Watermarking of Behavioral IPs: A Practical Approach

Jianqi Chen and Benjamin Carrion Schafer
The University of Texas at Dallas
Department of Electrical and Computer Engineering
jianchi.chen@utdallas.edu, schaferb@utdallas.edu

***Abstract*—This paper proposes a practical method to watermark behavioral IP (BIPs) for High-Level Synthesis (HLS), such that the watermark can be unequivocally retrieved at the generated RTL code, while being unremovable. The main approaches to watermark BIPs so far have focus on modifying the HLS process by e.g. introducing watermarking-aware scheduling or register binding algorithms. The main problem with these approaches is that they involve having full control over the HLS tool's internal behavior, which is not practically possible. Specifically, state-of-the-art HLS tools do not allow this type of controllability. Hence, these approaches are currently impossible to be implemented. On the other hand, commercial HLS tools make extensive use of synthesis directives in the form of pragmas. In this work we make use of these synthesis directives to assign operations in the source code to specific functional units given in the technology library in order to create the watermark. Experimental results show that our proposed method is effective in creating strong watermarks, while practical at the same time.**

I. Introduction

Design for Trust of Integrated Circuits (ICs) has emerged as an extremely important topic. Semiconductor companies were in the past vertically integrated and had therefore tight control over the entire design, manufacturing and distribution chain. However, this trend has shifted due to the large costs of setting their own production facilities. Most semiconductor companies are now fabless. They mainly focus on the architectural development of the Hardware (HW) and contract silicon foundries for the manufacturing process. This business model leads to these companies not having control over the manufacturing process, and hence, do not control e.g. how many ICs the foundry has actually manufactured and if the design has been maliciously modified to include backdoors (aka HW Trojan).

Much work in the hardware security community has focused on protecting the semiconductor companies from malicious alteration of third-party IPs (3PIPs) by untrusted vendors, or foundries, but comparatively little work has been done to protect the IP vendors from their IPs being illegally distributed. A study conducted by SEMI [1] highlighted that protection of IP rights is one of the most serious areas of concerns in the semiconductor industry. 90% of the companies that participated in the study reported having experienced some sort of IP violation. This includes infringement, counterfeiting, and technology theft. In 2012 the magnitude of the economic impact of IP violations in the electronic industry alone was estimated between $2 and $4 billion USD [2].

At the same time, most Integrated Circuits (IC) are now heterogeneous System on Chips (SoCs) comprised of multiple in-house developed IPs and 3PIPs integrated onto the same chip. It was suggested that complex SoCs will soon have over 30 dedicated accelerators [3]. In parallel to this, companies have also started to rely on High-Level Synthesis (HLS) to increase their design productivity mainly to design the hardware accelerators. HLS is a process that takes as input an untimed behavioral description and generates efficient RTL (Verilog or VHDL) that can execute it. The main steps in HLS are: (1) Resource allocation, (2) scheduling and (3) binding. In resource allocation, the behavioral description is analyzed, as well as a technology library that contains the area and delay of different functional units (FUs). This stage identifies the FUs in the technology library that are required to execute the different operations in the source code (i.e. additions and multiplications). Scheduling maps different portions of the code onto specific clock cycles based on the delay of the operations and the target frequency. Then, the binding stage assigns different FUs and registers to individual operations and variables in the source code. Finally, the RTL (Verilog or VHDL) is generated by creating an FSM that generates the control signals for every clock step and the datapath with all the FUs. This implies that every clock step generated in the scheduling stage is mapped to a specific FSM state.

So far, most approaches to watermark BIPs involve modifying the HLS process, in particular the binding stage such that a unique binding solution is generated. This approach leads to a good result as the search space grows exponentially with the number of registers in the design. Thus, being able to generate a unique solution serves as the watermark. The main problem with this approach is that HLS tools do not allow to control the register binding stage, and hence, this approach is not practical. Thus, practical methods are needed to allow BIP vendors to insert their watermarks in the generated RTL code. In summary, this paper makes the following contributions:

- Introduces a practical method to assign operations to specific functional units in a technology library through synthesis directives such that the generated RTL code can be unequivocally distinguished.
- Presents a watermarking method that takes a key as input and generates a unique RTL design using the key as seed for the watermarking method.

II. Motivational Example

Fig. 1 shows a motivational example. In particular, an example of a behavioral description that adds eight numbers together. The figure also shows a possible scheduling result.

978-3-9819263-5-4/DATE21/©2021 EDAA

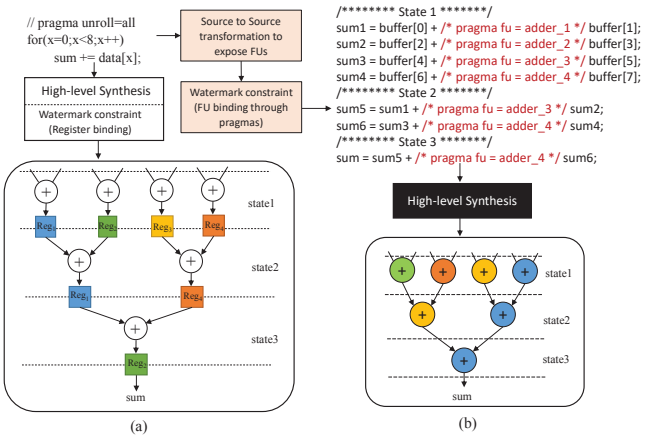

Fig. 1. Motivational example of sum of eight values (a) traditional approach and (b) proposed approach with HLS as black box.

Fig. 1(a) shows the traditional method to insert the watermark by modifying the synthesis process. In this case the register binding stage is modified such that the desired watermark is represented by a unique combination of registers used, where the same color registers represent shared register. As mentioned previously, this implies having to modify the HLS process. Fig. 1(b) shows our proposed approach that allows the use of any commercial HLS tool as it does not require to modify the synthesis process. The process is split into two phases. The first phase performs a source code to source code transformation in order to make all the operations in the code visible. Our proposed method then assigns individual FUs in the technology library to unique operations in the source code. One problem with this approach is that, as shown in the example, a single operation in the source code can lead to requiring multiple FUs. In this particular case, the for loop is unrolled, and one possible schedule is shown, where 4 adders are required, out of which two are re-used in cycle 2 and one in cycle 3. Thus, our proposed method has to be able to *see* all the additions in the source code to assign a unique FUs to them. Thus, a pre-processing step is required to *expand* the source code as shown.

To bind operations to FUs, out method relies on pragmas. Commercial HLS tools make extensive use of synthesis directives in the form of pragmas (comments). This allows designers to control how to synthesize loops (unroll or not), arrays (map to registers or RAM) and functions (inline or not) among others. One additional use of these pragmas is to map individual operations to specific FUs in the technology library. This implies that all of the operations have to be made visible based on the synthesis directives used in the source code. In this particular example, the user specified a pragma to unroll the loop. This leads to the expanded source code shown in Fig. 1(b) which has fully unrolled the loop making all of the additions now visible. As we will show later, one additional information that is needed is the clock cycle (FSM state) in which each operation is executed. This can be extracted from the synthesis report. Making use of this pragma, it is possible to generate a unique combination FU binding that serves as a watermark for the behavioral description.

III. Related Work

So far, the problem of IP trust has been approached from perspectives that are often impossible to be implemented in current VLSI design flows. This implies that companies have not adopted design for trust methodologies into their design methodologies.

Watermarking techniques have been proposed at nearly all design stages, starting from the algorithm-level [4], [5], behavioral-level [6]–[9], logic design [10]–[12] and physical design levels [13], [14].

Watermarking of behavioral IPs has been mainly addressed in three different ways. First, by inserting the watermark at the behavioral description itself before synthesis by e.g. setting unique DSP filter coefficients [4] or embedded codewords into DSP algorithms [5]. The main problem with this approach is that they are application dependent and can lead to significant overheads. Second, during the synthesis process, by e.g. modifying the register binding phase of the HLS process [6], [7], [9] or by modifying the logic to output a particular signature to the output ports during clock cycles in which the ports are unused [8]. This second family of watermarks require the modification of the HLS process. Finally, post-synthesis, as a post-processing step where e.g. buffers are inserted in the circuit [15], [16]. These techniques require deep knowledge of the circuit and often need manual design changes.

This work is different from previous work in behavioral IP watermarking as it is fully automated, application independent, and does not require to modify the internal mechanism of the synthesis process making it HLS tool agnostic.

IV. Proposed Method

A good watermark should exhibit the following properties [6]: (1) The designs functionality should not be altered by the watermark. (2) Low area, power, and net overheads. (3) The watermark should allow to unambiguously identify the owner. (4) It should be difficult to locate and (5) impossible to remove from the design. Finally, the watermark should be (6) easy to detect and (7) transparent to commercial EDA tools. Thus, watermarking at higher levels of abstractions are most beneficial because this already protects all consecutive tasks.

Fig. 2 shows an overview of the complete proposed flow including the inputs and outputs to each stage. The method is composed of four main steps. Step 1, makes all of the operations visible through a source to source transformation step. Then, step 2 synthesizes the new description in order to obtain the FU constraint file and the scheduling report that identifies which operations in the source code are scheduled in the same clock step. The method then continues by mapping individual operations in the source code to specific FUs in the technology library using synthesis directives based on the watermark given by the user using a set of predefined rules. Finally, the new instrumented description is synthesized (HLS). The details of our proposed method are as follows:

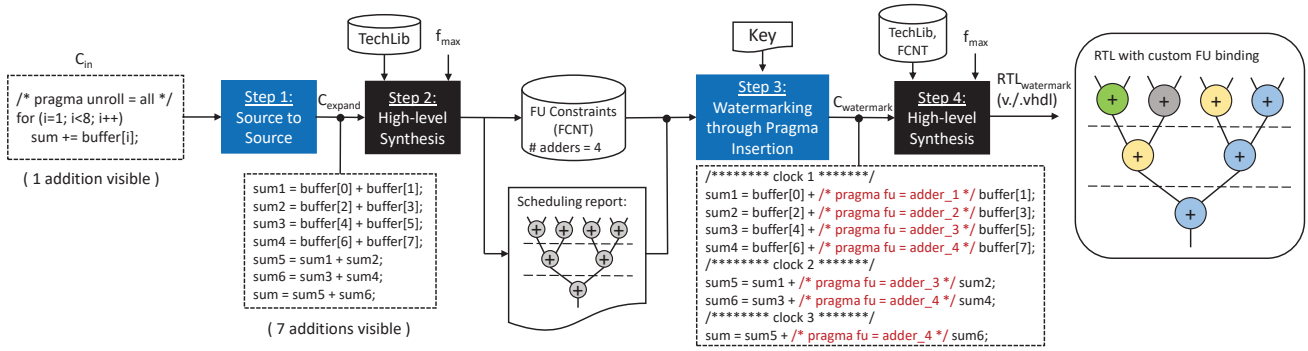

Fig. 2. Overview of the proposed method.

Step 1: Source to Source Transformation: This first step takes as input the behavioral description (C_{in}) to be watermarked and outputs a new behavioral description (C_{expand}) that makes all the operations that require a FU visible. As shown in the motivational example, a single operation in the C code can lead to multiple FUs in the generated RTL code. This is determined by pragmas inserted by the user in the behavioral description that forces the synthesizer to unroll loops completely, partially or not. Thus, this step parses the synthesis directives and outputs the new behavioral description based on this.

Step 2: HLS to Retrieve Scheduling Information: This next step takes as input the expanded behavioral description (C_{expand}) and synthesize it (HLS) in order to understand which operations are scheduled in which clock step. As mentioned previously, HLS assigns different parts of the source code to unique clock steps during the scheduling phase. This implies that operations scheduled in different clock steps can share the same FU. This depends on the delay of FUs, which basically, depends on the technology node targeted and the bitwidth of the FU. Based on the target maximum frequency selected by the user, the scheduler will assign more or less operations to the same clock cycle. HLS tools generate the RTL code, but also detailed reports. One of that reports is the scheduling report that indicates which lines of code are mapped to which clock cycle. In addition, the HLS tool generates a FU constraint file (FCNT) that specifies the types and number of FUs units used. The user can then manually overwrite this constraint file to generate smaller, but slower, or larger but faster circuits. The output of this step is therefore the scheduling report from the HLS process ($R_{schedule}$) and the FU constraint file ($FCNT$).

Step 3: FU binding for Watermarking: This step is the main watermarking insertion step. The key idea behind the step is to generate a very specific FU binding solution such that the generated RTL code is unique. Because the watermark is based on the binding of the FUs, it cannot be removed and is not detectable as it does not involve any additional hardware resources. Thus, making this a very strong watermark. This step, basically establishes the rules for the binding order of FUs to the operations in the C code.

The method takes as input the expanded C code (C_{expand}), the HLS scheduling report ($R_{schedule}$), the FU constraint file ($FCNT$) and the watermark key (key_{str}), and outputs a newly instrumented C code ($C_{watermark}$) with the pragmas that bind individual operations in the source code to FUs in the constraint file ($FCNT$). The main goal is to follow a specific sequence specified by the watermark key given by the BIP developer and to balance the usage of FUs such that the area of the muxes used to share the FUs is minimized. The watermark key is given as a string and is used as the seed that determines the order of the clock steps (FSM states) that will be bound first.

The FU binding algorithm is given in Algorithm 1 and the example used in the motivational section carried on to better illustrate the process shown in Fig. 3. The method starts by converting the given watermark, in this example the word 'watermark' (key_{str}) from text format into a big integer (key_{num}) by concatenating the ASCII code of all the characters (line 4 and Fig. 3(a)). This integer serves as the seed for the FU binding mechanism. The method then continues by extracting the state list (ST) and the FU list (FU) from $R_{schedule}$ and $FCNT$ file respectively. The key (key_{num}) is then used to re-order the FU list and state list by performing a modulo division key_{num} % FU and key_{num} % ST (line 5-15), and by consequently reducing the FU list and state list. This process is repeated until all the FUs and states are re-ordered as shown in Fig. 3. The new FU list (FU_{new}) and state list (ST_{new}) contain the order of the state that will be assigned to FUs first and the FUs that will be assigned first to the different operations in that state.

The proposed method then continues by binding FUs to the first state in the re-ordered state list ST_{new}. In the example shown in Fig. 3(c) this is state 2 bound to the first possible FU in FU_{new}, which in this case is add3. The main goal of this step is to balance the FUs usage such that the area of the muxes required to share the FUs is minimized (line 16-30). The process is repeated for all the states until all the operations in the source code have been bound to a particular FU.

Step 4: RTL with Watermark Generation: This last step takes as input the newly generated C code with the unique set of FU binding combination ($C_{watermark}$) and synthesizes it (HLS) to generate the watermarked RTL code (Verilog or VHDL) with a unique FU binding scheme ($RTL_{watermark}$).

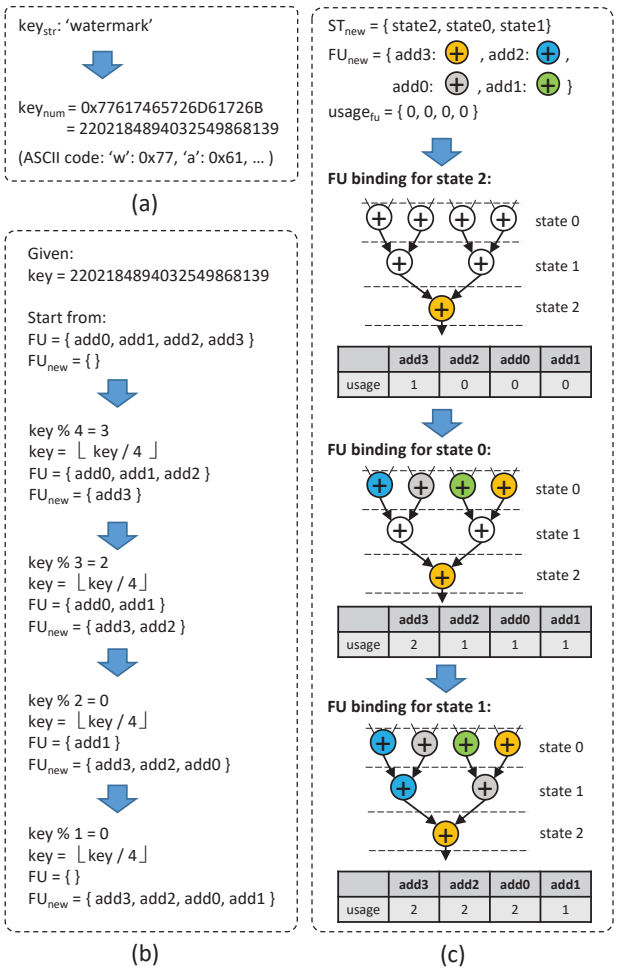

Fig. 3. Overview of watermarking insertion step (step 3). (a) ASCII code watermark key conversion into number. (b) Using key as seed for creating list of FUs and states that will be assigned first. (c) example of how FUs are bound in individual states.

Fig. 4 shows the schematic view of the two RTL circuits generated by the default FU binding and our proposed watermarked based binding solution using a commercial HLS tools [17]. We have labeled the FUs to highlight the differences. It can be observed that both circuits are very similar containing 4 adders (red components), but that these are used differently as shown by the interconnect, the registers and the muxes to which they are connected.

V. Security Analysis and Threat Model

The threat model used in this work assumes that the IP consumer can fully read the generated RTL code and can modify it to distort the watermark and claim ownership of the IP or insert his own watermark to claim ownership. The first case is extremely difficult because it would involve re-binding FUs at the RT-level. HLS tools carefully schedule each operation in each clock cycle and insert the exact amount of logic based on the delay of each operation in that particular clock cycle. Re-binding manually would most likely lead to

Algorithm 1: Watermarking through FU binding

input : $C_{expend}, R_{schedule}, key_{str}$
C_{expend}: the expanded C code
$R_{schedule}$: the HLS scheduling report
key_{str}: a string used as the key to generate watermark
output: $C_{watermark}$
$C_{watermark}$: C code with operations bound to FUs ∈ FCNT

```
/* Generate state list ST and functional unit list FU */
{ST, FU} = ReadInfo( C_expend, R_schedule )
/* Reorder ST and FU */
key_num = integer( key_str )
ST_new = Reorder( ST, key_num)
FU_new = Reorder( FU, key_num)
Function Reorder (list_old, key):
    list_new ← {}
    while list_old is not empty do
        index ← key % size( list_old )
        list_new.append( list_old[index] )
        remove list_old[index]
        key ← ⌊key / size( list_old )⌋
    end
    return list_new
/* FU binding */
usage_fu ← {0 for fu_i in FU_new}
for st_i ∈ ST_new do
    for op_j ∈ st_i do
        usage_min ← ∞
        for fu_k ∈ FU_new do
            if type( fu_k ) = type( op_j ) and
              usage_fu[k] < usage_min then
                select ← k
                usage_min ← usage_fu[k]
            end
        end
        usage_fu[select] ← usage_fu[select] + 1
        C_watermark.insert_pragma( op_j, fu_select )
    end
end
return C_watermark;
```

timing closure issues. Moreover, the attacker would need to re-bind the registers to which each FU writes its output and the FSM controller. Basically, having to re-do the entire circuit. Adding a new watermark would imply that the new RTL code has two watermarks. The original and the new one, thus, the IP owner can easily prove that its his IP.

One of the most important issues in watermarking is to prove that it is impossible, or very hard, for any designer to replicate the exact same FU binding. Proving that the problem grows superlinearly with the circuit size (number of FUs here) serves as that proof. For the proposed method, the search space is obtained by calculating all the possible combinations of FU binding to the each of the operations in each of the states as follows:

$$N = \prod_{st \in ST} \left(\prod_{fu \in FUtype} \binom{n_{fu}}{k_{fu,st}} \right) \quad (1)$$

where ST is the set of all the states, $FUtype$ is the set of all the types of available FUs, n_{fu} is the total number of a certain type of FU which is defined in FU constraint file ($FCNT$), $k_{fu,st}$ is the number of FUs needed in state st. For every state the number of different FU bindings is the product of combinations, and because it is independent for each state, the size of the whole search space is the product of every combination in each state. From this, it can be concluded that

TABLE I
MAIN CHARACTERISTICS OF A GOOD WATERMARK (WM) AND HOW OUR PROPOSED METHOD ADDRESSES IT.

Good Watermark Property	Addressed by proposed method
Design not altered	Watermark is part of the design not requiring additional HW
Low overhead	Results show that in many cases it leads to smaller area
Unambiguously identifiable	Examining the FUs used in each state identifies the watermark
Difficult to locate	WM cannot be located as it is part of the logic in the circuit
Impossible to remove	WN cannot be removed as the circuit would not work
Easy to detect	Looking at the FUs used in each state
Transparent to EDA tools	Commercial HLS tools provide option to bind operations to FUs

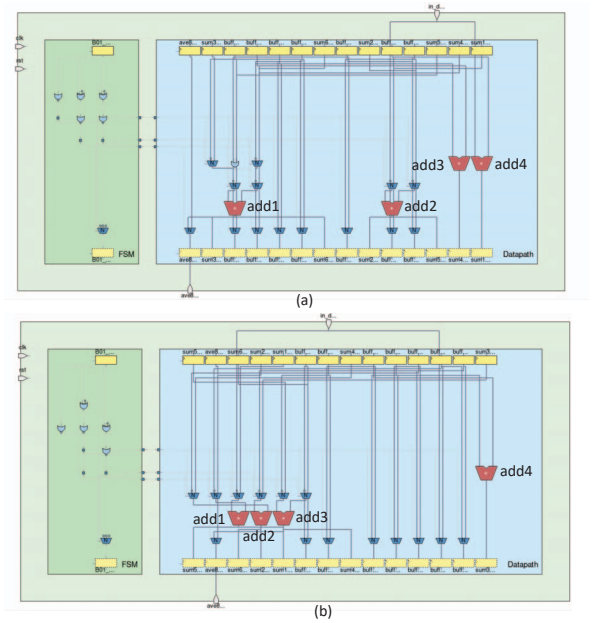

Fig. 4. Schematic view of synthesised example (a) using default FU binding and (b) watermarked binding

the search space even for the smaller benchmarks is very large and that it virtually impossible for someone to accidentally generate the exact same circuit.

Table I summarizes the main characteristics that a good watermark should display and how our proposed method addresses these. In particular the watermark is very easily retrievable. At the generated RT-level by visually inspecting the code, at the gate netlist level through logic simulation looking at the FSM control signals that enable the different FUs and at the post-silicon level through traditional post-silicon verification methods that allow to probe the same FSM signals that identify which FUs are being used in every FSM state.

VI. EXPERIMENTAL RESULTS

Six synthesizable SystemC benchmarks from S2CBench [18] are selected to test the proposed watermarking method. In particular, *fir* is a 16-tap FIR filter, *sobel* is an edge detection algorithm, *interp* is an interpolation filter, *adpcm* is an adaptive differential pulse code modulator, *decim* is a decimation filter, and *fft* is fast Fourier transform algorithm. The HLS tool used is CyberWorkBench [17] from NEC, and the target technology is Nangate 45nm technology. The target HLS frequency has been set in each case, such that

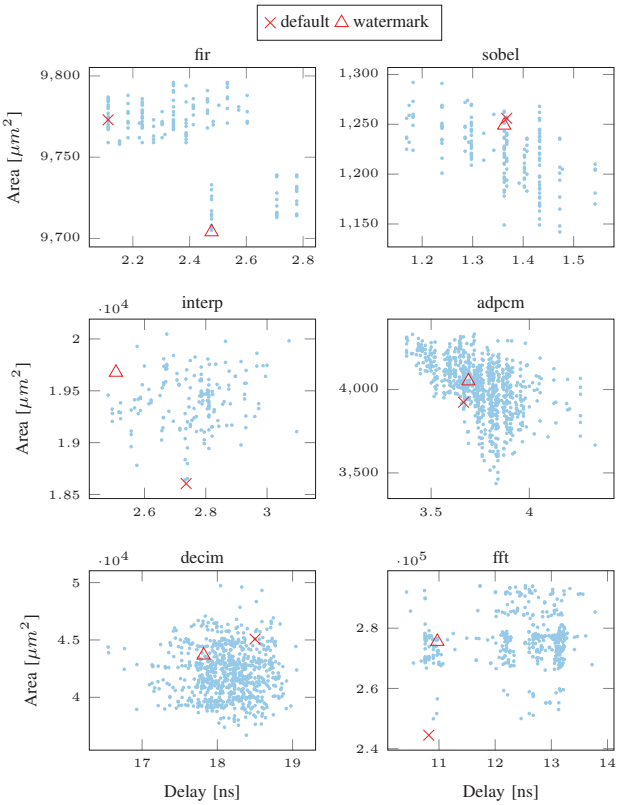

Fig. 5. Area vs. delay for the designs with different FU binding.

TABLE II
BENCHMARKS

Benchmark	# of lines	# of *op*	# of op_{unroll}	f_{target} [MHz]	# of states	search space
fir	62	3	23	400	3	137,998,080
sobel	94	5	13	667	3	598,950
interp	89	8	110	400	5	5.49e+64
adpcm	137	11	41	250	5	1.82e+10
decim	276	19	86	50	5	3.72e+30
fft	179	28	57	80	9	3.99e+36

no functional unit chaining happens. The main reason for this decision is to get comparable results across all of the benchmarks, although our proposed method can be applied to any target frequency. The experiments are conducted on an i7-6700 3.40GHz CPU with 16GB memory, running CentOS 7. The watermark key (key_{str}) in the synthesized circuit is the string 'watermark'.

Table II highlights the main characteristics of the benchmarks used in terms of lines of code, number of arithmetic operations (adders, multipliers and dividers) in the original code and once the loops have been unrolled (most likely way to synthesize any BIP to speed-up its execution), and the target synthesis frequency (f_{target}) used for each benchmark. The table also shows the number of states of the generated circuit (equal to the number of clock steps after scheduling), and on the last column, the total number of possible FU combinations for each of the benchmarks computed based on equation 1.

TABLE III
SUMMARY OF RESULTS

Benchmark	Simulated annealing Area [μm^2]	Simulated annealing Delay [ns]	Default Area [μm^2]	Default Delay [ns]	Watermark Area [μm^2]	Watermark Delay [ns]	Differences Δ_{Area} [%] SA vs. Default	Differences Δ_{Delay} [%] SA vs. Default	Differences Δ_{Area} [%] Watermark vs. Default	Differences Δ_{Delay} [%] Watermark vs. Default
fir	9,705	2.48	9,773	2.11	9,704	2.48	-0.72	18	-0.73	18
sobel	1,142	1.47	1,256	1.37	1,249	1.36	-9.1	7.3	-0.56	-0.73
interp	18,641	2.73	18,605	2.73	19,677	2.51	0.19	0.0	5.7	-8.1
adpcm	3,436	3.83	3,924	3.67	4,051	3.69	-12	4.4	3.2	0.54
decim	36,696	18.39	45,078	18.50	43,669	17.82	-19	-0.59	-3.1	-3.7
fft	249,947	10.90	244,485	10.82	275,564	10.97	2.2	0.74	13	1.4
Average							**-6%**	**5%**	**3%**	**1%**

Fig. 5 shows the area vs. critical path delay of the designs with different FU binding for each benchmark. In particular the figure shows the default FU binding generated by the HLS tools (red cross) and the FU binding when our watermarking technique is used. During the experimental results, it was found that the commercial HLS tool's default binding did not lead to the smallest area. Thus, we also implemented a simulated-annealing (SA) based search method that finds the binding solution that leads to the smallest possible circuit. The blue points in the graphs show the search space explored by the SA-based method, further highlighting the larger search space. All the solutions shown meet the specified maximum frequency. It should be noted that our proposed approach cannot be compared against previous works [6], [7], [9] because, as stated earlier, commercial HLS tools do not allow to control the register binding stage.

Table III summarizes the results when comparing the default synthesis solution, vs. the SA-based method that minimizes area and our proposed watermarking solution. The results in terms of overheads should be carefully interpreted, because it was observed that one of the reasons why the default HLS settings did not lead to the smallest circuit is that it also considers other design dimensions like routability, balancing the number of pin pairs between FUs.

From the results, it can be observed that our proposed method leads on half of the benchmarks to smaller and faster circuits. On average though, the proposed method leads to circuits 3% larger and 1% slower. The results also show that the default commercial HLS tools FU binding leads on average to solutions that are 6% larger but 5% faster than the SA-based solution

Based on this result, we can conclude that the proposed watermarking technique works well and that in many cases it can lead to even smaller and/or faster circuits than the default HLS synthesis.

VII. CONCLUSIONS

In this work we have introduced a practical method to watermark behavioral descriptions for HLS. The method is based on the use of synthesis directives in the form of pragmas that allow to bind unique operations in the source code to specific functional units in the technology library. To enable this watermarking approach, our proposed method needs to make all the operations in the source code visible and understand which of them are execute in the same clock step. Experimental results show that the watermarking scheme works well and that it can lead in many cases to circuits with smaller area and/or delay compared to the commercial HLS tool.

REFERENCES

[1] SEMI. (2008) Innovation is at risk: Losses of up to 4 billion annually due to ip infringement. [Online]. Available: http://www1.semi.org/en/innovation-risk-losses-4-billion-annually-due-ip-infringement

[2] ——, "Ip challenges for the semiconductor equipment and materials industry," online, October 2012. [Online]. Available: http://stg7.semi.org/sites/semi.org/files/docs/2012_IP_White_Paper_V2_SupAdd.pdf

[3] Y. S. Shao, B. Reagen, G. Wei, and D. Brooks, "The aladdin approach to accelerator design and modeling," *IEEE Micro*, vol. 35, no. 3, pp. 58–70, 2015.

[4] A. Rashid, J. Asher, W. H. Mangione-Smith, and M. Potkonjak, "Hierarchical watermarking for protection of dsp filter cores," in *IEEE Custom Integrated Circuits Conference*, May 1999, pp. 39–42.

[5] R. Chapman and T. S. Durrani, "Ip protection of dsp algorithms for system on chip implementation," *IEEE TSP*, vol. 48, no. 3, pp. 854–861, March 2000.

[6] I. Hong and M. Potkonjak, "Behavioral synthesis techniques for intellectual property protection," in *DAC*, June 1999, pp. 849–854.

[7] D. Kirovski and M. Potkonjak, "Local watermarks: methodology and application to behavioral synthesis," *IEEE TCAD*, vol. 22, no. 9, pp. 1277–1283, Sep. 2003.

[8] B. Le Gal and L. Bossuet, "Automatic low-cost ip watermarking technique based on output mark insertions," *Des. Autom. Embedded Syst.*, vol. 16, no. 2, p. 71–92, Jun. 2012.

[9] A. Sengupta, S. Bhadauria, and S. P. Mohanty, "Embedding low cost optimal watermark during high level synthesis for reusable ip core protection," in *2016 IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2016, pp. 974–977.

[10] S. Meguerdichian and M. Potkonjak, "Watermarking while preserving the critical path," ser. DAC '00. Association for Computing Machinery, 2000, p. 108–111.

[11] A. L. Oliveira, "Techniques for the creation of digital watermarks in sequential circuit designs," *IEEE TCAD*, vol. 20, no. 9, pp. 1101–1117, Sep. 2001.

[12] D. Kirovski, Yean-Yow Hwang, M. Potkonjak, and J. Cong, "Intellectual property protection by watermarking combinational logic synthesis solutions," in *ICCAD*, Nov 1998, pp. 194–198.

[13] A. B. Kahng, J. Lach, W. H. Mangione-Smith, S. Mantik, I. L. Markov, M. Potkonjak, P. Tucker, H. Wang, and G. Wolfe, "Constraint-based watermarking techniques for design ip protection," *IEEE TCAD*, vol. 20, no. 10, pp. 1236–1252, Oct 2001.

[14] R. D. Newbould, D. L. Irby, J. D. Carothers, J. J. Rodriguez, and W. Holman, "Watermarking ics for ip protection," *Electronics Letters*, vol. 38, no. 6, pp. 272–274, March 2002.

[15] A. K. Jain, L. Yuan, P. R. Pari, and G. Qu, "Zero overhead watermarking technique for fpga designs," in *GLSVLSI*, 2003, p. 147–152.

[16] Guangyu Sun, Zhiqiang Gao, and Yi Xu, "A watermarking system for ip protection by buffer insertion technique," in *ISQED*, March 2006, pp. 5 pp.–675.

[17] NEC, "Cyberworkbench v.6.1," 2015.

[18] B. Carrion Schafer and A. Mahapatra, "S2CBench:Synthesizable SystemC Benchmark Suite," *IEEE Embedded Systems Letters*, vol. 6, no. 3, pp. 53–56, 2014.