Locking the Re-usability of Behavioral IPs: Discriminating the Search Space through Partial Encryptions

Zi Wang and Benjamin Carrion Schafer
The University of Texas at Dallas
Department of Electrical and Computer Engineering
zi.wang5@utdallas.edu, schaferb@utdallas.edu

***Abstract*—Behavioral IPs (BIPs) have one salient advantage compare to the traditional RTL IPs given in Verilog or VHDL. The BIP can be used to generate RTLs with very different characteristics by simply specifying different synthesis directives. These synthesis directives are typically specified at the source code in the form of pragmas (comments) and control how to synthesize arrays (e.g. registers or RAM), loops (unroll or fold) and functions (inline or not). This allows a BIP consumer to purchase a BIP once and re-use it in future projects by simply specifying a different mix of these synthesis directives. This would obviously not benefit the BIP provider as the BIP consumer would not need to purchase the BIP again for future projects as oppose to IPs bought at the RT or gate-netlist level. To address this, this work presents a method to enable the BIP provider to lock the search space of the BIP such that the user can only generate micro-architectures within a specified search space. This leads to significant benefits to both parties: The BIP provider can now discriminate the BIP price based on how much of the search space is made visible to the BIP consumer, while the BIP consumer benefits from a cheaper BIP, albeit limited in its search space. This approach is made possible through partial encryptions of the BIP. Thus, this work presents a method that selectively fixes some synthesis directives and allows the BIP user to modify the rest of the directives such that the micro-architectures generated are guaranteed to be within the pre-defined search space.**

I. Introduction

High-Level Synthesis (HLS) has finally been embraced by most companies as part of their overall VLSI design flow. In particular, companies mainly use HLS to design their hardware accelerators. These accelerators execute dedicated tasks, orders of magnitude more efficient than general purpose solutions.

HLS takes as input an untimed behavioral description and converts it to efficient RTL (Verilog or VHDL). One key advantage of HLS compared to RTL VLSI design is that it enables the automatic generation of micro-architectures with unique design metrics without the need to modify the behavioral description. This is done by setting different mixes of synthesis directives in the form of pragmas (comments) at the source code. This allows designers to control how to synthesize arrays, loops and functions. Loops can be fully unrolled, partially unrolled, not unrolled or pipelined (folded). Arrays can be synthesized as registers, RAMs or expanded, and functions can be inlined or not. Setting different mixes of these pragmas leads to micro-architectures with unique area vs. performance and power trade-offs. Out of all these unique combinations the most important ones are the combinations that lead to Pareto-optimal designs (POD).

Raising the level of VLSI design abstraction to the behavioral level has created new opportunities for third party intellectual property (IP) vendors (3PIPs). These have started offering behavioral IPs (BIPs) to HLS users who constitute a much larger user base because they now also include algorithm developers as well as traditional VLSI designers. There are nevertheless new challenges associated with commercializing BIPs. One of the main challenge for BIP providers that is hampering the expansion of BIPs offering, is how to commercialize the IP without fully disclosing it. Traditional 3PIP vendors discriminate their IP pricing based on the amount of information disclosed. Thus, a 3PIP given in Verilog code is typically 10-100× more expensive than the same IP provided as a synthesized gate-netlist.

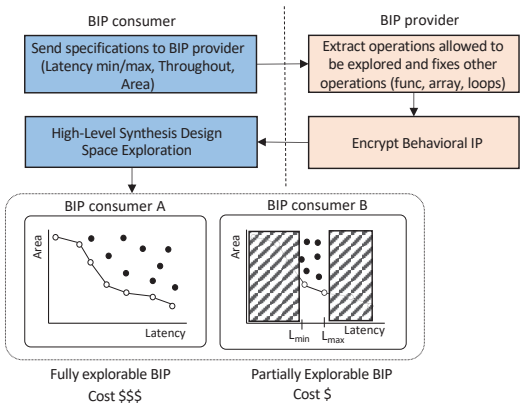

Fig. 1. Overview of Behavioral IP search space discrimination flow.

Although the BIP market is still in its infancy, we envision a similar price discrimination policy. Some commercial HLS vendors have started to enable the ability to encrypt behavioral descriptions that can in turn be synthesized with their HLS tool [1], [2]. This allows 3PBIP providers to control the amount of source code made visible to the BIP consumer, and hence, the cost of the BIP.

Selectively encrypting the BIP has the additional benefit of allowing the BIP provider to control the synthesis directives that the BIP user can set, as well as also the IO bitwidths. Thus, the cheapest option would be the fully encrypted BIP. This would allow BIP consumers to benefit from some of the advantages of HLS, including the generation of fast cycle-accurate simulation models, but this would severely limit the ability to perform HLS design space exploration as it does not allow the insertion of synthesis directives. The most expensive option would be the full disclosure of the source code which would allow the BIP user to maximize its re-usability.

To address this, this work proposes to create a framework that can automatically determine which synthesis attributes to fix and which ones to leave *explorable* in order to allow the BIP user to explore only a pre-defined search space range. The authors in [4] introduced this concept and did

978-3-9819263-5-4/DATE21/©2021 EDAA

some preliminary studies of the impact of encrypting the BIP on the search space.Fig. 1 highlights the envisioned flow. Two BIP consumers (A and B) contact the BIP provider each with different specifications in terms of minimum and maximum acceptable performance ranges (e.g. given in latency or throughput) or minimum or maximum area or power. The BIP provider, based on these constraints, fixes certain pragmas in the behavioral description and leaves other explorable. The modified BIP is then encrypted and sent to the BIP consumers who can then generate alternative micro-architectures only within the specified constraint range. Fig. 1 shows two exploration scenarios. In the first case the BIP consumer A can fully explore the search space, and hence, needs to pay more for the BIP, while in the second scenario, BIP consumer B, can only generate designs within the minimum (L_{min}) and maximum (L_{max}) specified latency. This limitation translates into a cheaper cost to consumer B. This scenario can be important for companies that have not fixed the overall system specifications and need some degree of flexibility in the generated RTL design, or need to generate RTLs within a given latency range but that also minimizes some other design metrics, e.g. reliability or temperature.

II. Motivational Example

The main motivation behind the proposed work is the observation that fully encrypting a BIP does not allow the insertions of synthesis directives, and thus, does not allow to explore the search space, while having full access to the BIP's source code implies that the BIP provider has to fully disclose the implementation details. This does not only limit the ability of the BIP provider to re-engage the BIP user in any future business, but also implies that the BIP can be illegally re-distributed to third parties. This suggests that fully disclosed BIPs have to be sold at much higher prices.

Fig. 2 shows a motivational example where a snippet of a program that computes the moving average of 16 numbers is used as an example. The figure shows that the code reads in a new value and computes its new average. Two pragmas can be inserted in the code. the first controls how the arrays that holds the 16 values to be averaged is synthesized (pragma1), while the second pragma controls how to synthesize the loop that adds the 16 numbers together (pragma 2). The actual values of the pragmas are stored in a header file (pragmas.h). This allows to decouple them from the source code.

The BIP provider can either encrypt the complete behavioral description or selectively encrypt portions of the source code. Different commercial HLS vendors allow different ways to do this [1], [2]. In the example shown in the figure, inserting an encryption start and end pragma allows to specify which part of the source code is encrypted. To encrypt the BIP commercial HLS tools provide an encryption tool for the BIP vendor and a unique encryption key which identifies that vendor. The encryption key is in turn used by the IP provider to encrypt the BIP. The HLS tools can then internally decrypt this behavioral description as it contains a database with all the legitimately issued keys. Although similar across

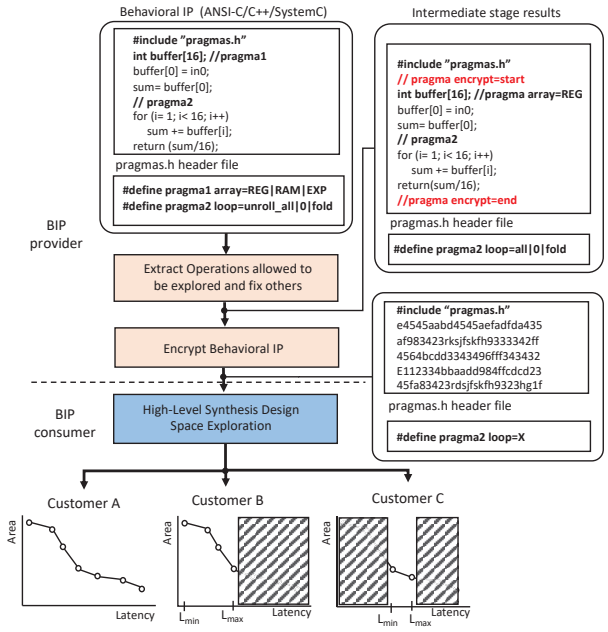

Fig. 2. Motivation for complete flow.

HLS vendors, this process is still tool and vendor specific. HLS tools that do not allow to encrypt the source code could rely on source code obfuscation. In source obfuscation a functional equivalent source file is generated, which is virtually impossible for humans to understand and extremely difficult to reverse-engineer.

As shown in Fig. 2, the BIP provider can now fix one of the synthesis directives, in this case the pragma to synthesize the array as a RAM, (pragma1 array=RAM), but leave the loop pragma unspecified. The array pragma is fixed in the main source code to be encrypted, while the loop pragma declaration is kept in the header file that will not be encrypted. The main behavioral description is in turn fully encrypted and the BIP then sent to the BIP consumer who can only modify the loop pragma.

The main goal is to be able to restrict the search space given a search space range (latency or area used in this work) by the BIP user (e.g. L_{min} and L_{max}), and thus, being able to discriminate the price that customer A, B or C pay for the BIP.

III. Proposed Framework

Fig. 3 shows an overview of the entire proposed flow that is composed of four main steps. The inputs to the flow are the BIP source (C_{IP}) created by the BIP provider and several constraints supplied by the BIP consumer. In particular, the search space of interest (SS_{int}) in either latency range or area range, e.g. $SS_{int} = [L_{min}, L_{max}]$ the target synthesis frequency (f_{target}) and the target technology library ($tech_{lib}$). The more restrictive SS_{int} is, the cheaper the BIP should be.

The flow then starts by performing a full HLS DSE on the BIP. For smaller BIPs a brute force search is possible, while for larger ones any of the heuristics introduced in the related work section can be used instead. The main goal of this first

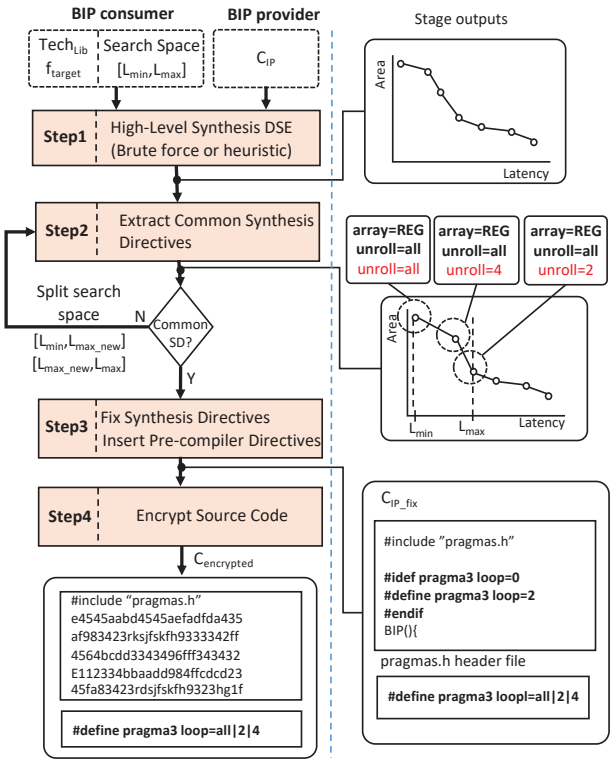

Fig. 3. Proposed flow overview.

step is to find the synthesis directives that lead to the Pareto-optimal designs (PODList). The method then continues by analyzing the exploration results and deciding which pragmas have to be fixed and to which ones can be left *explorable*. The BIP source code is then updated by fixing the synthesis directives and inserting some pre-compiler directives to limit the rest of the search space. This ensures that no designs outside of SS_{int} can be generated. The final step encrypts the BIP and outputs two files. The encrypted BIP ($C_{encrypted}$) and the unencrypted header file that allows the BIP consumer to modify the *explorable* operations. These steps are described in detail as follows:

Step 1: High-Level Synthesis Design Space Exploration: This first phase pre-characterizes the BIP's search space by performing a HLS DSE on the BIP using the target synthesis frequency (f_{target}) and technology library ($tech_{lib}$) provided by the BIP consumer. In this work we consider arrays, loops and functions. Thus, for every explorable operation in C_{IP}, our proposed method, exhaustively synthesizes (HLS) the BIP with all the possible synthesis directives that can be assigned to that particular operation or uses one of the previously developed heuristics for larger BIPs [3]. The method can also perform a pseudo-exhaustive exploration. This implies manually removing combination of pragmas that will never lead to a POD. The result of this step is a trade-off curve of Pareto-optimal designs $PODList$. Although time consuming this process is done offline.

Step 2: Extract Common Synthesis Directives: This second step is the core of the proposed flow. The method starts by extracting the Pareto-optimal designs (PODs) that are within the specified search space of interest (SS_{int}) from PODList, to $PODList_{SS}$. It then continues by comparing the synthesis directives that lead to these PODs and extracting the common subset of synthesis directives. The main ideas behind this method is to fix the synthesis directives (SD) that are common among all of the PODs $\in SS_{int}$, while allowing the BIP consumer to explorer the rest of the directives. This leads to two sets of directives, $SDList_{fix}$ and $SDList_{explorable}$.

In order for this method to lead to a feasible solution, all the PODs $\in PODList_{SS}$ need to share at least one common synthesis directive. If they do not share any directive, then it means that the desired search space is too large and cannot be restricted through pragmas. In this case the search space needs to be sub-divided into smaller search spaces. If the search space is too large, it groups the PODs with at least one common attribute together forming subgroups of PODs. The obvious disadvantage is that the BIP consumer would receive as many encrypted BIPs as unique search spaces are needed and would have to explore these separately

Fig. 3 shows an example of a BIP that can be explored by setting synthesis directives in three different operations. In particular, one array and two loops. In this case, three PODs are found in the desired search space between the L_{min} and L_{max} constraint. All three designs share the same synthesis directive for the array and the first loop (array=REG and loop=all). The third synthesis directive (second loop) differs in all three PODs. Thus, the search space can be controlled by fixing the first two pragmas and letting the BIP consumer to modify the third directive. This information is then passed to the third step that will automatically modify the BIP source code to encode this information into the BIP before it is sent to the BIP consumer.

Step 3: Fix Synthesis Directives and Insert Pre-Compiler Directives: This step separates the pragmas from the BIP source code through a header file. The pragmas.h header file contain the list of operations that the BIP consumer still can explore. In the example shown in Fig. 3, the second loop.

The important issue here is to limit the search space such that it guarantees that not micro-architectures outside of the desired range can be generated. A closer look to the example shown in Fig 3 reveals that the BIP user will be able to generate a new POD outside of the SS_{int} by setting the synthesis directive of the second loop to $SD_3 = \{unroll = 0\}$. This needs to be prohibited as the BIP consumer has not *paid* for that search space. Thus, any combinations of pragmas that lead to designs outside of the specified search space should not be allowed by out proposed method.

This can be achieved through pre-compiler directives that in this case are specified in the BIP source code that will be encrypted such that it cannot be removed later by the BIP consumer. Fig. 3 shows an example of how this looks like. A list of valid pragmas for the *explorable* operation is checked at

TABLE I
SUMMARY OF SEARCH SPACE RESTRICTIONS THAT OUR METHOD CAN IMPOSE USING ONLY POD AND POD + 5% AND 10% MARGIN.

		Only Pareto-optimal Designs (POD)		Pareto-optimal Designs + 5%		Pareto-optimal Designs + 10%	
Bench (BIP)	Total # PODs	# Largest Intervals	Avg # PODs/Interval	# Largest Intervals	Avg # PODs/Interval	# Largest Intervals	Avg # PODs/Interval
ave16	5	3	1.7	3	1.7	3	1.7
interp	3	2	1.5	2	1.5	2	1.5
VctAdd	6	2	3	2	3.0	2	3.0
CoTrans	9	3	3	2	4.5	2	4.5
iir	9	2	4.5	2	4.5	2	4.5
stdiv	13	4	3.3	4	3.3	3	4.3
decimation	15	4	3.8	3	5.0	2	7.5
fft	58	5	11.6	4	14.5	3	19.3
Avg.	**14.8**	**3.1**	**4.0**	**2.8**	**4.7**	**2.4**	**5.8**

compile time. If the user specifies any pragma option that leads to any design outside of the search space, then the pragma is forced to the closest valid pragma. In this example, if the user specifies $SD_3 = \{unroll = 0\}$, then SD_3 is forced to $SD_3 = \{loop = 2\}$. The synthesis directives that lead to POD outside of the valid search region are known, because the BIP provider has fully explored the search space in advanced and hence, can encode this information into the BIP.

Step 4: Source Code Encryption: This last step takes as input the newly instrumented source code of the BIP and encrypts it using the encryption tool of the target HLS tool. In case that the HLS tool does not have one, it can alternatively obfuscation the source code. The result of the complete flow is the encrypted BIP with limited search space that is sent to the BIP consumer ($C_{encrypted}$).

IV. EXPERIMENTAL RESULTS

Eight computationally intensive applications were taken from S2Cbench [5] to measure the effectiveness of our proposed method. These benchmarks represent the BIPs. The target technology used is Nangate open source 45nm and the target synthesis frequency set in all cases to 100Mhz (clock period of 10ns). Finally, the HLS tool used is CyberWorkBench v.6.1 [1] from NEC.

In these experiments we use the latency as constraint. To investigate the effectiveness of our proposed method, we first explore all of the benchmarks pseudo-exhaustively as described in the previous section. We then apply our proposed method with the objective of building the largest possible intervals. The main idea behind this is that larger regions can always be split into smaller regions by fixing a larger number of synthesis directives, while the largest intervals cannot be enlarged any further.

Three different sets of experiments are performed. In the first set, only the POD designs are used to build the explorable regions. In the second, we relax the constraint to include also micro-architectures that are within a 5% distance of any POD (5% Distance) and in the third case, we further relax this to allow micro-architectures within 10% distance of any POD (10% Distance). Basically, we use any designs within a 5% and 10% distance of any POD. The main idea is to investigate how relaxing the quality of the exploration influences the size of the explorable regions.

Table I shows the results. In particular, the total number of PODs found for each of the benchmarks. The table also shows the number of largest explorable regions found and the average number of PODs per interval when only PODs are allowed, PODs and micro-architectures within a 5% distance (POD+5%) and PODs and micro-architectures within a 10% distance (POD+10%).

From the results it can be observed that on average our proposed method can subdivide the total search space into 3.1 explorable regions, each with an average of 4.0 PODs when only the PODs are considered. In case that that quality constraint is relaxed, the number of intervals decreases to 2.8 and 2.4 on average, each with an average of 4.7 and 5.8 PODs for the 5% and 10% case.

In summary, these results show that selectively *hiding* synthesis directives provides a way for BIP providers to control the search space that a BIP consumer can explore. We do believe that this work could serve as a catalyst to the growth of the BIP industry in conjunction with generic encryption mechanisms that every HLS vendor supports.

V. SUMMARY AND CONCLUSIONS

This works illustrates the importance of synthesis directives in the form of pragmas (comments) for HLS design space exploration (DSE). Commercial HLS tools now provide methods for encrypting BIPs to protect BIP vendors from the unlawful use of these BIPs. The main problem is that encrypting the BIP severely affects the ability to explore the search space. The proposed work can be used to selectively control the search space and thus, lead to a new business model for BIP providers, selling BIPs that restricts the search spaces at a higher cost than BIPs restricted to a smaller search space.

REFERENCES

[1] NEC CyberWorkBench. (2017). [Online]. Available: www.cyberworkbench.com

[2] Mentor Catapult. (2017). [Online]. Available: www.mentor.com

[3] B. Carrion Schafer and Z. Wang, "High-level synthesis design space exploration: Past, present and future," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 1–12, 2019.

[4] Z. Wang and B. Carrion Schafer, "Partial encryption of behavioral ips to selectively control the design space in high-level synthesis," in *DATE*, 2019, pp. 642–645.

[5] B. C. Schafer and A. Mahapatra, "S2cbench: Synthesizable systemc benchmark suite for high-level synthesis," *IEEE Embedded Systems Letters*, vol. 6, no. 3, pp. 53–56, Sept 2014.