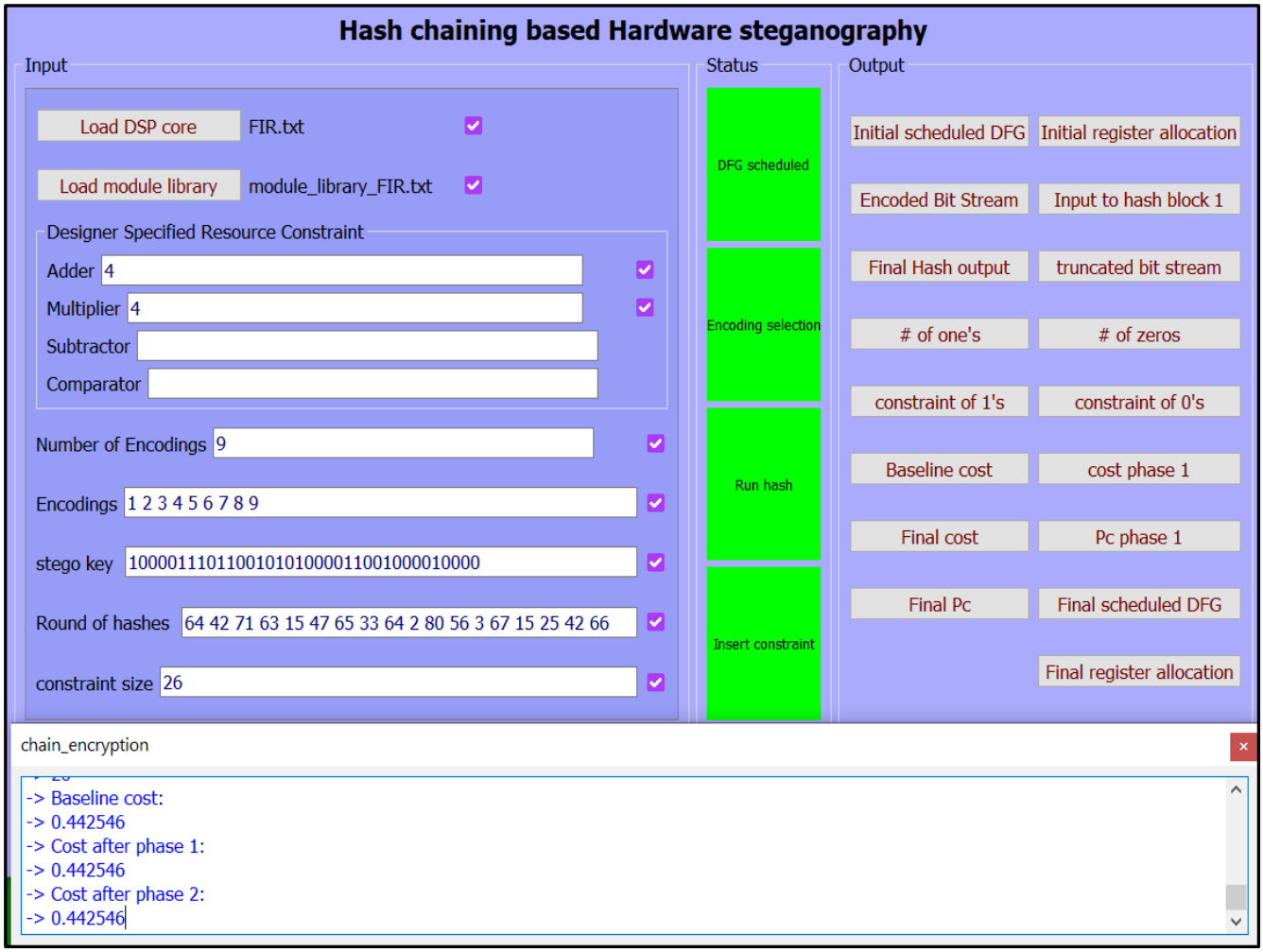

Fig. 16. *Snapshot of tool showing design cost pre and post embedding stego-constraints*

in Pc for increasing number of constraints is shown in Fig. 20. As shown in the figure, the Pc value decreases on increasing the size of stego-constraints (W=k1+k2), indicating higher strength of ownership proof. In addition, counterfeiting can be detected by inspecting the presence of stego-mark in a counterfeited IP/IC. The counterfeited IP/IC does not contain the authentic stego-mark of original vendor. Further, cloned IPs/ICs are detected by verifying the presence of authentic stego-mark in the IPs/ICs of different brands. The cloned IPs/ICs shall contain the authentic stego-mark of original owner, hence detected.

However, if embedded stego-mark or stego-constraints is compromised or leaked to an attacker, then the purpose of embedding steganography is defeated. Therefore, an embedded stego-mark is required to be highly secured and only the IP owner should be able to generate it. In order to obtain a highly robust stego-mark, the key-triggered hash-chaining steganography approach (Sengupta and Rathor, 2020) generates it using designer specified stego-key. Moreover, the encoded bitstreams used in the stego-constraints generation process also enhance the robustness of the stego-mark. Both the stego-keys and the encoded bitstreams are only known to the designer. For an attacker, to find the stego-constraints, both the stego-keys and encoded bitstreams need to be

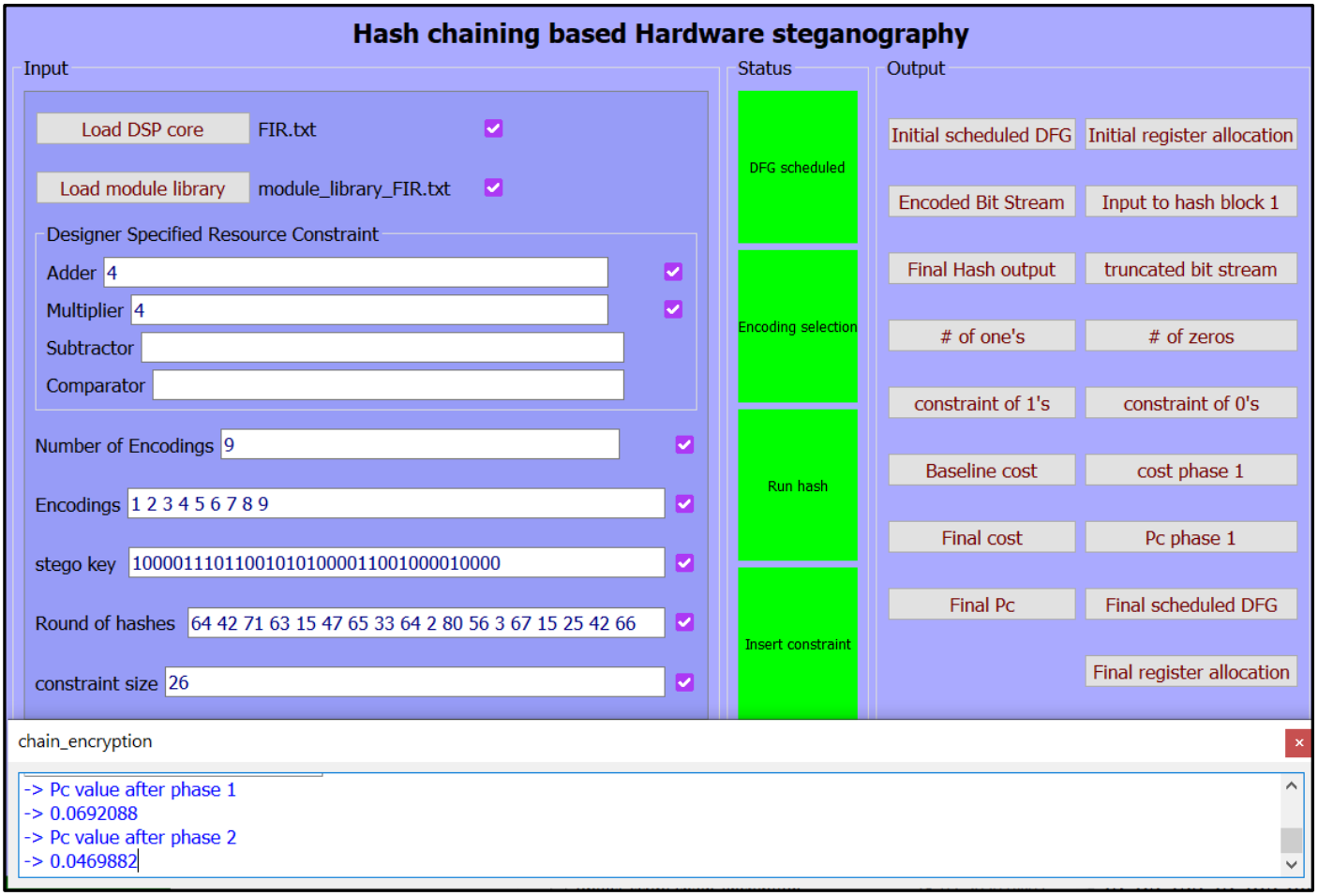

Fig. 17. *Snapshot of tool showing probability of coincidence (Pc) value post embedding stego-constraints corresponding to bits '0' and '1'*

Table 6. *Stego-constraints k1 and k2 and total constraints size W=k1+k2 for various DSP applications*

DSP applications	# of registers (h) before steganography	Resource constraints	Total constraint size (W=k1+k2)	#Constraints	
				k1 (#0s)	k2 (#1s)
DCT	8	1A, 4M	23	13	10
FIR	8	4A, 4M	26	20	6
JPEG_IDCT	29	12A, 12M	312	203	109
MPEG	14	3A, 7M	37	21	16
JPEG_sample	12	6A, 1M	51	30	21
EWF	7	2A, 1M	52	34	18

determined. From an attacker's perspective, the maximum possible stego-key size required to generate the stego-constraints is

reported in Table 7. Further, Table 7 also shows the required attacker's maximum effort to find out the valid key value. Additionally, an attacker is also required to find out the total encoded bits (in the bitstreams) used during stego-constraints generation. The attacker's effort of deriving actual encoded bits through brute force is reported in Table 7. The overall security in

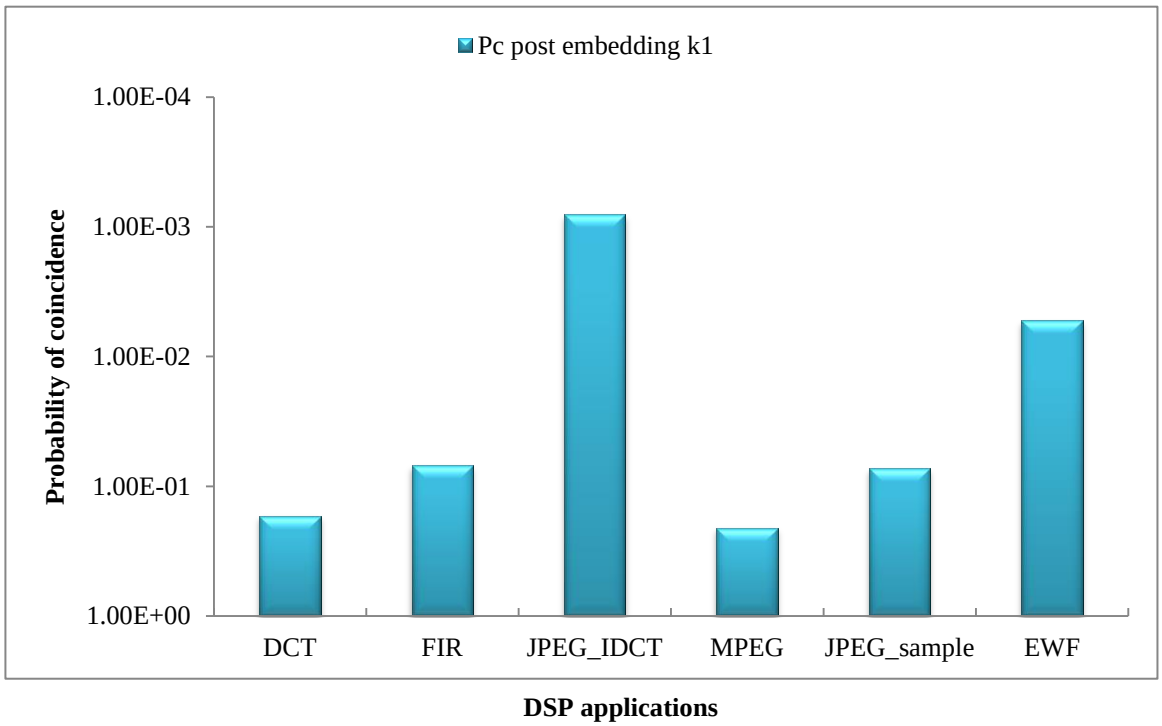

Fig. 18. *Probability of coincidence post embedding stego-constraints k1*

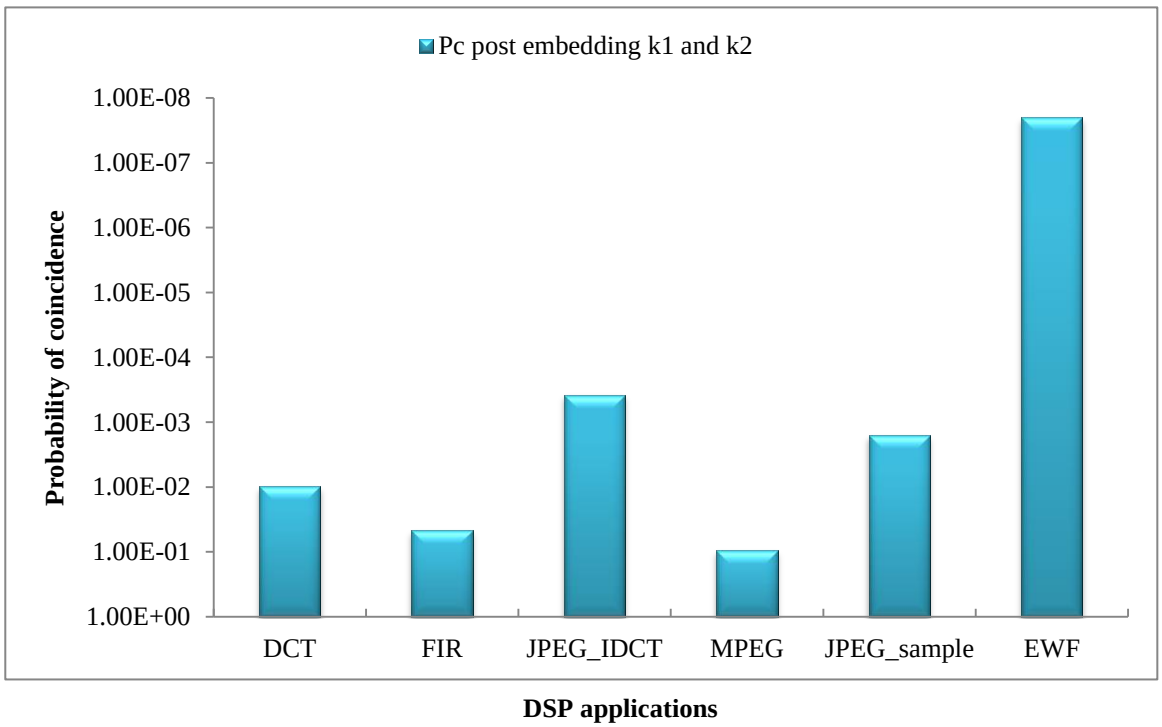

Fig. 19. *Final Probability of coincidence post embedding stego-constraints k1 and k2*

terms of attacker's total effort of finding stego-constraints is also shown in Table 7. As evident from the table, determining stego-constraints becomes almost infeasible for an attacker because of huge effort required to find out the stego-key value and encoded bits.

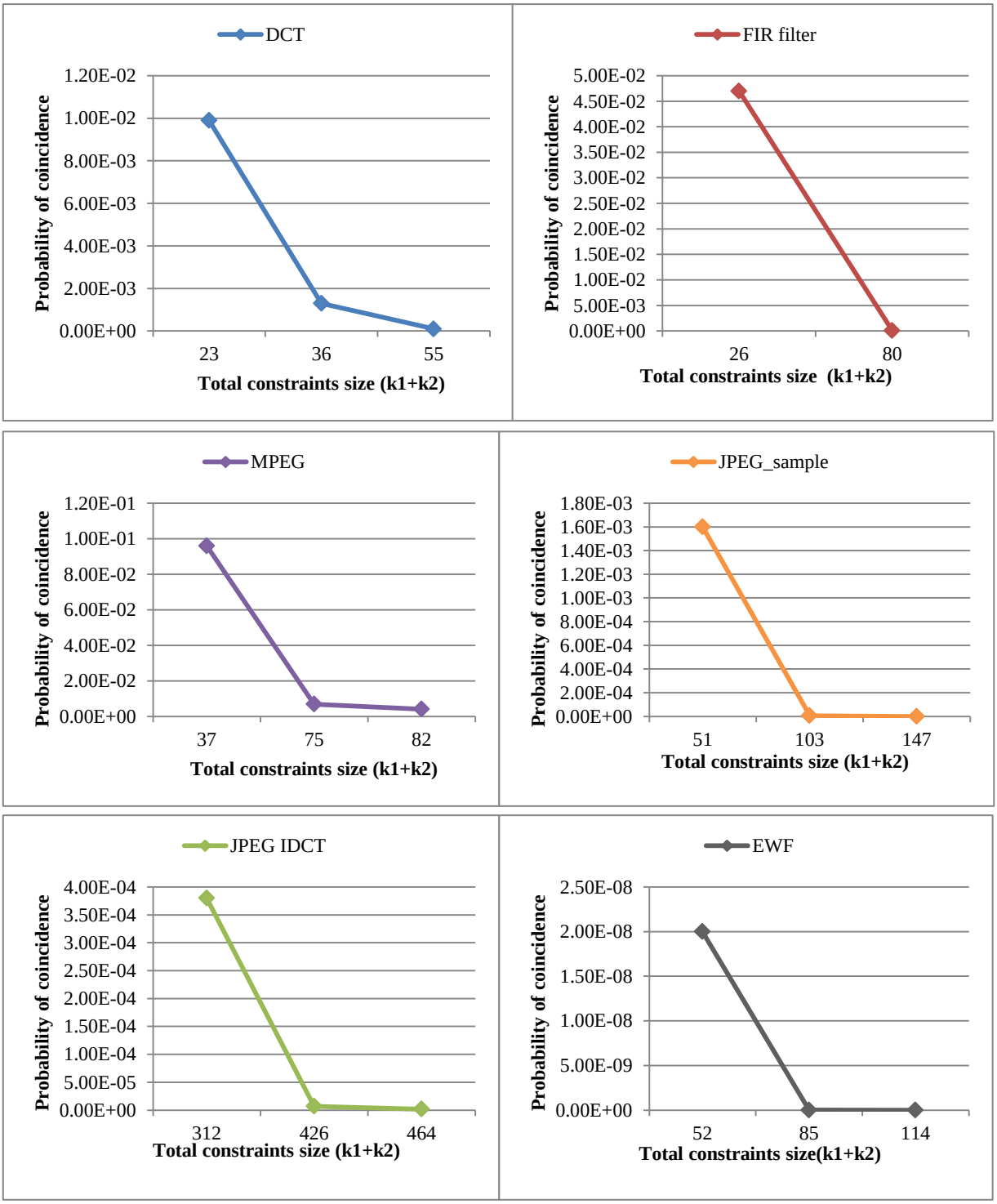

Fig. 20. *Impact of increasing stego-constraints size on probability of coincidence (Pc)*

Tamper tolerance: The key-triggered hash-chaining steganography approach (Sengupta and Rathor, 2020) is not vulnerable to tampering in contrast to watermarking based approaches (Sengupta and Bhadauria, 2016; Sengupta and Roy, 2017; Koushanfar *et al*., 2005). This is because, unlike watermarking, the steganography is free from signature digits combinations.

Security comparison with contemporary approaches: Watermarking and steganography are the contemporary approaches that secure an IP core against ownership abuse and counterfeiting/cloning. However, the purpose of watermarking approach fails if the designer signature gets leaked to an attacker. This is because, the watermarking approaches does not use secret key to generate the watermarking constraints. Similarly, steganography approach proposed by (Sengupta and Rathor, 2019a) does not employ a secret key to generate stego-constraints. Though the steganography approach proposed by (Sengupta and Rathor, 2019b) uses stego-key, however the key size is very small compared to the key-triggered hash-chaining steganography approach (Sengupta and Rathor, 2020). The comparison of key size with respect to (Sengupta and Rathor, 2019a; Sengupta and Rathor, 2019b; Sengupta and

Table 7. *Security analysis of key-triggered hash-chaining based steganography in terms of stego-key strength and attacker's total effort of finding stego-constraints*

DSP applications	Maximum key size using (1)	Attacker's maximum effort in terms of finding valid key (using (4))	Attacker's maximum effort in terms of finding encoded bits (using (6))	Attacker 's total effort (using (7))
DCT	491520	$>10^{147603}$	10^{2059}	$>10^{149662}$
FIR	192937984	$>10^{57939334}$	10^{2059}	$>10^{57941393}$
JPEG_IDCT	5.8153×10^{35}	$>10^{1.74\times10\text{^}35}$	10^{2059}	$>10^{1.74\times10\text{^}35}$
MPEG	7516192768	$>10^{2277634172}$	10^{2059}	$>10^{2277636231}$
JPEG_sample	283467841536	$>10^{85899345920}$	10^{2059}	$>10^{85899347979}$
EWF	584115552256	$>10^{177004712804}$	10^{2059}	$>10^{177004714863}$

Bhadauria, 2016) is shown in Table 8. As shown in the table, the key-size in the watermarking approach (Sengupta and Bhadauria, 2016) and steganography approach (Sengupta and Rathor, 2019a) is zero because of non-involvement of secret key to generate the secret constraints. Further, it is evident from the table that the key-triggered hash-chaining based steganography offers very high security in terms of robustness of the stego-mark (security of stego-constraints). This renders an attacker almost infeasible to find out the stego-constraints embedded into the design.

2. Design Cost Analysis *(Sengupta and Rathor, 2020)*

This sub-section discusses the impact of employing key-triggered hash-chaining driven steganography based security on design cost. Following equation is used to evaluate the design cost (Sengupta and Rathor, 2019b):

$$C_d(U_i) = \rho_1 \frac{L_d}{L_m} + \rho_2 \frac{A_d}{A_m} \qquad (9)$$

Where, $C_d(U_i)$ is the design cost of DSP cores for resource constraints U_i, further L_d and L_m are the design latency at specified resource constraints and maximum design latency respectively, A_d and A_m are the design area at specified resource constraints and maximum area respectively and ρ_1, ρ_2 are the weights which are fixed at 0.5. Variation in the design cost for increasing size of stego-constraints is shown in Fig. 21. As shown in the figure, the impact of increasing stego-constraint size on design cost is

Table 8. *Key size comparison of key-triggered hash-chaining based steganography with contemporary approaches*

DSP applications	Maximum key size in bits		
	key-triggered hash-chaining steganography approach	(Sengupta and Rathor, 2019b)	(Sengupta and Rathor, 2019a; Sengupta and Bhadauria, 2016)
DCT	491520	610	0
FIR	192937984	625	0
JPEG_IDCT	5.8153×10^{35}	785	0
MPEG	7516192768	620	0
JPEG_sample	283467841536	665	0
EWF	584115552256	640	0

either zero or nominal.

Design cost comparison with baseline: Design cost comparison with the baseline is shown in Fig. 22 for a particular constraint size (W). As shown in the figure, the design cost may increase by a marginal value because of possibility of increment in the number of registers required to embed the stego-constraints. However, no extra register is required for most of the DSP

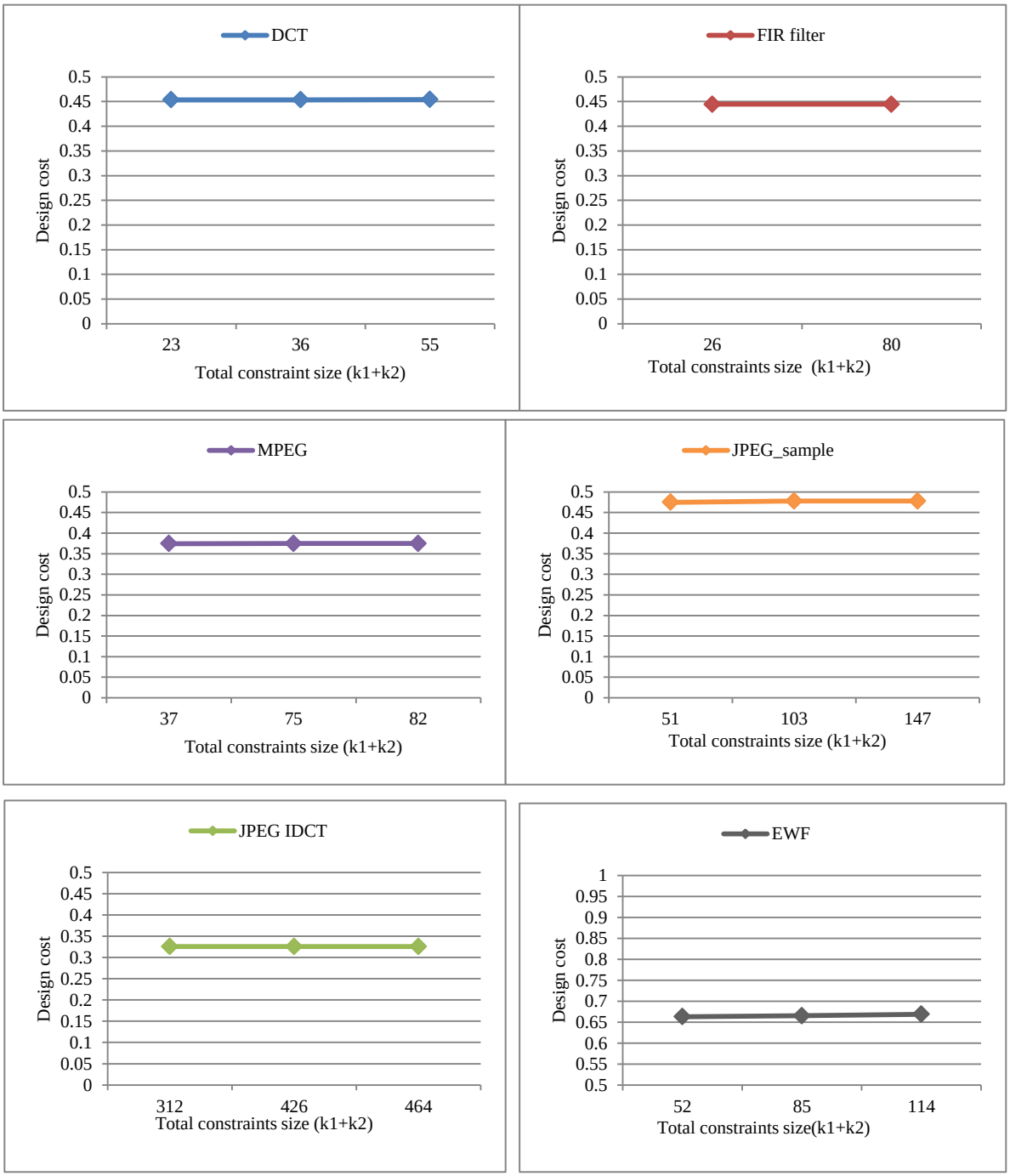

Fig. 21. *Impact of increasing stego-constraints size on design cost of key-triggered hash-chaining steganography*

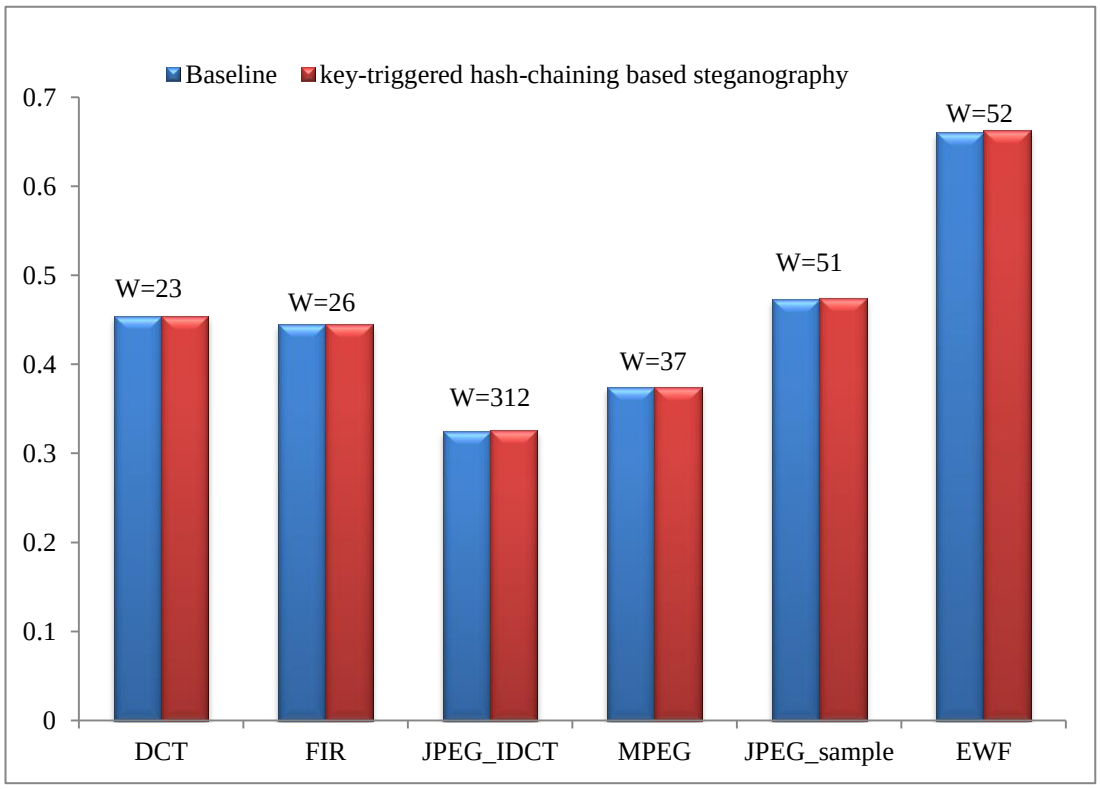

Fig. 22. *Design cost comparison of key-triggered hash-chaining steganography approach with baseline*

***Note**: W indicates the stego-constraints size*

applications. This indicates that the key-triggered hash-chaining steganography approach (Sengupta and Rathor, 2020) achieves very high security at almost zero overhead.

7.7. Conclusion

This chapter discusses a key-triggered hash-chaining based hardware steganography approach (Sengupta and Rathor, 2020) which offers very high security against false claim of IP ownership threat. Additionally, the key-triggered hash-chaining steganography approach is also capable to detect counterfeited/cloned IPs/ICs. The stego-mark generated through the key-triggered hash-chaining steganography approach is highly robust as it is produced using secret stego-key of very large size; designer selected encoded bitstreams and number of iterations of round function in each hash unit of the chaining process. The robustness of the stego-mark has been evaluated in terms of key size, attacker's total effort of finding stego-constraints and probability of coincidence. The case studies show that the key-triggered hash-chaining steganography approach provides higher security than contemporary approaches at trivial design overhead.

At the end of this chapter, a reader gains following concepts:

- Need of security of DSP hardware accelerators

- Key-triggeredhash-chaining based steganography methodology
- Various encodings of a DSP applications
- Role of encoded bitstreams of a DSP application in key-triggeredhash-chaining based steganography
- Role of hash units in key-triggeredhash-chaining based steganography
- Concept of regular and key-triggeredhash units
- Stego-embedder block in key-triggeredhash-chaining based steganography
- Detection of key-triggeredhash-chaining based steganography
- Security achieved using key-triggeredhash-chaining based steganography from an attcker's perspective
- Design process of obatining stego-embedded FIR filter core uing key-triggeredhash-chaining based steganography
- Analysis on case studies of various DSP applications, in terms of security and design cost

7.8. Questions and Exercise

1. Explain the threat model used in key-triggered hash-chaining based steganography.
2. Explain the role of encoded bitstreams of a DSP application in key-triggered hash-chaining based steganography.
3. Explain role of hash units in key-triggered hash-chaining based steganography.
4. Explain the concept of regular and key-triggered hash units.
5. Explain the function of stego-embedder block in key-triggered hash-chaining based steganography.
6. Explain the function of detection of key-triggered hash-chaining based steganography.
7. What is the significance of parallel switch blocks?
8. How many encoding algorithms can be used in the key-triggered hash-chaining based steganography?
9. What is the output bit size of the bit padding block? How is this size determined?
10. What is the output bit size of the parallel switch blocks?
11. What is the maximum key size of the stego-key block?
12. What is the rule of constructing 1024 bits through pre-processing?
13. Explain the different encoding rules used to encode a DSP application into bitstreams.
14. What is the attacker's maximum effort of finding the stego-key?
15. Determine the total encoded bits used in hash-chaining block to generate stego-constraints.
16. How to determine an attacker's total effort in determining the stego-constraints embedded into the design?
17. What is a KHC-Stego tool?
18. How many phases are used for embedding stego-constraints into the design?
19. Compare any hardware watermarking with key-triggered hash-chaining based steganography, in terms of security achieved and design overhead.

References

A. Sengupta, D. Kachave and D. Roy (2019a), 'Low cost functional obfuscation of reusable IP cores used in CE hardware through robust locking,' *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst*., vol. 38(4), pp. 604-616.

S. M. Plaza, I. L. Markov (2015), 'Solving the Third-Shift Problem in IC Piracy With Test-Aware Logic Locking,' *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*,vol. 34(6), pp. 961-971.

A. Sengupta (2017), 'Hardware Security of CE Devices [Hardware Matters],'*IEEE Consumer Electronics Mag*, vol. 6(1), pp. 130-133.

A. Sengupta (2016), 'Intellectual Property Cores: Protection designs for CE products, '*IEEE Consumer Electronics Mag*, vol. 5, no. 1, pp. 83-88.

R. S. Chakraborty and S. Bhunia (2009), 'HARPOON: An obfuscation-based SoC Design methodology for hardware protection,' *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst*., vol. 28(10), pp. 1493-1502.

A. Sengupta, S. Bhadauria (2016), 'Exploring Low Cost Optimal Watermark for Reusable IP Cores During High Level Synthesis,'*IEEE Access*, vol. 4, pp. 2198-2215,.

M. Yasin, J. J. Rajendran, O. Sinanoglu and R. Karri (2016), 'On improving the security of logic locking,' *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst*., vol. 35(9), pp. 1411-1424.

J. Zhang (2016), 'A practical logic obfuscation technique for hardware security,' *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 24(3), pp. 1193–1197.

B. Colombier and L. Bossuet (2015), 'Survey of hardware protection of design data for integrated circuits and intellectual properties,' *IET Computers & Digital Techniques*, vol. 8(6), pp. 274-287.

R. D. Newbould, J. D. Carothers, and J. J. Rodriguez (2002), 'Watermarking ICs for IP protection,' *IET Electron. Letters*, vol. 38(6), pp. 272–274.

A. Sengupta, S. P. Mohanty (2019), 'IP core and integrated circuit protection using robust watermarking', *Book: 'IP Core Protection and Hardware-Assisted Security for Consumer Electronics*', e-ISBN: 9781785618000, pp. 123-170.

A. Sengupta, D. Roy, S. P. Mohanty (2019b), 'Low-Overhead Robust RTL Signature for DSP Core Protection: New Paradigm for Smart CE Design,' *Proc. 37th IEEE International Conference on Consumer Electronics (ICCE),* pp. 1-6.

B. Le Gal, L. Bossuet (2012), 'Automatic low-cost IP watermarking technique based on output mark insertions,' *Design Autom.Embedded Syst*., vol. 16(2), pp. 71-92.

F. Koushanfar et al. (2005), "Behavioral synthesis techniques for intellectual property protection," *ACM Trans. Des. Autom. Electron. Syst*., vol. 10(3), pp. 523-545.

A. Sengupta, D. Roy (2017), 'Antipiracy-Aware IP Chipset Design for CE Devices: A Robust Watermarking Approach [Hardware Matters],'*IEEE Consumer Electronics Mag*, vol. 6(2), pp. 118-124.

A. Sengupta, D. Roy, S. P. Mohanty (2018), 'Triple-Phase Watermarking for Reusable IP Core Protection During Architecture Synthesis,'*IEEE Trans. Comput.-Aided Design Integr. Circuits Syst*, vol. 37(4), pp. 742-755.

A. Sengupta, R. Sedaghat, Z. Zeng (2010), 'A high level synthesis design flow with a novel approach for efficient design space exploration in case of multi-parametric optimization objective,' *Microelectronics Reliability*, vol. 50, Issue: 3, pp. 424-437.

R. Schneiderman (2010), 'DSPs evolving in consumer electronics applications, '*IEEE Signal Process. Mag*., vol. 27(3), pp. 6–10.

A. Sengupta and M. Rathor (2020), 'IP Core Steganography using Switch based Key-driven Hash-chaining and Encoding for Securing DSP kernels used in CE Systems', xyz.

A. Sengupta and M. Rathor (2019a), 'IP core steganography for protecting DSP kernels used in CE systems,' *IEEE Trans. Consum. Electron.*, vol. 65(4), pp. 506-515.

A. Sengupta and M. Rathor (2019b), 'Crypto-based dual-phase hardware steganography for securing IP cores,' *Lett. IEEE Comput. Soc*., vol. 2(4), pp. 32-35.

A. Sengupta and M. Rathor (2019c), 'Security of functionally obfuscated DSP core against removal attack using SHA-512 based key encryption hardware,' *IEEE Access*, vol. 7, pp. 4598-4610.

A. Sengupta, E. R. Kumar and N. P. Chandra (2019c), 'Embedding digital signature using encrypted-hashing for protection of DSP cores in CE,' *IEEE Trans. Consum. Electron*., vol. (3), pp. 398-407.

A. Sengupta (2020), 'Frontiers in Securing IP Cores - Forensic detective control and obfuscation techniques', *The Institute of Engineering and Technology (IET)*, ISBN-10: 1-83953-031-6, ISBN-13: 978-1-83953-031-9.