Chapter 04

Integrating Multi-key based Structural Obfuscation and Low-level Watermarking for Double Line of Defence of DSP Hardware Accelerators

Anirban Sengupta
Computer Science and Engineering
Indian Institute of Technology Indore

The chapter describes a double line of defense mechanism for securing hardware accelerators using key-based structural obfuscation and physical level watermarking. The presented approach discussed in this chapter is capable of securing against combined threat models of reverse engineering (leading to Trojan insertion) and IP piracy as preventive and detective control.

The chapter is organized as follows: Section 4.1 discusses about the background of the chapter; Section 4.2 presents the salient features of the chapter. Section 4.3 shows some practical applications applicable for this approach; Section 4.4 explains some contemporary approaches of this domain; Section 4.5 explains the details of the double line of defense process; Section 4.6 highlights the low-cost optimized multi-key based structural obfuscation process; Section 4.7 presents the KSO-PW tool of the presented approach; Section 4.8 discusses the case studies on DSP applications and Section 4.9 concludes the chapter.

4.1. Introduction

In this era of consumer electronics, digital signal processing (DSP) hardware accelerators have begun to dominate because of its vital role in image processing, audio processing, video processing and so forth applications. Today, nobody wants to compromise with the rate of video streaming and quality of videos. Moreover, high quality audio such 8-dimensional (D) audio are fascinating the consumers today. Various kinds of image filters have set their role in applications such robotics vision, biometric fingerprinting and medical imagery etc. Therefore the proliferating demand of high definition (HD) video, high quality audio and various kind of image filtering are the key reasons for blooming of DSP hardware accelerators in modern consumer electronics era. Apart from consumer’s applications, the utility of DSP hardware accelerators is well pronounced in several critical applications such as military, banking and healthcare etc. (Schneiderman, 2010; Sengupta, 2020).

So far we have discussed only application side of the DSP hardware accelerators. However, another side of DSP hardware accelerators is its design-for-security (DFS) that is also being given strong attention by industry and researchers today. The DFS perspective of a DSP hardware accelerator is vital for its usage in both critical and non-critical applications. This is because ensuring a secured design of DSP hardware accelerators builds up trust in hardware. This chapter focuses on design for security (DFS) perspective of DSP hardware accelerators. Now the question arises as to why the DFS is prevailing in modern system-on-chip (SoC) design technology. The key reason is the distribution of deign chain across the globe. In other words, various offshore entities (fabless design houses, SoC integrators and foundries) participate in the journey of an electronics system from its idea to physical existence (Sengupta, 2016; Sengupta, 2017; Plaza and Markov, 2015; Castillo *et al.*, 2007). During this journey, a DSP hardware accelerator design can be infected with malicious logic insertion by any untrustworthy design house involved in the design process (Zhang and Tehranipoor, 2011). For example, (i) a dishonest third party intellectual property (3PIP) vendor may covertly insert a Trojan horse at a safer place in the design and send Trojan infected IPs to the SoC integrator (ii) a dishonest SoC integrator may covertly insert a Trojan horse before sending the design to the fabrication unit or foundry (iii) an adversary in fabrication unit may insert the Trojan in the mask or by altering the dopant level. Thereby, a DSP hardware accelerator design can be compromised by an attacker using Trojan horse attack. This may lead to failure of hardware accelerators deployed in critical systems. Apart from Trojan threat, other threats such as counterfeiting and cloning are also becoming a challenge for trustworthy hardware designs (Sengupta *et al.*, 2019; Sengupta and Rathor, 2019a). This is because the economic temptation and intents of sabotaging the genuine vendor's reputation and revenue may push an adversary (untrusted entity in the design chain) towards counterfeiting and cloning of hardware designs (Sengupta and Roy, 2017; Sengupta and Mohanty, 2019).

The above discussion is the underlying reason of the ramification of the very large scale integration (VLSI) design process in the form of design-for-security (DFS). The DFS can be performed at various phases of design process viz. high (behavioural) level, register transfer level (RTL), gate level and physical/layout level. This chapter highlights on how a high level and the low level (physical level) can simultaneous be exploited to employ security against Trojan insertion, counterfeiting and cloning threats (Sengupta and Rathor, 2020). The design-for-security (DFS) at both high level and low level strengthens the trust in hardware designs. (Sengupta and Rathor, 2020) performed the high level DFS by employing multi-key based structural obfuscation during high level synthesis. Further, low level DFS has been performed by embedding watermarking at physical level. Thereby, (Sengupta and Rathor, 2020) integrated structural obfuscation at high level and watermarking at low level to provide double line of defence for securing DSP hardware accelerators against Trojan insertion, counterfeiting and cloning threats. The structural obfuscation (Sengupta *et al.*, 2017) based security during the high level design process ensures preventive control against Trojan insertion, counterfeiting and cloning threats. This is because to insert a Trojan horse or to counterfeit and clone the designs, an adversary performs reverse engineering (RE) to deduce the original structure and functionality of the design. On successful RE, an adversary becomes competent to insert malicious Trojan horse or counterfeit/clone the design. However, structural obfuscation alters the design structure in such a way that the RE becomes highly obscure for the attacker. Thus the structural obfuscation technique obstructs the reverse engineering performed by an adversary and provides security against Trojan insertion, counterfeiting and cloning threats. Further, the watermarking based security during the low level (physical level) design process ensures detective control against counterfeiting and cloning threats. This is because a robust watermark embedded into the designs enables detection of counterfeiting and cloning.

4.2. Salient Features of the Chapter

The chapter discusses the security of DSP hardware accelerators based on following key-points (Sengupta and Rathor, 2020):

- Discussion on design-for-security technique to generate highly secured DSP hardware accelerators using double line of defence against Trojan insertion, counterfeiting and cloning threats.
- Discussion on 1st line of defence using multi-key driven robust structural obfuscation, as preventive control against aforementioned hardware threats.
- Discussion on obfuscation process using following high level structural transformations which are executed sequentially: (i) key-driven loop unrolling (ii) key-driven partitioning (iii) key-driven redundant operation elimination (ROE) (iv) key-driven tree height transformation (THT) (v) key-driven folding knob based transformation.
- Discussion on 2nd line of defence using physical-level watermarking during early floorplanning of obfuscated DSP design. The watermark depends on vendor’s signature comprising of multiple variables, where each variable carries a robust mapping for conversion into respective watermarking constraints. The watermark insertion is overhead free regardless of the size of DSP design.

4.3. Some Practical Applications of DSP Hardware Accelerators for Modern Electronic Systems

In modern electronic systems such as television, digital camera, tablets, headsets, cell phones, laptops etc., DSP hardware accelerators have numerous practical applications. These applications of DSP hardware accelerators include filtering of digital data, attenuation, compression and decompression, audio and video encoding/decoding, speech recognition and so forth (Sengupta and Rathor, 2020). To facilitate these applications, DSP algorithms such as finite impulse response (FIR) filter, infinite impulse response (IIR) filter, discrete Fourier transform (DFT), fast Fourier transform (FFT), discrete wavelet transform (DWT), auto-regressive filter (ARF), discrete cosine transform (DCT), inverse discrete cosine transform (IDCT) etc. are executed as core functions. Because of data-intensive computations involved in theses DSP algorithms, it is efficient to realize them using hardware. Thereby DSP hardware accelerators are designed as dedicated processors such as application specific integrated circuits (ASICs) or reconfigurable logics in programmable devices such as field programmable gate array (FPGA) to execute data-intensive applications so that high performance can be achieved. Each kind of DSP hardware accelerator performs a specific function such as FIR filter core is used for signal attenuation, image processing, DCT core is used for transforming data from spatial to frequency domain during image compression, IDCT is used for transforming data from frequency to spatial domain during image de-compression, FFT core is used for fast transformation of data from time/spatial to frequency domain for image enhancement (e.g. biometric fingerprint image enhancement), DWT core is used for de-noising, data compression and feature extraction etc. Thus because of wide utility of DSP hardware accelerators, their demand in consumer electronics applications is proliferating.

4.4. Overview of Contemporary Approaches

This section discusses overview of contemporary approaches in two parts. The first part of the discussion includes structural obfuscation based contemporary approaches and the second part includes watermarking based contemporary approaches. The discussions on both parts are as follows (Sengupta and Rathor, 2020):

The structural obfuscation based approaches have been proposed for securing both sequential and combinational kind of circuits. For securing sequential circuits against piracy, structural transformation based obfuscation has been proposed by Li and Zhou, 2013. The authors performed following four operations to achieve best-possible obfuscation: (i) retiming (ii) re-synthesis (iii) sweep (iv) conditional stuttering. Further, (Chakraborty and Bhunia, 2009; Chakraborty and Bhunia, 2011) proposed obfuscation techniques to ensure security of designs against Trojan horse insertion. However, these approaches have not been proposed for securing larger designs such as DSP hardware accelerators. Their application is limited to small combinational and sequential circuits. Since their target hardware is not designed using high level synthesis (HLS) framework, therefore the DFS at high level is not possible. However, there are some other approaches which perform structural obfuscation for performing DFS at high level. For example, Lao and Parhi, 2015 performed structural obfuscation based DFS by applying folding transformation on the iterative data flow graph (DFG) of digital filters such as FIR. Further, Sengupta *et al.*, 2017 also performed structural obfuscation based DFS technique at high level using following transformations: loop unrolling (LU), logic transformation (LT), tree height transformation (THT), redundant operation elimination (ROE), and loop invariant code motion (LICM). This technique (Sengupta *et al.*, 2017) targets the security of DSP cores. Further, Sengupta *et al.*, 2018 targeted security of multi-media hardware accelerators such as joint photographic experts group (JPEG) compression processor for securing using THT based structural obfuscation. Further, Sengupta and Rathor, 2019b performed hologram motivated structural obfuscation by concealing one DSP architecture into another. In contrast to theses contemporary approaches, the structural obfuscation based DFS approach to be discussed in this chapter has following enhancements (Sengupta and Rathor, 2020): (i) multiple techniques of structural transformations have been performed (ii) all the applied techniques of structural transformations are driven through a designer's chosen key value, therefore resulting into higher control over the extent to which obfuscation can be applied (iii) an attacker requires to know both the correct keys and the employed multiple techniques of obfuscation to expose the true functionality of the design, hence resulting into higher security against RE attack (iv) applicability on both iterative and non-iterative algorithms of DSP (v) key-driven partitioning and key-driven folding-knob based transformation along with key-driven loop unrolling, key-driven ROE and key-driven THT based structural transformations.

To enable detection of counterfeiting and cloning, some watermarking based contemporary approaches have been proposed for DSP hardware accelerators. For example, Hong and Potkonjak, 1999 proposed watermarking technique based on binary encoding of author's signature. Further, Le Gal and Bossuet, 2012 proposed an In-synthesis watermarking technique which embeds authors signature as output marks. Furthermore, Sengupta and Bhadauria, 2016 proposed watermarking technique based on four variables author signature and Sengupta *et al.*, 2018 proposed watermarking technique based on seven variables author signature. Roy and Sengupta, 2019 proposed a watermark which is embedded at multiple levels of design abstraction such as high level and RTL. However in contrast to theses contemporary approaches, the low level watermarking to be discussed in this chapter has following

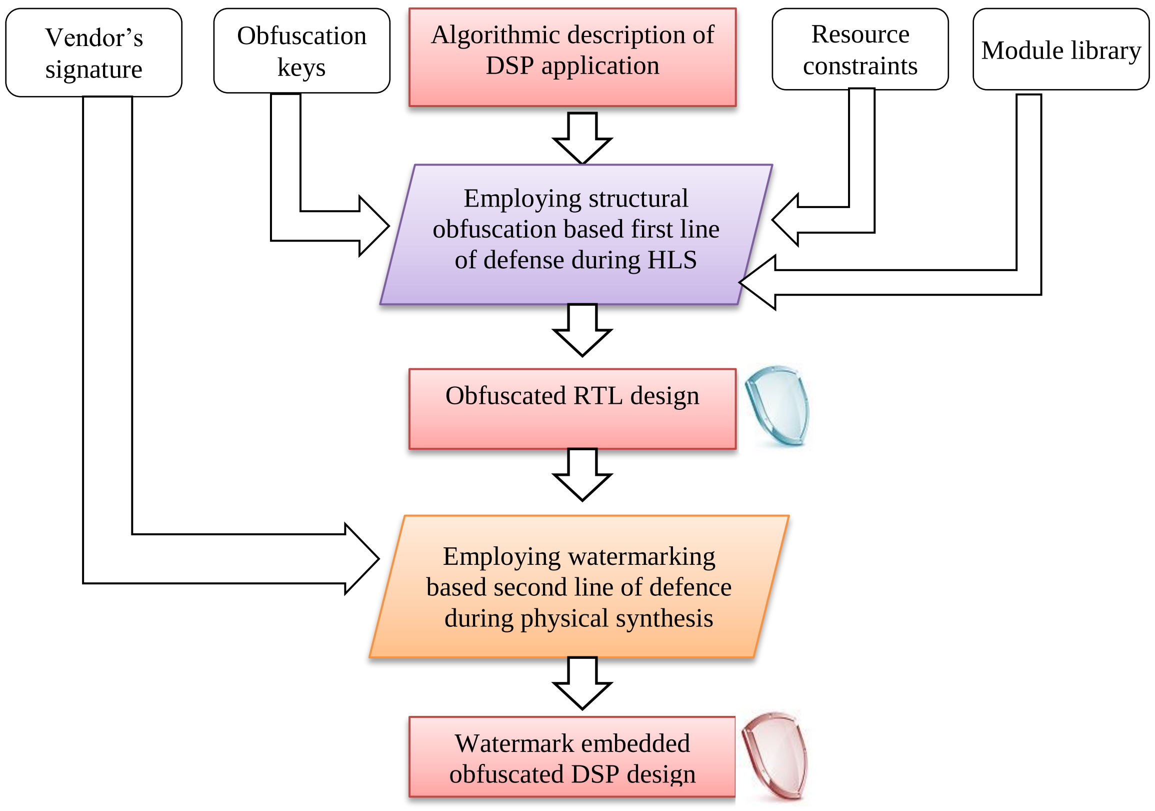

Vendor's signature
Obfuscation keys
Algorithmic description of DSP application
Resource constraints
Module library
Employing structural obfuscation based first line of defense during HLS
Obfuscated RTL design
Employing watermarking based second line of defence during physical synthesis
Watermark embedded obfuscated DSP design

Fig. 1. *Overview of multi-key based structural obfuscation and physical level watermarking based double line of defense (Sengupta and Rathor, 2020)*

differences (Sengupta and Rathor, 2020): (i) the low level watermarking proposed by Sengupta and Rathor, 2020 is embedded during floorplanning at physical level (ii) the author signature comprises of three distinct variables α, β and γ, where each variable has a robust mapping into watermarking constraints (iii) the embedding of watermark does not result into design cost overhead (iv) the physical level watermark to be discussed in this chapter has been embedded as a second line of defence, where first line of defence is employed using structural obfuscation. The physical level watermark as a second line of defence offers detective control in case the first line of defence is compromised. More explicitly, if an adversary nullifies the structural obfuscation by deducing the original functionality of an obfuscated design through RE then the physical level watermark acts as a second line of defence. Therefore if the attacker counterfeits or clones the design post compromising the structural obfuscation based security, then the physical level watermark based security helps in detecting counterfeiting and cloning.

4.5. Double Line of Defence using Structural Obfuscation and Physical Level Watermarking

Authors (Sengupta and Rathor, 2020) integrated structural obfuscation and physical level watermarking together to secure DSP hardware accelerators using double line of defence.

1. Top Down Perspective of the Approach

An abstract view of the structural obfuscation and physical level watermarking based double line of defence process is shown in Fig. 1. As shown in the figure, following inputs are required to generate a watermark implanted obfuscated DSP hardware accelerator as output (Sengupta and Rathor, 2020):

(i) Algorithmic description of a DSP application in the form of C/C++ or transfer function or mathematical relationship of inputs and output.

(ii) Resource constraints

(iii) Module library

(iv) Structural obfuscation (SO) secret keys (SO-key1, SO-key2, SO-key3, SO-key4, SO-key5)

(v) Vendor’s multi-variable signature comprising of three variables viz. α, β and γ.

As shown in the Fig. 1, structural obfuscation based first line of defence is employed during HLS process and requires algorithmic description of the DSP application, resource constraints, module library and structural obfuscation keys, as inputs. After employing the first line of defence, a structurally obfuscated RTL design is generated post-HLS. Thus obtained structurally obfuscated RTL design is fed as input along with the vendor’s signature to the second line of defence algorithm. The second line of defence algorithm is performed using watermarking based physical level synthesis. Post-watermarking, a watermark implanted obfuscated design is generated which is secured using double line of defence. The motivation of employing the double line of

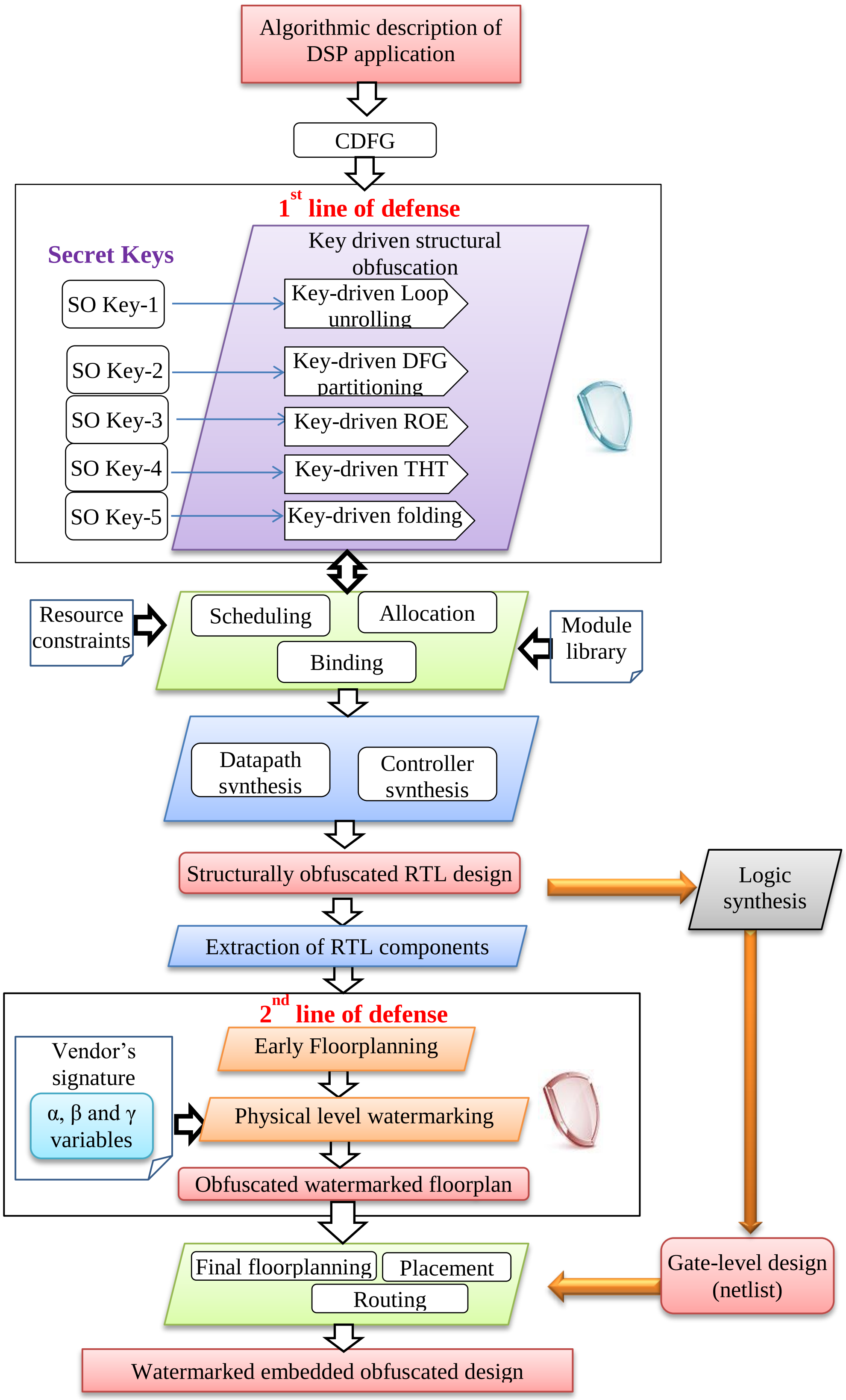


Fig. 2. *Secured design flow based on double line of defense approach (Sengupta and Rathor, 2020)*

defence is to secure DSP hardware accelerators against following threat scenarios: (i) Firstly, the Trojan insertion threat (resulting from reverse engineering) infecting third party IP cores (3PIP), which in turn compromises the security of SoC design. The first line of defence using structural obfuscation provides security against Trojan which can possibly be inserted in an untrustworthy regime such as foundry (ii) Secondly, counterfeiting/cloning threats that result into integration of fake designs or IP cores into SoC, hence compromising the security and reliability of an electronics system. The first and second line of defence ensures security against counterfeiting/cloning, where multiple SO-keys based structural obfuscation provides preventive control while physical level watermarking offers detective control. The second line of defence is not directly contextual unless the first line of defence is overtaken by an adversary. However somehow if an attacker de-obfuscates the obfuscated design and finds the correct functionality, only then doors are open for him/her for realizing malicious objectives of counterfeiting/cloning. In such a threat scenario, watermarking based 2nd line of defence secures the designs by enabling detective control over counterfeiting/ cloning.

A more informative secured design flow of the structural obfuscation and watermarking based double line of defence technique is shown in Fig. 2. As shown in the figure, the algorithmic description of DSP application is first represented in the form of control data flow graph (CDFG). Further, the CDFG is subjected to multiple secret SO-keys based structural obfuscations technique which performs following high level structural transformations (i) key-driven loop unrolling (LU) (ii) key-driven partitioning of CDFG (iii) key-driven ROE (iv) key-driven THT (v) key-driven folding knob based transformation. Post employing these multiple key-driven techniques, the CDFG is transformed in an obfuscated form. This transformed CDFG is subjected to scheduling, allocation and binding phases of HLS to generate an obfuscated design in the form of scheduled and allocated CDFG. Further, datapath and controller are synthesized to generate an obfuscated RTL circuit as shown in Fig. 2. Thus multiple secret SO-keys based structural obfuscation based first line of defence is employed. Now let's discuss how structural obfuscation prevents RE which may result into Trojan insertion, counterfeiting/cloning.

When a SoC or a standalone IC design is sent to an offshore foundry for fabrication, the design to be fabricated can be infected with Trojan (malicious logic) or it can be counterfeited/cloned. In order to insert a Trojan, or counterfeit /clone a design, an adversary first tries to interpret the true functionality and structure of design. To do so, the adversary performs RE. Once s/he successfully interprets the true functionality through RE, s/he can easily insert Trojan at safer places inside the design. The Trojan is inserted such that they remain dormant until the payload is activated by the trigger logic. The triggering is designed to occur only at rare events so that the Trojan logic cannot be detected typically during pre and post silicon simulation/validation. Therefore in order to evade detection of Trojan during validation, they are inserted at safe places in the design by an adversary. The insertion of Trojan at safe places in the design is only possible when an adversary successfully interprets the original functionality/structure of the design. Further once the functionality of design is known to the adversary, counterfeiting or cloning can be also executed. The structural obfuscation falls under the preventive control based DFS technique against Trojan insertion and piracy. This is because structural obfuscation aims to modify the design to such an extent that RE becomes arduous for an attacker to perform. Hence, the structural obfuscation technique thwarts RE and provides preventive control against Trojan insertion and piracy. The multiple SO-keys based structural obfuscation incurs very high amount of obscurity into the generated RTL/gate-level design structure (post-HLS) in terms of following modifications, without affecting functionality:

(i) Changes in the number of functional unit resources (such as multipliers, adders and subtractors) post obfuscation.

(ii) Changes in the interconnect-hardware (such as multiplexers and demultiplexers) in terms of their size and total count.

(iii) Changes in the total count of storage resources such as registers and latches.

(iv) Changes in the inter-connectivity of hardware resources.

So far, we have seen how structural obfuscation based first line of defence thwarts Trojan attack and piracy. Now let's move ahead in the double line of defence based secured design flow of hardware accelerators as shown in Fig. 2. As shown in the figure, a structurally obfuscated RTL datapath is generated post employing the first line of defence. Further in order to employ the second line of defence, firstly a set of RTL components is extracted from the structurally obfuscated RTL datapath. This set of RTL components is used to perform physical level watermarking. The watermarking is employed by performing a physical level design step referred as *early floorplanning* (proposed by Sengupta and Rathor, 2020). The early floorplanning is performed using the set of extracted RTL components. In other words, an early floorplan of RTL components is prepared. Further, this early floorplan is subjected to watermarking based on vendor's signature. The vendor's signature is a combination of three unique variables (α, β and γ), where mapping rules of each variable converts the signature into respective watermarking constraints or hardware security constraints. The watermarking constraints are implanted into the early floorplan of the design, thus resulting into an obfuscated watermarked floorplan. Further the obfuscated watermarked floorplan is subjected to final floorplanning, placement and routing phases of physical synthesis to obtain an obfuscated and watermarked layout file. The final floorplanning, placement and routing phases require gate-level netlist which is generated from logic synthesis of obfuscated RTL design. The embedded watermark in the early floorplan step (proposed by Sengupta and Rathor, 2020) of physical design flow enables detection against piracy/fake designs, hence acts as detective control as double line of defence.

2. Details of Double Line of Defence

As discussed earlier, double line of defence for DSP hardware accelerators has been deployed by integrating multiple SO-keys based structural obfuscation as a first line of defence and physical level watermarking as a second line of defence. This sub-section discusses the double line of defence mechanism in details. The flow chart of the double line of defence process is shown

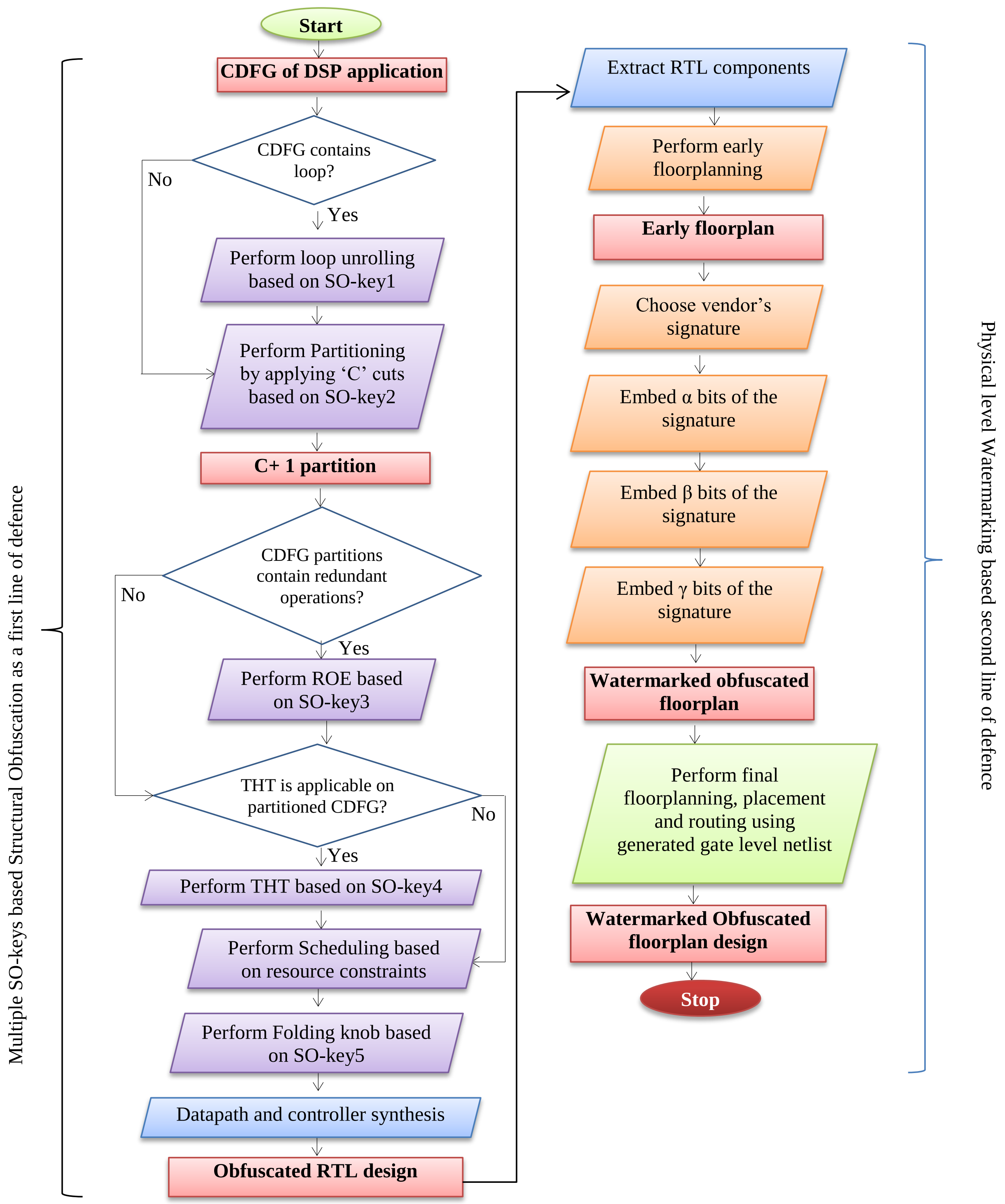
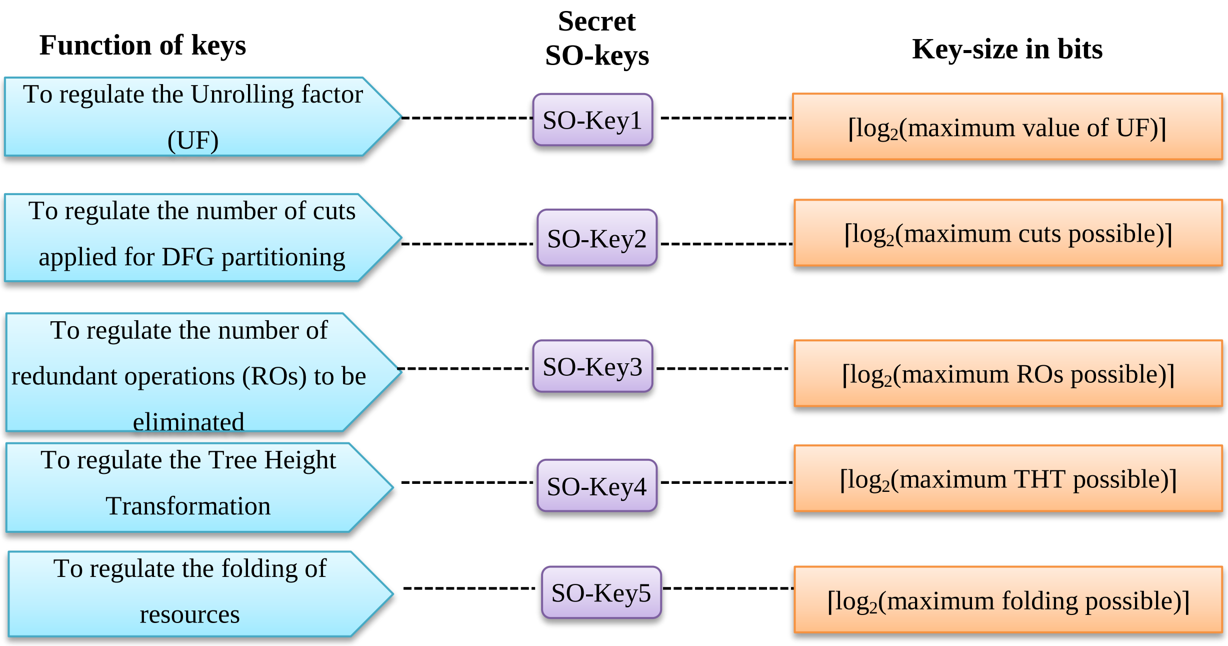

Start
CDFG of DSP application
CDFG contains loop?
No
Yes
Perform loop unrolling based on SO-key1
Perform Partitioning by applying 'C' cuts based on SO-key2
C+ 1 partition
CDFG partitions contain redundant operations?
No
Yes
Perform ROE based on SO-key3
THT is applicable on partitioned CDFG?
No
Yes
Perform THT based on SO-key4
Perform Scheduling based on resource constraints
Perform Folding knob based on SO-key5
Datapath and controller synthesis
Obfuscated RTL design
Multiple SO-keys based Structural Obfuscation as a first line of defence
Extract RTL components
Perform early floorplanning
Early floorplan
Choose vendor's signature
Embed α bits of the signature
Embed β bits of the signature
Embed γ bits of the signature
Watermarked obfuscated floorplan
Perform final floorplanning, placement and routing using generated gate level netlist
Watermarked Obfuscated floorplan design
Stop
Physical level Watermarking based second line of defence

Fig. 3. *Flow chart of the structural obfuscation and watermarking based double line of defence approach for securing DSP hardware accelerators (Sengupta and Rathor, 2020)*

in Fig. 3. As shown in the flow chart, the entire flow has been divided into two portions where the first portion depicts the flow of employing multiple SO-keys based structural obfuscation, as a first line of defence while the second portion shows the physical level watermarking, as a second line of defence. The detailed discussion of the approach with demonstration on DSP cores are given as follows (Sengupta and Rathor, 2020):

(a) Multiple SO-keys driven structural transformation based obfuscation- the first line of defence

The multiple SO-keys driven structural transformation based obfuscation requires following inputs: algorithmic description of



Function of keys
Secret SO-keys
Key-size in bits
To regulate the Unrolling factor (UF)
SO-Key1
⌈log$_2$(maximum value of UF)⌉
To regulate the number of cuts applied for DFG partitioning
SO-Key2
⌈log$_2$(maximum cuts possible)⌉
To regulate the number of redundant operations (ROs) to be eliminated
SO-Key3
⌈log$_2$(maximum ROs possible)⌉
To regulate the Tree Height Transformation
SO-Key4
⌈log$_2$(maximum THT possible)⌉
To regulate the folding of resources
SO-Key5
⌈log$_2$(maximum folding possible)⌉

Fig. 4. *Functions and Size of each SO-key used in structural obfuscation based first line of defence (Sengupta and Rathor, 2020)*

DSP application, multiple structural obfuscation secret keys (SO-key1 to SO-key5), module library and designer's specified resource constraints. As shown in the flow chart in Fig. 3, the process starts with the conversion of algorithmic description of DSP application into its CDFG representation. Further, multiple high level transformations are performed on CDFG in order to obtain a structurally transformed design leading to its equivalent structurally obfuscated design. Each high level transformation technique is driven through a designer's chosen secret SO-key which tailors the extent to which obfuscation has to be performed. Moreover, involvement of multiple secret keys in the structural obfuscation process renders the back engineering of the obfuscated design more complicated by an adversary who is assumed to be associated with an untrustworthy foundry. The function of five secret SO-keys and their sizes are shown in Fig. 4. Now let's see about each key-driven high level transformation based structural transformation/obfuscation technique one by one.

(i) Key-driven loop-unrolling based structural transformation

Loop unrolling is a high level transformation technique where loop body of an iterative DSP application is unrolled in order to incorporate parallelism in execution. The unrolling of loop body can be exploited to result into a structural transformation based obfuscation design. This is because upon loop unrolling, at circuit implementation level (RTL/gate level), the functional unit (FU) resource count, the inter-connect hardware resource (such as multiplexers and demultiplexers) count, storage elements (latches and registers) resource count change drastically. This leads to huge variations in the structure without changing the functionality. Hence, this transformation impedes the deduction of true structure and functionality of the design through RE by an attacker. The extent to which the loop unrolling has to be performed is regulated by a loop unrolling factor (UF).

The loop-unrolling based structural transformation employed by (Sengupta and Rathor, 2020) is driven by secret SO-key1

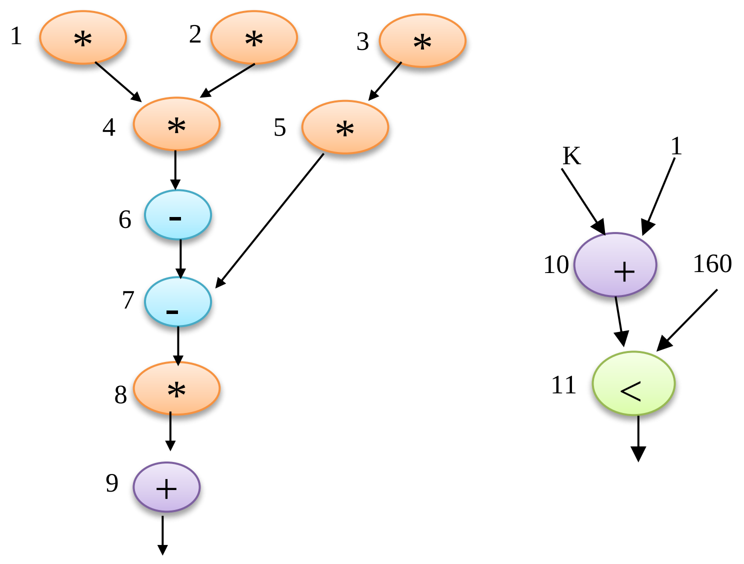


Fig. 5. *DFG of differential equation solver*

which acts as selected unrolling factor. Therefore, the SO-key1 size depends on the maximum value of unrolling factor. The maximum value of UF is chosen same as the maximum number of iterations (K) in a loop of DSP algorithm. For example, the maximum number of iterations in a 160-tap FIR filter is K=160. The SO-key1 size in bits is shown in Fig. 4.

A demonstration of loop-unrolling based transformation is shown using a loop based DSP application named ‘differential equation solver’. The DFG of differential equation solver is shown in Fig. 5, where total number of iteration is assumed to be 160. The loop unrolling based transformation is performed based on designer’s chosen secret value of SO-key1. Since K=160, therefore maximum UF value is also equal to 160. Hence the SO-key1 size is calculated to be $\lceil \log_2(160) \rceil = 8$ bits. For a designer’s chosen secret value of SO-key1= “00000100”, the UF value to be used for loop unrolling is 4 (which is a decimal equivalent of “00000100”). For UF value equal to 4, the loop unrolled graph of differential equation solver application is shown in Fig. 6. As shown in the figure, the loop body of differential equation solver algorithm has been unrolled four times. This increases the total number of operation in the application nearly four times. This leads to more

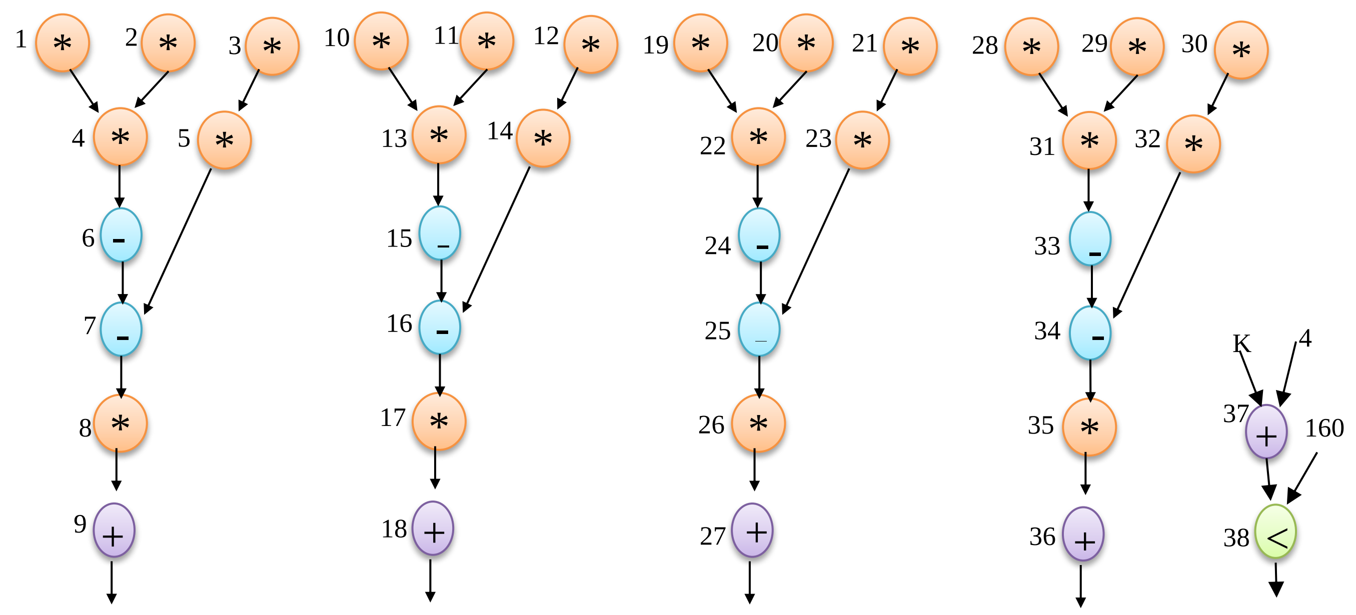


Fig. 6. *Loop unrolled DFG of differential equation after unrolling with UF=4*

utilization of FU resources, multiplexers, demultiplexers and registers. Hence, a drastic variation in RTL structure is incurred which results into structural obfuscation. This is how the first technique of structural transformation based obfuscation is performed.

(ii) Key-driven CDFG partitioning based structural transformation

CDFG partitioning is a high level transformation technique which can be exploited in such a way that an unrolled CDFG (in case of iterative DSP application) is partitioned in order to yield structural transformation based obfuscation. The impact of CDFG partitioning for structural obfuscation manifests in the RTL datapath in the form of modified interconnectivity of FU resources and modified size and number of multiplexers and demultiplexers. However, these modifications in the design do not impact any change in the functionality. Each partition of CDFG is further subjected to remaining structural transformation techniques in order to enhance obscurity.

The CDFG partitioning based structural transformation employed by (Sengupta and Rathor, 2020) is driven through designer's chosen secret SO-key2. The value of the chosen key determines the number of cuts to be performed on the CDFG in order to partition the graph; upon performing 'C' number of cuts, the target CDFG is partitioned into 'C+1' number of partitions. The way of applying cuts is again designer's selected. One of the ways that was used by (Sengupta and Rathor, 2020), is that the applied cuts should lead to such partitions where smallest one should comprise of minimum two nodes/ operations bearing a data dependency (even if partitions are of unequal size). Thus, the size of secret SO-key2 depends on the maximum number of cuts that can be performed on CDFG. The function and size of SO-key2 is highlighted in Fig. 4.

A demonstration of CDFG partitioning based transformation is shown using a DSP application named '8-point discrete

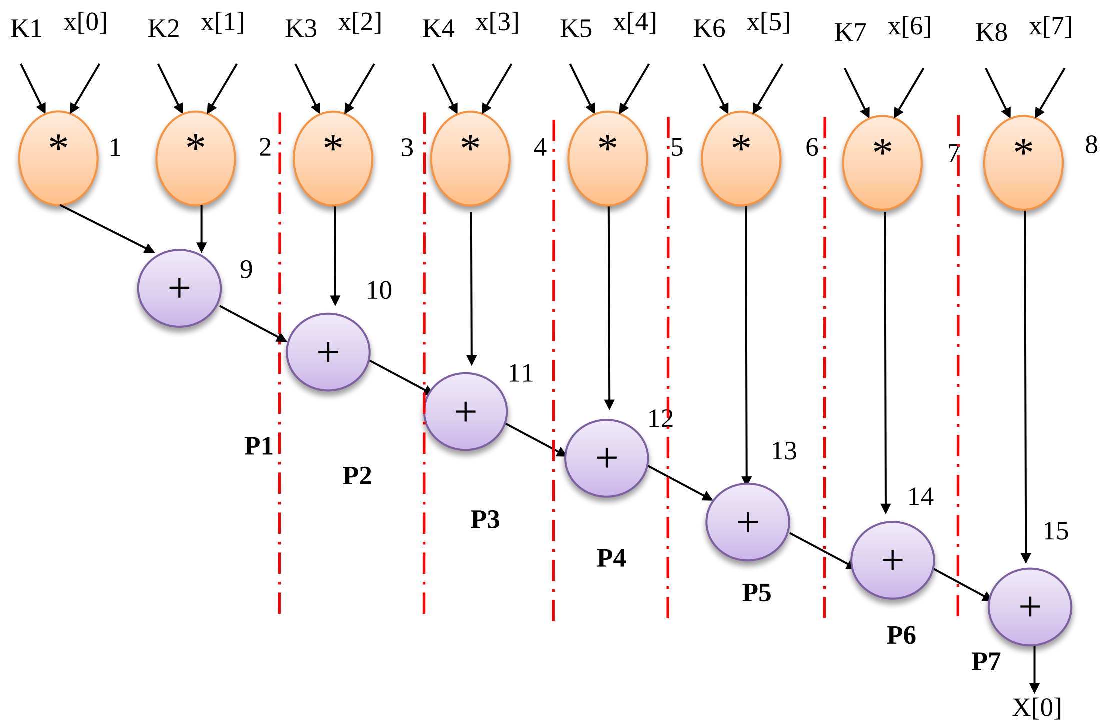


Fig. 7. *DFG of 8-point DCT core showing maximum 6 cuts can be possible with minimum two nodes in each partition. Due to 6 cuts, 7 partitions (P1- P7) could be yielded.*

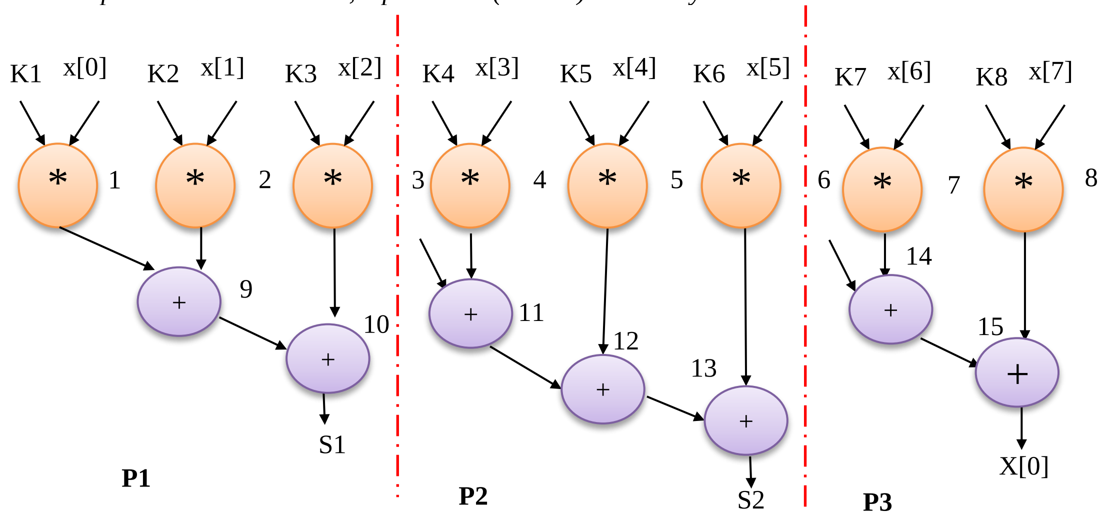


Fig. 8. *Partitioned DFG of 8-point DCT core*

Note: Red dotted lines show the cuts applied in partitioning

cosine transform (DCT)'. The generic equation of 8-point DCT is as follows:

$$X[0]=K1*x[0]+ K2*x[1]+ K3*x[2]+ K4*x[3]+ K5*x[4]+K6*x[5]+ K7*x[6]+ K8*x[7] \qquad (1)$$

Where, K1-K8 denote DCT coefficients, x[0]-x[7] indicate input values and X[0] indicates the 1st output sample of 8-point DCT. The corresponding DFG representation of 8-point DCT core is shown in Fig. 7. The DFG partitioning based transformation is performed based on designer's chosen secret value of SO-key2. Maximum six cuts can be applied to generate smallest possible partitions complying with the condition that there should be minimum two operations with data dependency in each partition. Therefore the size of SO-key2 is calculated to be $\lceil \log_2(6) \rceil = 3$ bits. For a designer's chosen value of SO-key2= "010", the number of cuts to be applied are 2 (which is a decimal equivalent of "010"). The DFG of 8-point DCT on applying C=2 cuts is shown in Fig. 7. Post applying two cuts, resultant three partitions P1, P2 and P3 are

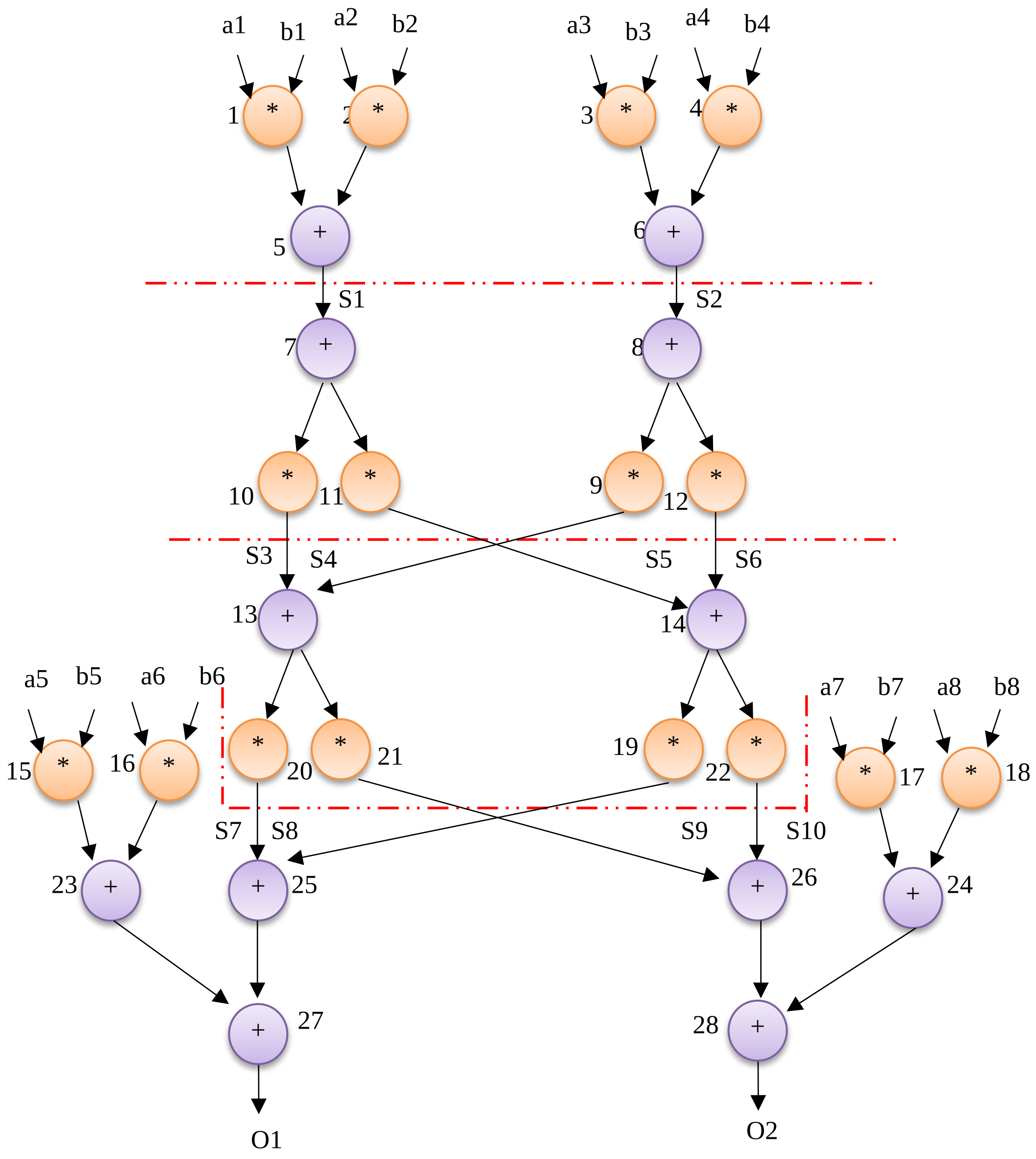


Fig. 9. *DFG of ARF application*

shown in Fig. 8. As shown in the figure, partitioning alters the interconnectivity of resources which leads to change in the size and number of interconnect hardware resources post behavioural synthesis, without affecting functionality.

(iii) Key-driven ROE based structural transformation

Redundant operation elimination (ROE) is a high level transformation technique that can be exploited to achieve structural obfuscation. Here such nodes or operations which generate same output because of having same inputs and same operation type are eliminated from the CDFG, in order to achieve structural transformation which in turn yields obfuscated design. To do so, firstly those operations are identified as redundant operations which have their inputs and operation type equivalent with another available node in the CDFG. Next the identified redundant nodes are removed without affecting the design function. The ROE based high level transformation technique contributes in structural obfuscation by incurring the following modifications in the RTL datapath while preserving functionality: modification in the interconnectivity of FU resources and size of multiplexers and demultiplexers.

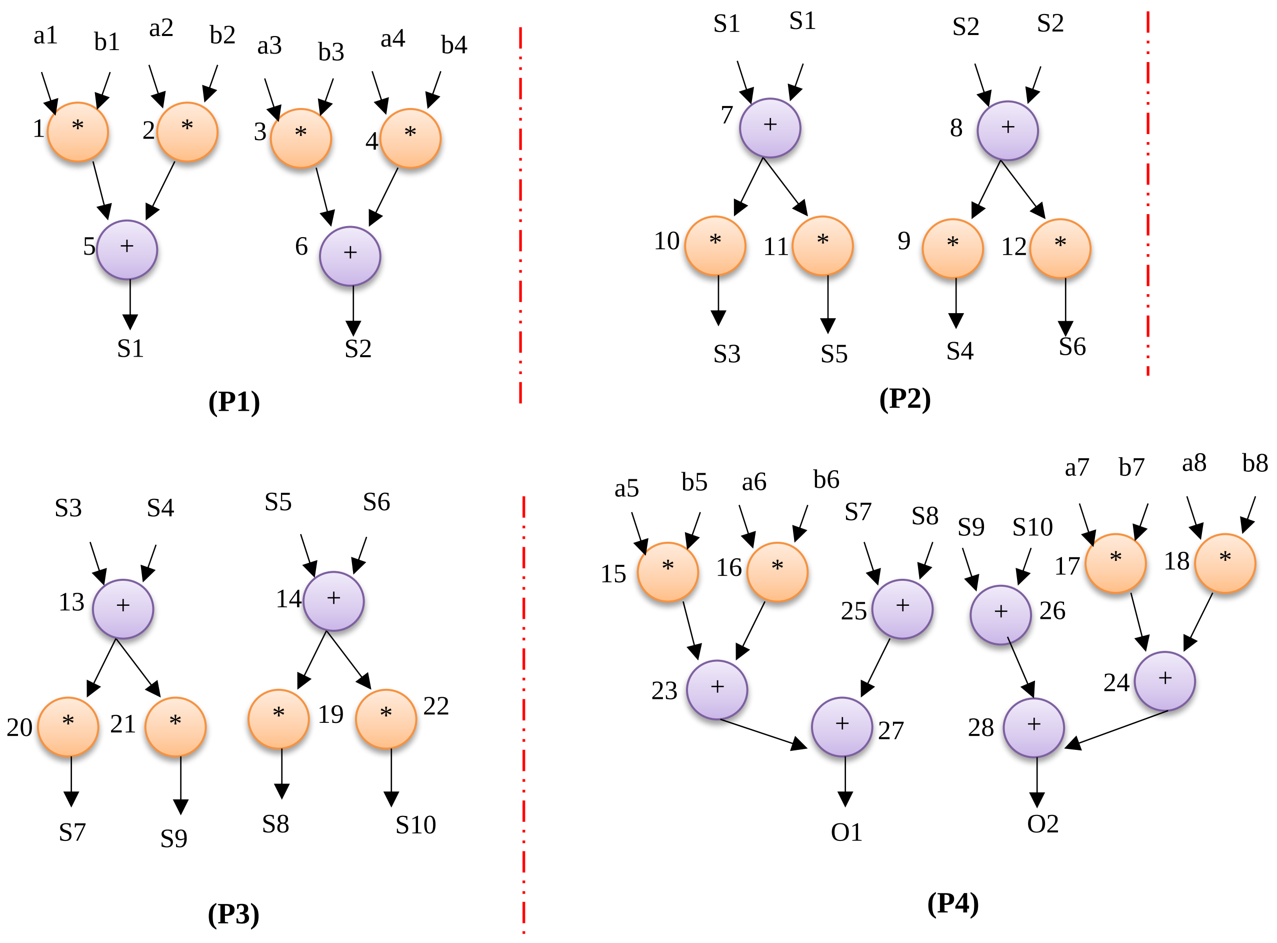


Fig. 10. *Partitioned DFG of ARF application*

The ROE based structural transformation employed by (Sengupta and Rathor, 2020) is driven through designer's chosen secret SO-key3. The value of the chosen key determines the number of redundant nodes that have to be removed from the partitioned CDFG. The size of SO-key3 depends on the maximum number of redundant nodes that have been identified across all the CDFG partitions. The function and size of SO-key3 is highlighted in Fig. 4.

The ROE based transformation cannot be applied on 8-point DCT core as redundant nodes are not present in the corresponding DFG partitions. Therefore, a demonstration of ROE based transformation is shown using auto-regressive filter (ARF) application. The DFG representation of ARF is shown in Fig. 9. The applied three cuts to generate its four partitions are also shown in Fig. 9. The resultant partitions P1, P2, P3 and P4 are shown in Fig. 10. The ROE based structural transformation is performed on the partitioned CDFG based on designer's chosen value of SO-key3. As shown in the partitioned DFG of ARF, there are total maximum four redundant nodes across the partitions. For a designer's chosen value

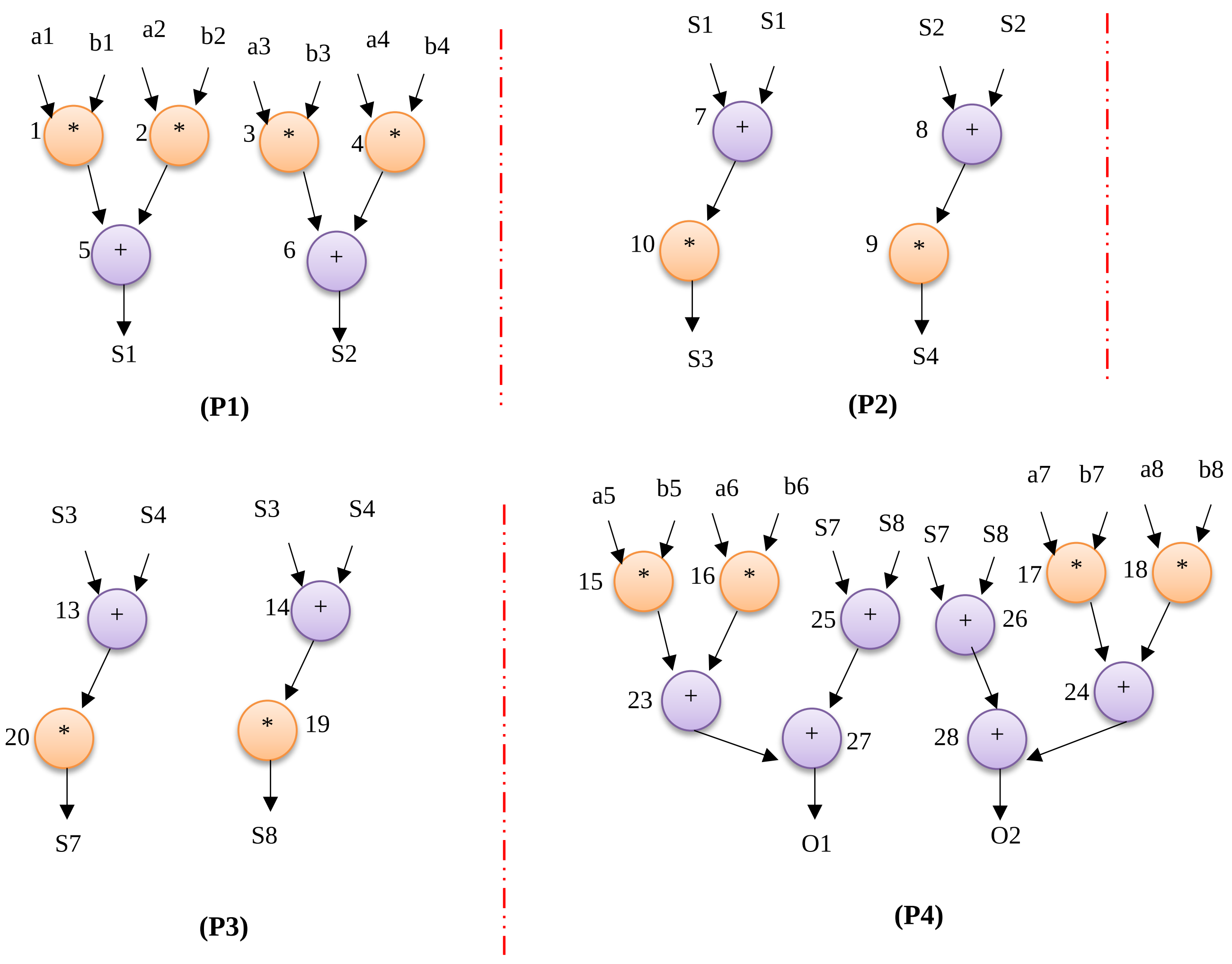


Fig. 11. *Partitioned DFG after ROE based transformation*

of SO-key3= “100”, the number of operations to be eliminated are 4 (which is a decimal equivalent of “100”). As shown in Fig. 10, operation number 11 and 12 are redundant in partition P2 as their inputs and operation types are same as operation number 10 and 9 respectively. Similarly, operation number 21 and 22 are redundant in partition P3 as their inputs and operation types are same as operation number 20 and 19 respectively. These redundant operations (number 11 and 12 in partition P2 and number 21 and 22 in partition P3) are eliminated. Post applying ROE based high level transformation; the transformed DFG of ARF is shown in Fig. 11. As shown in the transformed DFG post ROE, inputs of operation 14 changes from (S5, S6) to (S3, S4) and inputs of operation 26 changes from (S9, S10) to (S7, S8). Thus elimination of operations and changes in the interconnectivity of operations incur alterations in the RTL structure in terms of change in FU resource interconnectivity and size of interconnect hardware resources such as multiplexers and demultiplexers.

(iv) Key-driven THT based structural transformation

Tree height transformation (THT) is a high level transformation technique where height of a CDFG is subjected to alteration by breaking the sequential data dependency in the graph and forcing parallel dependency of some operations without affecting the design functionality. The alteration in the height of graph by modifying the data dependency of nodes incurs structural transformation based obfuscation in the RTL structure post HLS. The THT based high level transformation technique contributes in structural obfuscation by incurring the following modifications in the RTL datapath: modification in the interconnectivity of FU resources and size of multiplexers and demultiplexers.

The THT based structural transformation employed by (Sengupta and Rathor, 2020) is driven through designer’s chosen secret SO-key4. The value of the chosen secret key determines the number of partitions that have to be subjected to the THT. The size of SO-key4 depends on the maximum number of partitions that can be subjected to the THT. The function and size of SO-key4 is highlighted in Fig. 4.

A demonstration of THT based transformation is shown using 8-point DCT application. Earlier, it has been demonstrated that how partitions of DFG of 8-point DCT are generated (shown in Fig. 8). Since ROE is not possible in the partitioned DFG of 8-point DCT, therefore it is directly subjected to THT based transformation. The THT based structural transformation is performed on the partitioned CDFG based on designer’s chosen value of SO-key4. As shown in the partitioned DFG of DCT, there is maximum one partition (i.e. P2) which can be subjected to the THT. Therefore the size of SO-key4 is only 1 bit. For a designer’s chosen value of SO-key4= ‘1’, the partition P2 is subjected to the THT based structural transformation. The THT transformed partitioned DFG of DCT is shown in Fig. 12. As shown in the figure, the sequential data dependency of operation 12 in partition P2 has been converted to the parallel dependency with respect to the operation number 11. This

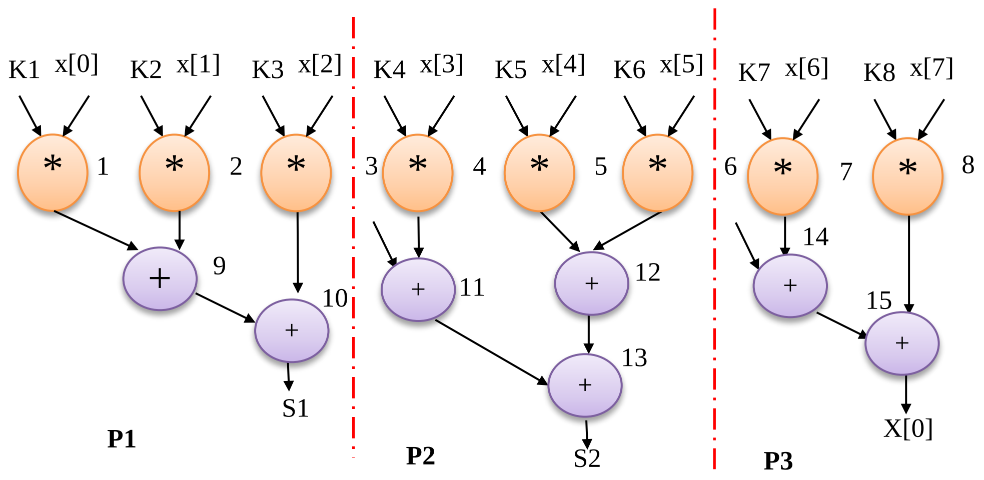


Fig. 12. *THT based obfuscation technique on partitioned DFG of 8-point DCT*

leads to alteration in the height of sub-graph in partition P2. Thus THT based structural transformation is performed to further enhance structural obfuscation in the controller design.

(v) Key-driven folding knob based structural transformation

Post performing loop unrolling, partitioning, ROE and THT based structural transformations; all operations of the obtained obfuscated partitioned CDFG are scheduled in control steps. The folding knob based structural transformation is performed post-scheduling of the obfuscated partitioned DFG to further enhance the obscurity in the design structure. Folding is a high-level transformation technique which can be exploited to achieve structural obfuscation; such that a common resource is shared among a set of operations that have the same operation type but are present in different control steps. **The number of operations in a set sharing the resource is called the 'folding factor'**. Thus the folding of FU resource units depends on the secret value of chosen 'folding factor'. Further, the number of folding transformation is controlled by the designer chosen secret value of SO-key5. **The value of the key5 denotes the number of folding transformations to be applied which is referred as the 'folding knob'**. The maximum size of SO-key5 depends on the maximum number of occurrences of folding transformation (i.e. maximum value of folding knob) across the scheduled DFG. The folding knob based transformation affects the number of FU resources and delay elements (also known as registers) post HLS, thereby structurally obfuscating the design without causing any change to the functionality.

The folding knob transformation has been demonstrated using 8-point DCT core. (Sengupta and Rathor, 2020) considered folding factor =2 for applying folding-knob based transformation. So far, THT obfuscated partitions of DFG of 8-point DCT have been explained in previous step. In order to perform folding knob transformation, the THT obfuscated partitions of DFG are scheduled as shown in Fig. 13. It is evident from the figure that there are maximum 5 occurrences of folding transformation based on folding factor =2. First folding transformation can be performed in scheduled partition P1 using

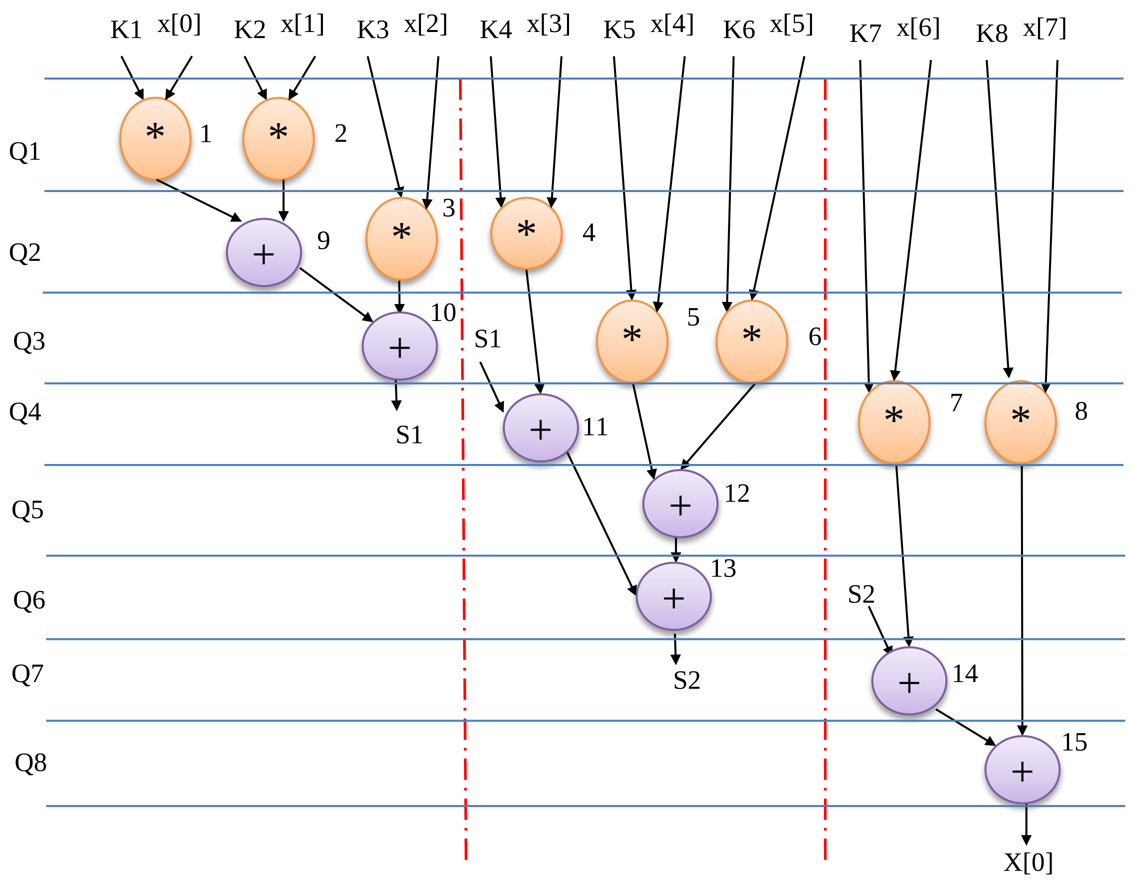


Fig. 13. *Scheduling of obfuscated partitioned DFG of DCT application using 2Multipliers and 1 Adder*

either (operation 1 and 2) or (operation 1 and 3) because of having same operation type and different control steps. Second folding transformation can be performed in scheduled partition P1 using (operation 9 and 10). Third folding transformation can be performed in scheduled partition P2 using either (operation 4 and 5) or (operation 4 and 6). Fourth folding transformation can be performed in scheduled partition P2 using any of following set of operations: (operation 11 and 12), or (operation 11 and 13) or (operation 12 and 13). Fifth folding transformation can be performed in scheduled partition P3 using (operation 14 and 15). For designer selected value of So-key5= "011", the value of folding knob is 3 i.e. folding transformation is to be applied at three occurrences in the scheduled graph. First folding transformation is applied using operation 1 and 3 in partition P1, hence they are essentially executed through same FU resource instance (i.e. M1) as shown in Fig. 14. The second folding transformation is applied using operation 4 and 6 in partition P2, hence they are essentially executed through same FU resource instance (i.e. M2) as shown in figure. The third folding transformation is applied using operation 11 and 12 in partition P2, hence they are essentially executed through same FU resource instance (i.e. A1) as shown in figure. The operations that are subjected to folding are encircled in loops as shown in Fig. 14. The allocation of other operations to the FU resources does not comply with the folding transformation. A possible allocation of remaining operations is shown in the Fig. 14. As shown in the figure, the obfuscated partitioned DFG of DCT application uses two multipliers and one adder for FU resource allocation.

Once multiple key driven techniques of structural obfuscation are performed, the individual obfuscated scheduled partitions of the graph are synthesized into individual obfuscated RTL datapath. The individual obfuscated RTL datapath of three scheduled DFG partitions (refer Fig. 14) of 8-point DCT are shown in Fig. 15. Further, these individual obfuscated RTL datapaths are integrated into a single obfuscated RTL datapath. Fig. 16 shows the integration of individual obfuscated datapath of scheduled DFG partitions into a single obfuscated RTL datapath. This obtained structurally obfuscated RTL

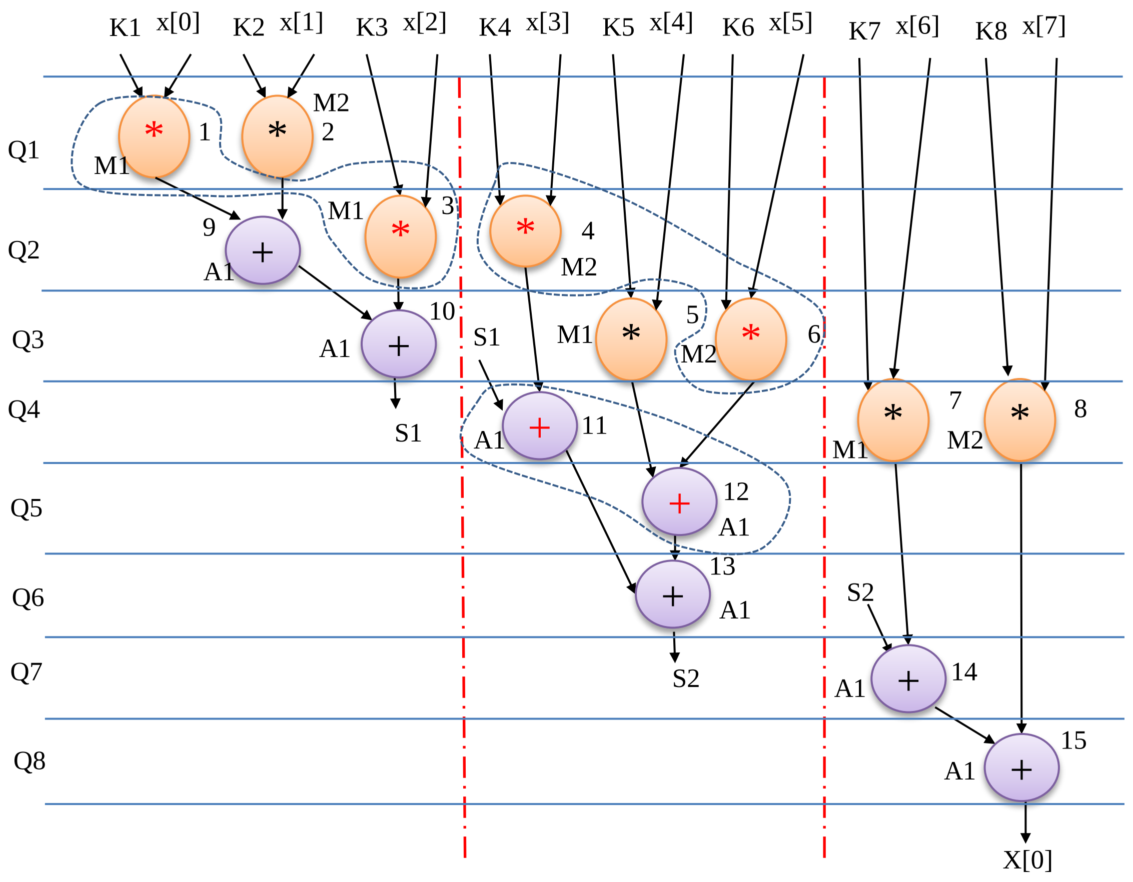


Fig. 14. *Folding knob based transformation on obfuscated partitioned DFG of 8-point DCT*

architecture is harder to reverse engineer by an adversary, hence it becomes secured against Trojan insertion and piracy threats.

(b) Multiple variables signature based physical level watermarking - the second line of defence

The watermarking based second line of defence is performed on the top of structurally obfuscated design. In this approach of second line of defence (Sengupta and Rathor, 2020), watermarking has been performed at physical level by embedding three variables vendor’s signature. To do so, physical level design step in the form of early floorplanning has been introduced by

(Sengupta and Rathor, 2020) in the physical design flow. In the early floorplanning step of physical design flow, an early

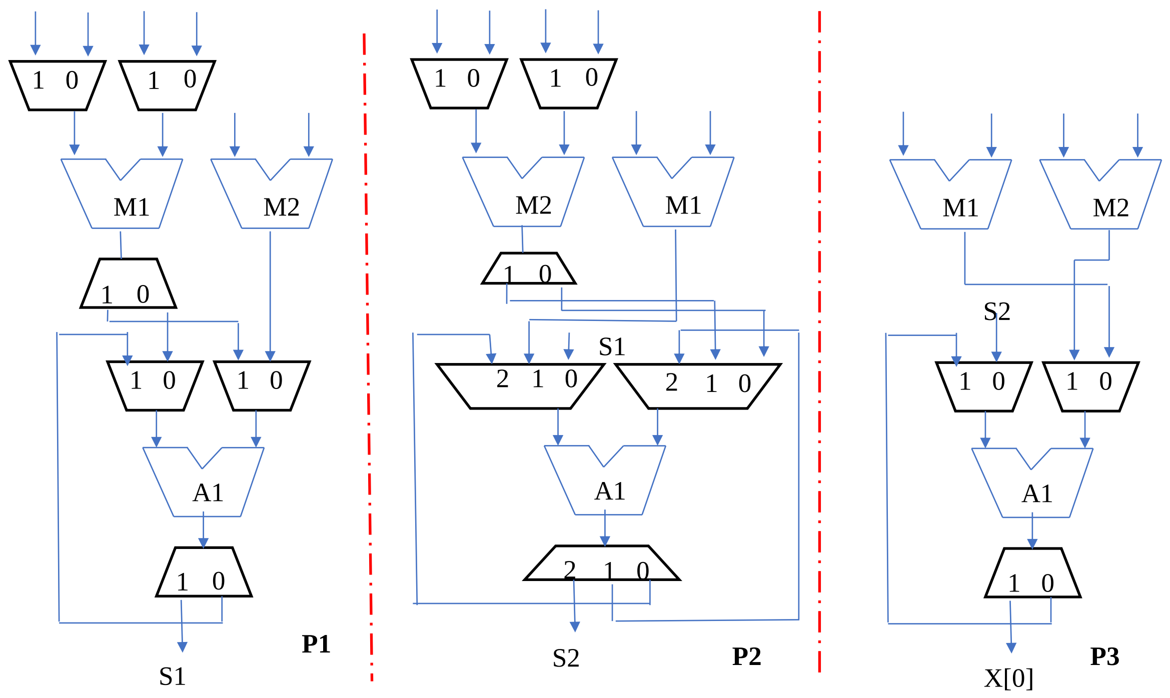


Fig. 15. *Obfuscated RTL datapath of individual scheduled DFG partitions of 8-point DCT*

floorplan of RTL components is prepared which is subjected to watermarking constraints. The overall process of accomplishing physical level watermark is illustrated using following steps:

(i) Formation of a set of RTL components

Firstly, RTL components are extracted from the structurally obfuscated datapath. A set 'R' of RTL components such as FUs (adders, multipliers, subtractors and comparator etc.), interconnect hardware resources (multiplexers and demultiplexers of various sizes) is formed. **The different type of RTL components in the set 'R' are arranged in the decreasing order of their size; and, different instances of the same RTL component are arranged in the set in increasing order of their instance number.** For example, a set of RTL components obtained from obfuscated RTL datapath of 8-point DCT (shown in Fig. 16) is given below:

R= *{M1, M2, d1, d2, d3, d4, x1, x2, x3, x4, x5, x6, x7, x8, A1, d5, d6, d7, d8, x9, 10, x11, x12, x13, x14, x15, x16}* (2)

The RTL components shown in the set 'R' have been ordered based on their size and instance number. Note: these components have been highlighted using red colour in the integrated obfuscated RTL datapath shown in Fig. 16.

Thus a set of RTL components is formed, which is used in the further steps of embedding physical level watermarking.

(ii) Preparing early floorplan

Once a set of RTL components is formed, it is utilized in preparing early floorplan. To prepare the early floorplan, the elements (i.e. RTL components) in the set 'R' are traversed from left to right. While traversing the set, each component of

RTL components of datapath of scheduled partition P1

FU resource	*Associated interconnect resources*
M1	*2:1 Mux1_M1, 2:1 Mux2_M1* *1:2 Demux1_M1*
M2	--
A1	*2:1 Mux1_A1, 2:1 Mux2_A1* *1:2 Demux1_A1*

+

RTL components of datapath of scheduled partition P2

FU resource	*Associated interconnect resources*
M1	--
M2	*2:1 Mux1_M2, 2:1 Mux2_M2* *1:2 Demux1_M2*
A1	*4:1 Mux1_A1, 4:1 Mux2_A1* *1:4 Demux1_A1*

+

RTL components of datapath of scheduled partition P3

FU resource	*Associated interconnect resources*
M1	--
M2	--
A1	*2:1 Mux3_A1, 2:1 Mux4_A1* *1:2 Demux2_A1*

RTL components of integrated datapath

FU resource	*Associated interconnect resources*
M1	*4:1 Mux1_M1 (x1) , 4:1 Mux2_M1 (x2)* *1:4 Demux1_M1 (d1)* *2:1 Mux1_M1 (x9), 2:1 Mux2_M1 (x10)* *1:2 Demux1_M1 (d5)*
M2	*4:1 Mux1_M2 (x3), 4:1 Mux2_M2 (x4)* *1:4 Demux1_M2 (d2)* *2:1 Mux1_M2 (x11), 2:1 Mux2_M2 (x12)* *1:2 Demux1_M2 (d6)*
A1	*4:1 Mux1_A1 (x5), 4:1 Mux2_A1 (x6)* *1:4 Demux1_A1 (d3)* *4:1 Mux3_A1 (x7), 4:1 Mux4_A1 (x8)* *1:4 Demux2_A1 (d4)* *2:1 Mux1_A1 (x13), 2:1 Mux2_A1 (x14)* *1:2 Demux1_A1 (d7)* *2:1 Mux3_A1 (x15), 2:1 Mux4_A1 (x16)* *1:2 Demux2_A1 (d8)*

Fig. 16. *Integration of all individual obfuscated datapaths into a single obfuscated RTL datapath of DCT*

Note*: 'x' and 'd' represent multiplexers and demultiplexers respectively. Numbering of components has been done in decreasing order of their size and increasing order of instance number* different type is selected according to their order of presence (in the set 'R') and placed in the floorplan in order to make it grow diagonally i.e. 'd' appears before 'x'. *Note: however selection within components of same size can be made randomly.* To facilitate the growth of floorplan, it is partitioned into equal sized blocks. The size of each block is given by the following:

*Block size = (largest component height in the set 'R') * (largest component width in the set 'R')* (3)

For the set of components of obfuscated RTL datapath of 8-point DCT core, the largest component height is of multiplier and the largest component width is also of multiplier. For the height=8 units and the width= 4 units of multiplier, the size of each block is 8×4.

Once first block is filled post placement of RTL components, further placement of components is performed in the second block which is placed just right to the first block in the floorplan. Further the third block is placed on the top of the first block in order to make diagonal growth of floorplan and maintain a rectangular envelop. Thus early floorplan is prepared using RTL components as shown in Fig. 17. The second line of defence in the form of physical level watermarking is performed on this generated early floorplan of the obfuscated design.

(iii) Selection of vendors signature and conversion into watermarking constraints

In order to embed vendor's watermark into the early floorplan, a secret signature comprising of three unique variables α, β and γ is selected. To enable embedding of signature into the floorplan, the mapping of signature variables into watermarking constraints is shown in Table 1. Based on these mapping rules, each digit of the signature is decoded and embedded into the

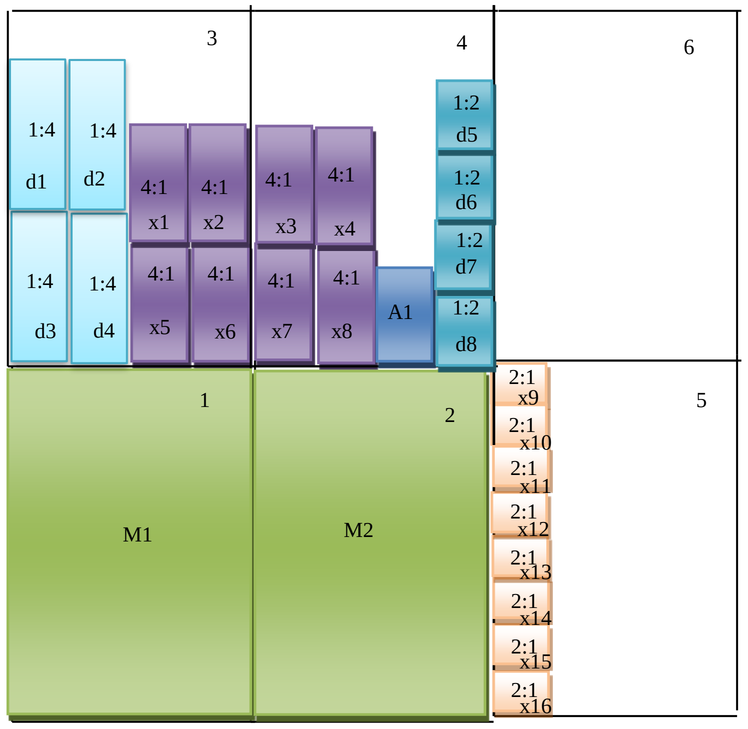


Fig. 17. *Early floorplan of obfuscated DCT design before embedding watermark*

floorplan one by one.

For embedding watermark into the floorplan of the obfuscated DCT design, let's assume the vendor's secret signature is: "α α β β γ β γ β".

(iv) Embedding of α digits into floorplan

As evident from the mapping rule of α variable given in the Table 1, the embedding of α digits of the signature is performed using FU resource components only. Therefore in order to embed α digits, firstly a subset 'R1' containing only FU components (multipliers, adders etc.) is extracted from the set 'R', where arrangement of FU resources follow the decreasing order of size and the increasing order of instance number. Further, the elements in subset 'R1' are traversed using two pointers 'p' and 'q' in order to embed each α digit. Initially, pointer 'p' indexes the 1st FU component in the subset 'R1' and pointer 'q' indexes the 2nd FU component. For each α digit, pth FU component is swapped with the qth component in the subset 'R1'. If swapping leads to placement of an odd FU component on the top of the even FU component in the floorplan, then one α digit is successfully embedded. Now, those FU components which participated in the embedding of α digit are eliminated from the subset 'R1'. However if the swapping does not place an odd FU component on the top of the even FU component (i.e. embedding of α digit is not possible) then the swapping is undone (previous placement of components in the floorplan is restored). Further, the pointer 'q' is subjected to increment by one. Again pth FU component is swapped with qth component for embedding of the α digit. For each swapping, 'q' is subjected to increment by one until it reaches to its maximum value (m) which is represented by the total number of FU components of same type in the subset 'R1'. When the maximum value is reached by pointer 'q', the pointer 'p' is incremented by one and pointer 'q' is set to 'p+1'. The pointer 'p'

Table 1. *Mapping of signature variables into watermarking constraints*

Variables in author's signature	Mapping rules to map into watermarking constraints
α	Odd FU resource needs to be placed on the top of even FU by swapping between two FU resources of same type
β	Odd multiplexer (Mux) needs to be placed on the top of even multiplexer by swapping between two multiplexers of same size
γ	Odd demultiplexers (Demux) needs to be placed to the right of even demultiplexer by swapping between two demultiplexers of same size

can go upto m-1. When the traversal of all components of a particular FU type is accomplished, pointers ‘p’ and ‘q’ are made to index the first two components of another FU type in the subset ‘R1’. This process is followed until the embedding of all α digits in the signature is accomplished.

In case of 8-point DCT design, the subset R1 extracted from set R is as follows:

$$R1=\{M1, M2, A1\} \qquad (4)$$

For embedding first α digit of the signature, the subset R1 is traversed to perform swapping between components of same size. Initially, components M1 and M2 are indexed by pointers ‘p’ and ‘q’ respectively. Therefore, components M1 and M2 are subjected to swapping. However, this swapping does not place an odd FU component on the top of even FU component (according to the embedding rule of α variable). Therefore this swapping is undone. Further swapping of components by increasing the pointer ‘q’ and/or ‘p’ in the subset ‘R1’ is not possible because of very limited number of FU components in the subset. Therefore embedding of any α digit is not possible in the floorplan of obfuscated 8-point DCT design.

(v) Embedding of β digits into floorplan

As evident from the mapping rule of β variable given in the Table 1, the embedding of β digits of the signature is performed using Mux components only. Therefore in order to embed β digits, firstly a subset R2 containing only Mux components of different sizes is extracted from the set ‘R’, where arrangement of Mux components follow the decreasing order of size and the increasing order of instance number. Likewise the embedding process of α digits, the elements in subset ‘R2’ are traversed using two pointers ‘p’ and ‘q’ in order to embed each β digit of the signature. Initially, pointer ‘p’ and ‘q’ index the 1st and 2nd Mux component of same size. For each β digit, pth Mux component is swapped with the qth Mux component of the same size in the subset ‘R2’. If swapping leads to placement of an odd Mux component on the top of the even Mux component in the floorplan, then one β digit is successfully embedded. **Now, those Mux components which participated in the embedding of β digit are eliminated from the subset ‘R2’. However if the swapping does not make embedding of β digit possible, then the swapping is undone**. During the embedding of β digits, the pointers ‘p’ and ‘q’ are incremented in the same manner as was discussed for embedding of α digits.

The embedding of β digits is demonstrated using early floorplan (shown in Fig. 17) of obfuscated DCT design. In order to do so, the subset R2 is extracted from set R which is given as follows:

$$R2=\{ x1, x2, x3, x4, x5, x6, x7, x8, x9, 10, x11, x12, x13, x14, x15, x16\} \qquad (5)$$

To embed 1st β digit of the signature, the subset R2 containing all Mux components is traversed and swapping is performed between components of same size. Initially, Mux components x1(4:1) and x2(4:1) are indexed by pointers ‘p’ and ‘q’ respectively, hence subjected to swapping for embedding 1st β digit. The **swapping of x1 and x2** will result into embedding of first β digit (as per the embedding/mapping rule of 'β'). This is because, it will place x1 (odd component) over x6 (even component). Since the embedding of 1st β digit has utilized Muxes x1 and x6, therefore they are removed from the subset R2. The modified subset R2 is as follows: R2= *{ x2, x3, x4, x5, x7, x8, x9, 10, x11, x12, x13, x14, x15, x16}*. Now for embedding next β digit, pointers 'p' and 'q' are made to index again first two components i.e. x2(4:1) and x3(4:1) respectively in the subset 'R2'. Now **swapping of x2 and x3** in the floorplan is performed which will not result into embedding of second β digit. This is because the swapping will cause the placement of x2 (even) on the top of x7 (odd) and x3 (odd) on the top of x5 (odd). Here, neither of the cases follows the embedding/mapping rule of 'β'. Therefore this swapping is undone. Now pointer 'q' is incremented by one and it starts indexing x4 in the subset 'R2'. Further, **swapping of x2 and x4** in the floorplan is performed which also will not result into embedding of second β digit. This is because the swapping will cause the placement of x2 (even) on the top of x8 (even) and x4 (even) on the top of x5 (odd). Here again, neither of the cases follows the embedding/mapping rule of 'β'. Therefore this swapping is also undone. Now pointer 'q' is again incremented by one and it indexes x5 in the subset R2. Now **swapping of x2 and x5** will result into embedding of second β digit (as per the embedding/mapping rule of 'β'). This is because, it will place x5 (odd component) over x2 (even component). Since the embedding of the β digit has utilized Muxes x2 and x5, therefore they are also removed from the subset R2. The modified subset R2 is as follows: R2= *{x3, x4, x7, x8, x9, 10, x11, x12, x13, x14, x15, x16}*. Now for embedding next β digit, pointers 'p' and 'q' are made to index again first two components i.e. x3(4:1) and x4(4:1) respectively in the subset 'R2'. **Swapping**

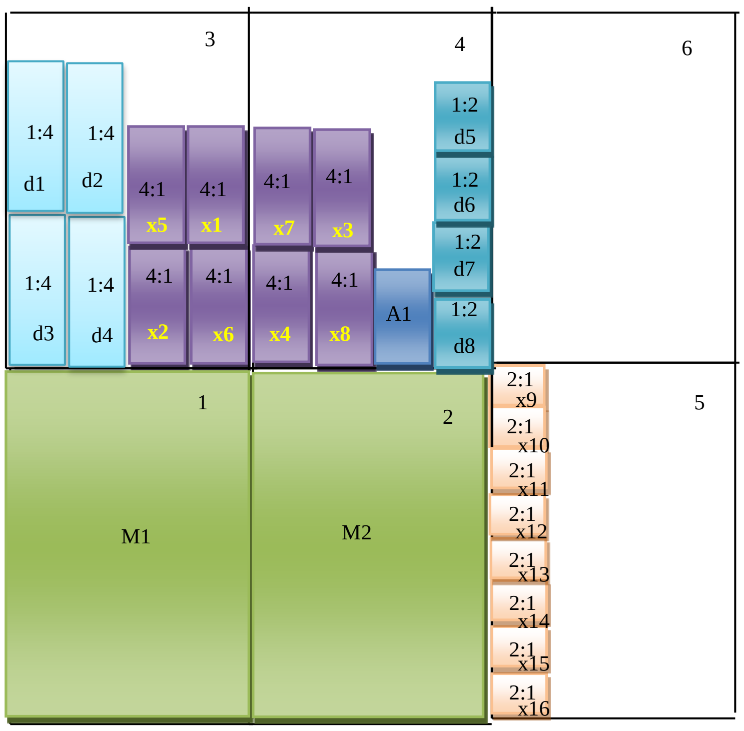


Fig. 18. *Intermediate watermarked floorplan post embedding β digits of the signature*

of x3 and x4 will result into embedding of third β digit. This is because, it will place x3 (odd component) over x8 (even component). Since the embedding of the β digit has utilized Muxes x3 and x8, therefore they are also removed from the subset ‘R2’. The modified subset ‘R2’ is as follows: R2= *{ x4, x7, x9, 10, x11, x12, x13, x14, x15, x16}.* Further, **swapping of x4 and x7** will result into embedding of 4th β digit. This is because, it will place x7 (odd component) over x4 (even component). This is how all four β digits of the signature are embedded in to the floorplan. The intermediate floorplan post embedding all β digits is shown in Fig. 18. As shown in the figure, the embedded β digits (i.e. odd Muxes on the top of even Muxes) have been highlighted using red colour.

(vi) Embedding of γ digits into floorplan

As evident from the mapping rule of γ variable given in the Table 1, the embedding of γ digits of the signature is performed using Demux components only. Therefore in order to embed γ digits, firstly a subset R3 containing only Demux components of different sizes is extracted from the set ‘R’, where arrangement of Demux components follow the decreasing order of size and the increasing order of instance number. Likewise the embedding rule of α digits, the elements in subset ‘R3’ are traversed using two pointers ‘p’ and ‘q’ in order to embed each γ digit of the signature. Initially, pointers ‘p’ and ‘q’ index the 1st and 2nd Demux component of same size. For each γ digit, pth Demux component is swapped with the qth component of same size in the subset ‘R3’. If swapping leads to placement of an odd Demux component to the right of an even Demux component in the floorplan, then one γ digit is successfully embedded. Now, those Demux components which participated in the embedding of γ digit are removed from the subset ‘R3’. However if the swapping does not make embedding of γ digit possible, then the swapping is undone. During the embedding of γ digits, the pointers ‘p’ and ‘q’ are incremented in the same manner as was discussed for embedding of α and β digits.

The embedding of γ digits in the floorplan of the obfuscated DCT design is discussed as follows. Firstly, the subset R3 is extracted from set R as given below.

R3= *{d1, d2, d3, d4, d5, d6, d7, d8}* (6)

To embed 1st γ digit of the signature, the subset ‘R3’ containing all Demux components is traversed and swapping is performed between components of same size. Initially, Demux components d1(1:4) and d2(1:4) are indexed by pointers ‘p’ and ‘q’ respectively, hence subjected to swapping for embedding 1st γ digit. The swapping of d1 and d2 will result into embedding of first γ digit (as per the embedding/mapping rule of ‘γ’). This is because, it will place d1 (odd component) right to the d2 (even component). Since the embedding of 1st γ digit has utilized Demuxes d1 and d2, therefore they are removed from the subset ‘R3’. The modified subset ‘R3’ is as follows: R3= *{d3, d4, d5, d6, d7, d8}.* Now for embedding next γ digit, pointers ‘p’ and ‘q’ are made to index again first two components i.e. d3(1:4) and x4(1:4) respectively in the subset ‘R3’.

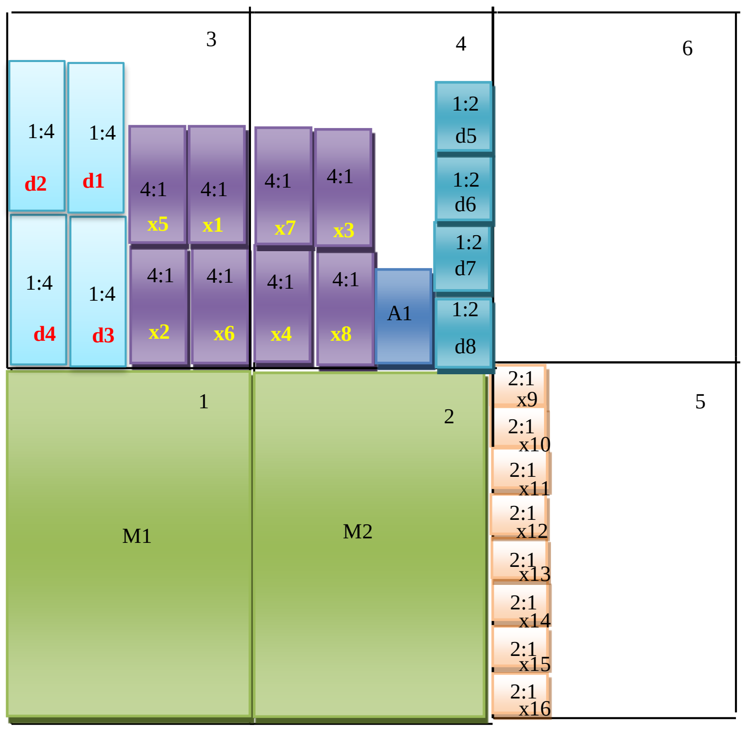


Fig. 19. *Final watermarked floorplan post embedding β and γ digits of the signature*

Now swapping of d3 and d4 in the floorplan is performed which will also result into embedding of γ digit. This is because, it will place d3 (odd component) right to the d4 (even component). This is how both γ digits of the signature are embedded in to the floorplan. The floorplan post embedding all γ digits is shown in Fig. 19. As shown in the figure, the embedded γ digits (i.e. odd Demuxes at the right to the even Demuxes) have been highlighted using Red colour. The floorplan in Fig. 19 is the final watermark embedded floorplan of the obfuscated RTL design of DCT core.

(c) Detection of Watermark

Detection of watermark into a design under-test proves the authenticity of the design. Absence of vendor's watermark into a design of the brand of a genuine vendor indicates that probably an adversary is misusing the brand of the genuine vendor to illegally sell counterfeited designs into the market. Hence, detection of the absence of watermark identifies the counterfeiting. Only those designs of vendor's brand would be proved authentic which contain the watermark (vendor's signature) of genuine owner. The detection of author's signature to identify his/her watermark is performed in following steps as shown in Fig. 20:

(i) Watermark constraints re-generation for verification

In this step, the watermarking constraints are regenerated from the vendor's signature comprising of α, β and γ variables. The re-generation of watermarking constraints is performed by applying vendor's /designer's specified mapping rules to the signature digits. Further, the presence of re-generated watermarking constraints is verified into the design.

(ii) Inspection of watermark into the design

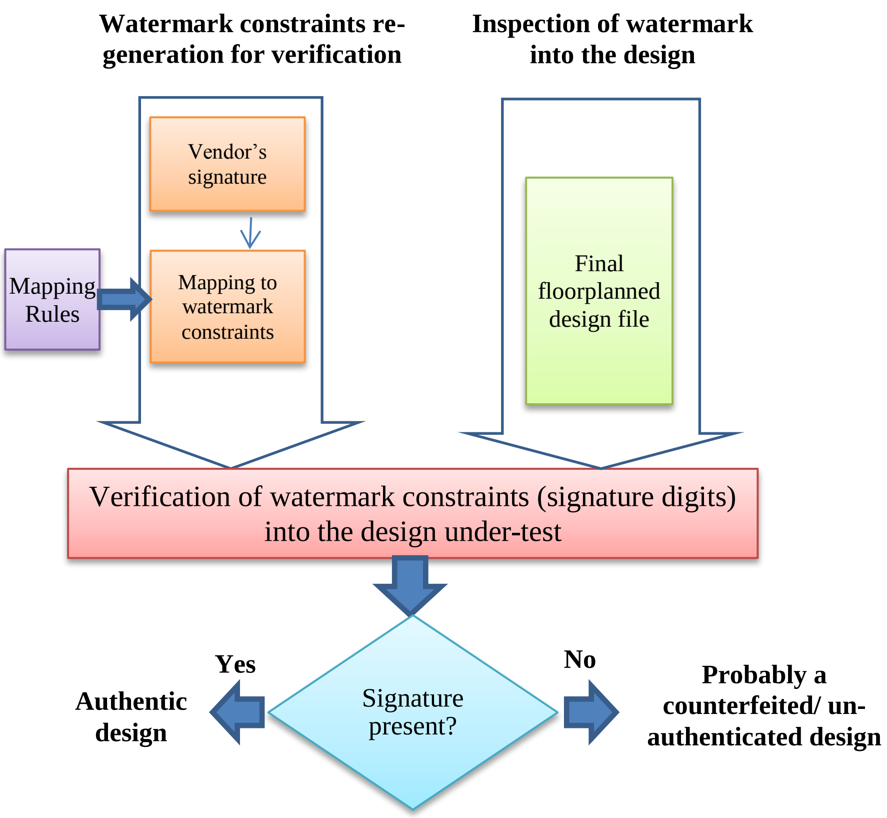

Watermark constraints re-generation for verification
Inspection of watermark into the design
Vendor's signature
Mapping Rules
Mapping to watermark constraints
Final floorplanned design file
Verification of watermark constraints (signature digits) into the design under-test
Yes
Authentic design
Signature present?
No
Probably a counterfeited/ un-authenticated design

Fig. 20. *Detection of signature digits embedded during physical level watermarking (Sengupta and Rathor, 2020)*

The final floorplanned design file obtained from the physical design tool is subjected to watermark detection. The following inputs are required to generate the final floorplanned design file during physical design process: (a) Verilog file of the DSP core design netlist (b) configuration file (c) watermarked early floorplan file of obfuscated design (this file contains the vendor's signature which represents the watermark) (e) library files (f) any other relevant files. In the floorplanned design file, orientations of RTL components are inspected to detect the watermark.

(iii) Verification of watermark constraints (signature digits)

If orientation of RTL components in the floorplanned design file is according to the watermark constraints (obtained post mapping of signature digits in first step), then the presence of original vendor's (author's) signature is verified in the design. Detection of author's signature (representing watermark) in the design of the brand of genuine author proves that the design is authentic. However, if the author's signature is missing in the design of the brand of genuine author, then probably those designs are counterfeited/ not authentic.

3. Key Size Analysis of the Structural Obfuscation

The total SO-key size (KS_{total}) of multi-structural transformation based obfuscation process is calculated as follows:

$$KS_{total} = \{\lceil(\log_2(U_m))\rceil + \lceil(\log_2 (C_m))\rceil + \lceil(\log_2 (R_m))\rceil + \lceil (\log_2 (T_m))\rceil + \lceil (\log_2 (F_m))\rceil\} \text{ bits} \quad (7)$$

Where, U_m indicates maximum value of UF, C_m indicates maximum cuts possible, R_m indicates maximum ROs possible, T_m indicates maximum THT possible and F_m indicates maximum folding possible. Driving the multiple techniques of structural transformations viz. loop unrolling, partitioning, ROE, THT and folding knob through individual SO-keys enhances the security level against RE (which results into Trojan insertion and piracy). Because of the involvement of multi-layered secret SO-keys in the structural transformations based obfuscation, the difficulty level of deducing true functionality and structure through RE by an attacker is augmented. This is because; now an attacker has to know both the multiple secret SO-keys and the structural transformation techniques to back-engineer the original functionality and structure. Further, finding true key combination among the exhaustive key combinations is difficult for the attacker. Hence, de-obfuscation becomes arduous for an attacker as the secret SO-keys, which are used to perform high level transformation based obfuscation, are not known to the attacker.

4.6. Low-cost Optimized Multi-key based Structural Obfuscation

This section presents a discussion on why a low cost and optimized version of multi-key based structural obfuscation is useful and how it is achieved. The discussion has been divided in following sub-sections:

1. Motivation for Low-Cost Optimized Structural Obfuscation

Selection of an optimal (resulting into low-cost) design from a vast number of functionally equivalent architecture designs requires extensive analysis of the design space. Exploring low-cost solution using design space exploration (DSE) framework, while simultaneously performing multiple structural transformations driven obfuscation based security is vital in two ways: (i) it minimizes the overhead impact caused due to employing structural obfuscation based security (ii) it ensures that the designer's area, power and delay budget is satisfied. This motivates the integration of DSE framework with the multiple high level transformations driven structural obfuscation based security technique. This section discusses how the particle swarm optimization (PSO) process (Mishra and Sengupta, 2014) is exploited to explore the design space in order to yield a low-cost obfuscated design without hampering the functionality of the DSP core. Below some approaches are mentioned where even though obfuscation has been used for security, however exploration of low cost solution has not been addressed.

A brief discussion on some approaches that secures DSP cores using obfuscation technique is as follows: Approaches proposed by (Chakraborty, Bhunia, 2009; Chakraborty, Bhunia, 2011) applied obfuscation based IP core security against Trojan insertion attacks. However, high level synthesis (HLS) framework has not been leveraged by these approaches to perform obfuscation. Further these approaches (Chakraborty, Bhunia, 2009; Chakraborty, Bhunia, 2011) do not target DSP cores and do not explore

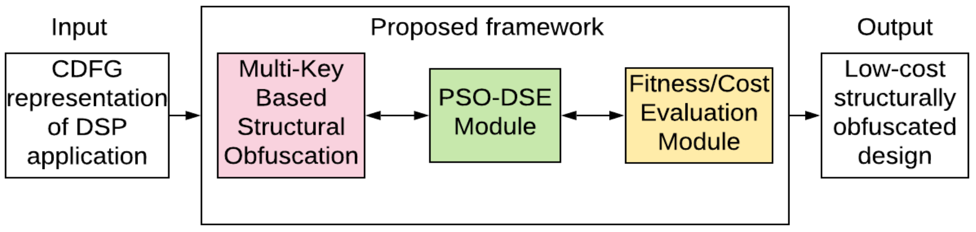

Input
CDFG representation of DSP application
Proposed framework
Multi-Key Based Structural Obfuscation
PSO-DSE Module
Fitness/Cost Evaluation Module
Output
Low-cost structurally obfuscated design

Fig. 21. *Overview of low-cost key-based structural obfuscation approach*

low-cost solution. In approaches proposed by (Lao and Parhi, 2015; Sengupta and Roy, 2017) structural obfuscation has been applied on DSP cores. However, these approaches do not explore the design space to ensure the low design cost. Therefore, this section presents a discussion on how the PSO-DSE is integrated with the key-based transformation driven structural obfuscation for DSP cores to explore a low-cost obfuscated design.

2. *High Level Perspective*

High level view of the methodology of generating low-cost obfuscated designs is shown in Fig. 21. It comprises of three major modules viz. key driven structural obfuscation module, PSO module and fitness/cost evaluation module. The structural obfuscation module employed in this approach accepts a CDFG (of the corresponding DSP application) and secret keys as inputs and produces a structurally modified but functionally equivalent CDFG design as output. The output of the structural obfuscation module serves as the input to the PSO-DSE module which determines the optimal resource configuration that would incur minimal design cost. PSO-DSE process uses the fitness/cost evaluation module for computing cost and finally a low-cost highly secured structurally obfuscated design is generated.

3. *Details of Methodology*

Figure 22 shows detailed description of the low-cost multi-key based structural obfuscation approach. The inputs are a CDFG representing a DSP application, module library, PSO control parameters (iteration count, swarm size, acceleration coefficient etc.) and secret multi-key values for obfuscation as shown in the figure. The first transformation i.e. key-based loop unrolling is applied to the input CDFG depending on its applicability; thereafter the CDFG is partitioned into as many partitions as per the specification of the designer's secret key. Further, redundant nodes elimination process is performed from the CDFG based on the designer's secret key, and then finally key-based tree height transformation is applied to the CDFG. Subsequently, PSO-DSE is applied to determine optimal resource constraints that would yield minimal design cost. Based on the optimal resource constraints, final scheduling is performed on the CDFG. The final obfuscation technique i.e. folding is performed on scheduled CDFG which results into structurally obfuscated scheduled CDFG of DSP core. Further upon performing datapath and controller generation of HLS, structurally obfuscated register transfer level (RTL) design is generated.

(a) Summary of key-driven structural transformation based obfuscation and their effect on gate count/design cost

(i) Loop Unrolling (Sengupta and Rathor, 2020): It is a high-level transformation technique used to obfuscate the loop-based DSP core by untwining the loop based on the secret SO-key1. The loop unrolling affects number of control steps in the controller and increases logic density, thereby causing a change in the gate count (and structure) without changing functionality.

(ii) Partitioning (Sengupta and Rathor, 2020): This high-level transformation technique is performed to obfuscate the CDFG by dividing it into a number of partitions depending on the secret SO-key2. Partitioning affects the count of multiplexers and demultiplexers in the final RTL, thereby changing the gate count (and structure) without causing any change to the functionality of the CDFG.

(iii) Redundant Operation Elimination (Sengupta and Rathor, 2020): This high-level transformation technique is performed to obfuscate the CDFG by removing redundant nodes from the graph based on the secret SO-key3. Elimination of redundant nodes affects the logic density thereby decreasing cost and also decreasing the gate count (and structure) without causing any change to the functionality.

(iv) Tree Height Transformation (Sengupta and Rathor, 2020): This high-level transformation technique is performed to obfuscate the CDFG by transforming the height of partitioned graph based on secret SO-key4. It leads to a decrease in the number of control steps, thereby decreasing gate complexity as well as count of the controller design and design cost.

(v) Folding (Sengupta and Rathor, 2020): It is a high-level transformation technique in which a common resource is shared among a set of nodes that have the same operation type but are present in successive control steps in the scheduled graph. The value of the secret SO-key5 specified by the user denotes the number of folding transformations to be applied which is referred as the folding knob. Sharing of resources enables maximum utilization of resources, thereby decreasing the cost.

(b) PSO-DSE based framework for generating low-cost design solution

A generic PSO has been mapped to DSE process in high level synthesis (HLS) framework and is integrated with key based structural obfuscation to generate low cost structurally obfuscated DSP design. The PSO-DSE process comprises of five steps namely (i) particle initialization (ii) velocity computation (iii) position computation (iv) fitness evaluation and (v) finding global and local best positions. Steps (ii)-(v) are repeated for each particle over the iterations, until either terminating criterion is met (e.g. 100 iterations are completed) or there is no change in the global best position for some successive iterations (e.g. 10). The detailed explanation of each step is given below (Mishra and Sengupta, 2014).

(i) *Particle Initialization:* Dimension of particle position equals number of types of resources (for example in this approach, it is a 4-D vector because there are four types of resources for the case studies tested viz. adder, subtractor, multiplier and comparator). Position of first particle from the population is initialized with maximum resource constraints i.e. R_{max}. Position of second particle is initialized with minimum resource constraints i.e. R_{min}. Position of the third particle is initialized to average of the positions of the first two particles i.e. $(R_{max}+R_{min})/2$. Then the position of the remaining particles is initialized with random values between minimum resource constraints and maximum resource constraints.

(ii) *Velocity Computation:* Velocity of a particle is computed using following equation (Mishra and Sengupta, 2014):

$$V_{new} = \omega * V_{old} + b_1 * r_1 * (R_{lb} - R_{curr}) + b_2 * r_2 * \left(R_{gb} - R_{curr}\right) \quad (8)$$

where ω is inertia weight which linearly decreases from 0.9 to 0.4 (to obtain global best), b_1 and b_2 are acceleration coefficients. b_1 linearly decreases from 3 to 2, b_2 linearly increases from 2 to 3 over the iterations of PSO. Further, r_1 and r_2 are random values between 0 and 1, R_{curr}, R_{lb} are the current and best position of the current particle and R_{gb} is the best position considering all the particles so far. Initial velocity for every particle is assigned to zero. Velocity computation is followed by velocity clamping using the following (Mishra and Sengupta, 2014):

$$V_{new}^{i} = \begin{cases} V_{max}^{i}, \ if \ V_{new}^{i} > V_{max}^{i} \\ -V_{max}^{i}, \ if \ V_{new}^{i} < -V_{max}^{i} \\ V_{new}^{i}, \quad otherwise \end{cases} \quad (9)$$

Where V^i means velocity of ith dimension, and V_{max}^{i} is given by $(R_{max}^{i}\text{-}R_{min}^{i})/2$.

(iii) *Position Computation*: Position is computed by moving in the direction of velocity. Position is calculated using the following (Mishra and Sengupta, 2014):

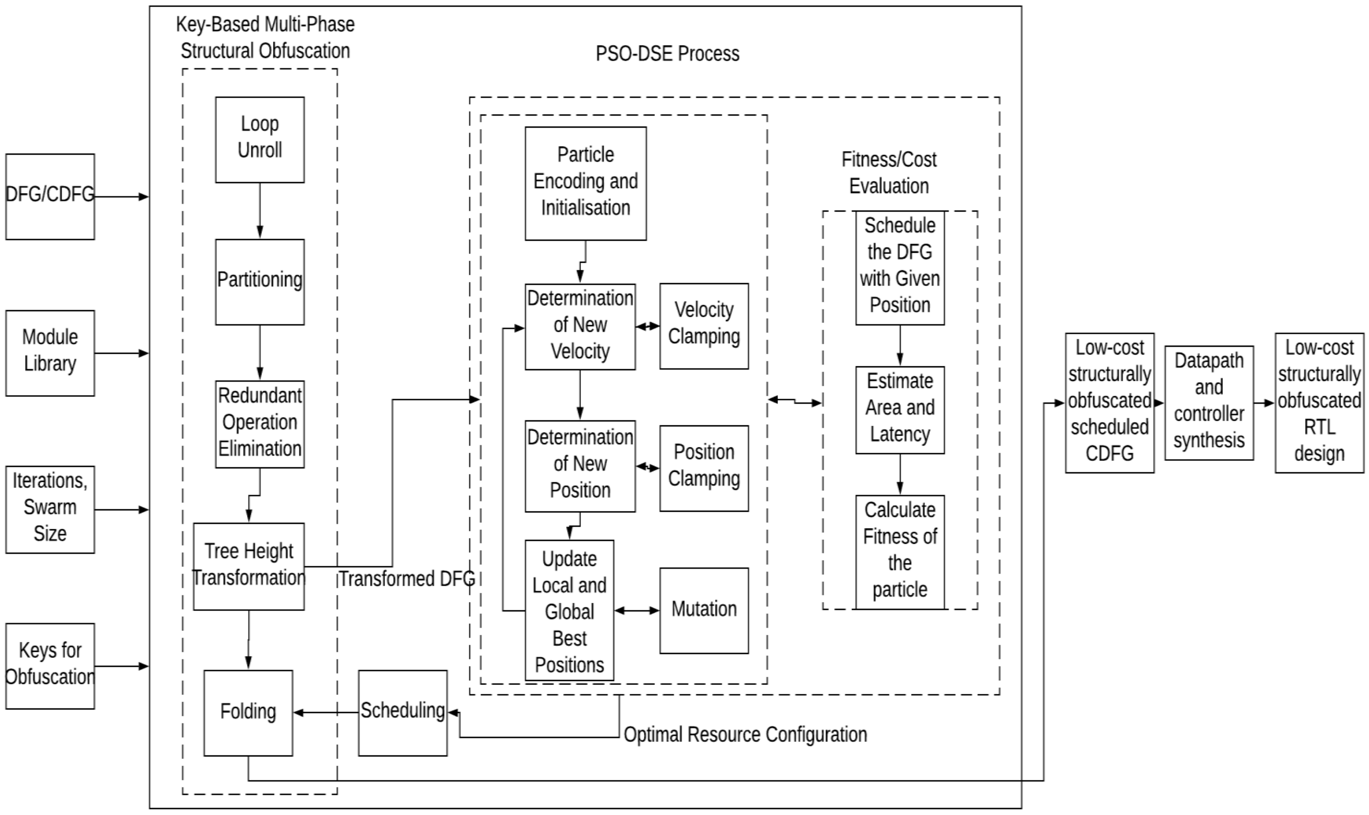
$$\mathrm{R_{new}} = \mathrm{R_{old}} + \mathrm{V_{new}} \quad (10)$$

If the new position calculated exceeds maximum resource constraints then clamping is applied by selecting a random value between minimum and maximum resource constraints for each dimension.

(iv) *Fitness Evaluation:* Fitness value for each particle in the population is computed using the following cost function (Mishra and Sengupta, 2014):

$$Cost = w_1 * \left(\frac{A}{A_{max}}\right) + w_2 * \left(\frac{L}{L_{\max}}\right) \quad (11)$$

Where, A and L are area and latency of the design with constrains of the particle and A_{max}, L_{max} are maximum possible area and latency. Further, w_1 and w_2 are the designer specified weights (each taken as 0.5).



DFG/CDFG
Module Library
Iterations, Swarm Size
Keys for Obfuscation
Key-Based Multi-Phase Structural Obfuscation
Loop Unroll
Partitioning
Redundant Operation Elimination
Tree Height Transformation
Folding
Scheduling
Transformed DFG
PSO-DSE Process
Particle Encoding and Initialisation
Determination of New Velocity
Velocity Clamping
Determination of New Position
Position Clamping
Update Local and Global Best Positions
Mutation
Fitness/Cost Evaluation
Schedule the DFG with Given Position
Estimate Area and Latency
Calculate Fitness of the particle
Optimal Resource Configuration
Low-cost structurally obfuscated scheduled CDFG
Datapath and controller synthesis
Low-cost structurally obfuscated RTL design

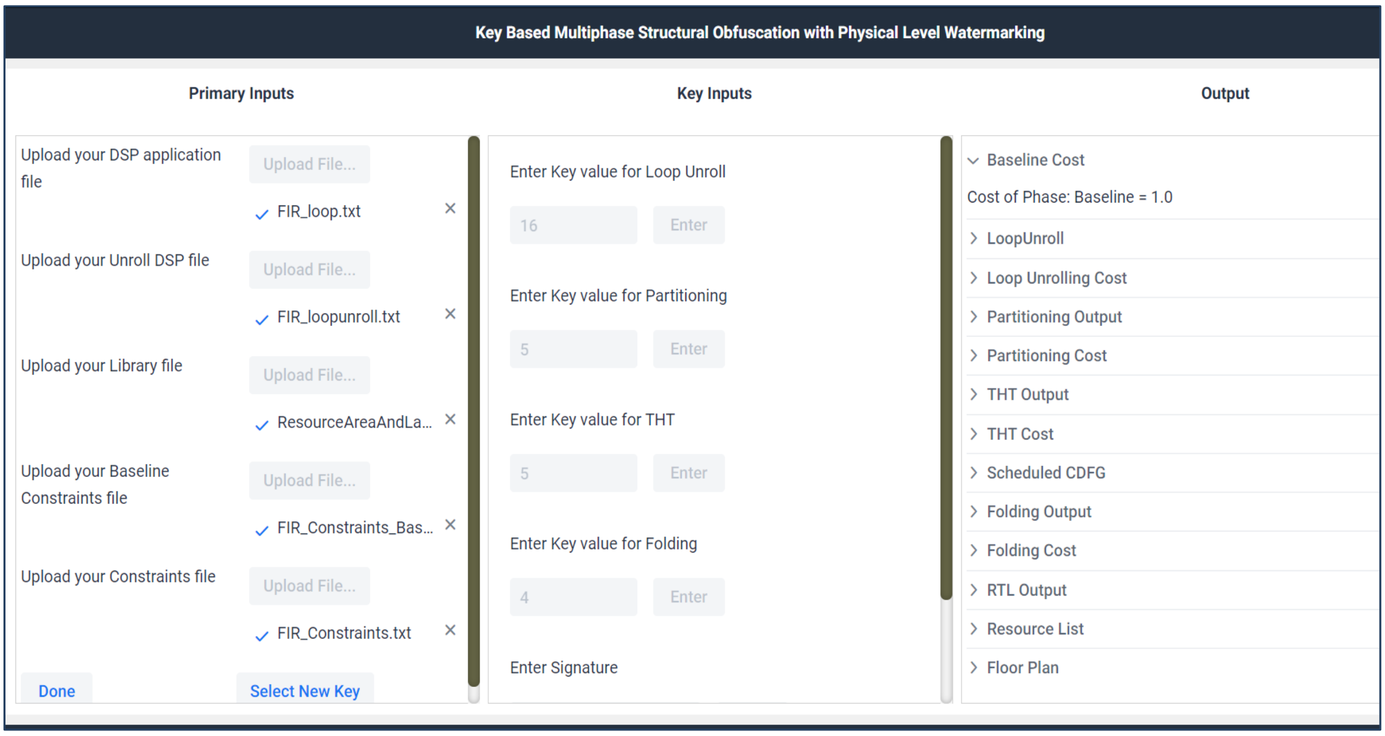
Fig. 22. *Details of low-cost key-based structural obfuscation approach*

(v) *Determining Local and Global Best*: Local best is the best position (i.e. the position at which the cost value for the particle is minimum) achieved by a particle so far. Global best is the position at which the cost value is minimum for the entire population (i.e. minimum cost of all the local bests).

(vi) *Mutation:* Mutation is applied in order to avoid getting stuck in local minima. It is applied on every fifth iteration with designer's choice of probability of mutation. And, it is applied on local best position of every particle. Further, mutation is applied by performing a left shift operation on every particle with even index, followed by position clamping. On every particle with odd index, mutation is applied by adding a random value (between minimum and maximum resource constraints) to the values at even index of the particle position vector and subtracting the same value from the values at odd index of the particle position vector, followed by position clamping. Then fitness/cost is evaluated at this new position obtained after mutation. If the cost value is lesser than value computed using the local best, then the local best is updated with the new position.

Thus low-cost structurally obfuscated design is generated post-performing PSO-DSE.

4.7. Structural Obfuscation and Physical level Watermarking Tool for Securing Hardware Accelerators


Key Based Multiphase Structural Obfuscation with Physical Level Watermarking

Primary Inputs

Upload your DSP application file — Upload File... ✓ FIR_loop.txt

Upload your Unroll DSP file — Upload File... ✓ FIR_loopunroll.txt

Upload your Library file — Upload File... ✓ ResourceAreaAndLa...

Upload your Baseline Constraints file — Upload File... ✓ FIR_Constraints_Bas...

Upload your Constraints file — Upload File... ✓ FIR_Constraints.txt

Key Inputs

Enter Key value for Loop Unroll: 16

Enter Key value for Partitioning: 5

Enter Key value for THT: 5

Enter Key value for Folding: 4

Enter Signature

Output

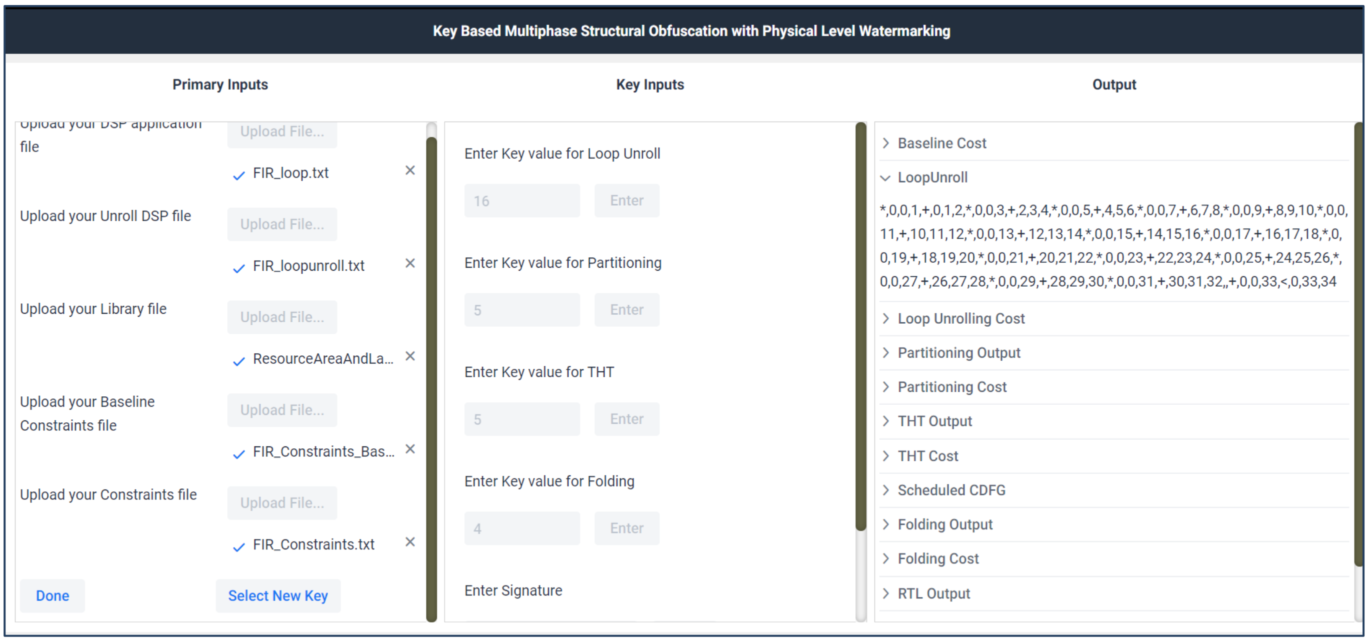
Baseline Cost

Cost of Phase: Baseline = 1.0

Fig. 23. *A snapshot of GUI of KSO-PW tool*

Authors have developed a ***KSO-PW tool*** (key-driven structural obfuscation and physical level watermarking tool) to simulate and analyse the functionality of structural obfuscation and physical level watermarking approach for securing DSP hardware accelerators. This tool provides a friendly graphical interface to users. A snapshot of the graphical user interface (GUI) of the tool is shown in Fig. 23, for FIR application input. 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 structural obfuscation and physical level watermarking based double line of defence approach. The panel in the middle shows the buttons for entering secret SO-keys (for multiple structural transformation techniques viz. loop unrolling, partitioning, ROE, THT and folding-knob) and vendor's signature for watermarking. Since ROE is not applicable for FIR application, therefore the button for entering the ROE key input remains invisible as shown in Fig. 23. It is noteworthy in the GUI of KSO-PW tool that for a DSP application, only those secret key-buttons are activated for which the corresponding structural transformation techniques are applicable. The KSO-PW tool accepts the DSP application input in the form CDFG along with module library file and resource constraints file. The tool shows outputs of all intermediate steps of structural obfuscation and watermarking approach and finally generated watermarked floorplan of the obfuscated design. Further it also shows design cost before performing structural obfuscation and watermarking (i.e. baseline design cost), design cost after intermediate steps and the final design cost.

Let's generate all the intermediate and final output of structural obfuscation and watermarking based double line of defence approach for 160-tap FIR filter core using the KSO-PW tool. Post feeding CDFG of 160-tap FIR filter as input, the secret SO-



Key Based Multiphase Structural Obfuscation with Physical Level Watermarking

Primary Inputs | Key Inputs | Output

Upload your DSP application file — Upload File... — ✓ FIR_loop.txt ×
Upload your Unroll DSP file — Upload File... — ✓ FIR_loopunroll.txt ×
Upload your Library file — Upload File... — ✓ ResourceAreaAndLa... ×
Upload your Baseline Constraints file — Upload File... — ✓ FIR_Constraints_Bas... ×
Upload your Constraints file — Upload File... — ✓ FIR_Constraints.txt ×
Done — Select New Key

Enter Key value for Loop Unroll — 16 — Enter
Enter Key value for Partitioning — 5 — Enter
Enter Key value for THT — 5 — Enter
Enter Key value for Folding — 4 — Enter
Enter Signature

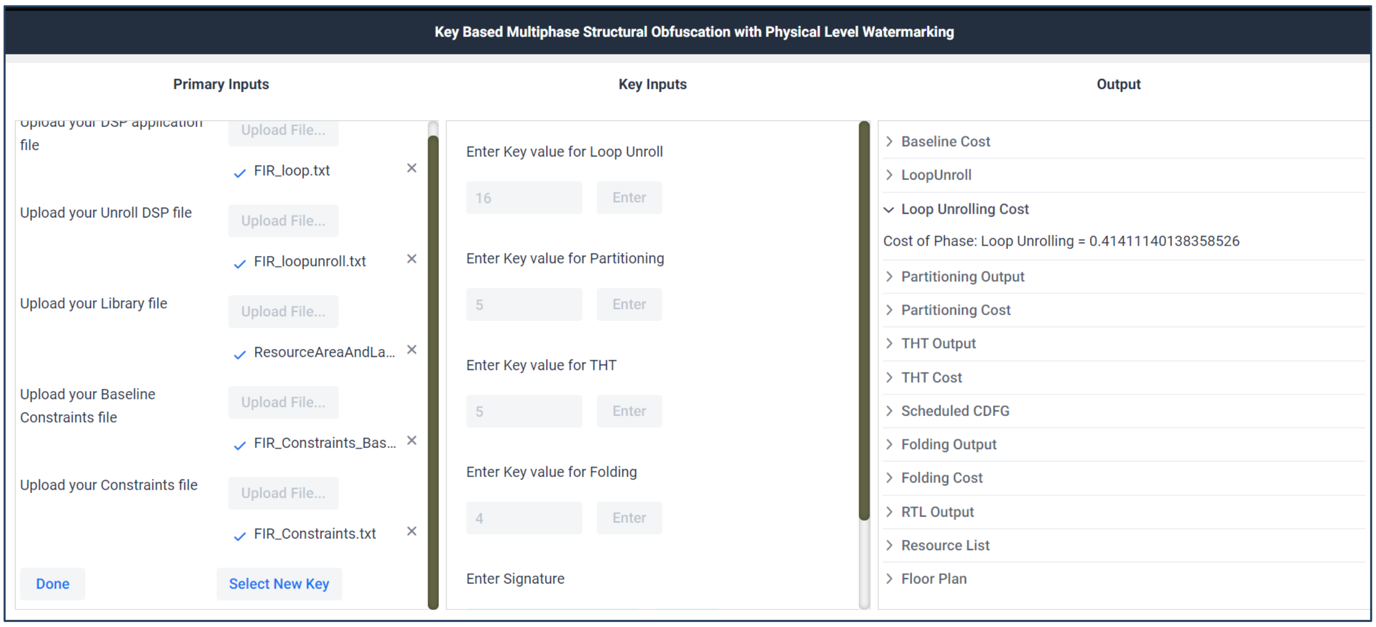
Baseline Cost

LoopUnroll

```
*,0,0,1,+,0,1,2,*,0,0,3,+,2,3,4,*,0,0,5,+,4,5,6,*,0,0,7,+,6,7,8,*,0,0,9,+,8,9,10,*,0,0,
11,+,10,11,12,*,0,0,13,+,12,13,14,*,0,0,15,+,14,15,16,*,0,0,17,+,16,17,18,*,0,
0,19,+,18,19,20,*,0,0,21,+,20,21,22,*,0,0,23,+,22,23,24,*,0,0,25,+,24,25,26,*,
0,0,27,+,26,27,28,*,0,0,29,+,28,29,30,*,0,0,31,+,30,31,32,,+,0,0,33,<,0,33,34
```

Loop Unrolling Cost
Partitioning Output
Partitioning Cost
THT Output
THT Cost
Scheduled CDFG
Folding Output
Folding Cost
RTL Output

Fig. 24. *Different SO-key values fed and output of loop unrolling of FIR filter application shown onto the tool*



Key Based Multiphase Structural Obfuscation with Physical Level Watermarking

Primary Inputs | Key Inputs | Output

Upload your DSP application file — Upload File... — ✓ FIR_loop.txt ×
Upload your Unroll DSP file — Upload File... — ✓ FIR_loopunroll.txt ×
Upload your Library file — Upload File... — ✓ ResourceAreaAndLa... ×
Upload your Baseline Constraints file — Upload File... — ✓ FIR_Constraints_Bas... ×
Upload your Constraints file — Upload File... — ✓ FIR_Constraints.txt ×
Done — Select New Key

Enter Key value for Loop Unroll — 16 — Enter
Enter Key value for Partitioning — 5 — Enter
Enter Key value for THT — 5 — Enter
Enter Key value for Folding — 4 — Enter
Enter Signature

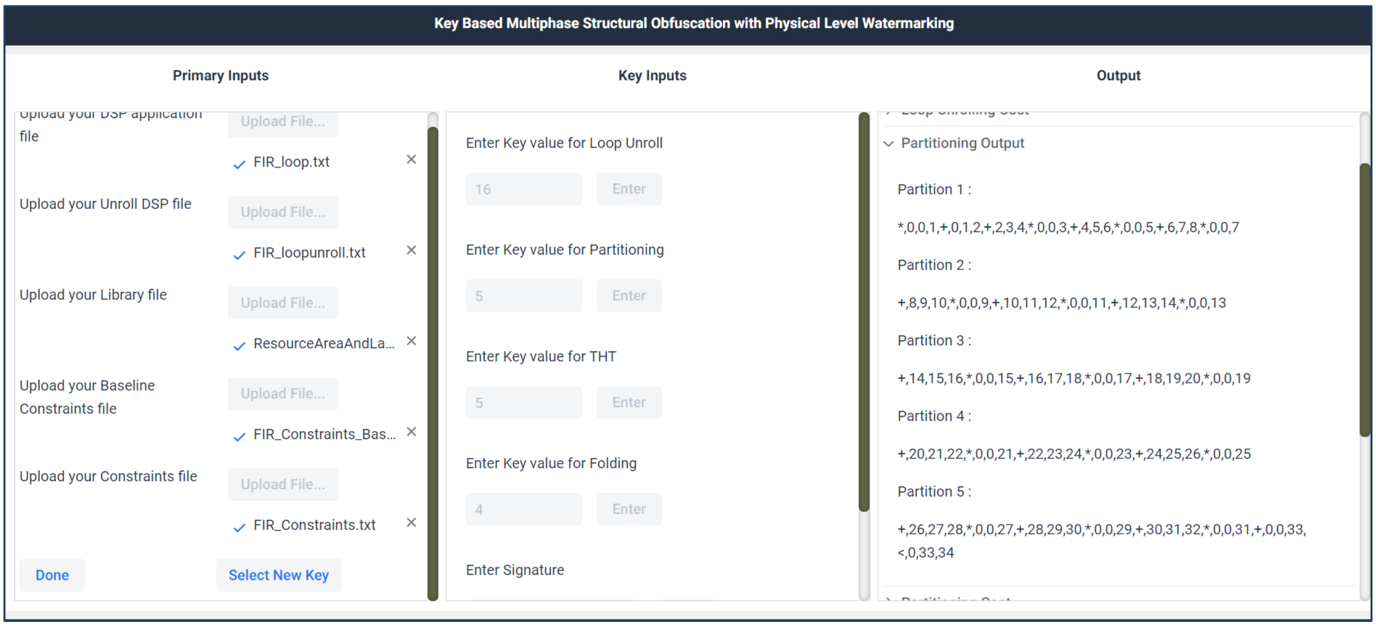
Baseline Cost
LoopUnroll
Loop Unrolling Cost

```
Cost of Phase: Loop Unrolling = 0.41411140138358526
```

Partitioning Output
Partitioning Cost
THT Output
THT Cost
Scheduled CDFG
Folding Output
Folding Cost
RTL Output
Resource List
Floor Plan

Fig. 25. *Design cost post loop unrolling of FIR filter application*

keys for loop unrolling, partitioning, THT and folding are fed to the tool as shown in Fig. 24. As sown in the figure, the entered SO-key1 for loop unrolling is 16 (i.e. UF=16). Therefore the loop body of 160-tap FIR filter application is unrolled 16 times. The output of loop unrolling is shown at the output panel in Fig. 24. The loop unrolling output shown onto the GUI is interpreted as follows: each opn has been denoted by four tuples: (operation type, input1 operation #, input2 operation # and current operation #). The design cost post performing loop unrolling is shown in Fig. 25. Further as shown in Fig. 26, the entered secret SO-key2 for partitioning is 5. Therefore the loop unrolled FIR filter application is divided into 5 partitions. The output of partitioning is


Key Based Multiphase Structural Obfuscation with Physical Level Watermarking

Primary Inputs | Key Inputs | Output

Upload your DSP application file — Upload File... — ✓ FIR_loop.txt
Upload your Unroll DSP file — Upload File... — ✓ FIR_loopunroll.txt
Upload your Library file — Upload File... — ✓ ResourceAreaAndLa...
Upload your Baseline Constraints file — Upload File... — ✓ FIR_Constraints_Bas...
Upload your Constraints file — Upload File... — ✓ FIR_Constraints.txt
Done — Select New Key

Enter Key value for Loop Unroll: 16 — Enter
Enter Key value for Partitioning: 5 — Enter
Enter Key value for THT: 5 — Enter
Enter Key value for Folding: 4 — Enter
Enter Signature

Partitioning Output

Partition 1 :

,0,0,1,+,0,1,2,+,2,3,4,,0,0,3,+,4,5,6,*,0,0,5,+,6,7,8,*,0,0,7

Partition 2 :

+,8,9,10,*,0,0,9,+,10,11,12,*,0,0,11,+,12,13,14,*,0,0,13

Partition 3 :

+,14,15,16,*,0,0,15,+,16,17,18,*,0,0,17,+,18,19,20,*,0,0,19

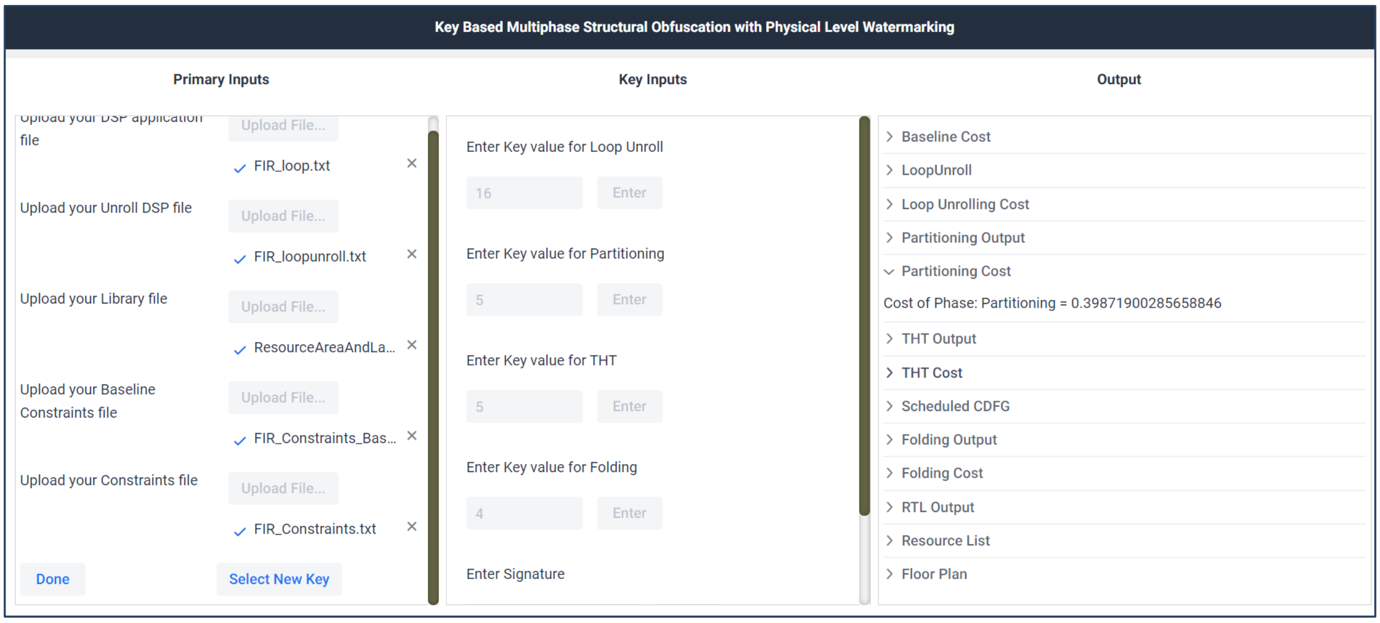
Partition 4 :

+,20,21,22,*,0,0,21,+,22,23,24,*,0,0,23,+,24,25,26,*,0,0,25

Partition 5 :

+,26,27,28,*,0,0,27,+,28,29,30,*,0,0,29,+,30,31,32,*,0,0,31,+,0,0,33,
<,0,33,34

Fig. 26. *Output of partitioning of loop unrolled FIR filter application shown onto the tool*


Key Based Multiphase Structural Obfuscation with Physical Level Watermarking

Primary Inputs | Key Inputs | Output

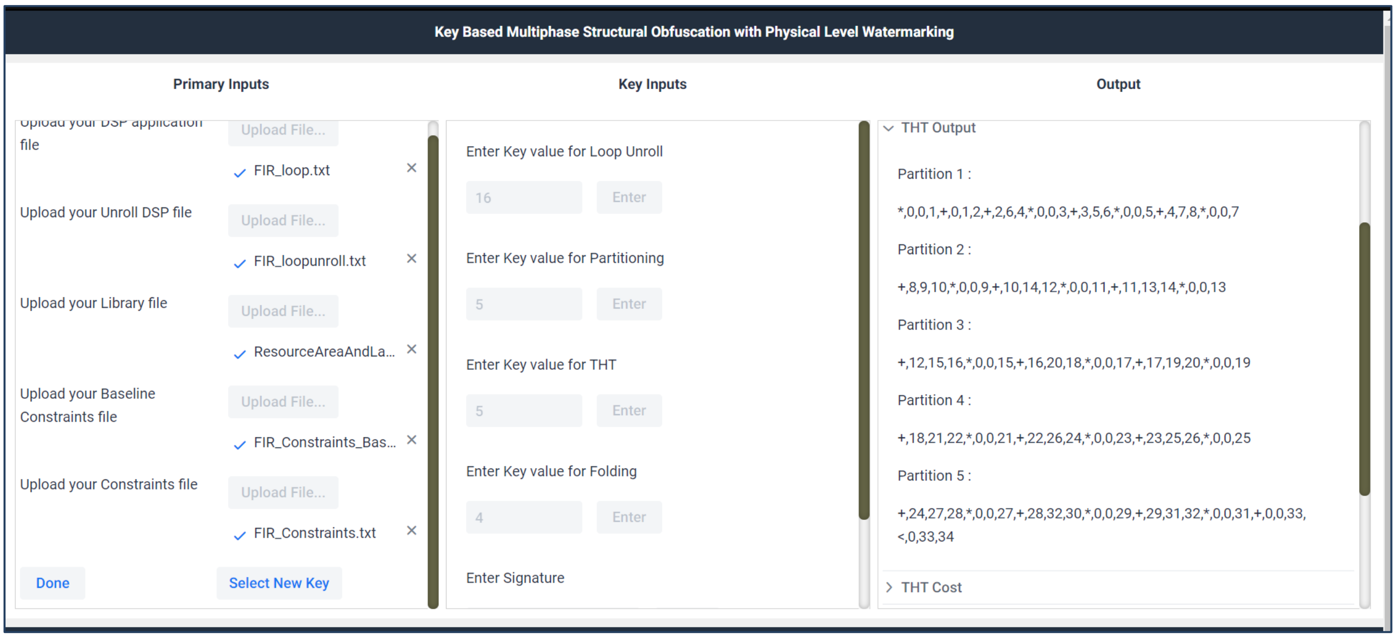
Upload your DSP application file — Upload File... — ✓ FIR_loop.txt
Upload your Unroll DSP file — Upload File... — ✓ FIR_loopunroll.txt
Upload your Library file — Upload File... — ✓ ResourceAreaAndLa...
Upload your Baseline Constraints file — Upload File... — ✓ FIR_Constraints_Bas...
Upload your Constraints file — Upload File... — ✓ FIR_Constraints.txt
Done — Select New Key

Enter Key value for Loop Unroll: 16 — Enter
Enter Key value for Partitioning: 5 — Enter
Enter Key value for THT: 5 — Enter
Enter Key value for Folding: 4 — Enter
Enter Signature

Baseline Cost
LoopUnroll
Loop Unrolling Cost
Partitioning Output
Partitioning Cost
Cost of Phase: Partitioning = 0.39871900285658846
THT Output
THT Cost
Scheduled CDFG
Folding Output
Folding Cost
RTL Output
Resource List
Floor Plan

Fig. 27. *Design cost post partitioning shown onto the tool*

shown on to the output panel in Fig. 26. The design cost post performing partitioning is shown in Fig. 27. As discussed earlier, ROE is not applicable on partitioned DFG of FIR applications as redundant nodes/operations are not present. Further as shown in Fig. 28, the entered secret SO-key4 for THT is 5. Therefore the THT is applied on all 5 partitions. The output of THT is shown onto the output panel in Fig. 28. As shown in the figure, partitions have modified post applying THT. The design cost post performing THT is shown in Fig. 29. Further, the CDFG of obfuscated partitioned FIR application is scheduled into 17 control steps. Scheduled partitioned CDFG is shown in Fig. 30, Fig. 31 and Fig. 32. The scheduling of operations in control steps 1 to 6 is



Partition 1 :

,0,0,1,+,0,1,2,+,2,6,4,,0,0,3,+,3,5,6,*,0,0,5,+,4,7,8,*,0,0,7

Partition 2 :

+,8,9,10,*,0,0,9,+,10,14,12,*,0,0,11,+,11,13,14,*,0,0,13

Partition 3 :

+,12,15,16,*,0,0,15,+,16,20,18,*,0,0,17,+,17,19,20,*,0,0,19

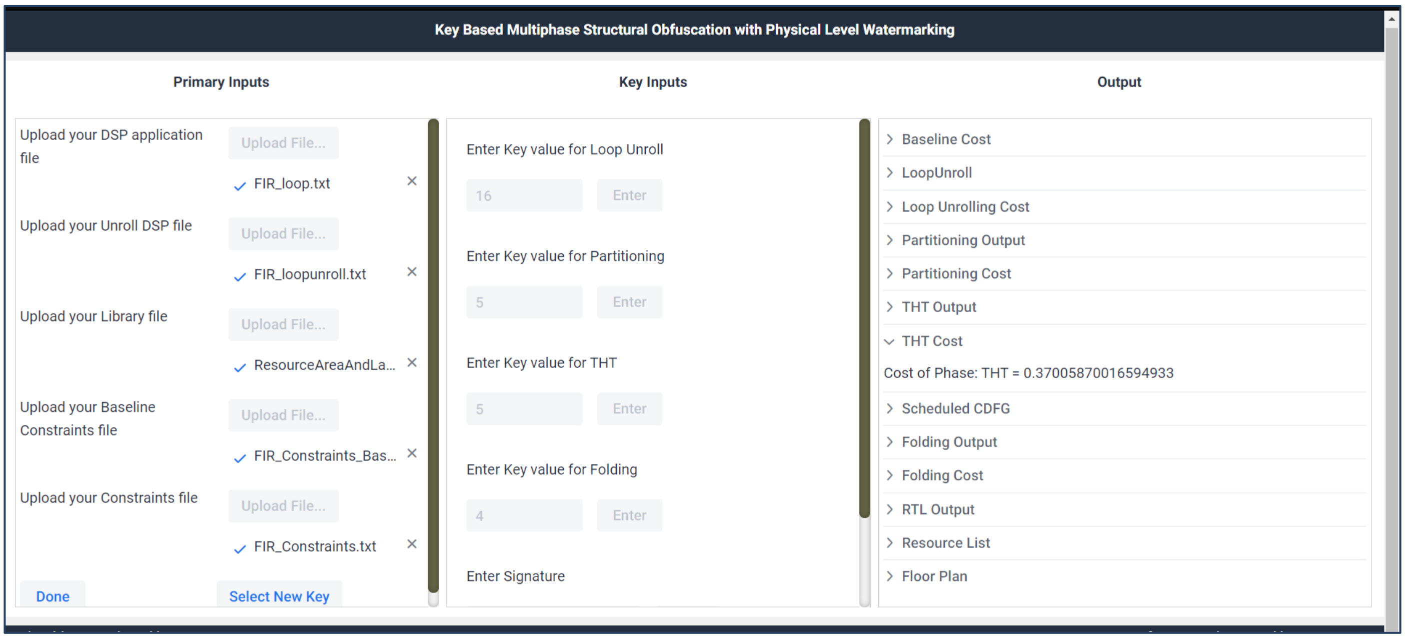
Partition 4 :

+,18,21,22,*,0,0,21,+,22,26,24,*,0,0,23,+,23,25,26,*,0,0,25

Partition 5 :

+,24,27,28,*,0,0,27,+,28,32,30,*,0,0,29,+,29,31,32,*,0,0,31,+,0,0,33,<,0,33,34

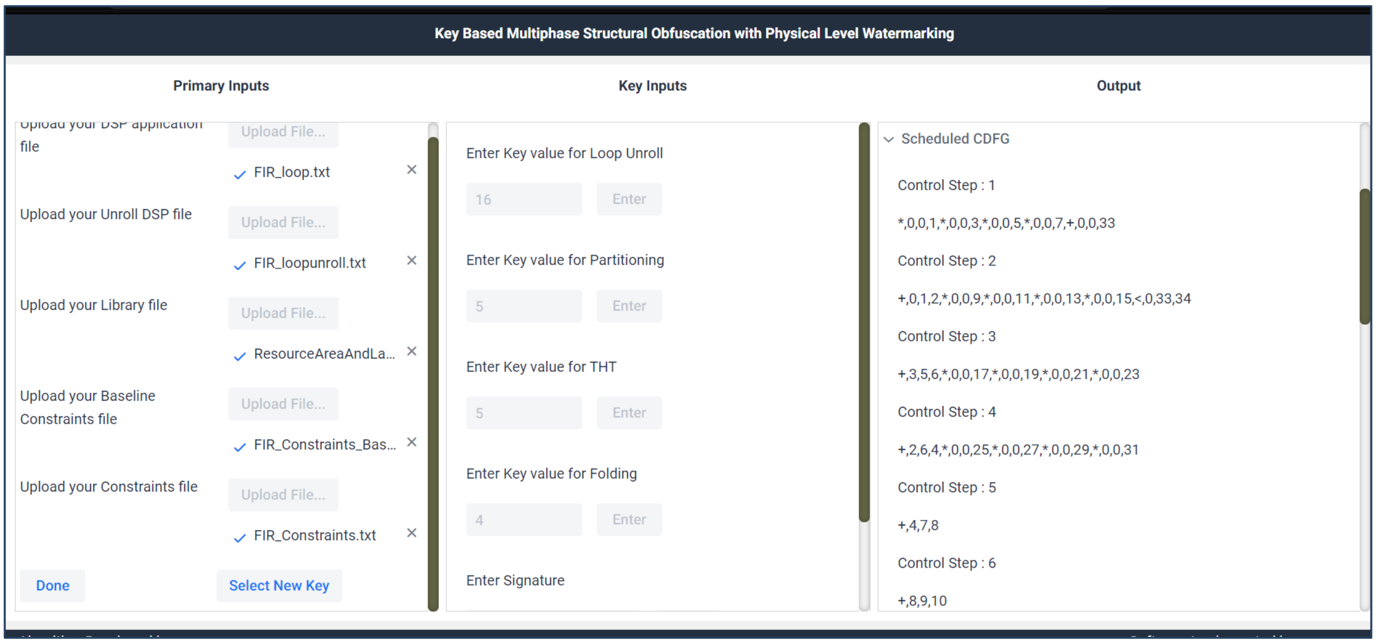
Fig. 28. *Output of THT shown onto the tool*



Cost of Phase: THT = 0.37005870016594933

Fig. 29. *Design cost post THT shown onto the tool*

shown in Fig. 30, scheduling in control steps 7 to 13 is shown in Fig. 31 and scheduling in control step 14 to 17 is shown in Fig. 32. Further as shown in Fig. 33, the entered secret SO-key5 for folding is 4. Therefore folding is performed on four folding occurrences in the scheduled partitioned CDFG of FIR filter application. The output of folding is shown onto the output panel in Fig. 33. As shown in the figure, operation/ node pairs (4, 8), (14, 12), (20, 18) and (26, 24) have been used for folding based on folding factor of 2. Therefore, both operations of same type in each pair are essentially allocated to the same FU resource unit. The design cost post performing folding is shown in Fig. 34. Further, Fig. 35 to Fig. 38 show the RTL output post-HLS of each


Key Based Multiphase Structural Obfuscation with Physical Level Watermarking

Primary Inputs

Upload your DSP application file — Upload File... — ✓ FIR_loop.txt ×

Upload your Unroll DSP file — Upload File... — ✓ FIR_loopunroll.txt ×

Upload your Library file — Upload File... — ✓ ResourceAreaAndLa... ×

Upload your Baseline Constraints file — Upload File... — ✓ FIR_Constraints_Bas... ×

Upload your Constraints file — Upload File... — ✓ FIR_Constraints.txt ×

Done — Select New Key

Key Inputs

Enter Key value for Loop Unroll: 16 — Enter

Enter Key value for Partitioning: 5 — Enter

Enter Key value for THT: 5 — Enter

Enter Key value for Folding: 4 — Enter

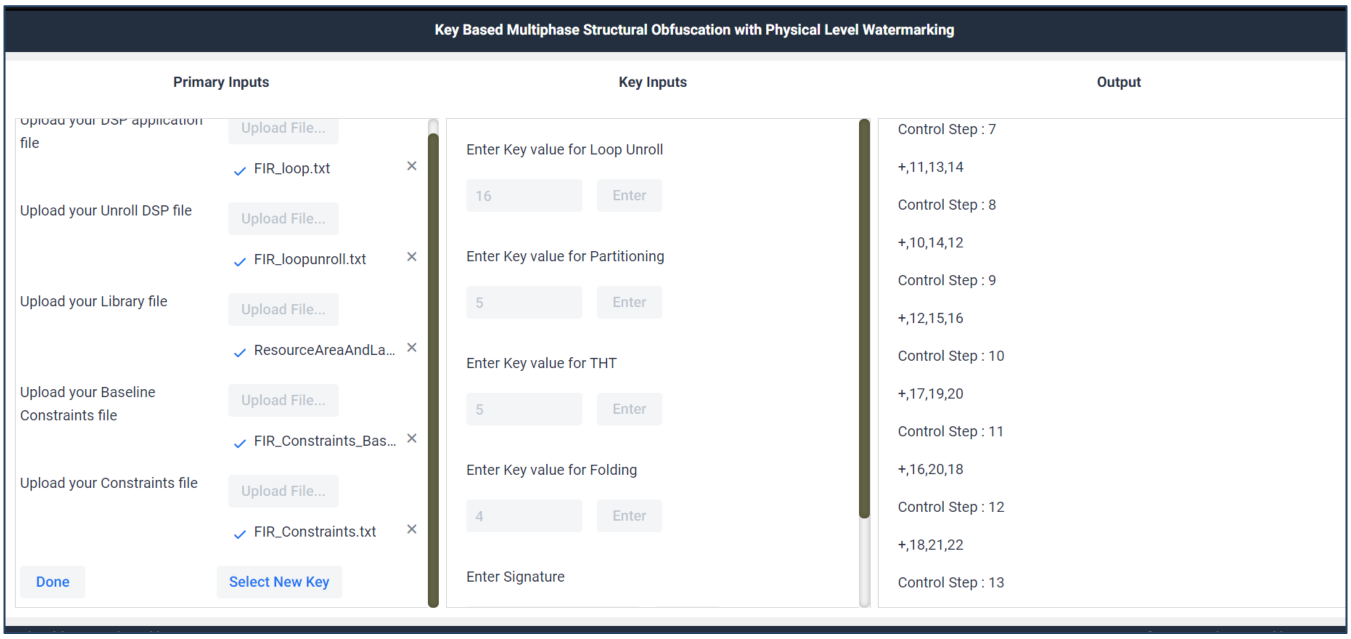
Enter Signature

Output

Scheduled CDFG

```
Control Step : 1
*,0,0,1,*,0,0,3,*,0,0,5,*,0,0,7,+,0,0,33
Control Step : 2
+,0,1,2,*,0,0,9,*,0,0,11,*,0,0,13,*,0,0,15,<,0,33,34
Control Step : 3
+,3,5,6,*,0,0,17,*,0,0,19,*,0,0,21,*,0,0,23
Control Step : 4
+,2,6,4,*,0,0,25,*,0,0,27,*,0,0,29,*,0,0,31
Control Step : 5
+,4,7,8
Control Step : 6
+,8,9,10
```

Fig. 30. *Output of scheduling shown onto the tool; excerpt-1: control steps 1 to 6*


Key Based Multiphase Structural Obfuscation with Physical Level Watermarking

Primary Inputs

Upload your DSP application file — Upload File... — ✓ FIR_loop.txt ×

Upload your Unroll DSP file — Upload File... — ✓ FIR_loopunroll.txt ×

Upload your Library file — Upload File... — ✓ ResourceAreaAndLa... ×

Upload your Baseline Constraints file — Upload File... — ✓ FIR_Constraints_Bas... ×

Upload your Constraints file — Upload File... — ✓ FIR_Constraints.txt ×

Done — Select New Key

Key Inputs

Enter Key value for Loop Unroll: 16 — Enter

Enter Key value for Partitioning: 5 — Enter

Enter Key value for THT: 5 — Enter

Enter Key value for Folding: 4 — Enter

Enter Signature

Output

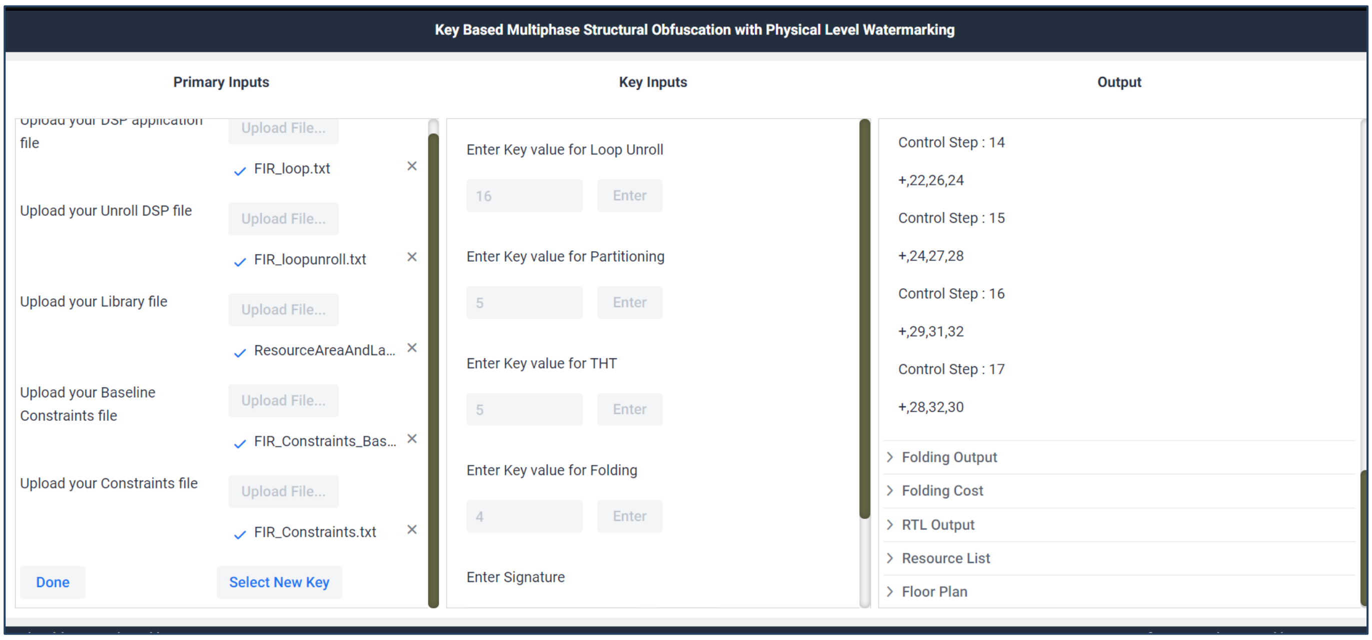
```
Control Step : 7
+,11,13,14
Control Step : 8
+,10,14,12
Control Step : 9
+,12,15,16
Control Step : 10
+,17,19,20
Control Step : 11
+,16,20,18
Control Step : 12
+,18,21,22
Control Step : 13
```

Fig. 31. *Output of scheduling shown onto the tool; excerpt-2: control steps 7 to 13*

scheduled partition. The RTL output of partition 1 is shown in Fig. 35. The RTL output of partition 2 and an excerpt of partition 3 are shown in Fig. 36. The RTL output of remaining excerpt of partition 3 and an excerpt of partition 4 are shown in Fig. 37. The RTL output of remaining excerpt of partition 4 and RTL output of partition 5 are shown in Fig. 38. The interconnection of different components in the RTL output shown onto the GUI (in Fig. 35 to Fig. 38) is interpreted as follows:

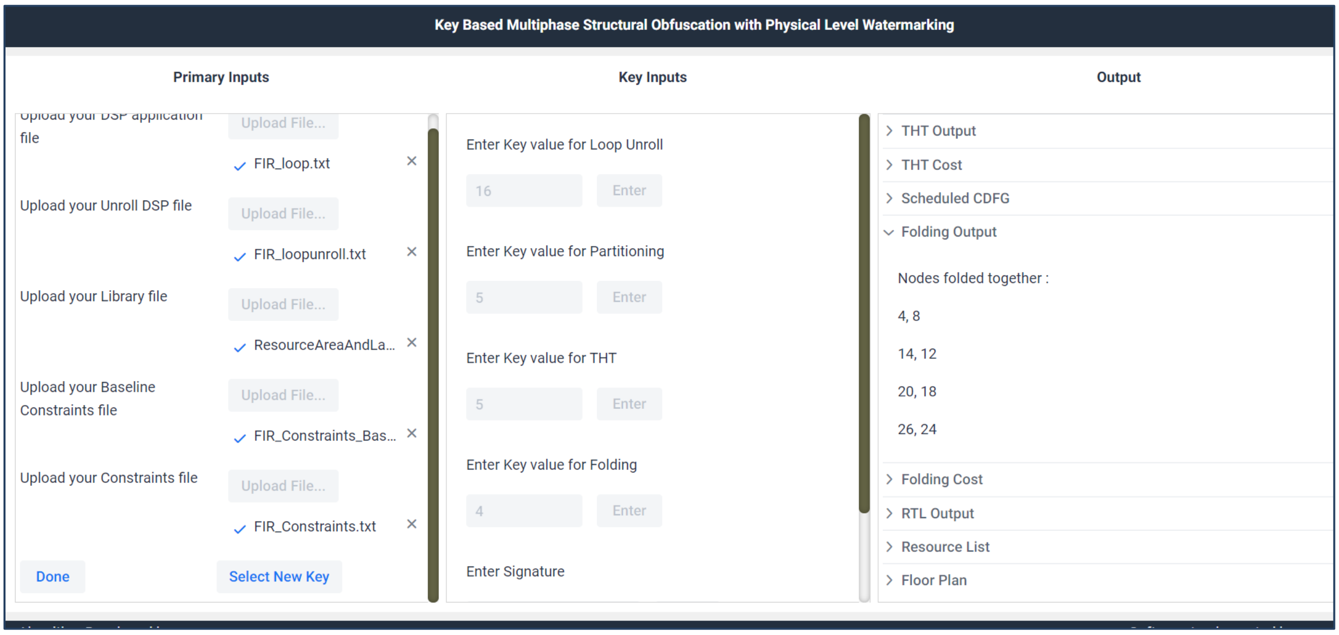
For M (multiplier): Input1 operation#, input2 operation #, (output operation #)

(‘0’ denotes ‘M’ gets primary input i.e. no intermediate input from an output of an operation)



```
Control Step : 14
+,22,26,24
Control Step : 15
+,24,27,28
Control Step : 16
+,29,31,32
Control Step : 17
+,28,32,30
```

Fig. 32. *Output of scheduling shown onto the tool; excerpt-3: control steps 14 to 17*



```
Nodes folded together :
4, 8
14, 12
20, 18
26, 24
```

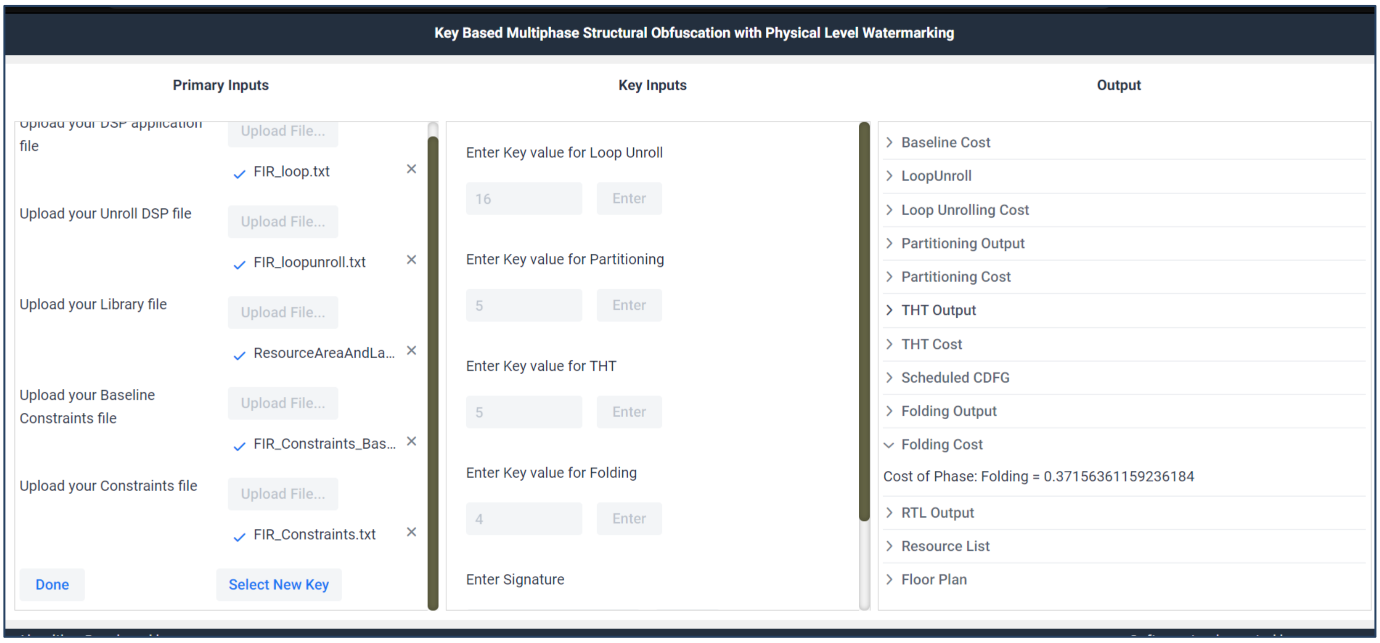
Fig. 33. *Nodes/operations subjected to folding shown onto the tool*

For A (adder): Input component 1, input component 2, output component

For Mux-x (4:1): 1st, 2nd, 3rd and 4th input of multiplexer respectively accepting input from operation #

For Demux-d (1:4): 1st, 2nd, 3rd and 4th output of demultiplexer respectively providing input to operation #

For C1 (comparator): Input operation#, output operation #


Key Based Multiphase Structural Obfuscation with Physical Level Watermarking

Primary Inputs | Key Inputs | Output

Upload your DSP application file — Upload File... — ✓ FIR_loop.txt ×
Upload your Unroll DSP file — Upload File... — ✓ FIR_loopunroll.txt ×
Upload your Library file — Upload File... — ✓ ResourceAreaAndLa... ×
Upload your Baseline Constraints file — Upload File... — ✓ FIR_Constraints_Bas... ×
Upload your Constraints file — Upload File... — ✓ FIR_Constraints.txt ×
Done — Select New Key

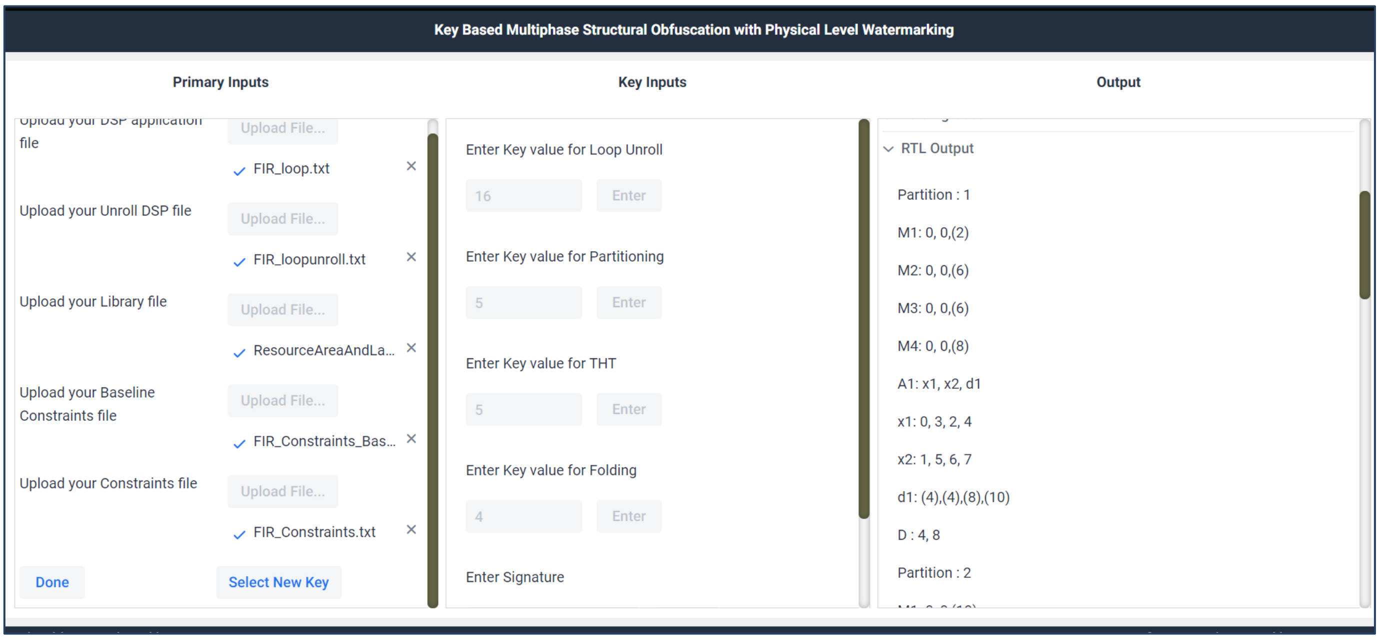
Enter Key value for Loop Unroll: 16 — Enter
Enter Key value for Partitioning: 5 — Enter
Enter Key value for THT: 5 — Enter
Enter Key value for Folding: 4 — Enter
Enter Signature

- Baseline Cost
- LoopUnroll
- Loop Unrolling Cost
- Partitioning Output
- Partitioning Cost
- THT Output
- THT Cost
- Scheduled CDFG
- Folding Output
- Folding Cost

Cost of Phase: Folding = 0.37156361159236184

- RTL Output
- Resource List
- Floor Plan

Fig. 34. *Design cost post folding shown onto the tool*


Key Based Multiphase Structural Obfuscation with Physical Level Watermarking

Primary Inputs | Key Inputs | Output

Upload your DSP application file — Upload File... — ✓ FIR_loop.txt ×
Upload your Unroll DSP file — Upload File... — ✓ FIR_loopunroll.txt ×
Upload your Library file — Upload File... — ✓ ResourceAreaAndLa... ×
Upload your Baseline Constraints file — Upload File... — ✓ FIR_Constraints_Bas... ×
Upload your Constraints file — Upload File... — ✓ FIR_Constraints.txt ×
Done — Select New Key

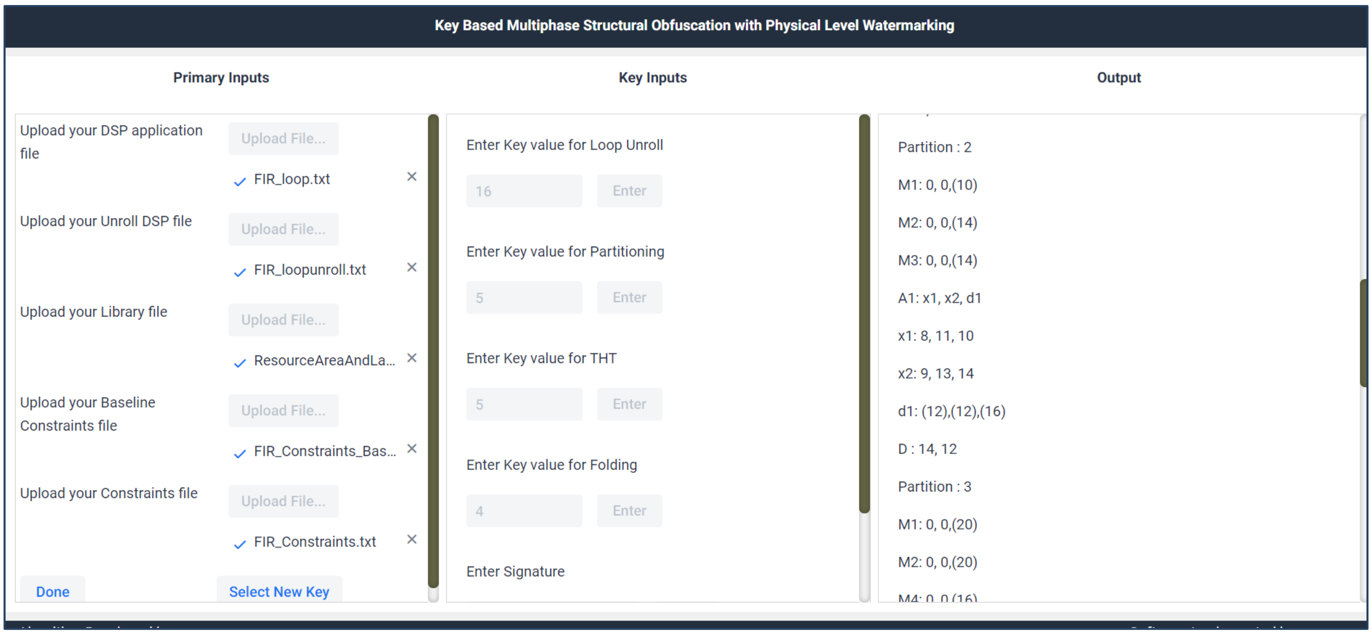
Enter Key value for Loop Unroll: 16 — Enter
Enter Key value for Partitioning: 5 — Enter
Enter Key value for THT: 5 — Enter
Enter Key value for Folding: 4 — Enter
Enter Signature

RTL Output

```
Partition : 1
M1: 0, 0,(2)
M2: 0, 0,(6)
M3: 0, 0,(6)
M4: 0, 0,(8)
A1: x1, x2, d1
x1: 0, 3, 2, 4
x2: 1, 5, 6, 7
d1: (4),(4),(8),(10)
D : 4, 8
Partition : 2
```

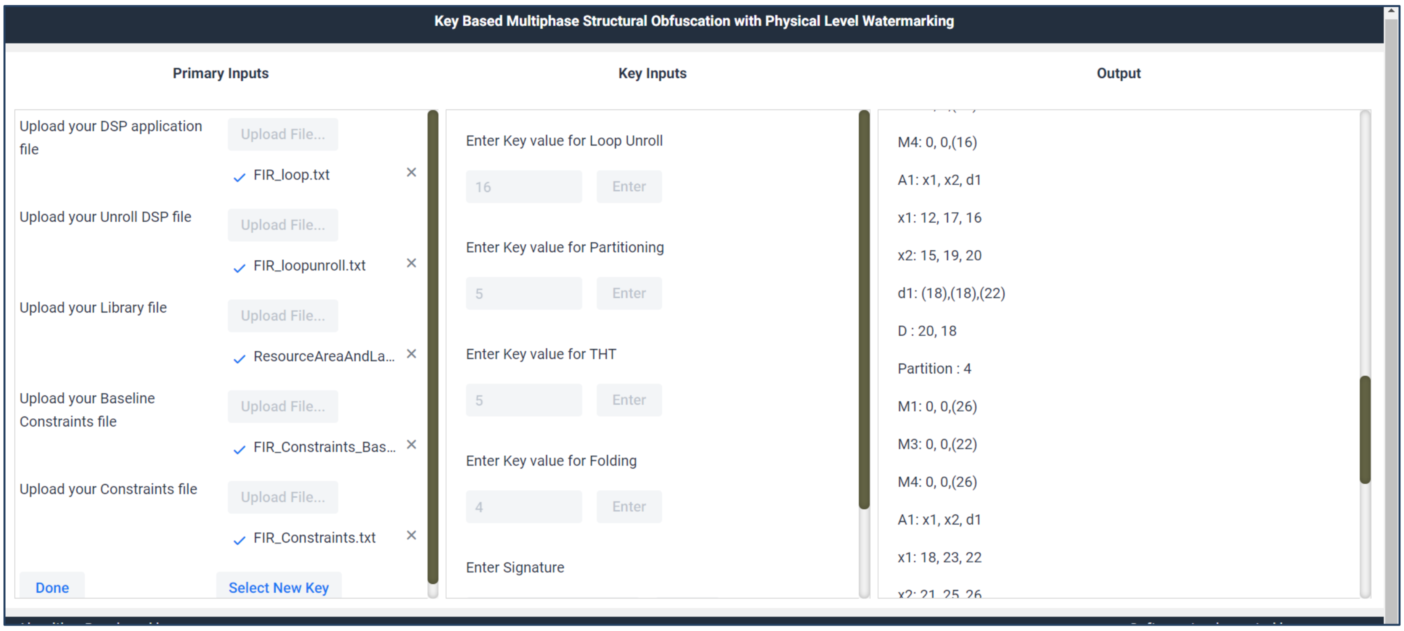
Fig. 35. *RTL output (of partition 1) of obfuscated FIR filter design shown onto the tool*

Further, the set of RTL components extracted from obfuscated RTL datapath of FIR filter for preparing an early floorplan is shown in Fig. 39. As shown in the figure, there are four instances of multiplier (i.e. M->1, M->2, M->3 and M->4), one instance of Demux 1:8 (i.e. d8->1), two instances of Mux 8:1 (i.e. x8->1, x8->2), one instance of comparator (i.e. C ->1), nine instances of Demux 1:4 (i.e. d4 ->1, d4 ->2, d4 ->3, d4 ->4, d4 ->5, d4 ->6, d4 ->7, d4 ->8, d4 ->9), eighteen instances of Mux 4:1 (i.e. x4 ->1, x4 ->2, x4 ->3, x4 ->4, x4 ->5, x4 ->6, x4 ->7, x4 ->8, x4 ->9, x4 ->10, x4 ->11, x4 ->12, x4 ->13, x4 ->14, x4 ->15, x4 ->16,


Key Based Multiphase Structural Obfuscation with Physical Level Watermarking

Primary Inputs		Key Inputs	Output
Upload your DSP application file	Upload File... ✓ FIR_loop.txt	Enter Key value for Loop Unroll: 16	Partition : 2
Upload your Unroll DSP file	Upload File... ✓ FIR_loopunroll.txt	Enter Key value for Partitioning: 5	M1: 0, 0,(10)
Upload your Library file	Upload File... ✓ ResourceAreaAndLa...	Enter Key value for THT: 5	M2: 0, 0,(14)
Upload your Baseline Constraints file	Upload File... ✓ FIR_Constraints_Bas...	Enter Key value for Folding: 4	M3: 0, 0,(14)
Upload your Constraints file	Upload File... ✓ FIR_Constraints.txt	Enter Signature	A1: x1, x2, d1
			x1: 8, 11, 10
			x2: 9, 13, 14
			d1: (12),(12),(16)
			D : 14, 12
			Partition : 3
			M1: 0, 0,(20)
			M2: 0, 0,(20)

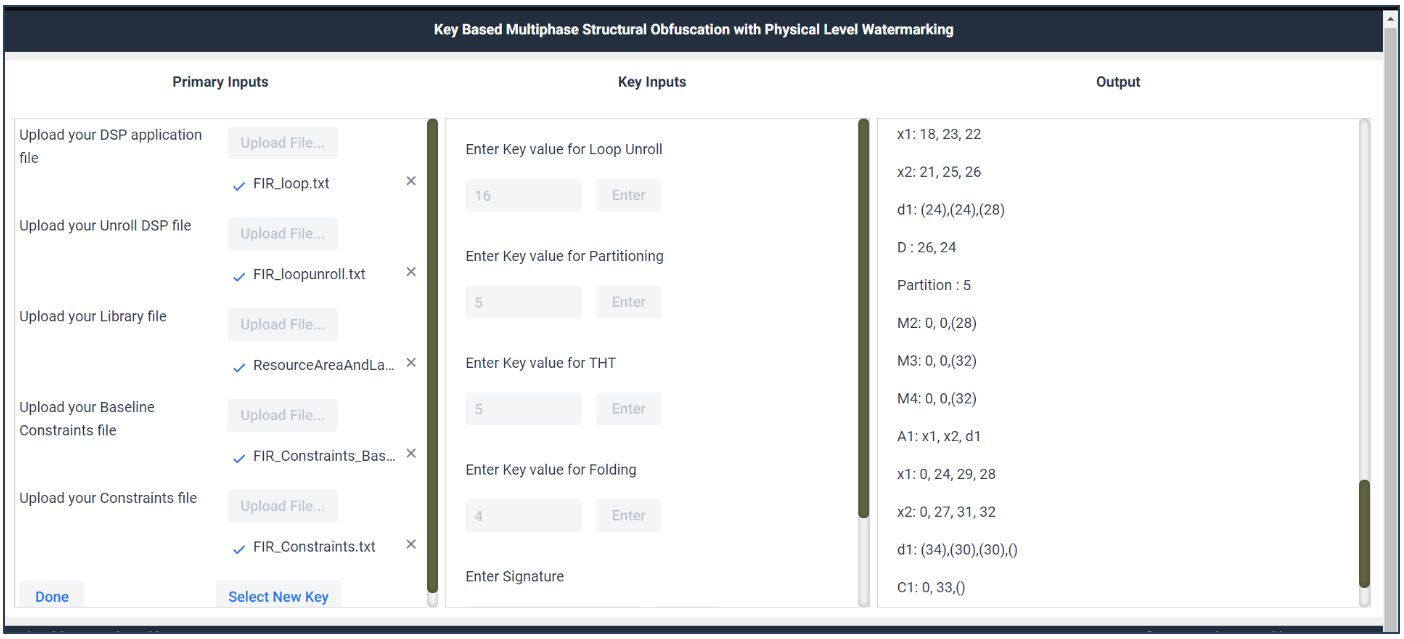
Fig. 36. *RTL output (of partition 2 and 3) of obfuscated FIR filter design shown onto the tool*


Key Based Multiphase Structural Obfuscation with Physical Level Watermarking

Primary Inputs		Key Inputs	Output
Upload your DSP application file	Upload File... ✓ FIR_loop.txt	Enter Key value for Loop Unroll: 16	M4: 0, 0,(16)
Upload your Unroll DSP file	Upload File... ✓ FIR_loopunroll.txt	Enter Key value for Partitioning: 5	A1: x1, x2, d1
Upload your Library file	Upload File... ✓ ResourceAreaAndLa...	Enter Key value for THT: 5	x1: 12, 17, 16
Upload your Baseline Constraints file	Upload File... ✓ FIR_Constraints_Bas...	Enter Key value for Folding: 4	x2: 15, 19, 20
Upload your Constraints file	Upload File... ✓ FIR_Constraints.txt	Enter Signature	d1: (18),(18),(22)
			D : 20, 18
			Partition : 4
			M1: 0, 0,(26)
			M3: 0, 0,(22)
			M4: 0, 0,(26)
			A1: x1, x2, d1
			x1: 18, 23, 22

Fig. 37. *RTL output (of partition 3 and 4) of obfuscated FIR filter design shown onto the tool*

x4 ->17, x4 ->18) and one instance of adder (A ->1). These RTL components and their instances have been arranged (as shown in Fig. 39) based on decreasing order of their size and increasing order of their instance number. An excerpt of floorplan is shown in Fig. 40, where orientations of components in block 8 and block 9 have been highlighted. To generate a watermarked floorplan, a six digit author's signature "αβαβγγ" is entered as shown in Fig. 40. On clicking on the "*View Final Floorplan*" button shown at output panel in Fig. 40, the final watermarked floorplan is generated in a new window. The final watermarked floorplan of FIR filter application is shown in Fig. 41. As shown in the figure, odd FU instance M->1 on the top of even FU instance M->4 shows the embedding of first α digit of the signature "αβαβγγ". Further, odd FU instance M->3 on the top of even FU instance M->2


Key Based Multiphase Structural Obfuscation with Physical Level Watermarking

Primary Inputs

Upload your DSP application file — Upload File... — ✓ FIR_loop.txt ×

Upload your Unroll DSP file — Upload File... — ✓ FIR_loopunroll.txt ×

Upload your Library file — Upload File... — ✓ ResourceAreaAndLa... ×

Upload your Baseline Constraints file — Upload File... — ✓ FIR_Constraints_Bas... ×

Upload your Constraints file — Upload File... — ✓ FIR_Constraints.txt ×

Done — Select New Key

Key Inputs

Enter Key value for Loop Unroll: 16 — Enter

Enter Key value for Partitioning: 5 — Enter

Enter Key value for THT: 5 — Enter

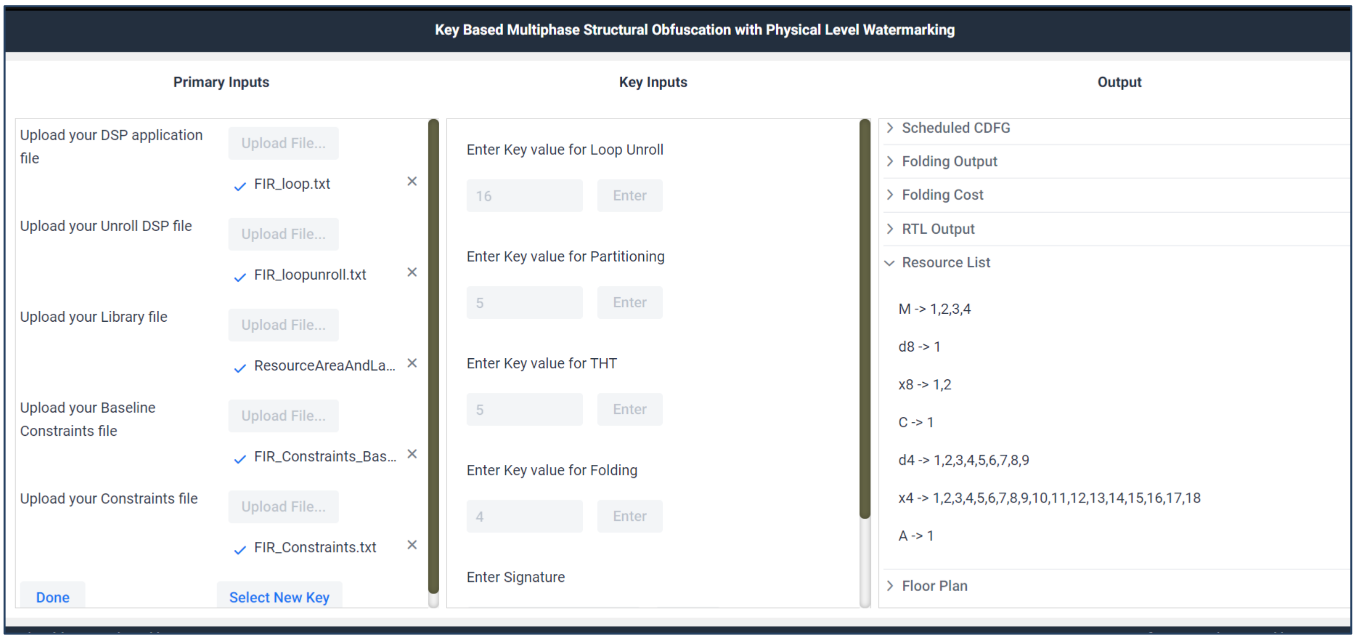
Enter Key value for Folding: 4 — Enter

Enter Signature

Output

```
x1: 18, 23, 22
x2: 21, 25, 26
d1: (24),(24),(28)
D : 26, 24
Partition : 5
M2: 0, 0,(28)
M3: 0, 0,(32)
M4: 0, 0,(32)
A1: x1, x2, d1
x1: 0, 24, 29, 28
x2: 0, 27, 31, 32
d1: (34),(30),(30),()
C1: 0, 33,()
```

Fig. 38. *RTL output (of partition 4 and 5) of obfuscated FIR design filter shown onto the tool*


Key Based Multiphase Structural Obfuscation with Physical Level Watermarking

Primary Inputs

Upload your DSP application file — Upload File... — ✓ FIR_loop.txt ×

Upload your Unroll DSP file — Upload File... — ✓ FIR_loopunroll.txt ×

Upload your Library file — Upload File... — ✓ ResourceAreaAndLa... ×

Upload your Baseline Constraints file — Upload File... — ✓ FIR_Constraints_Bas... ×

Upload your Constraints file — Upload File... — ✓ FIR_Constraints.txt ×

Done — Select New Key

Key Inputs

Enter Key value for Loop Unroll: 16 — Enter

Enter Key value for Partitioning: 5 — Enter

Enter Key value for THT: 5 — Enter

Enter Key value for Folding: 4 — Enter

Enter Signature

Output

- Scheduled CDFG
- Folding Output
- Folding Cost
- RTL Output
- Resource List

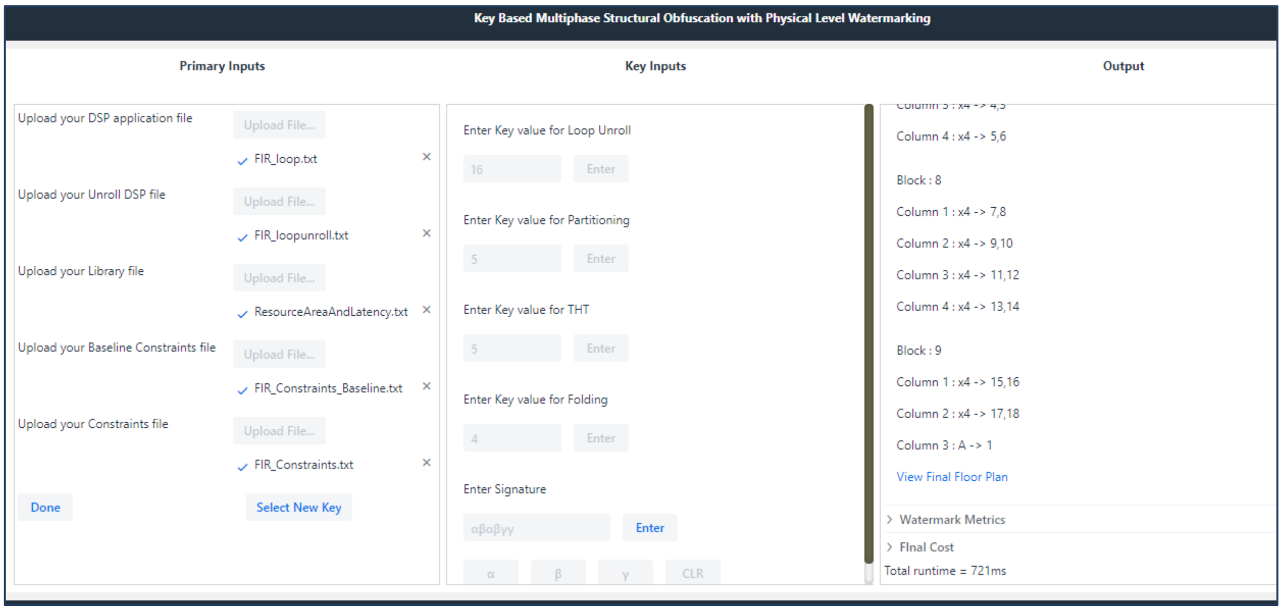
```
M -> 1,2,3,4
d8 -> 1
x8 -> 1,2
C -> 1
d4 -> 1,2,3,4,5,6,7,8,9
x4 -> 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18
A -> 1
```

- Floor Plan

Fig. 39. *Extracted RTL components (resource list) shown on to the tool*

shows the embedding of second α digit. Similarly, odd Mux instance x4->1 on the top of even Mux instance x4->2 shows the embedding of first β digit and odd Mux instance x4->3 on the top of even Mux instance x4->4 shows the embedding of second β digit. Further, odd Demux instance d4->1 at the right to even Demux instance d4->4 shows the embedding of first γ digit and odd Demux instance d4->3 at the right to even Demux d4->2 shows the embedding of second γ digit.

Thus the structural obfuscation and physical level watermarking can be simulated and analysed using the KSO-PW tool developed by the authors. This tool is useful for various kind of DSP hardware accelerator applications such as finite impulse


Key Based Multiphase Structural Obfuscation with Physical Level Watermarking

Primary Inputs

Upload your DSP application file — Upload File... — ✓ FIR_loop.txt ×

Upload your Unroll DSP file — Upload File... — ✓ FIR_loopunroll.txt ×

Upload your Library file — Upload File... — ✓ ResourceAreaAndLatency.txt ×

Upload your Baseline Constraints file — Upload File... — ✓ FIR_Constraints_Baseline.txt ×

Upload your Constraints file — Upload File... — ✓ FIR_Constraints.txt ×

Done — Select New Key

Key Inputs

Enter Key value for Loop Unroll: 16 — Enter

Enter Key value for Partitioning: 5 — Enter

Enter Key value for THT: 5 — Enter

Enter Key value for Folding: 4 — Enter

Enter Signature: αβαβγγ — Enter

α β γ CLR

Output

```
Column 3 : x4 -> 4,5
Column 4 : x4 -> 5,6

Block : 8
Column 1 : x4 -> 7,8
Column 2 : x4 -> 9,10
Column 3 : x4 -> 11,12
Column 4 : x4 -> 13,14

Block : 9
Column 1 : x4 -> 15,16
Column 2 : x4 -> 17,18
Column 3 : A -> 1
```

View Final Floor Plan

> Watermark Metrics

> Final Cost

Total runtime = 721ms

Fig. 40. *Feeding of author's signature into the tool to generate watermarked floorplan*

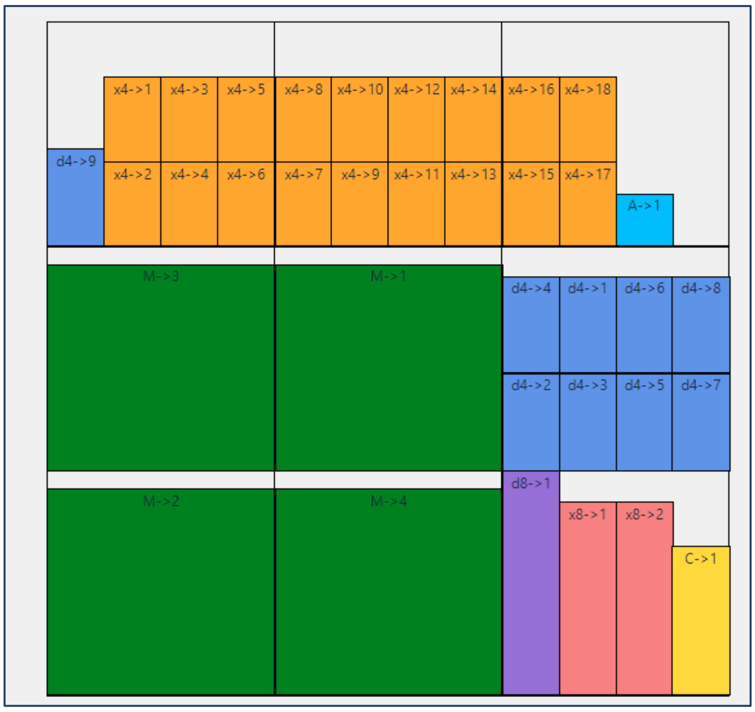


Fig. 41. *Final watermarked floorplan displayed by the tool*

response (FIR) filter, infinite impulse response (IIR) filter, discrete cosine transform (DCT), auto-regression filter (ARF) etc. In addition, the tool evaluates and shows the design cost pre and post performing double line of defence for hardware security.

4.8. Analysis of Case Studies

The double line of defence approach, proposed by (Sengupta and Rathor, 2020), offers security to DSP hardware accelerators without incurring any design cost overhead. This section discusses the security and design cost analysis of double line of defence approach based on multi-key driven structural obfuscation and physical level watermarking. The security analysis due to structural obfuscation has been discussed in terms of difference in gate count (that creates obscurity) incurred due to obfuscation and SO-key size. Further, security analysis due to physical level watermarking has been discussed in terms of probability of coincidence metric, tamper tolerance ability and brute-force attack analysis. Further, an analysis of total key size due to multi-key driven structural obfuscation and physical level watermarking has been discussed. Furthermore, the security and design cost analysis for low cost optimized multi-key based structural obfuscation have been discussed in a separate sub-section. This case study of security and design cost analysis for various DSP applications gives a deeper insight about the robustness of multi-key based structural obfuscation and physical level watermarking based double line of defence approach and its impact on overall design cost. The discussions are presented as follows:

1. Analysis of Case Studies for Double line of Defence - Structural Obfuscation and Physical level Watermarking

1.1. Security Analysis

The multiple SO-keys based structural obfuscation acts as a first line of defence and physical level watermarking acts as a second line of defence to secure DSP hardware accelerators against RE, Trojan insertion and piracy threats. The security achieved using both line of defence has been discussed separately as follows (Sengupta and Rathor, 2020):

(a) Security analysis of multi-key based structural obfuscation

As discussed earlier, the multiple SO-keys based structural obfuscation provides security to DSP hardware accelerators in the form of first line of defence. Here, the structural obfuscation based first line of defence acts as preventive control against Trojan (malicious logic) insertion and piracy threats. The preventive control is enabled because the employed multiple SO-keys based structural obfuscation incurs massive obscurity into the design structure, hence rendering the interpretation of true functionality and structure of the design highly complicated for an attacker. Therefore an attacker, who aims to RE to infect the design using Trojan insertion or aims to pirate the design, fails/gets hindered and hence becomes unable to realize his malicious intents. The robustness of structural obfuscation due to employing multiple structural transformations is measured in terms of difference in gate count pre and post-obfuscation. Fig. 42 highlights the difference in gate count pre and post employing structural obfuscation. The huge difference in gate count (that creates obscurity) shown in Fig. 42 indicates the robustness of multiple SO-keys based structural obfuscation technique. The change in gate count (without affecting functionality) incurred due to structural obfuscation is an indication of robustness of obfuscation because of following reasons: (i) the change in gate count post obfuscation does not follow any particular pattern (for inferring the original structure/functionality), but rather depends on the obfuscation techniques employed and designer's chosen SO-keys (ii) the change in gate count not only makes a difference in the count but also results into alterations in the gates connectivity, hence leads to an obfuscated netlist.

As shown in Fig. 42, the amount of gates changed due to obfuscation depends on the type and size of the application (as evident from the figure, there is no fixed pattern in the gate count difference for different DSP application). This is because different applications have different operation count and data dependency. Thus the nature of the applications determines whether a particular type of obfuscation technique (such as loop unrolling, partitioning, ROE, THT and folding knob) is applicable or not and further to what extent it is applicable. Thereby for different DSP applications, the applied structural transformation techniques of structural obfuscation creates different kind of modifications in the resource interconnectivity, the number of resources, size and count of the Muxes/Demuxes and storage elements, which in turn modifies the overall gate count (creating obscurity while preserving functionality). For example, the gate count of IIR and ARF filters reduces post applying multiple SO-keys based structural obfuscation. The reason is that the applied obfuscation techniques do not result into larger size multiplexers and demultiplexers with respect to the non-obfuscated (baseline) counterpart. Therefore, the gate count post obfuscation reduces; whereas the gate count of FIR and DCT core is augmented post-obfuscation as shown in Fig. 42 (Sengupta and Rathor, 2020).

In addition, the multiple secret SO-keys used to regulate the process of structural obfuscation play a vital role in enhancing the robustness of obfuscation. The reasons are as follows: (i) each structural transformation technique is regulated using a

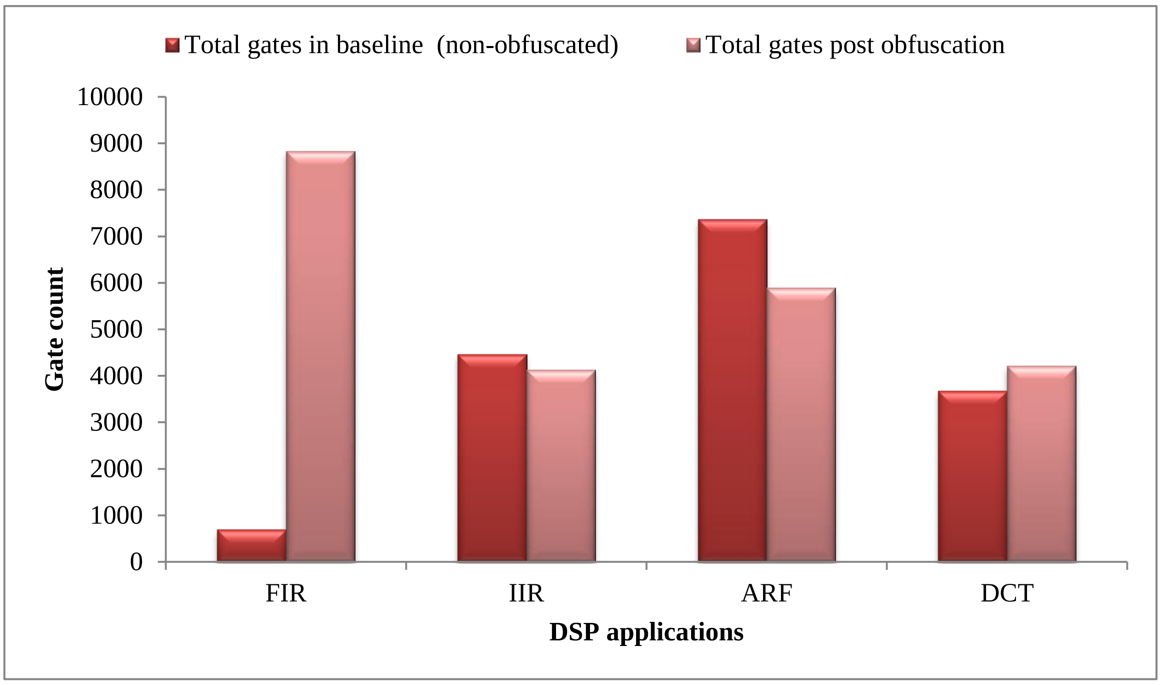


Fig. 42. *Strength of structural obfuscation in terms of difference in gate count*

specific secret key value which decides the extent to which the structural transformation has to be performed (ii) only being aware of the applied transformations cannot help an adversary in performing RE. An attacker needs to aware of both the applied structural transformation techniques and the secret SO-keys used (iii) each individual secret SO-key contributes to augment the total size (space) of key, hence it becomes challenging to find an exact correct key among exhaustive possibilities. Therefore, the incorporation of secret SO-keys in the structural obfuscation process enhances the security level manifold. Fig. 43 (a) highlights the total SO-key size for different DSP applications (Sengupta and Rathor, 2020).

(b) Security analysis of physical level watermarking

As discussed earlier, the physical level watermarking provides security to DSP hardware accelerators in the form of second line of defence. Here, the physical level watermarking based second line of defence acts as detective control against counterfeiting and cloning threats. The detective control is enabled by embedding vendor's secret signature into the early floorplan of the physical design process. The embedded watermark is detected to identify counterfeited and cloned designs. The robustness of the watermark embedded at physical level has been analysed by (Sengupta and Rathor, 2020) in terms of the following: (a) probability of coincidence (b) tamper tolerance (c) brute-force analysis (d) key bits required to represent the total space of vendor's secret signature. Let us see the discussion on each analysis one by one.

The probability of coincidence (Pc) metric is measured as follows (Sengupta and Rathor, 2020):

$$Pc = \left(\prod_{i=1}^{\alpha} \frac{1}{\{\sum_{k1\in F_p} \frac{k1(k1-1)}{2}\}-x++}\right) * \left(\prod_{j=1}^{\beta} \frac{1}{\{\sum_{k2\in X_q} \frac{k2(k2-1)}{2}\}-y++}\right) * \left(\prod_{k=1}^{\gamma} \frac{1}{\{\sum_{k3\in D_r} \frac{k3(k3-1)}{2}\}-z++}\right) \tag{12}$$

Where, 'k1' denotes total number of FU resource components of type F_p, where p denotes the total types of FU resources; 'k2' denotes number of multiplexers of size X_q, where q denotes different sizes of multiplexers in the design; 'k3' denotes number of demultiplexers of size D_r, where r denotes different sizes of demultiplexers in the design. The range of variables x, y and z are as follows: $0\leq x\leq \alpha-1$, $0\leq y\leq \beta-1$, $0\leq z\leq \gamma-1$, where x, y and z variables are incremented with the embedding of each digit of signature variables α, β and γ respectively. The interpretation of individual terms in the formula is as follows:

$\{\sum_{k1\in F_p} \frac{k1(k1-1)}{2}\}$ indicates all swapping pairs corresponding to all types of FU resource components in the set extracted from obfuscated RTL design (i.e. swapping pairs of all multiplier instances+ swapping pairs of all adder instances + swapping pairs of all instances of p th FU resource).

$\{\sum_{k1\in F_p} \frac{k1(k1-1)}{2}\} - x++$ indicates **remaining** swapping pairs after embedding an α digit.

$\frac{1}{\{\sum_{k1\in F_p} \frac{k1(k1-1)}{2}\}-x++}$ indicates probability of obtaining/detecting one pair of FU modules corresponding to an α digit.

$\left(\prod_{i=1}^{\alpha}\frac{1}{\{\sum_{k1\in F_p}\frac{k1(k1-1)}{2}\}-x++}\right)$ indicates probability of obtaining/detecting all pairs of FU modules corresponding to all embedded α digits in a non-watermarked design by an attacker.

$\{\sum_{k2\in X_q}\frac{k2(k2-1)}{2}\}$ indicates all swapping pairs corresponding to all sizes of Mux components in the set extracted from obfuscated RTL design (i.e. swapping pairs of all instances of Mux of one size + swapping pairs of all instances of Mux of next size + swapping pairs of all instances of Mux of qth size).

$\{\sum_{k2\in X_q}\frac{k2(k2-1)}{2}\}-y++$ indicates **remaining** swapping pairs after embedding an β digit.

$\frac{1}{\{\sum_{k2\in X_q}\frac{k2(k2-1)}{2}\}-y++}$ indicates probability of obtaining/detecting one pair of Mux modules corresponding to an β digit.

$\left(\prod_{j=1}^{\beta}\frac{1}{\{\sum_{k2\in X_q}\frac{k2(k2-1)}{2}\}-y++}\right)$ indicates probability of obtaining/detecting all pairs of Mux modules corresponding to all embedded β digits in a non-watermarked design by an attacker.

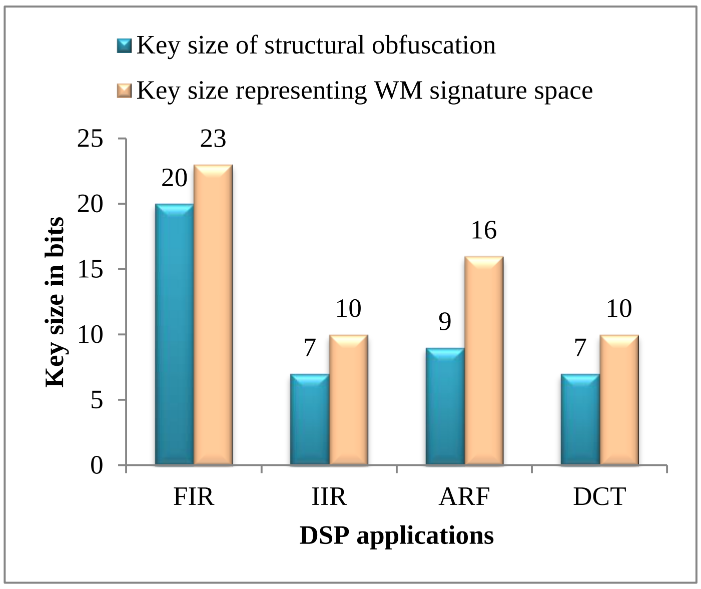
$\{\sum_{k3\in D_r}\frac{k3(k3-1)}{2}\}$ indicates all swapping pairs corresponding to all sizes of Demux components in the set extracted from obfuscated RTL design (i.e. swapping pairs of all instances of Demux of one size + swapping pairs of all instances of Demux of next size + swapping pairs of all instances of Demux of rth size).

$\{\sum_{k3\in D_r}\frac{k3(k3-1)}{2}\}-z++$ indicates **remaining** swapping pairs after embedding an γ digit.

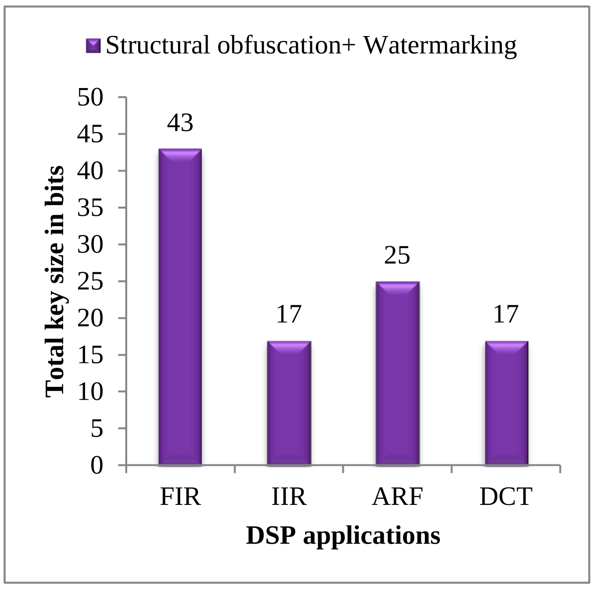
$\frac{1}{\{\sum_{k3\in D_r}\frac{k3(k3-1)}{2}\}-z++}$ indicates probability of obtaining/detecting one pair of Demux modules corresponding to an γ digit.

$\left(\prod_{k=1}^{\gamma}\frac{1}{\{\sum_{k3\in D_r}\frac{k3(k3-1)}{2}\}-z++}\right)$ indicates probability of obtaining/detecting all pairs of Demux modules corresponding to all embedded γ digits in a non-watermarked design by an attacker.

The probability of coincidence captures the probability of coincidently finding same signature in a non-watermarked design. If the probability of coincidence value is high, the watermarked is considered weak. Therefore lower Pc value is desirable, which indicates that large amount of digital evidence is embedded into the design, hence indicating high robustness of watermark. Fig. 44 shows the probability of coincidence obtained using physical level watermarking. The figure shows that significantly lower Pc is achieved for all DSP applications. The reason of obtaining lower Pc is that the signature digits corresponding to the multiple variables (i.e. α, β and γ) have been embedded using different types and size of RTL components in the early floorplan stage. Thus a stronger detective control based security is achieved by embedding a robust physical level watermark (Sengupta and Rathor, 2020).



(a)



(b)

Fig. 43. *Key size analysis (a) key size of structural obfuscation and key size of watermarking (b) total key size of double line of defence approach (Sengupta and Rathor, 2020)*

The tamper tolerance capability of embedded physical level watermark is measured in terms of total signature combinations representing signature space of watermark. Because of embedding watermark corresponding to multiple signature variables (α, β and γ), the signature space of watermark is significantly high. Therefore, the attacker's effort of finding correct signature, to eliminate the signature digits by tampering watermarking constraints, becomes significantly high. Hence ability to tolerate the tampering, caused by the adversary, is high. The formula to estimate the tamper tolerance capability (T^P) is as follows (Sengupta and Rathor, 2020):

$$T^P = Z^Q \tag{13}$$

Where, Z represents the number of signature variables used in the watermark and Q represents the size of the vendor's signature. The value of Z^Q represents the signature space of watermark (i.e. total possible combinations of signature), which in turn shows the tamper tolerance capability of the watermark. Fig. 45(a) depicts the tamper tolerance capability (using eq. (13)) of the physical level watermark for vendor's chosen signature strength. As shown in the figure, very high value of tamper tolerance is achieved. This indicates that the physical level watermark proposed by (Sengupta and Rathor, 2020) is strong against tampering.

Further, the security against removal attack is ensured using brute-force attack analysis of the signature. The security against removal attack on signature using brute force analysis in measured in terms of probability of finding the valid signature within exhaustive signature combinations (signature space). Hence the security is measured using following formula (Sengupta and Rathor, 2020):

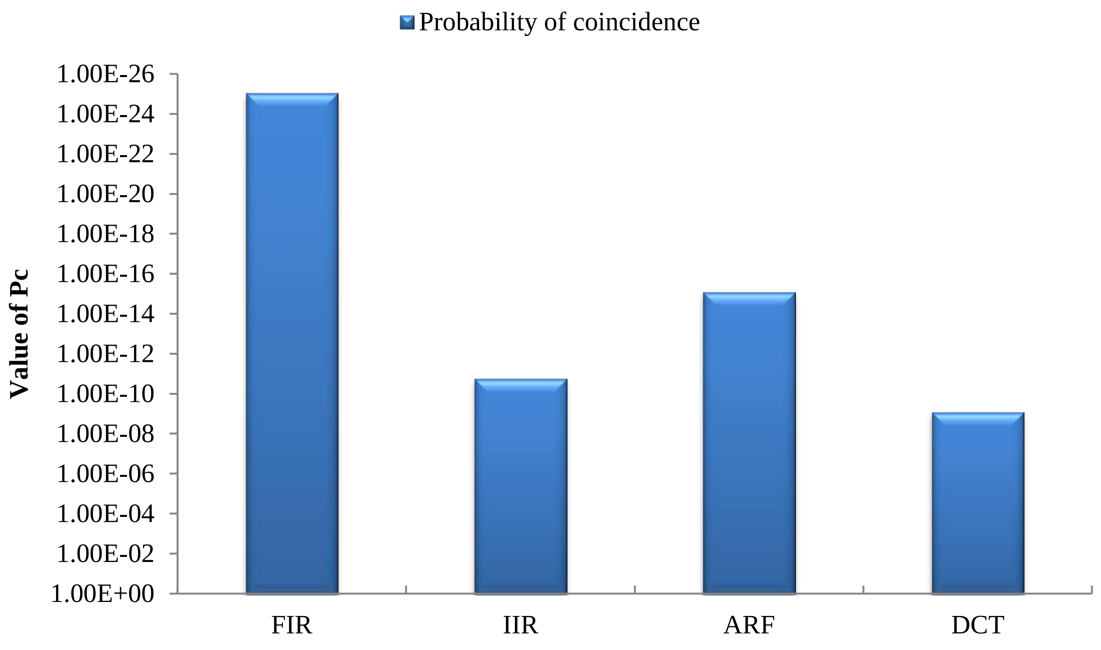


Fig. 44. *Robustness of physical level watermarking in terms of Pc (Sengupta and Rathor, 2020)*

$$S^B=1/Z^Q \qquad (14)$$

Where, S^B indicates the probability of finding correct signature by an attacker using brute force analysis and Z^Q represents the signature space of watermark. Lower the value of S^B, higher is the security against removal attack on signature. Fig. 45(b) depicts the brute force analysis using the security metric given in eq. (14). As shown in the figure, very low probability of finding correct signature using brute force analysis is achieved. Hence, it indicates the strong security against removal attack

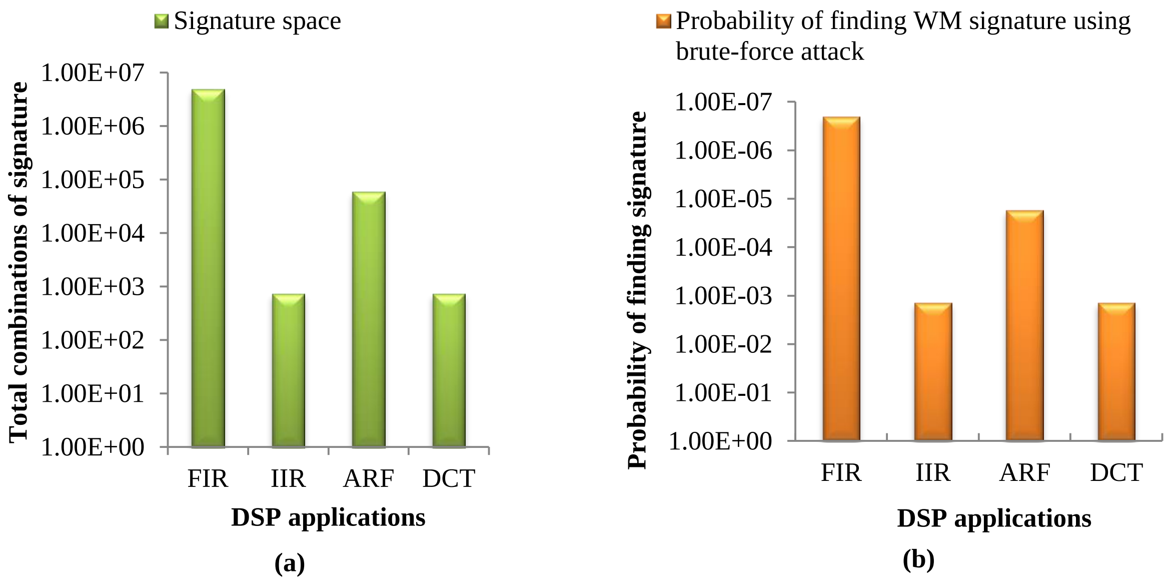


Fig. 45. *Security analysis of physical level watermarking in terms of (a) Tamper tolerance analysis (b) Brute force attack analysis (Sengupta and Rathor, 2020)*

on signature by an attacker (Sengupta and Rathor, 2020).

Further, the number of bits required to represent the total space of the signature embedded during physical level watermarking is calculated using ⌈(log$_2$ (Z^Q)⌉. This formula gives the key size in bits which captures the signature space of watermark. Fig. 43 (a) shows the key size required to represent the signature space. Further, Fig. 43(b) shows the total key size of the double line of defence approach which sums up the key size of structural obfuscation and the required key size to capture the whole signature space of the physical level watermark. It indicates the hardship of an attacker in terms of finding correct key of structural obfuscation and the valid signature of physical level watermark (Sengupta and Rathor, 2020).

1.2. Design Cost Analysis

It is important to ensure that the employed security mechanism should not incur excessive design overhead. A security mechanism that results into minimal or zero design overhead is considered effective and practical. Therefore, the design cost of the double line of defence approach needs to be analysed. Following equation is used to evaluate the design cost (Sengupta and Rathor, 2020):

$$C_d(U_i) = \rho_1 \frac{L_d}{L_m} + \rho_2 \frac{A_d}{A_m} \qquad (15)$$

Where, $C_d(U_i)$ is the design cost calculated 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. The analysis of design cost post-employing each line of defence is discussed as follows:

(a) Design cost analysis of multi-key based structural obfuscation

For evaluation of the design cost of multi-keys based structural obfuscation, the design area and latency are calculated using 15 nm NanGate library (Sengupta *et al*., 2020). The calculation of deign area is based on area of resources in the obfuscated design and calculation of latency is based on the scheduling of structurally obfuscated design (Sengupta *et al*., 2020). Fig. 46 compares the design cost post-employing structural obfuscation with respect to the baseline (un-obfuscated) counterpart. As shown in the figure, the obfuscation mechanism incurs zero design overhead for most of the applications. This is because the applied high level transformations during structurally obfuscating the design, also results into a sort of optimization in the structure as by-product. As shown in Fig. 46, the design cost of FIR filter application reduces significantly. The reason is the applicability of loop unrolling based structural transformation which causes parallelism of operations. This leads to substantial reduction in the design latency, hence reducing the overall design cost. Further, design cost of some applications

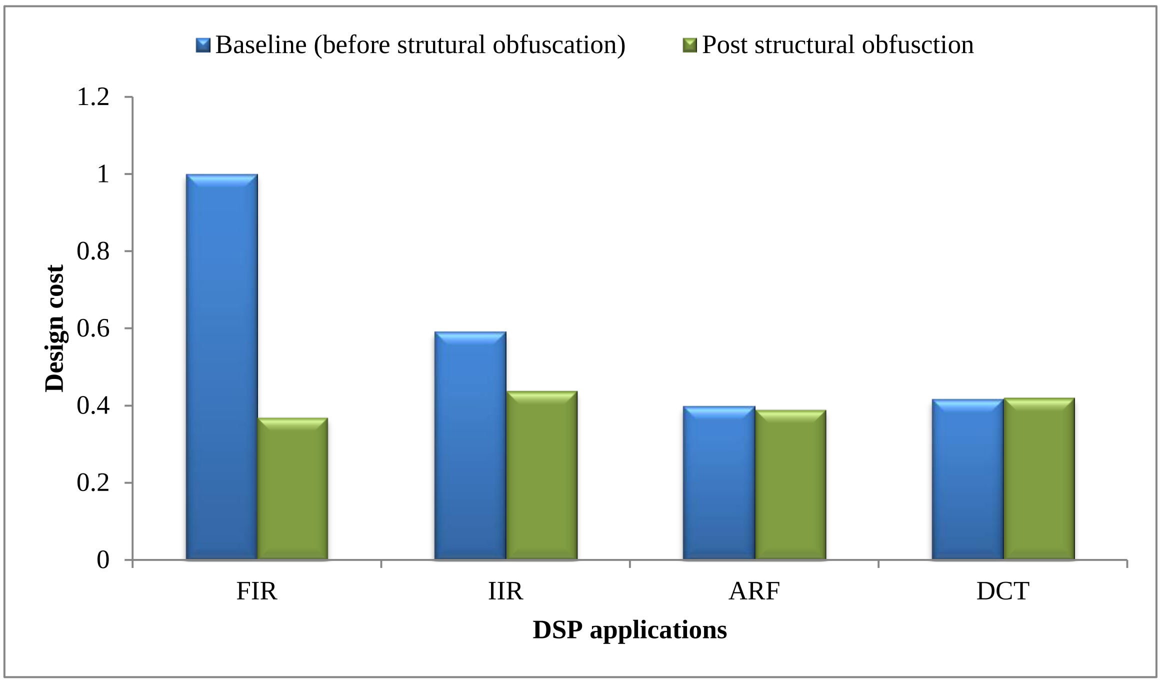


Fig. 46. *Design cost comparison pre and post structural obfuscation with respect to baseline (Sengupta et al., 2020)*

(such as DCT) slightly increases post-obfuscation. This is due to the nature of target application and applicability of different obfuscation techniques affecting the Mux/Demux size and their count.

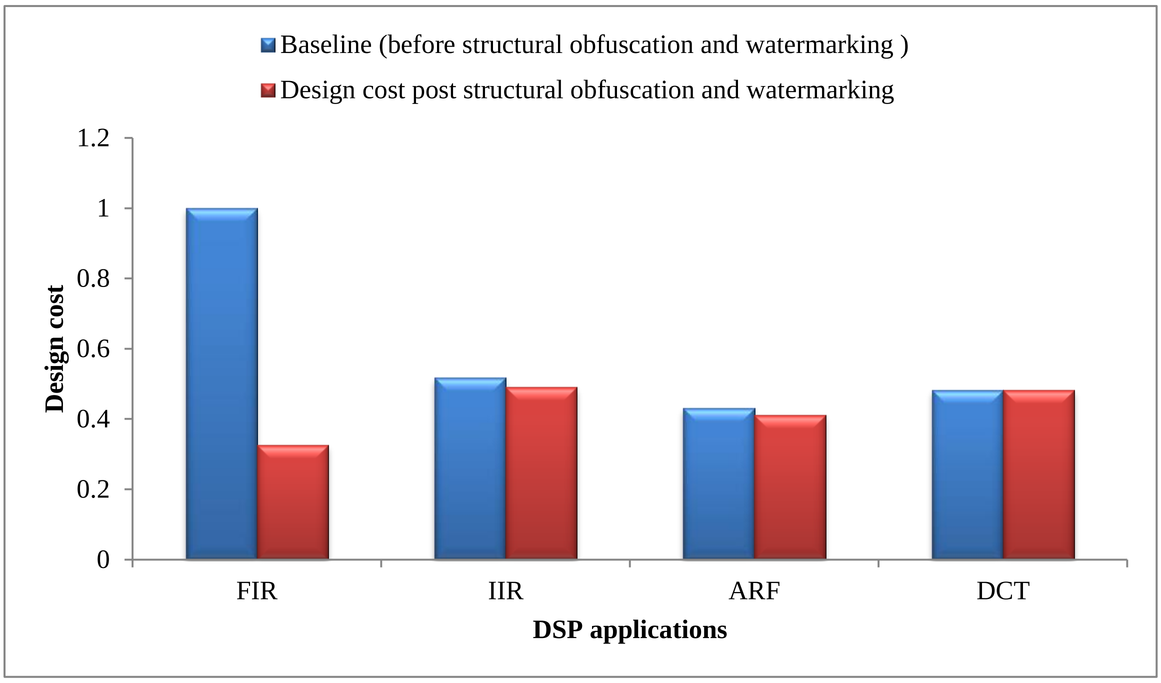


Fig. 47. *Design cost comparison pre and post structural obfuscation and physical level watermarking with respect to baseline (Sengupta and Rathor, 2020)*

(b) Design cost analysis of structural obfuscation and physical-level watermarking

For evaluation of design cost post-structural obfuscation and physical level watermarking, the design area is measured in terms of the area of the enveloping rectangle of the floorplan design (Sengupta and Rathor, 2020). Whereas, scheduling of design is exploited for determining the latency. The comparison of design cost of structurally obfuscated watermarked design with the baseline (un-obfuscated) design is shown in Fig. 47. As shown in the figure, the structural obfuscation and physical level watermarking based double line of defence mechanism incurs zero design cost overhead. Moreover, the design cost post employing double line of defence reduces because of reduction in either design latency or floorplan area post employing structural obfuscation. The impact of physical level watermarking on design cost is nil. This is because; the physical level watermark has been embedded into the floorplan by swapping the RTL components of same type and same size. Therefore, no design cost overhead incurs. Let's analyse the case study on FIR filter application. The cost of the obfuscated FIR filter design is significantly lesser than the baseline design. The underlying reason is the substantial reduction in the latency post-employing structural obfuscation. The reduction in latency is achieved because of key driven unrolling factor based loop unrolling transformation which causes more parallelization of the operations (due to duplicate iterations of loop body) during scheduling, hence resulting into lesser delay. This kind of parallelization of operations is absent in the baseline (un-obfuscated) FIR filter design, therefore it has more delay and hence larger design cost than the obfuscated version. Further for other DSP applications shown in Fig. 47, loop unrolling based structural transformation is not applicable. Therefore the design latency does not change post-structural obfuscation. However, the structural obfuscation results into a slight

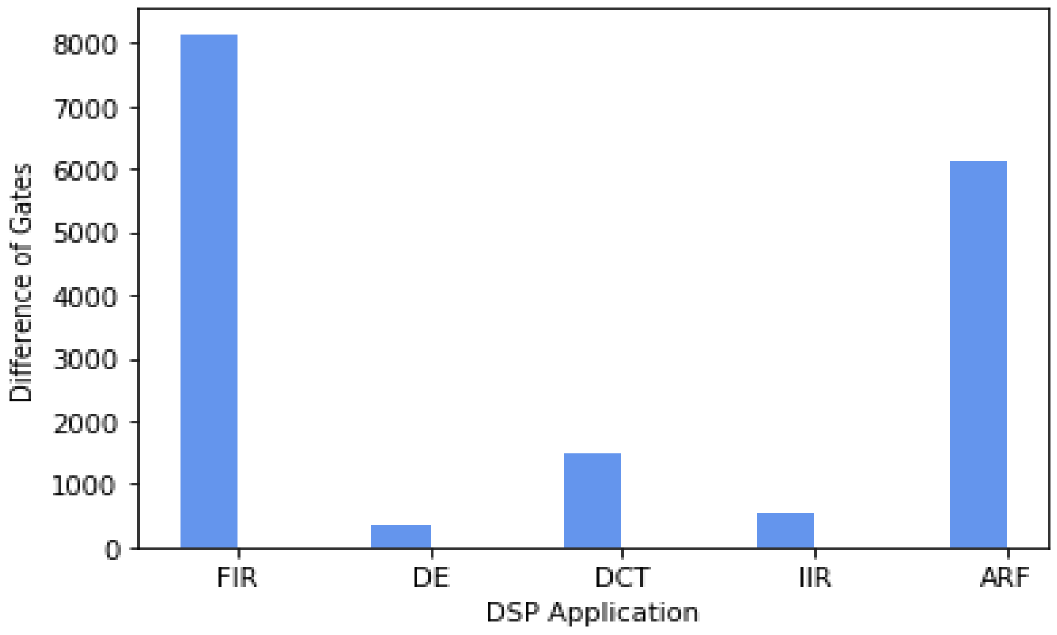


Fig. 48. *Security of multi-key based structural obfuscation in terms of number of gates affected*

decrement in the area of enveloping rectangle of the structurally obfuscated floorplan. The area is reduced because of reduction in the sizes of Muxes and Demuxes post-obfuscation. In general, the type and size of DSP applications and the applicability of structural transformation together determine the increment/decrement in the interconnect hardware resources (size and number of the Muxes and Demuxes), storage resources and FU resources (Sengupta and Rathor, 2020).

2. *Analysis of Case Studies for Low Cost Optimized Multi-key based Structural Obfuscation*

The analysis of the low-cost multi-key based structural obfuscation process (presented in section 4.6) has been discussed, for various DSP applications, in terms of design cost and security.

2.1. Security Analysis

The multi-key based structural obfuscation approach secures a DSP design against RE by obfuscating it in terms of structural transformation resulting into affecting larger amount of gates structurally (while preserving functionality), such that it becomes un-obvious to interpret to an attacker or outsider. The gates are affected in terms of change in gates interconnectivity and change in total gate count post-obfuscation. Fig. 48 depicts the number of gates transformed (change in the gate count) due to this obfuscation process, with respect to its equivalent un-obfuscated design. As shown in the figure, significant change in gate count post-applying obfuscation is achieved thereby making it appear un-obvious or non-meaningful during inspection. Higher the number of gates transformed (i.e. change in the gate count), more is the obfuscation expected in the design, thereby more difficult it is for an attacker to interpret it functionally; thus ensuring higher security. For example, a very high increase in the gate count of the obfuscated FIR filter design compared to the baseline design is observed owing to the loop unrolling transformation being applied on it along with other successive transformations.

2.2. Design Cost Analysis

The impact of PSO based DSE on multi-key based structural obfuscation approach has been analysed in terms of design cost using eq. (11). Fig. 49 compares the cost of the obfuscation approach without PSO-DSE vs. the cost of the obfuscation approach with PSO-DSE, for different DSP cores. The design cost of multi-key based structurally obfuscated design with PSO-DSE module is achieved to be lesser than that without PSO-DSE module. This is because, PSO based DSE produces an optimal architecture (resource configuration) which is used to schedule the structurally obfuscated design. On overage, 6.58% reduction is achieved upon integrating multi-key based structural obfuscation approach with PSO-DSE process.

Further, Fig. 50 depicts the comparison of baseline cost with the cost of the multi-key based structural obfuscation approach with PSO-DSE module. Because of using optimal architecture obtained using PSO-DSE, lower design cost is achieved for the

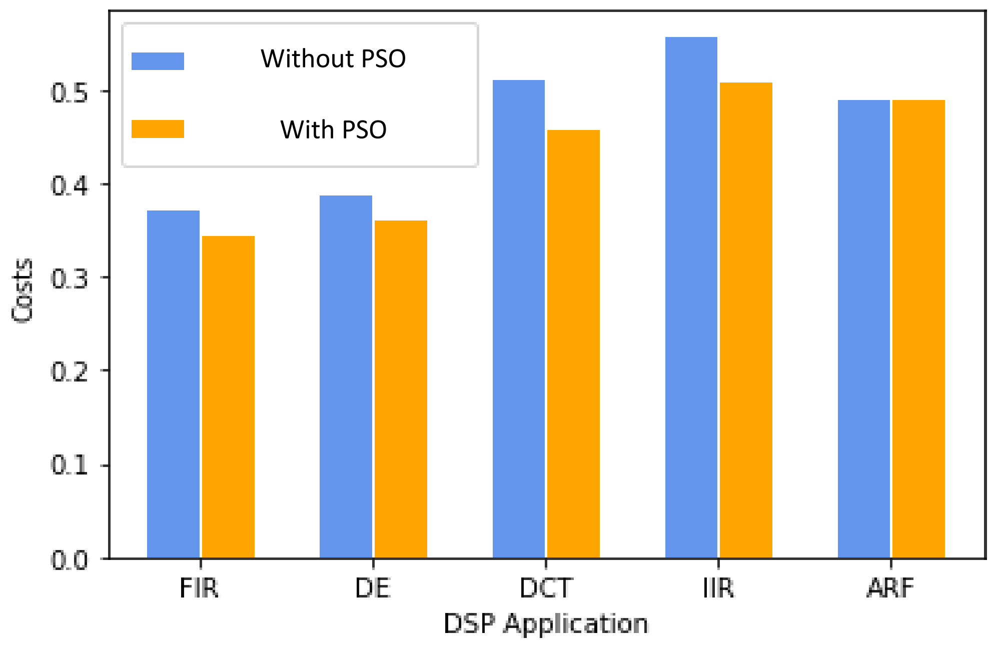


Fig. 49. *Design cost analysis of multi-key based structural obfuscation approach with and without PSO-DSE*

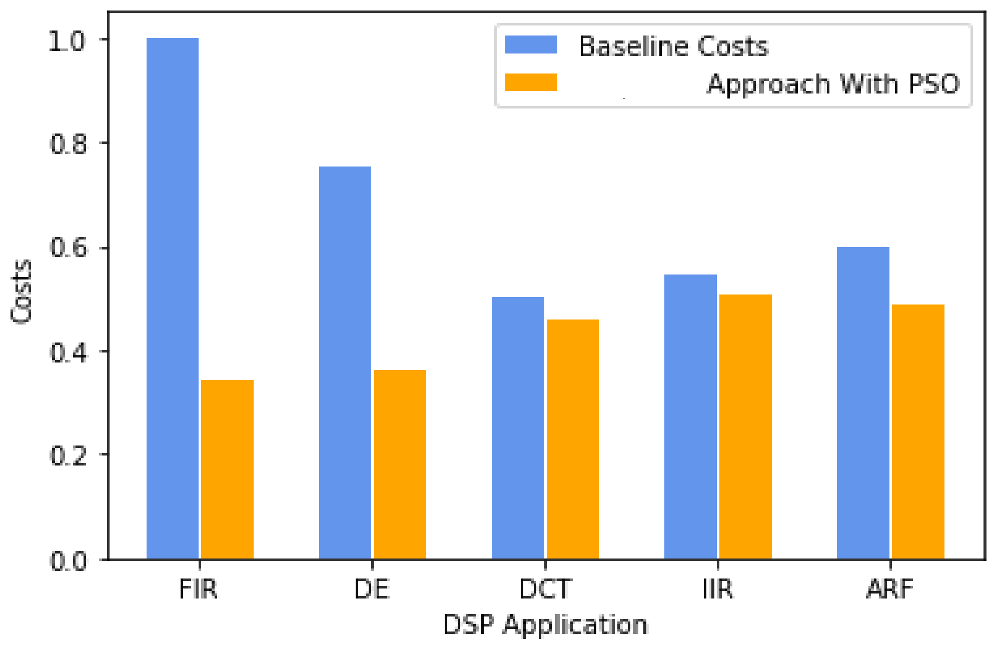


Fig. 50. *Design cost comparison of multi-key based structural obfuscation approach with the baseline*

obfuscation approach, compared to baseline design cost. A drastic reduction in design cost is achieved in case of finite impulse response (FIR) filter and differential equation (DE) applications as shown in Fig. 50. This is because in these applications, being loop-based, loop unrolling based transformation was applied which contributed to the huge reduction in delay. Since loop unrolling increases parallelization, therefore execution delay is decreased.

4.9. Conclusion

The design-for-security (DFS) aspect in the VLSI design process has become very important because of potential hardware threats such as Trojan insertion and piracy. This chapter discusses DFS using a double line of defence technique to offer enhanced security to DSP hardware accelerators. The first line of defence based on multiple SO-keys driven structural obfuscation provides preventive measure and the second line of defence based on physical level watermarking provides detective measure against the hardware threats. Because of employing multiple techniques of structural transformations, each driven through a key value, incurs huge obscurity in the design structure, hence resulting into a robust structural obfuscation. Since the employed high level transformation techniques also optimize the design structural, therefore the probability of incurring design overhead due to structural obfuscation is almost zero. Further, an author's watermark is embedded into early floorplan of obfuscated design during physical design process. The embedded watermark has a larger signature space because of comprising of multiple variables. Therefore the watermark is highly robust because it results into very low probability of coincidence, high tamper tolerance ability and high security against the removal attack. In addition, the embedding rules of physical level watermark are such that it does not result into design overhead. Additionally, PSO based extensive design space exploration has been applied to yield a low cost structurally obfuscated design with an average improvement of 6.58% in the design cost compared to the baseline.

At the end of this chapter, the following concepts are communicated to readers:

- Importance and applications of DSP hardware accelerators in electronics systems
- Hardware threats to DSP hardware accelerators
- Need of design-for-security in VLSI design flow
- A double line of defence based DFS technique to secure DSP hardware accelerators
- First line of defence using SO-keys driven multiple high level transformations based structural obfuscation
- Physical level watermark during early floorplanning
- Detection of physical level watermarking
- Demonstration of the process of employing multi-key based structural obfuscation
- Demonstration of the process of generating a watermarked floorplan of an obfuscated design by embedding author signature.
- Importance of PSO-DSE integration with the security algorithm.
- Obtaining optimal architecture using PSO-DSE to apply structural obfuscation based security.
- Security and design cost analysis of double line of defence for various DSP applications

- Security and design cost analysis of the low-cost multi-key based structural obfuscation approach with respect to baseline version.

4.10. Questions and Exercise

1. What is partitioning? What is the rule for partitioning?
2. How is key-driven loop unrolling performed?
3. What is key-driven folding knob based transformation?
4. What is double line of defense employed for hardware accelerators?
5. What are the inputs required for designing watermark implanted obfuscated DSP hardware accelerator?
6. How many secret keys are used for obfuscation and what is the role of each?
7. What impact does key-based structural obfuscation have on the RTL/gate-level design structure?
8. What is the concept of ‘early floorplanning’ and how is it useful?
9. In what sequence are the transformations in structural obfuscation performed? How would it vary if the transformation sequence is changed?
10. Explain the key-size of each secret keys used.
11. Demonstrate key-based loop unrolling on 16 tap FIR.
12. Demonstrate key-based partitioning on 16 tap FIR.
13. Demonstrate key-based THT on 16 tap FIR.
14. Derive the generic expression of 8-point DCT computation function.
15. How is the RTL circuit of key-based structurally obfuscation design generated?
16. What is the difference between folding factor and folding knob?
17. Explain encoding algorithm of physical level watermarking.
18. Explain physical level watermark detection algorithm. What are inputs required for this process?
19. How do you evaluate the total key-size of structurally obfuscated designs?
20. Explain the flow of low-cost optimized structural obfuscation process.
21. What is velocity clamping in PSO-DSE?
22. In the KSO-PW tool, what inputs are required? What is the output generated?
23. Give examples of DSP hardware accelerator applications where key-based loop unrolling is not applicable.
24. Give examples of DSP hardware accelerator applications where key-based THT is not applicable.
25. Give examples of DSP hardware accelerator applications where key-based ROE is not applicable.

References

E. Castillo, U. Meyer-Baese, A. Garcia, L. Parilla, A. Lloris (2007), ‘IPP@HDL: Efficient intellectual property protection scheme for IP cores,’*IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 15, no. 5, pp. 578–590.

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.

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,.

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.

D. Roy, A. Sengupta (2019), ‘Multilevel Watermark for Protecting DSP Kernel in CE Systems [Hardware Matters],’*IEEE Consumer Electronics Mag,* vol. 8(2), pp. 100-102.

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.

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.

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

X. Zhang and M. Tehranipoor (2011), ‘Case study: Detecting hardware Trojans in third-party digital IP cores,’ *IEEE International Symposium on Hardware-Oriented Security and Trust,* San Diego CA, pp. 67-70.

Y. Lao and K. K. Parhi (2015), ‘Obfuscating DSP Circuits via High-Level Transformations’ *IEEE Transactions on Very Large Scale Integration (VLSI) Systems,* vol. 23, (5), pp. 819-830.

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.

A. Sengupta, D. Roy, S. P. Mohanty and P. Corcoran (2018), ‘Low-Cost Obfuscated JPEG CODEC IP Core for Secure CE Hardware,’ *IEEE Transactions on Consumer Electronics*, vol. 64(3), pp. 365-374.

A. Sengupta, D. Roy, S.P. Mohanty, and P. Corcoran (2017), ‘DSP design security in CE through algorithmic transformation based structural obfuscation,’ *IEEE Trans. Consum. Electron.*, vol. 63, no. 4, pp. 467–476.

A. Sengupta and M. Rathor (2019b), ‘Protecting DSP Kernels Using Robust Hologram-Based Obfuscation,’ *IEEE Transactions on Consumer Electronics*, vol. 65, no. 1, pp. 99-108.

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.

L. Li and H. Zhou (2013), ‘Structural transformation for best-possible obfuscation of sequential circuits,’ in Proc. *HOST*, Austin, TX, , pp. 55-60.

RS Chakraborty, and S. Bhunia (2009), ‘Security against hardware Trojan through a novel application of design obfuscation,’ in proc. of the *International Conference on Computer-Aided Design, ACM*, pp. 113-116.

RS Chakraborty, and S. Bhunia (2011), ‘Security against hardware Trojan attacks using key-based design obfuscation,’ *Journal of Electronic Testing*, 27, no. 6, pp. 767-785.

A. Sengupta, D. Roy (2017), ‘Protecting an intellectual property core during architectural synthesis using high-level transformation based obfuscation,’ *Electronics Letters*, vol. (13), pp. 849-851.

V. K. Mishra, and A Sengupta (2014), ‘MO-PSE: Adaptive multi-objective particle swarm optimization based design space exploration in architectural synthesis for application specific processor design,’ *Elsevier Journal on Advances in Engineering Software*, 67, 111–124.

I. Hong and M. Potkonjak (1999), ‘Behavioral synthesis techniques for intellectual property security,’ in Proc. *DAC*, pp. 849–854.

A. Sengupta, M. Rathor, S. Patil and G. H. Naukudkar (2020), ‘Securing Hardware Accelerators using Multi-Key based Structural Obfuscation,’ *IEEE Letters of the Computer Society*, accepted.

A. Sengupta & M. Rathor (2020), ‘Enhanced Security of DSP circuits using Multi-key based Structural Obfuscation and Physical-level Watermarking for Consumer Electronics systems,’ *IEEE Trans. Consum. Electron.*, doi: 10.1109/TCE.2020.2972808.

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